ISNM International School of New Media,
an affiliated institute of the University of Lübeck
Academic Director: Prof. Dr.-Ing. Andreas Schrader

Ocean: Towards Web-scale Context-aware Computing

A community-centric, wide-area approach for in-situ, context-mediated component discovery and composition

DISSERTATION
For fulfillment of requirements for the Doctoral Degree of the University of Lübeck
from the Faculty of Technology and Natural Sciences

Submitted by
Darren Vaughn Carlson, M.Sc.
From Coon Rapids, Minnesota, United States of America
Lübeck, May 2009

Erstberichterstatter:	Prof. Dr.-Ing. Andreas Schrader
Zweitberichterstatter:	Prof. Dr. rer. nat. Stefan Fischer
Vorsitz des Prüfungsausschusses:	Prof. Dr.-Ing. Alfred Mertins
Tag der mündlichen Prüfung:	15.07.2009

Author's Declaration

I hereby declare that the work presented in this dissertation, except as acknowledged in the text and references, is my own original work and that it has not been submitted for academic recognition or credit at this or any other university.

Darren Vaughn Carlson
Luebeck, May 2009

Abstract

Interrelated advances in data communication networks, distributed systems and mobile computing are rapidly altering the domain of network-based software. Today computing systems are no longer confined to conventional mainframe, enterprise and desktop scenarios. Rather, the emergence of powerful mobile devices, embedded systems and wireless computer networks enable software to operate across a broad range of non-traditional computing environments. Such advances are recognized as important foundations for creating mobile distributed systems capable of dynamically integrating environmental capabilities and accommodating changing user requirements. Towards this end, context-awareness has emerged as an important design approach for mediating the integration of algorithmic or structural components at runtime. However, while many everyday environments present unprecedented opportunities for adaptive systems, current context-aware approaches remain consigned to small-scale deployments and research prototypes; existing primarily within isolated islands of niche functionality that are far removed from everyday use.

Over the last decade, the explosive rise of the Internet and World Wide Web (Web) has resulted in a ubiquitous fabric of data communications and distributed computation. Importantly, modern Web architecture addresses many of the middleware requirements of large-scale networked systems by accommodating multiple trust domains, unanticipated load and independent component deployment. Moreover, the Web's low entry-barrier and non-proprietary standards have made its communication protocols, functional apparatus and device support ubiquitous. Further, the Web's underlying architectural model proven remarkably capable of accommodating a variety of problem domains, including sophisticated cross-domain component interoperation. However, despite the emergence of a vast ocean of contextually-relevant Web content and services, conventional Web architecture has proven difficult to exploit by existing context-aware systems.

This dissertation presents a hybrid computing approach, called Ocean, which aims to capture the entrepreneurial spirit of modern Web architecture as a means of supporting large-scale context-aware systems. Unlike existing approaches, Ocean addresses the key challenges facing real-world networked software by emphasizing user participation and community-based computation. Ocean defines a conceptual model for augmenting existing Web-based software components (Resources) with expressive contextual metadata as a means of facilitating in-situ discovery and integration. Further, Ocean defines a complimentary software architecture that provides simple, widely accessible and scalable mechanisms for distributed applications to discover and compose contextually-relevant Resources at runtime. Towards these ends, Ocean extends emerging community-centric computing techniques such as collaborative annotation, open plug-in contribution, volunteer-based computing and recommender systems. By leveraging community participation, Ocean aims to support the emergence of a new class of hybrid Web applications capable of dynamic context-aware adaptation.

Zusammenfassung

Die Fortschritte in Datennetzen, verteilten Systemen und Mobilkommunikation führen zu rasanten Veränderungen im Bereich Netzwerk-basierter Software. Heutige Rechenanlagen sind nicht länger auf traditionelle Großrechner-, Server- oder Desktop-Systeme beschränkt. Durch die Entwicklung von leistungsstarken Mobilgeräten, eingebetteten Systemen und drahtlosen Datennetzen kann Software auf einer großen Bandbreite von nicht-konventionellen Rechnersystemen eingesetzt werden. Diese Fortschritte sind die Grundlage für die Entwicklung von mobilen verteilten Systemen mit der Möglichkeit der dynamischen Integration von Umgebungsressourcen und der Anpassung an sich stetig ändernde Nutzeranforderungen. In diesem Zusammenhang hat sich Kontextsensitivität als ein wichtiges Gestaltungskonzept für die Vermittlung der Integration algorithmischer oder struktureller Komponenten zur Laufzeit entwickelt. Während jedoch viele alltägliche Umgebungen beispiellose Möglichkeiten für adaptive Systeme bieten, bestehen heutige kontextsensitive Ansätze aus relativ kleinen Installationen und Forschungsprototypen, die zumeist auf Insellösungen in Nischenanwendungen fern der täglichen Nutzung beschränkt sind.

Innerhalb des letzten Jahrzehnts hat die explosionsartige Entwicklung des Internets und des World Wide Web zu einer allgegenwärtigen Struktur von Datenkommunikation und verteilter Rechenleistung geführt. Die moderne Web-Architektur adressiert bereits viele der Anforderungen an hoch skalierte vernetzte Systeme durch die Anpassung an eine Vielzahl von gesicherten Bereichen, nicht vorhersagbarem Lastverhalten und der Verwendung unabhängiger Komponenten. Darüber hinaus haben der einfache Web-Zugang und die Verwendung nicht-proprietärer Standards zu einer universellen Verbreitung der Kommunikationsprotokolle, Funktionsbausteine und Geräte-Unterstützung geführt. Zudem hat sich das zugrundeliegende Architekturmodell des Webs als bemerkenswert geeignet für die Anpassung an eine Reihe von Herausforderungen erwiesen; insbesondere für die domänenübergreifenden Interaktion zwischen Komponenten. Trotz der Entstehung einer immensen Vielzahl kontextuell relevanter Inhalte und Dienste, hat sich allerdings herausgestellt, dass die konventionelle Webarchitektur nicht systematisch von existierenden kontextsensitiven Systemen nutzbar ist.

In dieser Dissertation wird das Ocean Framework als ein hybrider Ansatz vorgestellt. Ziel ist die Nutzung des Unternehmergeists im modernen Web zur Unterstützung von hoch skalierten kontextsensitiven Systemen. Im Gegensatz zu existierenden Ansätzen, adressiert Ocean die zentralen Herausforderungen von vernetzter Software in realen Umgebungen durch die betonte Einbeziehung der Nutzer und gemeinschaftlich bereitgestellter Ressourcen. Das Ocean Framework definiert ein konzeptionelles Modell für die Anreicherung existierender Web-basierter Softwarekomponenten mit ausdrucksstarken kontextuellen Metadaten zur Ermittlung und Integration von Ressourcen im jeweiligen Kontext. Zudem definiert Ocean eine ergänzende Softwarearchitektur, die einfache, weithin erreichbare und skalierende Mechanismen für verteilte Anwendungen bereitstellt, um während der Laufzeit kontextuell relevante Ressourcen zu entdecken oder zu erzeugen. Zu diesem Zweck erweitert Ocean aufkommende Techniken zur Unterstützung von Nutzergemeinschaften, wie gemeinschaftliche Kommentierungen, offene Schnittstellen für die Integration von Zusatzprogrammen, freiwillig bereitgestellte Rechenressourcen und Empfehlungssysteme. Durch die vorteilhafte Nutzung der Partizipation von Nutzergemeinschaften zielt Ocean auf die Unterstützung der Entwicklung einer neuen Klasse von hybriden Web-Anwendungen, die sich durch die Fähigkeit der dynamischen kontextsensitiven Adaption auszeichnet.

Acknowledgements

This dissertation could not have been possible without the generous support of several people. First, I would like to thank my primary advisor, Prof. Dr.-Ing. Andreas Schrader, for his encouragement, tireless support and his uncanny ability to ask the difficult questions that improved this research immeasurably. Next, I would like to thank my secondary advisor, Prof. Dr. rer. nat. Stefan Fischer, for his gracious assistance during my candidature. I am also grateful to Hans-Christian Fricke for assisting with the German translation of the abstract. Finally, I would like to thank my wonderful family for supporting me along the way.

In memory of my grandmother,
Marie Maude Montgomery.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Motivation

Interrelated advances in data communication networks, distributed systems and mobile computing are rapidly altering the domain of network-based software. Today computing systems are no longer confined to conventional mainframe, desktop and enterprise scenarios. Rather, the emergence of powerful mobile devices, embedded systems and wireless computer networks enable software applications to operate across a broad range of non-traditional computing environments. These advances are recognized as important foundations for devising mobile distributed systems capable of dynamically accommodating changing environmental factors and user requirements [73, 290]. Towards this end, context-awareness has emerged as an important design approach for mediating the integration of algorithmic or structural software components at runtime. Such capabilities are increasingly recognized as important for "dissolving the traditional boundaries for how, when, and where humans and computers interact" [208] and allowing computing systems to "weave themselves into the fabric of everyday life until they are indistinguishable from it" [353]. However, despite considerable interest and significant research effort, current context-aware systems remain consigned to small-scale deployments and research prototypes; existing primarily within isolated islands of niche functionality that are far removed from everyday use.

The ongoing development of isolated application scenarios remains an artifact of the ambitious goals common to many context-aware computing projects. Indeed, many of the first explorations in the field were developed before most of the enabling infrastructure was commercially available; requiring that researchers create and deploy the sophisticated computer networks, communication protocols and computing devices required for a given application scenario [345, 347]. A few early examples of context-aware applications included employee locators [2], telephone call routing systems [345] and techniques for seamlessly transferring running software applications to the nearest compatible terminal [267]. Along with a range of similar efforts, these early context-aware applications represented important explorations of computing systems that could autonomously change their behavior and functioning based on who or what was around them. Importantly, while many of these early projects existed as research prototypes, their intriguing results inspired a number of important advances in network engineering, mobile computing and distributed systems [109].

Throughout the last decade, many of the technologies anticipated by early context-aware projects have become commercial realities. Increasingly capable and inexpensive microprocessors, memory systems and secondary storage products have given rise to a vast array of mobile computing devices such as laptops, mobile phones and personal data assistants (PDAs). Further, accelerated by the explosive growth of the Internet and World Wide Web (Web), high-speed data communication networks are becoming increasingly ubiquitous throughout many everyday environments. In addition, technical advances such as the Global Positioning System (GPS), two-dimensional bar-codes, radio frequency identification (RFID) and a diverse range of sensor types allow commodity devices to detect and model a variety of useful contextual information [49]. Together, these interrelated advances

are providing researchers unprecedented opportunities for enhancing real-world mobile software with widely available context-awareness features.

For example, the Massachusetts Institute of Technology (MIT) Locale project[1] explores how context information can be used to automatically change mobile phone settings based on user-definable criteria such as location. Using Google's Android Mobile Platform[2] as a foundation, Locale allows users to define *situations*, which specify the conditions where a phone's settings should be automatically changed. For example, a user may define geographic locations around her workplace and a nearby movie theater where her mobile phone's ring mode will be automatically set to vibrate or silent respectively. In other locations, Locale can be used to allow incoming calls from specific groups of people; forwarding everyone else to voice-mail or another number. Locale users may also specify that wireless networking should be deactivated and screen brightness should be reduced when a mobile phone's battery power reaches 20% of capacity. Although many of these features have been previously explored in research prototypes, Locale is uniquely designed for widespread deployment on commodity hardware. An overview of MIT's Locale application is shown in Figure 1.

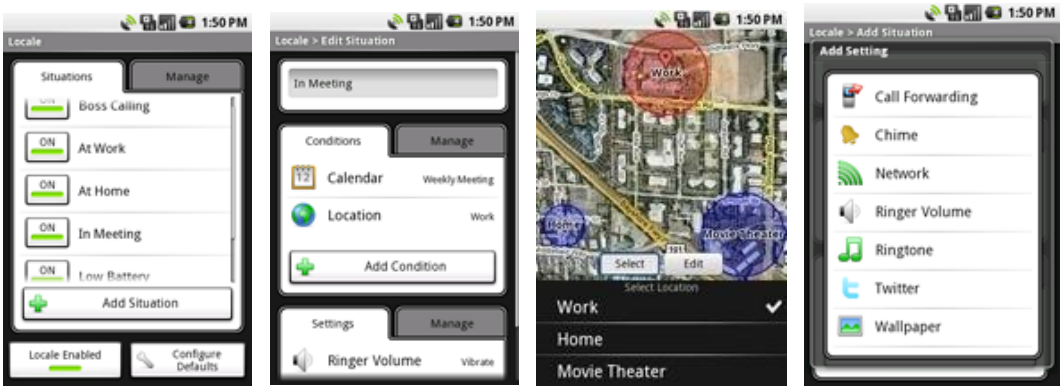


Figure 1: An overview of MIT Locale (from [168])

While applications such as Locale demonstrate the maturity of mobile computing and the viability of existing context-sources, advanced context-aware systems remain elusive outside of research laboratories and small-scale deployments [79]. Notably, advanced context-aware scenarios often involve *compositional adaptation*, whereby algorithmic or structural components are discovered and integrated into the application at runtime [208]. These types of adaptive approaches often combine mobile computing with key aspects of distributed and component-based systems, including techniques such as distributed object communications and late-binding. However, conventional distributed computing techniques often assume a relatively static execution environment, which are typically inappropriate for the rapidly changing environmental characteristic of mobile computing [62]. Unlike traditional distributed computing scenarios, where the available networks and distributed components are well-known a-priori, context-aware systems face a number of additional challenges that derive from the complex interactions often present at the intersection of multiple computing domains. Table 1 provides an overview of the key challenges facing context-aware computing and identifies related focus areas.

[1] http://www.androidlocale.com
[2] http://code.google.com/android/

Challenge	Focus area	Motive
Heterogeneity	Distributed systems	• Allowing a variety of services. • Supporting different types of devices, networks, systems, and environments.
Scalability	Distributed systems	• Enabling large-scale deployments • Increasing the number of resources and users
Dependability and security	Mission-critical and distributed systems	• Avoiding failures that are more frequent and more severe than acceptable. • Providing availability, confidentiality, reliability, safety, integrity, and maintainability.
Privacy and trust	Mobile computing	• Defining the trustworthiness of interacting components.
Spontaneous component interoperation	Mobile computing	• Allowing interaction with a set of components that can change both identity and functionality. • Permitting association and interaction.
Mobility	Mobile computing	• Providing application and data access anywhere, anytime. • Enabling the user environment to go along with the user.
Context acquisition and modeling	Context-aware computing	• Perceiving the user's state and surroundings. • Inferring and representing context information.
Context management	Context-aware computing	• Provisioning context information and modifying system behavior as a result.

Table 1: Key context-aware computing challenges (based on [73])

As discussed throughout Chapter 2, many of the challenges presented in Table 1 have been addressed in isolation by related work from a variety of disciplines. For example, distributed computing research has developed various techniques for distributing and coordinating processing across multiple autonomous networked computers. In such systems, the *heterogeneity* concerns listed above derive from the often diverse underlying hardware platforms, operating systems, runtime software and communication protocols common to networked execution environments. Further, in distributed scenarios, where networked components span multiple device types and may be provided by different vendors, the *dependability and security* of components and interactions become increasingly problematic [115]. Moreover, the broad scope of many distributed systems often requires the development of techniques for overcoming *scalability* issues (e.g. caching strategies, protocol

layering and functionality placement). However, as discussed in section 2.2.1, current distributed systems research provides a broad range of codified approaches that address many of these concerns, including the development of techniques such as remote communication, fault tolerance, high-availability, remote information access and security [288].

Similarly, mobile computing research investigates software systems designed for portable operation and wireless communications [109]. Notably, the portability of many modern devices implies effective resolution of several fundamental issues such as power management, interface limitations and resource constraints. Moreover, mobile networking places additional demands on issues such as network address migration, disconnected operation and proximate resource management. In many cases, mobile systems must also overcome network heterogeneity and significant security risks while simultaneously imposing fewer demands on users whose attention may be limited. In this regard, mobile environments often exacerbate the heterogeneity, scalability and dependability/security issues common to conventional distributed computing [109]. However, as discussed in section 2.2.3, current mobile computing research addresses many of these issues through wireless networking technologies; techniques for mobile information access; support for adaptive applications; and system-level energy saving mechanisms.

Finally, as discussed in section 2.3, issues related to context acquisition, modeling and management are being addressed by ongoing efforts in context-aware computing. Context-aware computing refers to a generalized approach for building software applications capable of examining and adapting to an individual's (often changing) context and requirements [293]. Such systems can be understood as an evolutionary synthesis of mobile and distributed computing, whereby additional aspects of adaptive computing are often involved [208]. However, unlike conventional adaptive computing scenarios, where participating applications and distributed components are well-known and relatively static over time, context-aware scenarios are often characterized by rapidly changing execution environments, where the available computational resources may fluctuate unpredictably as contextual factors change (e.g. the user moves). For example, many encountered networked entities may offer some measure of remotely accessible information or computation; revealing useful context information or offering clients the ability to discover and utilize various functionalities [169, 314, 342]. Additional variations include the "processors available for a task, the devices accessible for user input and display, the network capacity, connectivity, and costs may all change over time and place. In short, the hardware configuration is continually changing. Similarly, the computer user may move from one location to another, joining and leaving groups of people, and frequently interacting with computers while in changing social situations" [290].

As context-aware computing often involves complex aspects of distributed, mobile and adaptive computing, supportive middleware is well-recognized as an essential requirement for constructing non-trivial systems [147]. Broadly, context-aware systems acquire and model contextual information from the user's environment as a means of facilitating parameter or compositional adaptation within a domain-specific application. Foundationally, most context-aware middleware provides support for automated acquisition and modeling of context information from the user's physical environment as a mechanism of informing adaptation [145]. Moreover, similar to middleware for conventional distributed systems, middleware for context-aware systems often provide the necessary infrastructure

for coordinating communication between participating distributed components. As such, most context-aware approaches extend *existing* distributed computing models such as those common to enterprise scenarios. Examples of such models include Remote Procedure Call, Distributed Object Communications and Message-Oriented-Middleware (see section 2.2.1).

As detailed in Chapter 2, a variety of approaches have been developed to address the requirements of context-aware computing. While varying in overall scope, existing systems are typically designed with the assumption that the underlying network infrastructure, participating hardware devices, application constituents and context mechanisms are contained within a limited administrative domain and are well-known a-priori [97]. As such, many mandate expensive and invasive deployment of context instrumentation; require domain-specific network configuration; rely on specially outfitted mobile devices; adopt domain-centric middleware; and lack support for spontaneous *cross-domain* component interoperation (see section 3.2). Moreover, the considerable expense and effort required to devise, implement and deploy such systems promotes a top down development style intended to address niche problem domains where the requisite support infrastructure can be readily provided [62, 97]. While such approaches have been successful in small-scale deployments and research prototypes, many are inappropriate for large-scale, real-world environments. Indeed, several recent surveys [22, 62, 106] indicate that existing systems generally fail to provide ubiquitous accessibility and often incur intractable scaling costs that inhibit widespread reuse; resulting in a pronounced lack of developer adoption and end-user participation. Hence, current context-aware systems exist within isolated islands of niche functionality that are far removed from everyday use [79, 288].

1.2 Towards Ubiquity

While isolation is common in current context-aware systems, a variety of recent advancements are providing the foundations for improved ubiquity. For example, as detailed in section 3.2.1, while the deployment of dedicated context instrumentation is often infeasible for large-scale scenarios, many everyday environments are becoming increasingly saturated with existing sources of useful contextual information. Common examples include GPS signals, GSM cell tower identifiers, media access control (MAC) addresses, RFID tags, barcodes, accelerometer data, light intensity, etc. Related, modern mobile devices such as laptops, mobile phones and PDAs are increasingly capable of detecting such *ambient information* through the use of inbuilt hardware. In this regard, commodity devices are increasingly recognized as platforms for performing local context modeling and coordinating the actions of complex distributed systems [268].

In our previous context-aware computing approach, called Aladin [49], we explored how commodity devices can be used to acquire and model a diverse range of context information using local hardware and dynamically installed software plug-ins. Based on a select set of architectural abstractions, Aladin provides a client-centric foundation for developing context-aware systems capable of wide-area operations (i.e. operating across multiple administrative domains). To validate our approach, we developed three diverse application models based on the resultant Aladin Framework, including a mobile interactive cinema platform [50], a museum tour-guide system [49], and a pervasive multiplayer tangible game [148]. Additional related work indicates that such client-centric approaches can be effectively adapted to large-scale, heterogeneous environments [139, 268].

Given cross-domain context acquisition and modeling, large-scale context-aware systems presuppose a foundation of network accessibility encompassing the intended operational area [73]. In this regard, current Internet infrastructure represents an increasingly ubiquitous substrate of data communications. The success of the Internet's architecture is often attributed to several key internetworking techniques [341, 358]. First, the Internet is based on a single logical addressing scheme that is universal adopted. Second, modularity is promoted through the five-layer TCP/IP protocol suite (i.e. the hourglass model), which features "a generic packet (datagram) delivery mechanism as a separate layer at the hourglass' waist" [358]. Finally, Internet architecture promotes scalability and extensibility by placing "most 'intelligence' (i.e., information about end-to-end communication, control) squarely in the end points" [358]. As described in [223], these techniques affirm that the Internet's "goal is connectivity, the tool is the Internet Protocol, and the intelligence is end to end rather than hidden in the network." In terms of promoting large-scale networked systems, the Internet's basic design philosophy has proven remarkably effective at supporting "constant innovation and entrepreneurial spirit at the physical substrate of the network as well as at the application layer" [358].

Related, conventional Web architecture addresses many Internet-scale application requirements through a distributed middleware model derived from a "coordinated set of architectural constraints that restricts the roles and features of architectural elements, and the allowed relationships among those elements" [105]. As discussed in section 3.2.6, the Web's principle constraints include a universal addressing scheme; connector layering; intermediary caching (via metadata and idempotent methods); the uniform interface; and stateless component interactions [106]. Importantly, the Web's architectural model supports key aspects of visibility, reliability, and scalability. As richly described in [106], "visibility is improved because a monitoring system does not have to look beyond a single request datum in order to determine the full nature of the request. Reliability is improved because it eases the task of recovering from partial failures [176]. Scalability is improved because not having to store state between requests allows the server component to quickly free resources, and further simplifies implementation because the server doesn't have to manage resource usage across requests."

Based on its underlying architectural model, the Web has inarguably achieved its original design goal of providing "a shared information space through which people and machines could communicate" [33]. However, its popularity has also resulted in a number of ancillary effects that have important implications for large-scale context-aware computing. First, the Web's design addresses scalability beyond geographic dispersion by incorporating techniques to accommodate multiple trust domains, unanticipated load and allow independent component deployment. Second, as evidenced by the tremendous number of Web-based applications, the Web's application model has proven remarkably capable of accommodating a variety of application domains. Third, the distributed architecture designed to support the Web's application model is sufficiently flexible to support a variety of non-hypermedia application scenarios [266]. Fourth, the Web's low entry barrier and non-proprietary standards have made its communication protocols, functional apparatus and device support ubiquitous. Fifth, the Web's increasing ubiquity has resulted in significant developer adoption; promoting the emergence of a broad array of development toolkits, application frameworks and related knowhow. Sixth, increasing developer adoption has resulted in an explosion of Web-based

information and computation. Seventh, the generality of the Web's underlying architecture, combined with the rise of open data exchange formats (e.g. XML) is rapidly enabling machine-based processing of Web-based computation. Finally, the proliferation of machine processable Web-based computation is increasingly recognized as often having semantic associations to real-world contexts [179].

Based on the preceding observations, Internet infrastructure and Web architecture ostensibly represent compelling foundations for the development of large-scale, real-world context-aware systems; however, these technologies have proven difficult to exploit in traditional context-aware scenarios [97]. As described in section 3.2, the current lack of Internet-scale systems highlights the fundamental conflicts arising between the scope of current context-aware research and the requirements of large-scale network architectures. For example, by emphasizing connectivity and the end-to-end principle of system design [279], Internet infrastructure reveals very little of the contextual semantics required by context-mediated adaptation strategies. In this regard, potential context information – such as information regarding the underlying communications hardware, network topologies and the physical location of components – is intentionally hidden from end-systems as a means of providing "the illusion of a single, seamlessly connected network where the fragmented nature of the underlying infrastructure and the many layers of protocols remain largely transparent to the user" [358]. Further, conventional Web architecture provides only limited forms of context-mediated component discovery (based on its hypermedia application model) [113, 114, 357]. Problematically, in current Web architecture, context information such as location, proximate devices and inferred activity cannot be effectively used to mediate component discovery, selection and interoperation. This is unfortunate, since much of the information and computation on the Web is increasingly recognized as having semantic relationships to real-world environments [37, 177, 182].

1.3 Thesis Statement

This dissertation contends that traditional context-aware computing approaches are ill-suited for building truly ubiquitous networked systems and generally fail to promote significant developer adoption and end-user participation. We posit that the increasingly rich sources of context information contained within everyday environments provide a foundation for cross-domain context acquisition and modeling. Further, we argue that the ubiquity of Internet infrastructure and Web architecture represent compelling foundations for the development of new classes of context-aware Web applications capable of in-situ, cross-domain component discovery and interoperation. We suggest that extending existing infrastructure will enable developers to employ existing knowhow and tools to create large-scale, context-aware systems without the need for prohibitive infrastructure. Finally, we argue that increasing developer adoption will lead to significant end-user participation; allowing for the application of community-based computation as a means of addressing the key challenges facing large-scale network architectures.

1.4 Research Issues

Based on the motivating factors presented in section 1.1 and the trends introduced in section 1.2, we aim to develop a context-aware computing approach that captures the entrepreneurial spirit of modern Web architecture as a means of supporting large-scale context-aware systems. As detailed in section 4.3.1, distributed components in conventional Web architecture are represented by an architectural

abstraction known as *Resource*. As described in [105], any information that is important enough to be named can be modeled as a Resource (e.g. an image, newsfeed, software release, Web page, search result, etc.) Based on the Resource abstraction, Web architecture has been designed primarily to support the requirements of an Internet-scale distributed hypermedia application [105, 106, 341]. As such, conventional Resources provide very little information regarding their potential associations to real-world contexts. Rather, in the current Web model, the selection of components for runtime composition is based on two limited forms of context-mediation: Resource-mediated and metadata-mediated (see section 4.2). Briefly, in the Resource-mediated form, context information such as Web page text and associated hyperlinks is delivered to users encoded within a Resource's Representation. In the metadata-mediated form, machine-processable information such as HTTP headers and HTML metadata tags are sent to Web agents alongside requested Representations. Together, these basic context-mediation techniques allow the Web's hypermedia model to act as an "engine of application state" [105] where dynamic Resource composition is based on user-directed hyperlink navigation.

While Web-based context-mediation supports the requirements of hypermedia applications, its transactional nature prevents the component pre-filtering required by compositionally adaptive systems [29, 208]. Problematically, the Web's hypermedia model mandates that Resource Representations are requested and consumed *before* context information can be extracted from them. For example, in a prototypical Web page interaction, embedded hyperlinks are used to trigger additional application states. Importantly, transitions to new application states *precede* the delivery of additional context information (e.g. HTTP headers and Web page text); limiting available components to those encoded within delivered Representations. In contrast, complex compositional applications rely on the *pre-filtering* of potential application constituents before composition occurs [21, 29, 192, 231]. Although techniques have been developed to provide some measure of Web-centric component pre-filtering [82, 192, 289], existing approaches lack support for expressive component descriptions based on arbitrary context types; require significant context domain expertise on the part of application developers; and often mandate computationally expensive translation of native context data into low-fidelity intermediary formats. Hence, in order to provide a foundation for Web-centric context-aware computing, extensible and expressive techniques for Resource contextualization are required.

Large-scale context-aware systems presuppose distributed middleware that is ubiquitous, highly scalable and supportive of a wide variety of problem domains [38, 323]. Importantly, real-world distributed applications must be capable of operating across multiple administrative domains, and "continue operating when subjected to an unanticipated load, or when given malformed or maliciously constructed data, since they may be communicating with elements outside their organizational control" [106]. In this regard, cross-domain operation implies that application constituents may be developed, deployed and managed by multiple external organizations. Additional considerations include the scalability of component interactions, generality of interfaces, independent component deployment, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems [105]. Finally, any widely-accessible context-aware application model must be capable of accommodating a wide range of developer skill-levels. While conventional Web architecture addresses these requirements, its application model does not inherently support the

rich contextual semantics required by context-aware applications. Hence, in order to support Web-scale, real-world context-aware systems, ubiquitous middleware approaches must be developed to support scalable component discovery and interoperation based on real-world context information.

Given a sufficiently ubiquitous middleware approach, cross-domain context-aware systems must be able to discover and interoperate with relevant distributed components at runtime without extensive prior knowledge [96, 180]. As described in section 2.2.1, traditional distributed component architectures rely on *process-centric* descriptions of a component's end-point addresses, available methods and associated data-types (using technologies such as Corba IDL or WSDL). However, although modern development environments simplify the creation of remotely accessible methods, the resultant proliferation of domain-specific interfaces can reduce the probability of component interoperation [225, 334, 336]. Several recent surveys [21, 62, 97] indicate that the majority of current context-aware systems adopt such process-centric interoperation (PCI) styles. PCI techniques such as those epitomized by Corba and SOAP Web Services, make the coupling between callers and components clear and unambiguous; however, in large-scale scenarios, interactions between distributed components may suffer from architectural mismatch [115], where domain specific method syntax, sequencing and semantics prevent widespread reuse due to a lack of widespread understanding of a given interface [334]. While dynamic interactions between specialized interfaces can be resolved in small-scale distributed systems (e.g. enterprise scenarios) they become problematic in larger scenarios where component interfaces cannot be known a-priori [334]. Moreover, PCI techniques rely on complex infrastructure, highly skilled developers, platform specific mobile code and significant tooling, which are recognized as antithetical to widespread developer adoption [145].

Given an effective approach for spontaneous, cross-domain component interoperation, techniques must be developed to support *wide-area* component contextualization and discovery. In this regard, two approaches for service-discovery are commonly used. First, proximate interactions between networked systems are commonly accomplished using *service discovery protocols* such as the Service Location Protocol (SLP) [135], Zero Configuration Networking (Zeroconf) [314], Universal Plug and Play (UPnP) [169] and JINI [342]. While differing in scope and approach, service discovery protocols are generally designed to facilitate service advertising (e.g. publishing a service's capabilities and interfaces via multicast) and service discovery, whereby appropriate components can be located and selected based on desirable characteristics. However, as described in section 2.3.5, while some service discovery protocols have achieved limited commercial adoption (notably Zeroconf and UPnP), most are insufficient for supporting wide-area scenarios due to limitations in service density, network accessibility (i.e. local-link constraints) and protocol interoperability.

In wide-area service integration scenarios, component registries are typically used to mediate dynamic binding and interactions between the loosely-coupled elements of a distributed application [29, 308]. Within component registries, associative metadata are used to describe important attributes of distributed constituents such as addressing information, interface descriptions and supported data types. Distributed applications discover potential application constituents by querying the component registry using a search protocol. The registry uses incoming queries to perform a component lookup using previously created metadata as a filtering mechanism. Although some component registries have begun to address context-aware component discovery, a number of inherent problems have

prevented their widespread adoption (see section 4.2). Briefly, most approaches adopt the previously introduced PCI style, which is recognized as inhibiting cross-domain spontaneous component discovery and interoperation [225, 334, 336]. Next, most context-enhanced registries define intermediary metadata formats, which may not be known be all participants or capable of expressing the fidelity of native context information. Next, most registry architectures provide a restricted set of metadata types and do not provide for the contribution of new types by external domain-experts. Finally, existing registry architectures do not provide the contribution mechanisms necessary for creating and maintaining the vast amounts of semantic metadata required by context-aware Web-based scenarios.

In wide-area component interoperation scenarios, context-aware applications may encounter situations where constituent discovery result in sub-optimal results due to a lack of sufficient query terms. Notably, if the metadata used to contextualize distributed components differs significantly from the query terms provided by client applications, search effectiveness is diminished and contextually-relevant components may remain invisible. In information retrieval (IR), a similar phenomenon, known as *word mismatch*, has been studied with regard to text-based search. Word mismatch refers to the situation where "the users of IR systems use different words to describe the concepts in their queries than the authors use to describe the same concepts in their documents" [374]. As described by Furnas et al., people use the same words to describe a search object less than 20% of the time [113]. Section 7.2.2 describes a similar phenomenon, called *context mismatch*, which refers to the situation where context-aware applications may not be capable of generating the context information necessary to facilitate effective component discovery. Notably, in real-world scenarios, the context data modeled from the user's environment may be highly heterogeneous, unsystematically organized and unpredictably available. Furthermore, the context data available to a given application is often inherently limited to the capabilities of the executing mobile device [49, 268]. In IR research, *query expansion* techniques have been developed to help improve query results by augmenting impoverished queries with supplemental search terms [51]; however, query expansion has not been effectively applied context-aware component discovery.

Due to the rapidly increasing number of networked devices and components, *information overload* represents another critical challenge for Web-scale context-aware applications. Similar to Web search scenarios, where common query terms may result in an overwhelming number of search results [51, 205], component discovery may result in an overwhelming number of relevant results; making effective selection more difficult. Importantly, information overload can become increasingly problematic in information-saturated environments, such as popular tourist locations or dense urban environments [51]. Recently, personalization techniques such as *recommender systems* have emerged as a promising approach for reducing information overload in complex filtering and selection scenarios [264]. The principle objective of a recommendation system is to help users select relevant items from among a large set of similar items by generating suggestions or predicting the utility of a specific item for a given user [339]. Recently, context information has been used within recommender systems as a means of helping users filter and find useful information such as providing a list of nearby restaurants according to a model of the user's preferences [207]. Preliminary results have shown how context-aware systems can be used to enhance service discovery and provisioning in

mobile scenarios [269]; however, recommender algorithms have proven difficult to exploit in context-aware systems due to their comparatively low end-user participation [207].

Based on the research challenges presented above, this dissertation presents a hybrid context-aware computing approach, called Ocean, which aims to capture the entrepreneurial spirit of modern Web architecture as a means of supporting large-scale, real-world context-aware systems. Unlike existing approaches, Ocean addresses the key challenges facing large-scale networked systems by emphasizing user participation and community-based computation. Ocean defines a comprehensive conceptual approach for augmenting existing Web-based software components (Resources) with expressive contextual metadata as a means of facilitating in-situ discovery and integration. Related, Ocean defines a complimentary software architecture that provides simple, accessible and scalable mechanisms for distributed applications to discover, select and compose contextually-relevant Resources at runtime. Towards these ends, Ocean extends several community-centric approaches such as collaborative annotation, domain expertise contribution, volunteer-based context modeling and recommender systems. By reappropriating existing context sources and leveraging community participation, Ocean aims to support the emergence of new classes of hybrid context-aware Web application capable of dynamic, cross-domain component discovery and interoperation.

1.5 Dissertation Structure

Based on the motivation presented in the preceding sections, Chapter 2 substantiates our approach by presenting relevant background material and related. Section 2.2 begins with an overview of relevant computing approaches and an introduction into data communication networks. Section 2.2.1 discusses several key distributed computing approaches and section 2.2.2 describes the application of these approaches in service oriented architectures and Internet-based Web service scenarios. Next, section 2.2.3 discusses how increasing device mobility has motivated the development of adaptive mobile systems that often confound traditional distributed systems techniques. Based on this background, section 2.3 begins a discussion of context-aware computing. We begin by introducing a conceptual framework intended to guide the discussion of a representative notable state-of-the-art systems. Section 2.3.1 explores key notions of context and discusses how context-awareness has become well-recognized as an important mechanism for guiding software adaptation in complex computing environments. Section 2.3.2 describes current approaches for context acquisition. Section 2.3.3 describes key methods of context modeling and representation. Section 2.3.4 describes major techniques for context management and provisioning. Finally, section 2.3.5 describes approaches for context-aware component interoperation.

Based on the background and related work presented in Chapter 2, the foundations of the Ocean approach are presented in Chapter 3. The chapter begins with an introduction into the key challenges facing large-scale context-aware systems in section 3.2. Importantly, this section also introduces several related advances that have begun to address each challenge in isolation. As described in section 3.2.1, client-centric context acquisition and modeling approaches have shown how context information can be extracted from heterogeneous large-scale environments without the need for prohibitive instrumentation or infrastructure. Next, as described in section 3.2.3, the tremendous growth and popularity of the Internet has resulted in an increasingly ubiquitous substrate of data

communications. Next, as described in section 3.2.5, modern Web architecture provides a ubiquitous, Internet-scale middleware architecture whose communication protocols and functional apparatus are broadly adopted. Finally, as described in section 3.2.7, the Web's underlying architectural style (formalized as Representational State Transfer [105]) provides an promising approach for supporting cross-domain component interoperation. Based on these advances, section 3.3 describes Ocean's overall scope and section 3.4.1 introduces the design principles intended to guide the derivation of the Ocean architecture. Section 3.4.2 describes the key architectural constraints intended to align Ocean with conventional Web architecture. Next, section 3.5 derives the overall Ocean approach by describing its principle architectural abstractions, application model, component contextualization and discovery techniques, registry architecture and support for community-based computation. The chapter concludes with a discussion of Ocean's principle stakeholders in section 3.6 and the Ocean Reference Implementation in section 3.7, which is intended to validate core aspects of the Ocean approach.

Based on the approach specifications presented in Chapter 3, Ocean's core architectural abstraction, called the *Contextualized Resource*, is presented in Chapter 4. Briefly, the Contextualized Resource abstraction extends the Web's conventional Resource model with an expressive contextual metadata model used to facilitate in-situ discovery and integration. The chapter begins by presenting background and related work specific to context-mediation in REST-based architectures in section 4.2. Next, the Contextualized Resource abstraction is presented in section 4.3. Section 4.3.2 presents the Context Metadata abstraction, which is intended to encapsulate the syntax and semantics of diverse context domains. Related, section 4.3.3 provides a theoretical discussion of similarity modeling as pertaining to the wide-area Resource Discovery mechanisms discussed in subsequent chapters. The chapter concludes with an illustrative Contextualized Resource example in section 4.4.

Building on the Contextualized Resource abstraction, Chapter 5 discusses key ideas regarding Web-scale context-aware computing using the Ocean approach. As discussed throughout this chapter, Ocean provides a simple, accessible and scalable mechanism for mobile applications to discover, select and compose contextually-relevant Web Resources at runtime. Based on this philosophy, section 4.1 provides background and related work related specific to our approach derivation presented in section 3.5. Next, section 5.2.2 presents Ocean's Web-centric application model as an extension of the client-centric mashup style presented in section 5.2.1. Notably, the Ocean application model defines a new class of context-aware Web application capable of spontaneous cross-domain component discovery and interoperation. Section 5.3 describes the Ocean Contextualized Resource Registry (Ocean Registry), which is used to adapt the Ocean application model to the requirements of conventional Web architecture. This section introduces the Ocean Registry's principle separation of concerns, software architecture and notable APIs. In order to promote the contribution of Context Metadata implementations within the Ocean Registry, section 5.4 describes an adaptation of the Java Community Process intended to facilitate participation by external Context Domain Experts. Finally, in order to promote large-scale Resource contextualization, section 5.5 concludes the chapter by introducing Ocean's open Contextualized Resource contribution model, which allows *any* Contextualizer to contextualize *any* Resource with *any* combination of Ocean Metadata.

Chapter 6 presents the Ocean Registry's Persistence and Discovery Frameworks, which are intended to support wide-area Contextualized Resource contextualization and discovery. Briefly, the Persistence Framework allows Contextualized Resources to be efficiently stored and indexed for rapid retrieval according to domain-specific techniques. Section 6.2 provides background and related work regarding database techniques and related similarity search approaches. Next, section 6.3 presents the Ocean Registry's persistence model by describing Ocean's relevant abstractions and illustrating how Contextualized Resources can be effectively indexed using an example. Next, section 6.4.1 describes Ocean's discovery protocol, which allows applications to query the Ocean Registry for contextually-relevant Resources using native context data (NCD) as query terms. This section describes the Resource Discovery API and its associated request/response formats. Ocean's Resource Discovery approach is presented in section 6.4. This section presents background and related work regarding existing similarity search techniques and presents Ocean's associated Discovery Framework. This section further describes how contextually-relevant Resources are discovered using the search protocol previously introduced. Notably, the Discovery Framework operates in conjunction with the aforementioned Persistence Framework; allowing the NCD within Discovery Requests to be effectively compared with persisted Context Metadata within the Ocean Registry's shared data store.

Chapter 7 discusses how Ocean's Web-scale focus introduces two critical challenges for Resource discovery and selection. The first challenge relates to context-mismatch, whereby Resource discovery performance is degraded due to mismatches between the Context Metadata used to describe a Web Resource and incoming NCD-based query terms. Section 7.2 addresses context-mismatch by defining a mechanism that automatically expands Discovery Requests with additional, contextually-relevant query terms extracted from the shared query information provided by diverse members of the Ocean end-user community. The second challenge relates to information overload, whereby Resource discovery performance may be degraded due to extremely large numbers of undifferentiated query results. Section 7.3 addresses information overload by describing Ocean's Resource personalization approach, which automatically predicts a user's affinity for a given Resource based on previously modeled preferences information from similar Ocean end-users. Notably, both query expansion and Resource personalization are available as optional, privacy-aware enhancement features that can be used either alone or in combination to help improve discovery search results.

Chapter 8 presents an Ocean application scenario as a method of illustrating how Ocean's various contributions form an integrated whole. In addition, the example scenario provides further validation of Ocean's large-scale focus by integrating real-world native context sources, significant Contextualized Resource information and more realistic models of Ocean community behavior. First, section 8.2 describes our experimental setup, including the development of an Ocean application development environment, called Ocean Studio, and an embedded Ocean reference implementation (RI) that is designed for rapid prototyping. Next, section 8.3 provides an overview our example application, called LinkFlow, which forms the conceptual foundation for the remainder of the chapter. Section 8.4 describes our data acquisition methodology and related toolset designed to capture large numbers of native context sources and real-world Contextualized Resources. Section 8.5 presents a validation of the basic LinkFlow scenario within the Ocean Studio development environment. Section 8.6 describes the integration of the previously acquired native context sources and Contextualized

Resource data into Ocean Studio. Notably this section also describes the resultant query performance reduction due to context mismatch and information overload. Section 8.7 discusses how the aforementioned query challenges can be addressed by applying Ocean's query expansion and Resource personalization techniques.

Chapter 9 concludes the dissertation with a summary of contributions and a discussion of future research directions.

Chapter 2
On Context-aware Computing

2.1 Introduction

To help motivate the Ocean approach, this chapter presents background material and related work regarding context-aware computing. As discussed shortly, context-aware computing represents an evolutionary synthesis of several domains of computer science, including data communication networks, distributed systems and mobile computing. As such, a comprehensive treatment of each sub-domain remains outside of the scope of this dissertation. Rather, this chapter discusses the background material directly related to the development of large-scale context-aware systems. First, section 2.2 begins by discussing the rise of computer networking and the key techniques that have led to the rapid proliferation of network infrastructure. Based on increasingly ubiquitous communication networks, section 2.2.1 describes important approaches for distributing computation across multiple autonomous networked computers. Related, section 2.2.2 describes techniques for accommodating distributed computing in Internet-based scenarios through service-oriented computing and SOAP Web services. Finally, section 2.2.3 describes how recent trends in mobile computing and wireless networks have motivated in the development of adaptive systems capable of altering their runtime behavior and capabilities to better fit the characteristics of a mobile user's current situation.

The aforementioned background material is used to introduce the field of context-aware computing in section 2.3. This section describes major techniques and technologies for adapting mobile and embedded systems to the characteristics of highly dynamic mobile computing scenarios. Notably, this section is structured by the introduction of a unified conceptual framework that allows context-aware systems to be decomposed, categorized and analyzed. Based on this conceptual framework, a representative sampling of context-aware computing research is presented. First, section 2.3.1 defines key notions of context and context-awareness. Next, section 2.3.2 describes important approaches in context acquisition. Section 2.3.3 describes techniques for context modeling and representation. Section 2.3.4 presents key techniques for context-management and provisioning. Finally, section 2.3.5 describes major approaches for facilitating context-aware component interoperation.

2.2 Background

Over the past several decades, increasingly powerful computational capabilities have been steadily diffusing into everyday objects, devices and environments. Today our homes, cars, offices and public spaces contain an ever-increasing assortment of embedded systems, mobile devices and computer networks. Although such trends appear recent, the dissemination of computation away from individual devices has been evident since the beginnings of general purpose computing. Early computing systems were designed necessarily with all hardware resources, processing instructions and data persistence isolated within a single hardware platform. The rapid proliferation of such *stand-alone* computers gave rise to a set of well-understood techniques for *sequential computing*, where computational tasks are solved through the step-by-step serial application of processing instructions [13]. Importantly, the development of sequential computing produced a framework for specifying algorithms, comparing their performance and understanding their inherent limitations such as lower-

bound runtime and the notion of NP-completeness. However, although stand-alone computing provided a straightforward means of solving common computational problems, most sequential algorithms were unable to dynamically exploit additional computing resources and often suffered from intrinsic inefficiencies imposed by their inability to work on different parts of a problem simultaneously [38].

Improvements in computer hardware resulted in the emergence of *parallel computing* as a mechanism to address the inefficiencies of sequential algorithms. In parallel computing, different processing instructions are carried out simultaneously within a single computer system by exploiting the notion that larger computational problems can often be divided into smaller ones that can be solved concurrently. Parallel processing architectures typically coordinate task partitioning, scheduling and processing using local intra-process communication (IPC), which occurs within a shared memory address space on the same machine using techniques such as pipes, first-in-first-out queues or shared memory. In many cases, parallel architectures incur beneficial properties such as increased computational performance, improved modularity, and fault isolation [36]. As a consequence, advanced parallel computing techniques – such as instruction pipelining, multithreading, task scheduling and multiprocessor hardware architectures – have become common features of most modern hardware platforms, operating systems, compilers and programming languages [132].

The advent of parallel computing coincided and often supported related advancements in the field of data communication networks throughout the late 1960s and early 1970s. During this time, the costs associated with mainframe computing, combined with the increasing availability of micro-computers, spurred increased interest in designing computer networks capable of interconnecting autonomous computers. In this regard, the development of network-based communications was motivated by a diverse set of goals, including resource sharing (e.g. sharing a mainframe computer or file server); increased reliability (i.e. providing alternative sources of data or computation); increased computational performance through parallelization; and improved human communications (e.g. file sharing and electronic mail) [323]. Important networking challenges included reliable data exchange; selection of appropriate communication paths; congestion control; deadlock control; and security [325].

Resolution of the aforementioned challenges requires a communication model that imposes a high degree of cooperation between participating entities. Cooperation is often achieved through the specification of *network protocols* that dictate the rules and conventions of communication [323]. In this regard, the complexity of network-based communication is mitigated through the use of *protocol layering* whereby communication sub-tasks are encapsulated by a structured set of interrelated protocols commonly referred to as a *communications architecture* or *protocol suite* [248]. The exploration of such architectures resulted in the development of two broad classes of transmission technologies, including broadcast and point-to-point. Briefly, in *broadcast networks*, a number of computers (hosts) communicate using a shared communications medium (or link) by means of medium access control (MAC) protocols that help the hosts share the link fairly and efficiently. In broadcast networks, short messages (called *packets* in some contexts) are sent by any machine to all other hosts connected to the shared link [323]. In contrast, *point-to-point networks* support multiple

connections between pairs of hosts by way of an unshared serial link and related protocols. In point-to-point scenarios, transmitted data must pass through (often multiple) intermediaries on the way to a given destination.

The commercial success of both broadcast and point-to-point network designs resulted in the development of broad range network types, which are commonly classified according to physical size [323]. Common classifications include *personal area networks* (PANs) that are intended to serve a single person at a range up to 1 meter; *local area networks* (LANs) that often represent privately owned networks at the organization scale; *metropolitan area networks* (MANs) that operate at the scale of cities or groups of corporate offices; and *wide area networks* (WANs) that span large geographical areas such as countries or continents. A more detailed illustration of this basic classification scheme is shown in Figure 2.

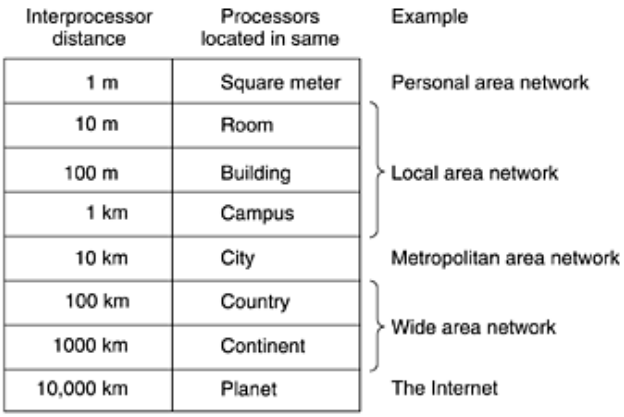


Figure 2: Classification of computer networks based on physical size (from [323])

The rapid proliferation of incompatible network infrastructure motivated the exploration of internetworking techniques, whereby computer-based communications could span multiple heterogeneous physical networks. The dominant communications architecture in this regard is the Transmission Control Protocol/Internet Protocol (TCP/IP) protocol suite that was first proposed by Cerf and Kahn in [56]. Briefly, TCP/IP combines several interrelated advancements into a relatively simple, flexible and robust internetworking scheme that has come to form the foundation of the modern Internet. At its core, the TCP/IP model aims to optimize available channel capacity, minimize transmission latency and improve communication robustness by adopting packet-based statistical multiplexing (*packet switching*) as its principle communications model [248]. Second, the TCP/IP approach enhances the basic packet switching model by defining a standardized set of abstractions intended to encapsulate the technology-dependent communication functions (primitives) of various underlying network types. Accordingly, the TCP/IP architecture encapsulates complex communication sub-tasks into multiple self-contained layers; each providing well-defined service interfaces to layers above and below. By introducing a platform agnostic *virtual network* between underlying network technologies and higher-level network users, the TCP/IP model provides "users the illusion of a single, seamlessly connected network where the fragmented nature of the underlying infrastructure and the many layers of protocols remain largely transparent to the user" [358].

While a detailed description of Internet architecture can be found elsewhere [323], a brief description of its three fundamental design principles provide insight into its design. First, as described above, *layering* is used to break up complex tasks into relatively simple subtasks that can be structured and arranged separately. Subtasks are implemented by modules that can be arranged into a vertical stack, where each layer provides a well-defined set of services to its adjoining layers and structures communication through a specific convention, or protocol. Secondly, Internet architecture has been influenced by the *end-to-end argument in system design*, which states that "certain required end-to-end functions can only be performed correctly by the end-systems themselves. A specific case is that any network, however carefully designed, will be subject to failures of transmission at some statistically determined rate. The best way to cope with this is to accept it, and give responsibility for the integrity of communication to the end systems" [279]. The end-to-end principle positions complex, application-specific functionality at the end-systems while retaining relatively simple and scalable core network characteristics. The Internet's final design principle is the use of a single logical addressing scheme that is shared by all participating nodes. As described in [223], these principles affirm that the Internet's "goal is connectivity, the tool is the Internet Protocol, and the intelligence is end to end rather than hidden in the network." As evidenced by the Internet's explosive growth, rapid performance increases and proliferation of software services, its basic design philosophy has proven remarkably effective at supporting "constant innovation and entrepreneurial spirit at the physical substrate of the network as well as at the application layer" [358]. The Ocean approach is strongly influenced by these design principles, as described in section 3.4.2.

2.2.1 Key Aspects of Distributed Computing

The increasing availability of inexpensive computing hardware combined with emerging computer networks and internetworking techniques led to the development of techniques for distributing computation across multiple autonomous networked computers. The resultant field of research, known as *distributed systems* or *distributed computing*, encompasses a wide range of approaches and algorithms for distributing and coordinating remote processing. Generally, distributed computing refers to any "computing system in which a number of components cooperate by communicating over a network" [44]. While such definitions are often vague and contentious, Ghosh [120] presents several well-recognized criteria for distributed systems, including having more than one sequential process; message-based IPC; disjoint address space; and, collective goals whereby distributed processes interact with each other to address a common task. Ghosh also identifies several benefits of distributed systems, including geographic distribution; performance improvements beyond parallel computing; enhanced resource sharing; and enhanced fault-tolerance. Coulouris et al. identify several additional beneficial properties in [74], including system extensibility; location transparency whereby local and remote information can be accessed in a unified way; failure transparency whereby system failures can be automatically masked; and replication transparency whereby software and data can be replicated on multiple machines invisibly.

A distributed system can be described in terms of software architecture. Broadly, *software architecture* has been defined by Fielding as "an abstraction of the run-time elements of a software system during some phase of its operation. A system may be composed of many levels of abstraction and many phases of operation, each with its own software architecture" [105]. Fielding further

describes that software architectures are used to structure and decompose computational tasks into smaller, more manageable parts through encapsulation. Encapsulated elements are selected, arranged and configured to achieve a set of system properties that fulfill a system's requirements. Properties can include functional properties (e.g. system behavior) and quality attributes (e.g. ease of evolution, reusability of components, efficiency, and dynamic extensibility). Evaluations of software architecture include metrics such as *correctness* (that the system indeed fulfills the specified requirements) and *efficiency* (that the system fulfils the specified requirements at a low cost – e.g. low computational complexity, communication efficiency, etc.) Properties are induced within a system through the application of software engineering principles, which represent a style of arranging and configuring architectural elements. Accordingly, software engineering principles induce a "coordinated set of architectural constraints that restricts the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style" [105]. Table 2 provides an overview of the four principle architectural elements of software architecture as defined by Fielding in [105].

Element	Description
Component	An abstract unit of software instructions and internal state that provides a transformation of data via its interface.
Connector	An abstract mechanism that mediates communication, coordination, or cooperation among components.
Data	Elements of information which are transferred from a component, or received by a component, via a connector.
Configuration	The structure of architectural relationships among components, connectors, and data during a period of system run-time.

Table 2: The principle elements of software architecture (from [105])

With reference to Table 2, distributed systems differ from stand-alone architectures in several notably ways. First, components in a distributed system may reside on (potentially several) addressable networked devices that are known as *nodes* or *hosts*. Nodes are generally autonomous (i.e. maintain their own semi-independent agenda) and heterogeneous (i.e. comprise a diversity of hardware platforms, operating systems and runtime software). Hence, units of distributed computation provide their own internal stand-alone software architecture that is configured to collaborate with other networked entities via network-based messages. Accordingly, a distributed architecture can be described as a finite collection of entities that communicate using messages in order to "achieve a common goal; for example, to perform a given task, to compute the solution to a problem, to satisfy a request either from the user (i.e., outside the environment) or from other entities" [282]. Distributed architectures are also characterized by connectors that are often several orders of magnitude slower than in-memory IPC techniques and may be subject to significant variations in communications quality due of the characteristics of the underlying network (e.g. delay, jitter and loss) [323]. Moreover, the capabilities of entities in a distributed architecture may not be uniform. For example,

entities may vary in terms of available processing capability, local memory or access to connected peripherals such as input and output hardware. Consequently, distributed architectures require additional measures to ensure interoperability (e.g. data passed between entities cannot often rely on proprietary representation such as language specific data types [334]). Additionally, the design and configuration of distributed systems is often significantly more complex than stand-alone architectures. Several important factors in this regard are listed below as presented in [325]:

1. **Broadcasting and synchronization:** If information must be made available to all processes, or all processes must wait until some global condition is satisfied, it is necessary to have a message passing scheme that somehow touches all processes.
2. **Election:** Some tasks must be carried out by exactly one process of a set, for example, generating output or initializing a date structure. If, as is sometimes desirable or necessary, no process is assigned this task a priori, a distributed algorithm must be executed to select one of the processes to perform the task.
3. **Termination detection:** It is not always possible for the processes in a distributed system to observe directly that the distributed computation in which they are engaged has terminated. Termination detection is then necessary in order to make the computed results definitive.
4. **Resource allocation:** A node may require access to a resource that is available elsewhere in the network, though it does not know in which node this resource is located. Maintaining a table that indicates the location of each resource is not always adequate, because the number of potential resources may be too large for this, or resources may migrate from one node to another. In such a case the requesting node may inquire at all or some other nodes about the availability of the resource, for example using broadcasting mechanisms.
5. **Deadlock detection and resolution:** If processes must wait for other processes (which is the case if they share resources, and also if their computation relies on data provided by other processes) a cyclic wait may occur, in which no further computation is possible. These deadlock situations must be detected and proper action must be undertaken to restart or continue the computation.
6. **Distributed file maintenance:** When nodes place read and write quests for a remote file, these requests may be processed in an arbitrary order, and hence provision must be made to ensure that each node observes a consistent view of the file or files. Usually this is done by time stamping requests, as well as the information in files and ordering incoming requests on the basis of their time stamps.

Based on the above challenges, several computational models and infrastructure technologies have been developed to aid in creating distributed applications. Early *ad-hoc* approaches were generally based on low-level communication abstractions (e.g. sockets), which were used to manually organize communications between distributed entities. Ad-hoc approaches require that developers specifically adapt the underlying communication techniques to the requirements of a given application. While suitable for some application types, several deficiencies have been identified. According to Buschmann et al. [44], the resultant application code is often tightly coupled to the underlying

operating system and socket APIs; requiring costly manual programming effort when porting an application to different platforms. Related, creating distributed communication frameworks using sockets can also result in paradigm mismatches, which derive from fundamental platform differences (e.g. object-oriented versus function-oriented socket APIs). In this regard, the reuse of ad-hoc techniques is often limited as developers rely on the creation of custom data-structures and application-specific method invocation that may not be widely known or applicable across problem domains.

The proliferation of increasingly heterogeneous network infrastructure, combined with the increasing need to create platform independent networked applications, led to the development of *Structured Communication* techniques beginning in the mid-1970s. Structured Communications decouples the intricacies of network-based IPC from higher-level application logic by providing a common communications infrastructure that allows remote applications to interact much like they would in shared-memory environments. Structured Communications infrastructure provides common abstractions such as procedures and data-types, which shield developers from machine-level data representations and encapsulate network-based IPC within a familiar programming model. Historically significant Structured Communication approaches include Sun Microsystems' Remote Procedure Call (RPC) [320] and the Distributed Computing Environment (DCE) [273]. A high-level overview of Remote Procedure Call is shown in Figure 3.

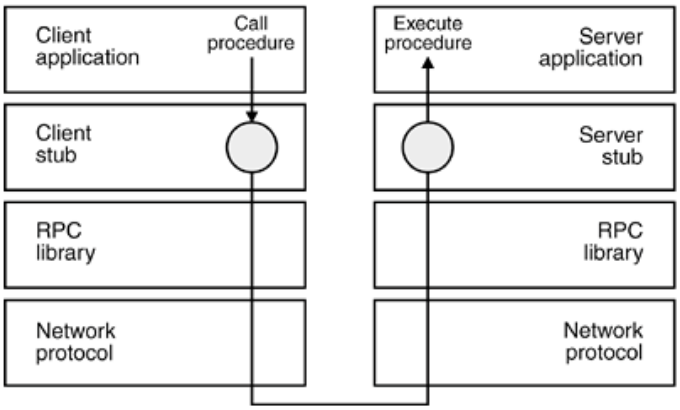


Figure 3: Overview of Remote Procedure Call (from [186])

As traditional procedure-based programming gave way to object-oriented programming (OOP) in the early 1990s, the confluence of RPC-based distributed computing and component-based architectures resulted in the development of *Distributed Object Communications* (DOC) middleware. DOC middleware extends Structured Communications' notion of platform independence with object-oriented techniques for distributing reusable services efficiently and robustly over heterogeneous nodes [44]. DOC middleware provides network-aware versions of common OOP abstractions (e.g. objects, properties, methods and parameters) along with related infrastructure for marshaling, transporting and unmarshalling object representations between remote entities [13]. Similar to OOP's object-based encapsulation model, DOC middleware relies upon precise (and often fine-grained) interface descriptions that establish interface contracts between client and server regarding the mechanisms and data-types necessary for remote interactions. Notable DOC middleware technologies

include the Java-specific Remote Method Invocation (RMI) [319] and the largely language independent Common Object Requesting Broker Architecture (CORBA) [235]. In DOC middleware, significant infrastructure such as CORBA's Object Request Broker (ORB) is used to provide transparent IPC between distributed objects through integrated support for naming, object binding, communication protocols, exception handling, transaction support, security, etc. A high-level overview of ORB-based distributed communications is shown in Figure 4.

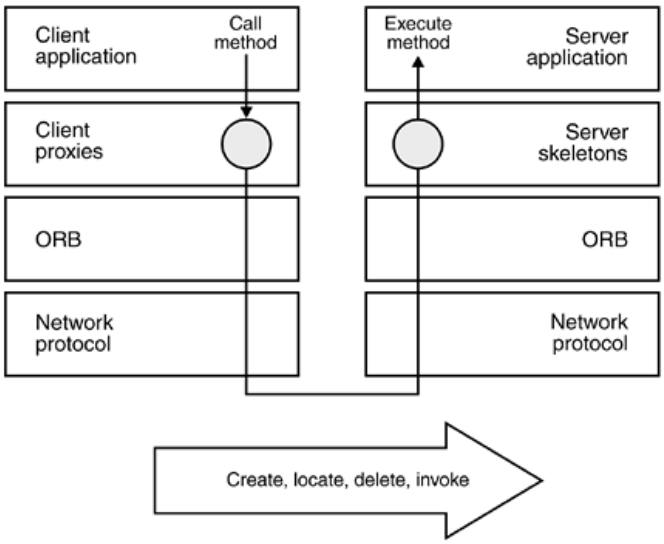


Figure 4: Overview of ORB-based distributed communications (from [186])

The RPC techniques and DOC middleware approaches previously described are generally based on the request/response communications model, where requests flow from client to server and responses flow back from the server to the calling client. However, such approaches are often ill-suited for scenarios where a distributed application must respond to external stimuli and events [44]. Problematic issues in this regard include *designated communication* whereby clients must know the identity of a suitable server (resulting in tight coupling between sender and recipient); *point-to-point communication*, whereby client are restricted to communications with one server at a time; and *brittle interface semantics*, whereby slight changes in interface definitions may break dependant applications. As discussed in section 3.2, these issues are important limiting factors with regards to large-scale, cross domain context-aware systems.

To address some of the aforementioned challenges, *Message-Oriented Middleware* (MOM) has been proposed as an alternative method of structuring distributed applications. Although variations exist (e.g. UDP multicast-based systems), most MOM approaches are based on asynchronous message queuing, whereby senders transmit data to receivers without blocking to wait for a response [44]. Asynchronous message services are typically provided by an abstraction known as a *message queue*, which allows messages to be published and persisted until picked up by a receiver (or group of receivers). Notable MOM implementations include IBM's MQSeries [154] and BEA's MessageQ [239]. Unlike many interface-centric approaches, MOM's often rely on *payload semantics*, whereby a relatively simple messaging interface (e.g. `sendMessage()` and `receiveMessage()`) is used to manage flows of self-describing messages (e.g. based on XML-based headers). In the basic message

queuing model, point-to-point communications are used to asynchronously deliver messages from a single sender to a single receiver. More sophisticated implementations are organized using a publish-and-subscribe model that introduces a *topic* abstraction, which is used to provide anonymous publication of messages to any number of receivers who have registered interest in a given topic. As described in section 3.5, the concept of payload semantics plays an important role in the Ocean application model. An overview of the two example MOM approaches is provided below in Figure 5.

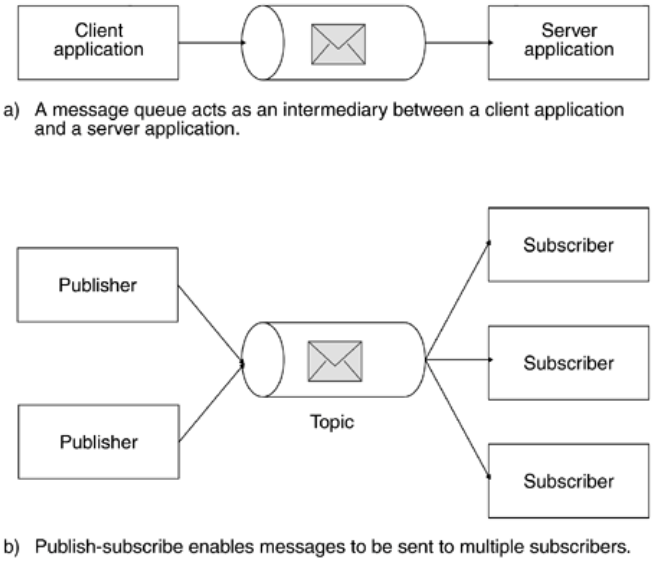


Figure 5: Overview of two example message-oriented-middleware approaches (from [186])

2.2.2 Service Oriented Computing and SOAP Web Services

The widespread adoption of DOC middleware and rapid rise of the Internet as an important global communications infrastructure have motivated two related (yet often divergent) trends in distributed computing [257]. The first trend, known as *Enterprise Application Integration* (EAI), involves devising methods for integrating (often numerous) legacy software systems contained within an organization into coherent (and often large) distributed systems. The second trend involves developing techniques for cross-organization integration of enterprise data among external customers and partners (often via the Internet). Accommodating the autonomy, heterogeneity and complexity implied by such requirements has led to the emergence of a distributed computing style, known as *Service Oriented Computing* (SOC), which aims to abstract business functionality away from monolithic applications in order to yield distributed systems that are easier to build, maintain and extend [335].

SOC defines a distributed computing model whereby applications are constructed by combining pre-existing, self-contained and loosely-coupled units of functionality known as *services*. In SOC, services can be defined as self-describing, modular, atomic pieces of computation that adhere to well-known (and often self-describing) interfaces. While conceptually similar to objects in OOP, where data and methods are encapsulated to promote modularity and reuse, services generally offer a higher degree of abstraction and interface granularity. SOC implementations (also known as *Service Oriented Architectures* or SOAs) typically extend related advances in component-based middleware

such as Corba's Component Model [233] and Enterprise Java Beans[3]. Briefly, component middleware provides support for dynamic assemblies of components along with comprehensive deployment and lifecycle management in the form of component *containers*. As described in [44, 234], containers are used to provide unified deployment and runtime mechanisms such as data persistence, event notification, transaction support, replication, load balancing, security, etc. Further, containers also define a collection of runtime policies (e.g. transaction, persistence, security, and event delivery) and are responsible for initializing and providing runtime management of components.

SOAs typically extend component middleware approaches with additional facilities for component distribution, discovery and runtime interaction. These mechanisms are introduced by Vinoski in [335] as:

- *Service registries*, in which services advertise their locations and capabilities, and where consuming applications go to find those services;
- *Service repositories*, in which developers store metadata, such as contract descriptions and policies, for use at both service design and deployment times;
- *Service definition languages*, which developers use to define service contracts; and
- *Service platforms*, which provide design-time and runtime support for service creation, deployment, and execution.

Based on these foundational concepts, SOAP web services (SOAP-WS) have emerged as a popular approach for building SOA using Internet-based infrastructure [369]. Similar to conventional DOC middleware, SOAP-WS aim to "provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks" [371]. More specifically, SOAP-WS define a set of protocols and frameworks for achieving service discovery, interoperation and coordination using common Internet protocols and standards such as HTTP and XML. At its foundation, SOAP represents "a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data-types, and a convention for representing remote procedure calls and responses" [369]. Accordingly, the SOAP standard is used to provide a distributed interoperation approach consisting of: a common type system; a service description language; addressing model; bindings to lower-level transports; security or routing frameworks; and mechanisms for breaking message into segments like header and body (known as *framing*). While the SOAP model is transport agnostic, bindings have been developed for common protocols such as HTTP, SMPT, and others (with HTTP being the most common [266]). Moreover, SOAP is message exchange agnostic, allowing applications to utilize a variety of interaction patterns, such as request/response, request/multiple response, etc. [266]

To achieve application-level component discovery and interoperability, SOAP-WS utilize several specifications in addition to SOAP. Notably, the *Web Services Description Language* (WSDL) is used to describe the operations a networked component supports (e.g. how messages are used during

[3] http://java.sun.com/products/ejb/

method invocation) along with the associated type and structure of the messages conveyed. As described in [67], WSDL provides an XML grammar that describes network services as sets of endpoints that operate on messages that contain either document-oriented or procedure-oriented information. Importantly, supported operations and messages are described abstractly and then bound to a specific network protocol and message format as a means of defining endpoints.

Several other standards are commonly used in conjunction with WSDL and SOAP. First, service discovery and binding is facilitated through an additional specification called *Universal Description, Discovery and Integration* (UDDI) [231]. The UDDI specifications describe an Internet-based registry architecture intended to allow organizations to publish and discover services using SOAP as an interrogation protocol and WSDL as service description metadata (see section 4.2). Additional notable specifications include *WS-Addressing*, which provides a "transport-neutral mechanisms to address Web services and messages" [370]; *WS-Transaction*, which provides a set of specifications intending to provide "protocols for coordinating the outcome of distributed application actions" [241]; and, *WS-Security*, which provides "security functions such as integrity and confidentiality in messages implementing higher-level Web services applications" [240].

2.2.3 The Influence of Mobility

The widespread deployment of distributed computing infrastructure such as Java RMI and SOAP-WS, has paralleled related advances in mobile computing and wireless networks. Beginning in the early 1990s, advances in semiconductor electronics, battery technologies and materials science led to the emergence of small computing systems that could be easily carried or embedded within physical objects. Simultaneously, wireless networking technologies began transitioning from research prototypes into commercially available products. The resultant proliferation of wireless networking infrastructure and related mobile device support motivated interest in developing software systems capable of addressing the unique challenges imposed by mobile scenarios. As described by Forman and Zahorjan in [109], these principle challenges included portability, mobile networking, and wireless communications. Briefly, *portability* refers to the development of small, lightweight computing systems. Notable portability challenges include resource constraints (e.g. limited storage or screen size); power limitations; and increased security risks. Next, *mobile networking* refers to the ability of a device to change locations while maintaining network connectivity. Notable mobility challenges include address migration; location-dependent information; and locality migration. Finally, *wireless communication* refers to techniques for connecting mobile devices using wireless networks. Notable wireless communication challenges include disconnection; low bandwidth operation; high bandwidth variability; network heterogeneity and enhanced security risks.

Over the last decade, many of the aforementioned challenges have been addressed by significant research efforts in the field of mobile computing. As discussed by Satyanarayanan in [288], portability has been addressed through the development of a variety of power management techniques, including variable-speed processor scheduling [355]; energy-aware adaptation [108]; and energy-sensitive memory management [193]. Next mobility has been addressed through the development of a variety of address migration techniques, including Mobile IP [252] and the widespread adoption of the Dynamic Host Configuration Protocol (DHCP) in wireless networks

[323]. Finally, wireless communication has been addressed through the development of a variety of networking technologies, including several broadly adopted 802.11 wireless communication standards [323]; energy-aware mobile ad-hoc networking technologies and routing protocols [133]; bandwidth-adaptive file access [215]; selective control of data consistency [326]; and techniques for disconnected operation [182].

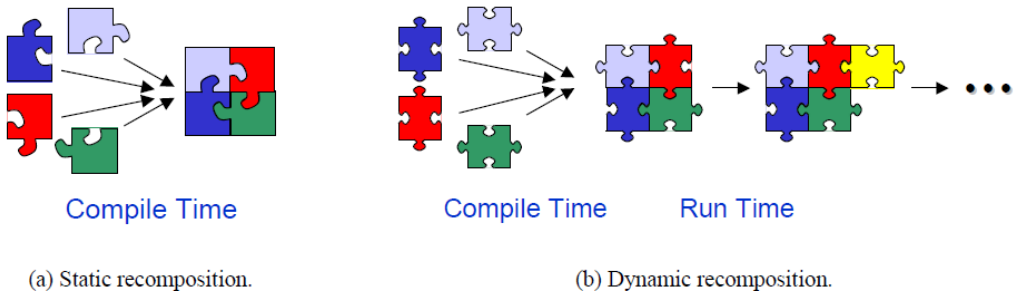
The mobile computing techniques and technologies presented above have motivated increasing interest in software systems capable of dynamically adapting their runtime behavior and capabilities to fit the characteristics of a mobile user's current situation [293]. Two principle approaches for runtime adaptation have been explored. First, *parameter adaptation* relies on a predetermined set of program variables that can be "tuned" to help adjust an application's runtime behavior. A well-known example of parameter adaptation is TCP's flow control approach, which responds to detected network congestion by dynamically adapting its retransmission strategy using a preconfigured set of control window values [323]. Although parameter adaptation is often effective and relatively easy to implement, it lacks the ability to incorporate additional algorithms and system components after an application has been deployed. In contrast, *compositional adaptation* allows for algorithmic or structural system components to be integrated into a software application during runtime. Broadly, compositional adaptation "enables dynamic recomposition of the software during execution – for example, to switch program components in and out of a memory-limited device or to add new behavior to deployed systems" [208].

Adaptive composition is based on three fundamental concepts. First, adaptive systems are derived from the *separation of concerns* principle, which promotes program modularity by breaking an application into distinct features with little functional overlap [249]. A separation of concerns is used to promote the independent development of application logic and cross-cutting concerns, which cannot typically be cleanly decomposed (e.g. quality of service, fault tolerance, security, etc.) Second, adaptive systems extend notions of *computational reflection*, whereby a program can reason about, and possibly alter its runtime behavior [204]. Computational reflection generally involves both *introspection*, where an application observes its behavior, and *intercession*, where an application acts on its observations and modifies its behavior in response. For example, an application may use computational reflection to discover, select and dynamically load a communication protocol better-suited to the current network conditions [208]. Finally, adaptive systems typically extend notions of *component-based design* (CBD), which refers to a method decomposing software systems into self-describing, self-contained executable units of functionality.

As noted by McKinley et al. in [208], CBD can be delineated into two basic approaches: static and dynamic. In static approaches, components are selected and composed at compile time (i.e. early binding). In dynamic approaches, components are added, removed and configured at runtime through the use of component resolution and late binding (also known as dynamic binding). Generally, dynamic binding approaches are supported by specific programming language constructs (e.g. Python and various Java derivatives) and related host and middleware infrastructure (e.g. the Java virtual machine and Open ORB[4]). Notably, academia and industry have developed a variety of methods and

[4] http://openorb.sourceforge.net/

mechanisms for supporting dynamic compositional adaptation (please see [209] for a detailed overview). An overview of static versus dynamic recomposition is presented in Figure 6.



(a) Static recomposition. (b) Dynamic recomposition.

Figure 6: Overview of static (a) versus dynamic (b) recomposition (from [209])

Dynamic recomposition, as illustrated above, is often reliant on the principle of *loose coupling*, which refers to architectural strategies for minimizing the dependencies between the functional units of a software system [186]. Examples of such dependencies can include hard-coded method calls between service implementations or reliance on proprietary messaging formats. Several examples of tight versus loose coupling are provided below in Table 3 as adapted from Krafzig et al. [186].

Level	**Tight coupling**	**Loose coupling**
Physical coupling	Direct physical link required	Physical intermediary
Communication style	Synchronous	Asynchronous
Type system	Strong type system (e.g., interface semantics)	Weak type system (e.g., payload semantics)
Interaction pattern	OOP-style navigation of complex object trees	Data-centric, self-contained messages
Control of process logic	Central control of process logic	Distributed logic components
Service discovery and binding	Statically bound services	Dynamically bound services
Platform dependencies	Strong OS and programming language dependencies	OS- and programming language independent

Table 3: Tight versus loose coupling (adapted from [186])

Although widespread adoption of the approaches summarized in this chapter have produced a relatively seamless substrate of data communications, portable computing platforms and adaptation mechanisms, the development of *mobile distributed systems* (MDS) has often proven more difficult [293]. Unlike conventional distributed systems, where the application scenarios are well-understood and distributed components are known a priori, mobile users often encounter rapidly changing and

highly diverse execution environments that place unique demands on networked mobile software [293]. An overview of the challenges faced by MDS is presented below as adapted from [73]:

- **Heterogeneity:** Mobility increases heterogeneity concerns for distributed systems on several levels. Mobile users encounter a comparatively diverse set of computational resources which may vary widely in terms of capabilities, interaction mechanisms and arrangement within a physical or virtual space. Moreover, mobile and pervasive devices also vary widely in terms of hardware platform, operating systems and runtime software. In addition, sources of environmental computation may be unsystematically organized, unpredictably available and not inherently interoperable.
- **Scalability:** Mobile scenarios are often characterized by unpredictable numbers of users, devices and operational areas [109]. Moreover, components within mobile scenarios may pursue autonomous agendas. Consequently, mobile software must address aspects of both structural scalability and load scalability. As described by Bondi in [38], structural scalability refers to a system's ability to expand in a given dimension without major modifications – e.g. accommodating flexible routing schemes – whereas load scalability refers to a system's ability to handle increasing amounts of work without degradation in system performance – e.g. additional service interaction or increasing data traffic.
- **Dependability and security:** The autonomy of mobile scenarios often introduces additional dependability and security concerns. Existing failure-detection and recovery strategies, such as check-pointing, compensation, isolation, or reconfiguration must be adapted for mobile use. Further, network connectivity and distributed services may be provided by unknown parties. Hence, large-scale distributed applications must be capable of operating across multiple trust domains, and "continue operating when subjected to an unanticipated load, or when given malformed or maliciously constructed data, since they may be communicating with elements outside their organizational control" [106].
- **Spontaneous interoperation:** The dynamism characteristic of mobile scenarios often results in spontaneous encounters with relevant information or computation. As noted by Edwards et al., "Interoperability among a group of devices, applications, and services is typically predicated on those entities having some degree of prior knowledge of one another" [96]. However, given the large operational range characteristic of mobile scenarios combined with a proliferation of domain-specific component interfaces and related data-types often results in architectural mismatch, where implicit and often conflicting assumptions made by component designers inhibit spontaneous cross-domain component interoperation [115].

2.3 Related Work in Context-aware Computing

Based on the issues and challenges presented in the last section, new computing techniques were seen as necessary for adapting mobile and embedded systems to the dynamic and complex characteristics of the physical world [208, 287]. In this regard, *Ubiquitous Computing* (Ubicomp) emerged as a novel "method of enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user" [354]. Attracting interest across a broad range of disciplines – including computer science, human computer interaction, engineering, the social sciences, philosophy, and anthropology – the Ubicomp model advocates "an extended form of mobile computing in which users employ many different mobile, stationary and embedded computers over the course of the day. In this model computation does not occur at a single location in a single context, as in desktop computing, but rather spans a multitude of situations and locations covering the office, meeting room, home, airport, hotel, classroom, market, bus, etc. Users might access their computing resources from wireless portable machines and also through stationary devices and computers connected to local area networks" [290].

An central aspect of the Ubicomp model is *context-aware computing*, which refers to a generalized approach for building software applications capable of examining and adapting to an individual's (often changing) context and requirements [293]. Context-aware systems represent evolutionary syntheses of distributed systems and mobile computing that may involve sophisticated parameter and compositional adaptation. However, unlike conventional adaptive computing scenarios, where the distributed applications and participating components are well-known and relatively static over time, Ubicomp scenarios are characterized by rapidly changing execution environments where the available computational resources may fluctuate and may be unknown a priori [145]. In this regard, notable variations include the "processors available for a task, the devices accessible for user input and display, the network capacity, connectivity, and costs may all change over time and place. In short, the hardware configuration is continually changing. Similarly, the computer user may move from one location to another, joining and leaving groups of people, and frequently interacting with computers while in changing social situations" [290].

As context-aware computing involves complex aspects of distributed, mobile and adaptive computing, supportive middleware is well-recognized as an essential requirement for non-trivial systems [147]. Broadly, context-aware systems acquire and model contextual information from the user's environment as a means of facilitating adaptation within a domain-specific application. Similar to conventional distributed computing middleware, context-aware middleware often provides infrastructure for facilitating data communications between participating distributed components. As such, most context-aware approaches extend *existing* distributed computing models such as Remote Procedure Call, Distributed Object Communications and Message-Oriented-Middleware (see section 2.2.1). In addition, most context-aware middleware provides support for automated acquisition and modeling of context information from the user's physical environment as a means of informing application adaptation. However, despite the similarities between systems, a variety of approaches are often used to address heterogeneous "requirements and conditions such as the location of sensors (local or remote), the amount of possible users (one user or many), the available resources of the used

devices (high-end-PCs or small mobile devices) or the facility of a further extension of the system" [21].

Researchers have attempted to derive generalized conceptual frameworks whereby context-aware systems can be decomposed, categorized and analyzed. In this regard, layering is often utilized as a structuring principle to assure desirable system properties, such as separation of concerns, modularity, extensibility and flexibility [117]. As such, the functionality of context-aware systems can often be segmented into well-defined layers as a means of decoupling low-level context acquisition and modeling tasks from higher-level information management, dissemination and application adaptation tasks. For example, low-level sensing and data retrieval can be encapsulated by a specific layer (or layers) similar to the physical and link layers of the TCP/IP model [323]. While some discrepancies exist, prominent conceptual frameworks differ only by the level of detail. For example the Henricksen framework [146] simply suggests that low-level data acquisition details should be placed into the lowest layers, whereas the Mäntyjärvi framework [185] specifies separate layers for sensor measurement, preprocessing and quantization. Further, each framework suggests a mechanism for managing (and potentially storing) context information that can be used by a context-aware application or group of applications. Finally, application-level adaptation decisions and behavior are uniformly encapsulated into the highest layer or layers (with the Henricksen framework providing a separate "decision support tools" layer). Figure 7 provides a layered conceptual framework that is intended to guide the discussion of context-aware systems throughout the remainder of this chapter.

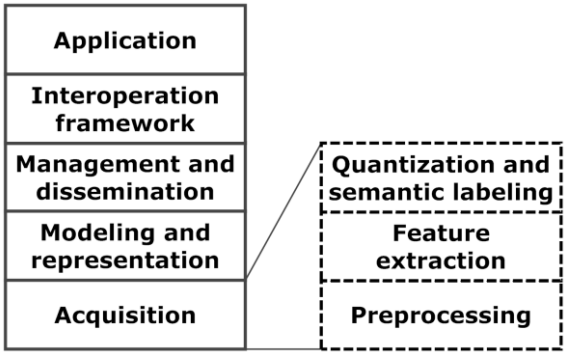


Figure 7: Layered conceptual framework for context-aware applications

2.3.1 Defining Context and Context-awareness

When discussing the operational details of context-aware systems, definitions of "context" are often intuitively understood, yet difficult to apply when engineering practical systems. In this section, we summarize findings from Dey and Abowd [89], who provide insight into the various conceptualizations of the terms context and context-awareness. One of the first general descriptions of context was introduced by Schilit and Theimer [292], who included basic notions such as location, proximate users and objects (plus related state changes). Their definition is similar in spirit to the definition provided by Brown et al. in [42], where additional aspects such as time, temperature and season are considered. Dey [88] includes several user-centric notions of context, such as the user's orientation, focus of attention and emotional state. Related, Franklin and Flaschbart [103] include the user's situation whereas Ward et al. [348] view context from the application's standpoint.

The early context definitions presented above have been criticized as overly domain-specific and difficult to apply [145]. To overcome domain-specificity, researchers have attempted to categorize the types of functionality that a context-aware system might provide. The first such categorization was provided by Schilit [293] who provided four classifications for context-aware applications based on whether the system manually or automatically obtains information and execute commands on behalf of the user. Specifically, systems that manually obtain contextually relevant information are classified as *proximate selection* applications. In these systems, available information or services are made easier to choose based on contextual information such as location or proximity (e.g. displaying a list of nearby printers). Similarly, systems that allow users to manually execute service commands are classified as *contextual command* applications. Next, systems that automatically discover and bind to contextually relevant computation are categorized as *automatic contextual reconfiguration* applications. Finally, systems that automatically execute commands based on contextually bound services are classified as *context-triggered* action applications.

Pascoe [250] provides a more elaborated version of Schilit's classification scheme, which includes descriptions the fundamental features of a context-aware system. Pascoe's taxonomy implies a hierarchical application of context-awareness features, whereby contextual sensing (or *acquisition*) is introduced as the foundation of a context-aware system. Contextual sensing refers to an application's ability to discover relevant information from the environment such as sensor data or user preferences. Additionally, an application may undergo *contextual adaptation*, whereby the application alters its state based on the discovery of context information. An important contextual adaptation strategy is identified as context-based *resource discovery*, whereby contextual information leads to the discovery of relevant distributed services. Finally, discovered services may prompt an application to undergo *contextual augmentation*, whereby the services are dynamically composed into the application at runtime.

Context-aware feature taxonomies have led to more applicable definitions of context and context-awareness. Perhaps one of the most widely adopted operational definitions was suggested by Dey and Abowd [89], who defined context as "any information that can be used to characterize the situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves." Importantly, this definition highlights the central role of the application as the arbiter of context information significance; reducing system-level reliance on considerations such as context detection, determination and response. Related, Dey and Abowd further differentiate between primary and secondary context. Briefly, a primary context answers questions such as who, what, when, and where and may involve information such as location, identity, time, and activity. Primary context information may lead to additional sources of derived context information, known as secondary context information. For example, understanding a person's identity (i.e. primary context information) may help a system discover the person's address (i.e. secondary context information).

Dey and Abowd also provide a domain-neutral definition for *context-awareness*, whereby a system is considered context-aware if it "uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task" [89]. Importantly, this definition helps clarify the generality of contextual information and its relevance to many existing software applications. For

example, a traditional Web browser might be viewed as context-aware if it exploits user preferences to control issues related to default text size and home page. However, user preference information arguably represents an impoverished type of contextual information.

More sophisticated context-aware systems generally utilize techniques for *automatically* discovering and utilizing context information. In this regard, Hendrickson provides several additional distinctions between context, context model and context information, which are applicable in more complex systems. These definitions are summarized below as presented in [145]:

- The *context* of a task is the set of circumstances surrounding it that are potentially relevant to its completion.
- A *context model* identifies a concrete subset of the context that is realistically attainable from sensors, applications and users, and able to be exploited in the execution of the task. The context model that is employed by a given context-aware application is usually explicitly specified by the application developer, but may evolve over time.
- *Context information* is a set of data, gathered from sensors and users, that conforms to a context model. This provides a snapshot that approximates the state, at any given point in time, of the subset of the context encompassed by the model.

Based on the above definitions, Hendrickson goes on to describe several important characteristics of context information in [146]. In her work, she classifies context information into four principle classes, including sensed, static, profile and derived. Briefly, *sensed* information refers to information gleaned from a user's environment through the use of physical sensors such as accelerometers, radio frequency receivers, etc. Such information is often highly dynamic, rapidly changing and may be affected by issues such as faulty connections, sensor drift, mis-calibration, wear and tear and humidity [185]. *Static* information refers to relatively fixed information such as a device's local resources or available communication channels. Static information is generally provided using domain-specific means (e.g. registry settings). *Profile* information refers to manually created data provided by the user. Profile information is often highly accurate for limited time periods, but can be affected by staleness if the user fails to continually update the profile. Finally, *derived information* refers to information that is automatically interpreted from other context types such as sensor data, user profiles or application-specific mechanisms. Notably, derived information is often error prone because of low-level error propagation and brittle heuristics. Table 4 provides an overview of the properties of the aforementioned context information types as presented in [146].

Context type	Persistence	Quality issues	Source of inaccuracy
Sensed	Low	May be inaccurate, unknown or stale	Sensing errors; sensor failures or network disconnections; Delays introduced by distribution or the interpretation process
Static	Forever	Usually none	Human error
Profile	Moderate	Prone to staleness	Omission of user update in response to changes
Derived	Variable	Subject to errors and inaccuracies	Imperfect inputs; use of a crude or oversimplified derivation mechanism

Table 4: Typical properties of context information (from [146])

The context information types presented above are subject to a variety of errors and may exhibit considerable uncertainty with regard to quality [146]. Hendrickson describes four principle types of information quality problems, including unknown, ambiguous, imprecise and erroneous. Briefly, *unknown* refers to information that is not known or not understood. *Ambiguous* refers to the possible variance between distinct information describing the same attribute (e.g. two distinct sensor values for a given attribute). *Imprecise* refers to information that represents correct, yet inexact values for a given attribute (e.g. proximity values with relatively high margins of error). Finally, *erroneous* refers to information that does not match the state of the attribute being measured (e.g. a disconnected temperature sensor). Related, several measures for specifying the uncertainty of context information have been proposed, including quality measure specification [173], confidence metadata [196] and semantic quantization [185].

The characteristics described above emphasize the domain-specificity and complexity of context information. For example, sensors have been developed to detect a wide variety of environmental attributes such as acceleration, temperature, light, image data, radio frequency signals, etc. Interactions with sensors often involves domain-specific issues such as electrical specifications, interaction protocols, data integrity measures and post-processing mechanisms (e.g. sensor fusion) [349]. Moreover, interacting with a given sensing system requires a precise understanding of its underlying physical organization along with a detailed understanding of the temporal and uncertainty constrains of the information in question [216]. For example, sensors may be part of a device's local capabilities or deployed throughout the target environment as collections of semi-autonomous sensing equipment capable of self-organization and adaptation. The raw information obtained from a sensing system may require aggregation or other post-processing. For example, relatively high-order context information such a device's physical position may be derived from a GPS receiver that aggregates low-level timing signals and calculates associated time-of-arrival variance using two-dimensional trilateration [237]. Notably, the use of sophisticated contextual information often necessitates the participation of domain-experts capable of effectively modeling and representing such information types [173]. Accordingly, Ocean provides dedicated abstractions in this regard (see section 4.3.2).

2.3.2 Context Acquisition

A primary feature of most context-aware systems is an ability to obtain useful environmental information that may be relevant to the interactions between a user and an adaptive application. As discussed in the last section, context information can be classified as sensed, static, profile and derived. To accommodate the unique requirements of each information type, context-aware applications typically employ sensors that are adapted to the characteristics of the information under consideration. (It should be noted that "sensor" refers to any element capable of capturing contextual information.) Sensors generally encapsulate the domain-expertise required for managing and interpreting low-level sensor data. Such encapsulation insulates a software system from the low-level considerations inherent to many sensor types. For example, an RFID sensor may produce a stable and accurate list of tags currently in its energizing field by handling low-level details such as energy level adaptation, response aggregation and collision avoidance [301].

Indulska and Sutton [157] have classified sensors into three main types, including physical, logical and virtual. *Physical sensors* refer to devices capable of capturing physical attributes from the environment (also known as context atoms [185]). Physical sensors are capable of capturing a wide variety of attributes (e.g. light, sound, pressure, temperature, switch position, etc.) using domain-specific measurement hardware and associated software control systems. *Virtual sensors* are generally pure software systems that capture context atoms from the user's local computing environment. Virtual sensors have been used to capture user profile information, mouse clicks, keyboard input, email addresses and raw scheduling data [21]. Finally, *logical sensors* use low-level context atoms from both physical and virtual sensors to derive higher-order context information that better reflects the situation of the user. In this respect, logical sensors are similar to the "derived" context information type discussed by Hendrickson [146]. Examples of derived information from logical sensors may include contacted email addresses or scheduling conflicts, whereas derived information from physical sensors may include physical position estimations based on an analysis of the signal strength values from multiple radio sources [188].

In order to utilize sensor-derived information within a context-aware system, raw measurement data generally require transformation into more meaningful information [333]. Such transformations are often sensor-specific and adhere to the following steps, as described by Korpipaa et al. [185]. First, *measurement* refers to the capture of relatively unstructured sensor data flows such as electrical signal measurements. Next, *preprocessing* refers to the construction of more structured data that contain a certain number of samples (e.g. time dimension quantization) and generic features for each time interval. Next, *feature extraction* delineates "interesting" segments of the data (provided by the preprocessing phase) into distinct sets that can be analyzed together. Finally, *quantization* is then applied to extracted feature sets in order to produce more meaningful context information. Notably, the quantization method employed is dependent on the characteristics of the feature data and may include approaches such as fuzzy sets or crisp limits [101]. Figure 8 illustrates two example quantization approaches applicable to audio sample data.

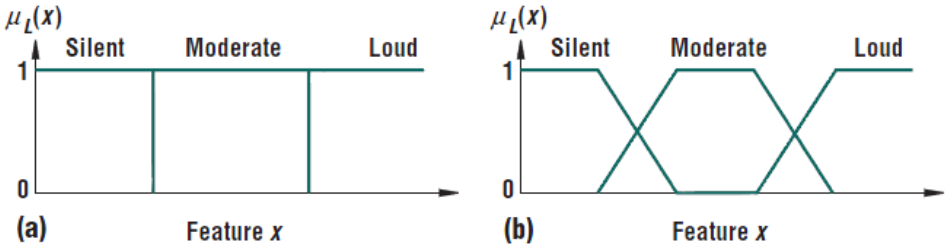


Figure 8: Examples of (a) crisp and (b) fuzzy context quantization (from [185])

Once sensor data has been acquired and transformed, it must be integrated within a software system in order to drive runtime adaptation. Early context-aware systems such as Xerox Parc's PARCTAB system [347] relied on dedicated acquisition techniques such as direct sensor access [65], whereby client applications are hardcoded to utilize specific sensing equipment, device drivers and specially developed instrumentation. For example, PARCTAB represented a Ubicomp infrastructure composed of custom-built hardware devices in three form-factors: inch-scale Tabs; foot-scale Pads; and yard-scale Liveboards. These devices, which were available throughout the Xerox PARC facilities, were connected through a dedicated infrared network capable of both data transfer and device position estimation. Initially designed with 25 infrared network cells, PARCTAB was eventually extended to 50 cells and became the foundation for several context-aware applications that allowed researchers to study the effects of software services capable of continuous connectivity and contextualized use.

Similar to PARCTAB, many early context-aware systems utilized direct sensor access as their primary context acquisition method. For example, Active Badge [345] utilized direct sensor access techniques to provide routing of telephone calls based on the known locations of people within office buildings. Developed in the early 1990s by the Olivetti Research Lab in Cambridge England, the Active Badge system features a lightweight, power-efficient identity badge design that helps locate the wearer by emitting a unique infrared (IR) signal every ten seconds. Additional system infrastructure and instrumentation is used to received and identify emitted IR pulses using a custom-built network of infrared receivers positioned in rooms and hallways. Further refinement of the Active Badge system resulted in Active Bat [140], which improved positioning accuracy and speed. Additionally, Schilit extended PARCTAB with a service-based application architecture supporting management of time-varying resources, dynamic configuration, opportunistic interaction and contextual customization [293]. Shortly thereafter, context-aware tour-guide systems began to emerge such as Cyberguide [1] and Lancaster GUIDE [78], which provided tourist information to users through the wireless delivery of contextualized media to dedicated mobile devices.

More recently, the prohibitive expense of many positioning systems motivated further research into developing more cost-effect solutions. In this regard, MIT's Cricket system [258] provides fine-grained ultrasonic positioning information based on low-cost, off-the shelf components. Similarly, Patel et al. [317] proposed a sub-room localization technique that exploits residential power-lines as a universally available positioning infrastructure. Further, Rehman et al. [332] developed an indoor localization system, called CILoS, which is based on pre-existing code division multiple access

(CDMA) mobile-phone infrastructure and is capable of accurately differentiating between floors of a multi-floor building.

While direct sensor access approaches are relatively straightforward to construct and are often effective in limited application scenarios, they offer little in terms of extensibility and reuse [21]. For example, the Smart Floor [242] is capable of identifying and localizing users based on footstep force profiles detected by specially engineered floor tile; however, significant instrumentation requirements has limited its widespread use. Accordingly, beginning in the late 1990s several projects began to explore the application of software modularization to the task of context acquisition and processing. For example, Georgia Tech's Cyberdesk project [88] addressed dynamic software integration with a Java-based application framework in order to enable context-aware discovery and integration of software modules. Cyberdesk's supported context types included physical location, nearby objects, time, application data, etc. Notably, Cyberdesk represents one of the first Ubicomp projects to provide an application programming interface (API) intended for service developers.

Other projects have explored techniques for abstracting rich contextual data from high-level applications. Inspired by Cyberdesk, Salber et al. proposed the Context Toolkit [276], which provides a component-based architecture and distributed infrastructure designed to support the aggregation of sensor data through several abstractions. First, *Widgets* provide simplified access to context information through reusable and customizable sensor wrappers. Next, *Interpreters* help transform low-level context data into more useful high-order information (e.g. translating raw sensor data into location coordinates or room identifiers). Finally, *Aggregators* combine multiple types of sensor data into unified context information. Notably, Widgets incorporate support for context histories, although neither user profile information nor uncertainty representation is provided. Related, the Technology for Enabled Awareness (TEA) project [294], proposed by Schmidt et al., espouses a simple, yet flexible layered architecture designed to encapsulate a heterogeneous set of underlying sensors. TEA provides several context information aggregation methods along with a set of mechanisms for quantizing acquired low-level data into more meaningful high-order data using rule-based algorithms, statistical methods and neural network techniques.

2.3.3 Context Modeling and Representation

Sensor measurement, preprocessing, feature extraction and quantization result in structured information that requires semantic labeling before it can be used by a context-aware application [185]. In context-aware software, semantic labeling is generally known as *context modeling*. Broadly, a context model encapsulates the syntax and semantics of a given context acquisition domain as machine-readable and (often) temporally constrained *native context data* (NCD). The format of NCD can range from application-specific data structures that may be only useful for a single application type to well-known industry standards that have been developed and ratified by organizations such as the International Organization for Standardization (ISO). Importantly, a context model represents a mechanism for capturing environmental information such that it "can be used to characterize the situation of entities [in order to] provide relevant information and/or services to the user, where relevancy depends on the user's task" [89]. As such, resultant NCD must be capable of expressing the often complex characteristics of an environment or situation, as discussed in section 2.3.1. Additional

aspects of context models include partial validation, expressions of incompleteness, ambiguity and applicability to existing environments [316].

The majority of context-aware computing research has focused on developing specific context acquisition and management systems rather than widely applicable context modeling techniques [145]. Consequently, most current context models are defined for the requirements of a given research project. However, recent work in has begun to address more generalized modeling approaches that allow domain-experts to express complex context information and provide related mechanisms for syntax parsing, feature extraction, interpretation and comparison [146]. While existing context-aware systems rarely support externally developed context modeling techniques, they are increasingly regarded as important for modeling real-world environments [49]. As such, the following sections summarize the findings of Strang and Linnhoff-Popien [316], who investigated several notable context modeling approaches and related NCD formats.

2.3.3.1 Key-Value Models

A popular context modeling approach is the key-value model, which represents contextual information as collections of keys and associated values. Keys are generally text-based entities that denote well-understood context designations such as temperature or location. Values refer to the quantized context data associated with the keys. Importantly, the key-value approach relies on a shared understanding of the key elements along with the syntax and semantics of the value data. Early context-aware systems utilized the key-value model because of its relative simplicity (see Schilit et al. [290]); however its flexibility has promoted adoption in more recent context-aware frameworks such as Capeus [280] and popular service discovery protocols (e.g. SLP [136] and JINI [342]). However, while widely used, the key-value model suffers from a lack of expressiveness [145] and a lack of sophisticated structuring [316], which limits its use in wide-area context-awareness scenarios.

2.3.3.2 Markup-Scheme Models

The Markup scheme context model refer to hierarchically structured contextual attributes and associated values; often taking the form of generic XML-based data or more complex representations such as the Standard Generic Markup Language[5]. Markup schemes are commonly used for modeling static or dynamic profile information (see section 2.3.1). Examples of markup-based profiles include the Composite Capabilities/Preferences Profile (CC/PP) [344] and the User Agent Profile[6]. Markup models also may integrate contextual attributes with existing semantic models. For example, the Comprehensive Structured Context Profiles (CSCP) proposed by Held et al. [142], combines profile information with semantic representations based on RDF/S [367]. An example CSCP profile is shown below in Figure 9, as presented in [142].

[5] http://www.w3.org/MarkUp/SGML/
[6] http://www.wapforum.org

```
<?xml version="1.0" encoding="UTF-8"?>
  <rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:cscp="context-aware.org/CSCP/CSCPProfileSyntax#"
    xmlns:dev="context-aware.org/CSCP/DeviceProfileSyntax#"
    xmlns:net="context-aware.org/CSCP/NetworkProfileSyntax#"
    xmlns="context-aware.org/CSCP/SessionProfileSyntax#"
    <SessionProfile rdf:ID="Session">
      <cscp:defaults rdf:resource=
        "http://localContext/CSCPProfile/previous#Session"/>
      <device><dev:DeviceProfile>
      <dev:hardware><dev:Hardware>
      <dev:memory>9216</dev:memory>
      </dev:Hardware></dev:hardware></dev:DeviceProfile>
      </device>
    </SessionProfile>
  </rdf:RDF>
```

Figure 9: A context representation based on the CSCP profile (from [142])

2.3.3.3 Ontology-based Models

The integration of RDF as a component within markup-based models parallels related efforts in ontology-based context models and related reasoning algorithms. Ontology-based models attempt to encapsulate a specific domain of knowledge by formalizing and standardizing a set of concepts and associated relationships. The concrete encapsulation of semantics and relationships in this way is referred to as an *ontology*, which has been defined as "a formal explicit description of concepts in a domain of discourse (or classes), properties of each class describing various features and attributes of the class, and restrictions on properties" [65]. The usefulness of an ontology is often evaluated in terms of its descriptive power, support of reasoning algorithms, encoding scheme (i.e. NCD format) and overall adoption (i.e. shared understanding) [315]. In this regard, ontology-based context models generally derive from the requirements of probabilistic reasoning approaches such as Bayesian networks [260], fuzzy logic [185], or ad-hoc situation determination [329].

Ontology-based models are often used as inputs to reasoning algorithms, which attempt to infer contextually-relevant knowledge as a means of mediating system behavior. One of the first uses of ontologies in context-aware computing was described by Öztürk and Aamodt [245], who proposed normalizing and combining various knowledge domains based on an analysis of psychological studies related to recall and recognition. Additional notions of encoding have also been explored by approaches that exploit the generalized Web Ontology Language (OWL) as a means of developing domain-specific ontologies [344]. Important ontologies specific to context-awareness include the Context Ontology Language (COOL) [315], the Standard Ontology for Ubiquitous and Pervasive Applications (SOUPA) [65] and the Ontology for Mobile Device Sensor-Based Context Awareness [184]. Examples from the Ontology for Mobile Device Sensor-Based Context Awareness are shown in Table 5 (from [184]).

	Context type	Context	Value	Confidence	Source	Attributes
1	Environment:Light:Type	Natural	-	1.0	Device Sensor	Confidence = Fuzzy membership
2	Environment:Temperature	Warm	21	0.8	Device Sensor	Confidence = Fuzzy membership Valueunit = Celsius
3	Device:Activity:Placement	Athand	-	1.0	Device Sensor	Confidence = Crisp

Table 5: Examples from the Ontology for Mobile Device Sensor-Based Context Awareness

2.3.3.4 Logic-based Models

Logic-based context models attempt to encapsulate domain knowledge into more formalized definitions of facts, expressions and rules. Early logic models emerged from artificial intelligence research, where context information became increasingly viewed as useful for reducing the number of assumptions required by reasoning techniques [185]. Logical-models have also been used to constrain reasoning about the intended goals of a human user by modeling a subset of the environmental state [119]. Akman and Surav [5] used logic-models in deriving the Extended Situation Theory, which investigated the use of context-derived situations as a means for enhancing natural language analysis. Such approaches develop "mathematically-based tools to analyze, in particular, the way context facilitates and influences the rise and flow of information" [87]. Additionally, Work by Gray and Salber [127] investigated the use of logical-models as inputs to first-order predicate logic, while Bacon et al. [16] modeled location as a fact type in a rule-based inference engine used to mediate the behavior of a multimedia system. A logic model example from Situational Theory is shown in Figure 10.

If it freezes, Ovett will be cold.

This utterance expresses an instance of the constraint that, if a person's environment is freezing, and that person is scantily clad (such as a runner), then that person will be cold. More precisely, let S and T be the situation-types:

$$S = [e^{\cdot} \mid e^{\cdot} \mid= \langle\langle \text{freezing}, t^{\cdot}, 1 \rangle\rangle \wedge \langle\langle \text{present-in}, p^{\cdot}, e^{\cdot}, t^{\cdot}, 1 \rangle\rangle \wedge \langle\langle \text{scantily-clad}, p^{\cdot}, t^{\cdot}, 1 \rangle\rangle]$$

$$T = [e^{\cdot} \mid e^{\cdot} \mid= \langle\langle \text{cold}, p^{\cdot}, t^{\cdot}, 1 \rangle\rangle]$$

where $e^{\cdot}$ is a situation parameter, $t^{\cdot}$ is a temporal parameter, and $p^{\cdot}$ is a parameter for a person. Then the described situation for u_3 is the world and the propositional content is:

$$w \mid= (S \Rightarrow T)[f]$$

where f anchors $p^{\cdot}$ to $SO = c_{u3}$(Ovett).

Figure 10: An example of a logical context model based on Situational Theory (from [87])

2.3.3.5 Graphical Models

The representation of context semantics and relationships has been further explored by the development of graphical context models. Existing graphical modeling techniques such as the Unified Modeling Language (UML) [110] are widely used in software engineering as a means of expressing relationships between elements in software systems. As the UML is sufficiently general, it has also been used to model contextual information. For example, Bauer [25] used UML to model context information in air traffic management scenarios. A more complex example is provided by Henricksen et al. [145], who proposed the Context Modeling Language (CML) as an extension of Object-Role Modeling (ORM). Broadly, ORM defines a methodology and a related graphical means of modeling complex contextual relationships as sets of entities constrained by facts [145]. In ORM, facts represent informative statements about a relationship between entities in a context situation. Examples of fact constraints include "is of type" and "permitted to use." Importantly, facts can be used to represent the various elemental context types (see section 2.3.1) such as sensed, static, profile and derived. While ORM is useful for graphical modeling, an XML-based representation of the CML, called XCML [270], has been proposed as an approach for transforming graphical ORM diagrams into runtime context model representations suitable for dissemination within context-aware frameworks. An example context model representation based on ORM is shown in Figure 11.

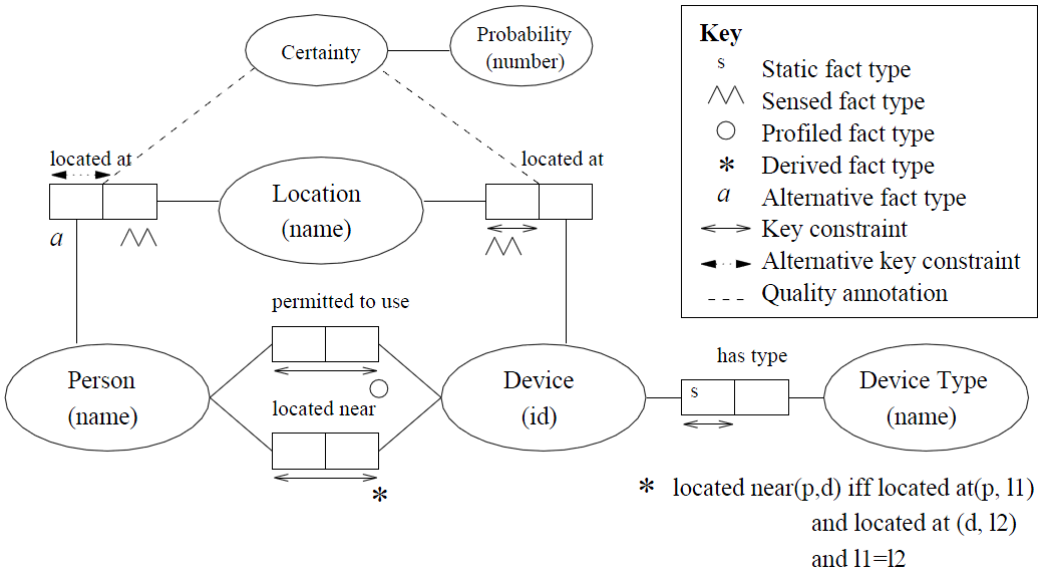


Figure 11: A graphical context model based on Object Role Modeling (from [146])

2.3.3.6 Object-oriented Models

While graphical context models capture context information visually, object-oriented context models attempt to represent context information as machine-processable objects that expose well-known interfaces and data-types [316]. Typically, object-oriented models rely on domain-specific representation formats that may imply language-specific data types, serialization mechanisms and framework support. For example, Bouzy and Cazenave [40] developed an object-oriented model which represents characteristics of a 4000 year old game, termed "Go," which is popular in Japan, China and Korea. Their approach, termed "Computer Go," uses objects to model Go-specific context information (e.g. temporal, spatial and goal) as a means of reducing the number of assumptions

required by game-play reasoning techniques. Related, location based systems utilize a variety of location sensing techniques, including triangulation, scene-analysis and proximity (see section 4.3.2). In such systems, object-oriented models have been used to represent values such as time-of-flight [258] or provide native representations of standard geo-location markup models [237]. Additionally, the Technology for Enabled Awareness (TEA) project [294] provides an object-based infrastructure to supports mobile devices that interact simultaneously with multiple context sources. TEA addresses the often complex requirements of physical and logical sensor types through the definition of an object-oriented contextual model known as a *cue*. In TEA, cues encapsulate sensor measurement, preprocessing, feature extraction and quantization and may be combined or aggregated in order to model more complex situations. As an example, TEA aggregates time-series sensor data as a means of constructing higher-order context information using Kohonen's Self-Organizing Maps (see Figure 12).

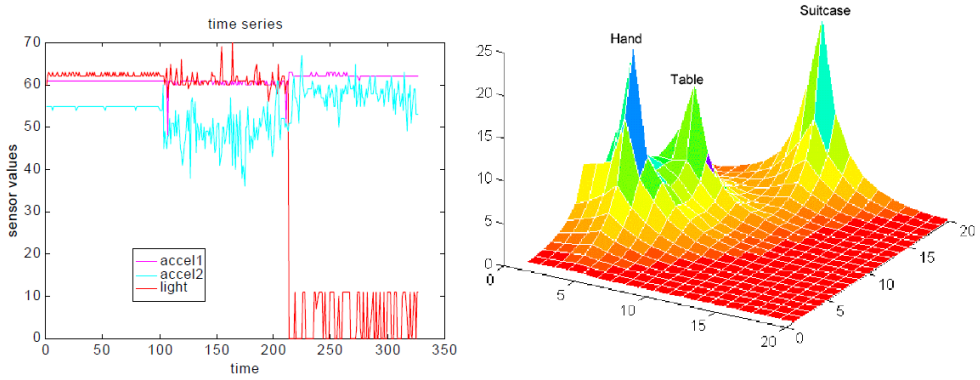


Figure 12: Time series sensor data (left) and Kohonen Clustering of sensor readings (right) (from [294])

2.3.4 Context Management and Provisioning

Provided that contextual information can be acquired, modeled and represented, it is well-recognized that additional approaches are necessary for managing and provisioning context information so that it can be used by adaptive applications [21, 146]. As previously discussed, early context-aware systems were often designed to accommodate a limited set of context instrumentation and associated representation formats; hence, such systems were often tightly coupled with the underlying sensing technologies [276]. However, as the scope of context-aware systems expanded, the complexity of building non-trivial systems increased dramatically [147]. Notably, mobile distributed systems require techniques for acquiring reliable sources of context information in uncertain environments, where sensor sources may be rapidly changing, noisy, partially true and heterogeneous [185]. In this regard, the complexity inherent in managing and provisioning such information is seen as a major obstacle for the development of context-aware software [146].

Over the years, researchers have purposed a variety of context management and provisioning approaches. As discussed by Riva [268], several categories of approaches can be identified in this regard, as shown in Figure 13. Briefly, *internal context provisioning* localizes context acquisition and related management systems within a single device platform. Next, *external centralized context provisioning* utilizes a federated service infrastructure to aggregate context data from dedicated sensors located within the environment; providing additional services such as context processing,

reasoning and dissemination. Finally, *external distributed context provisioning* combines federated infrastructure with additional peer-to-peer mechanisms for direct, inter-device context information sharing. This section describes a representative sampling of important approaches from each category.

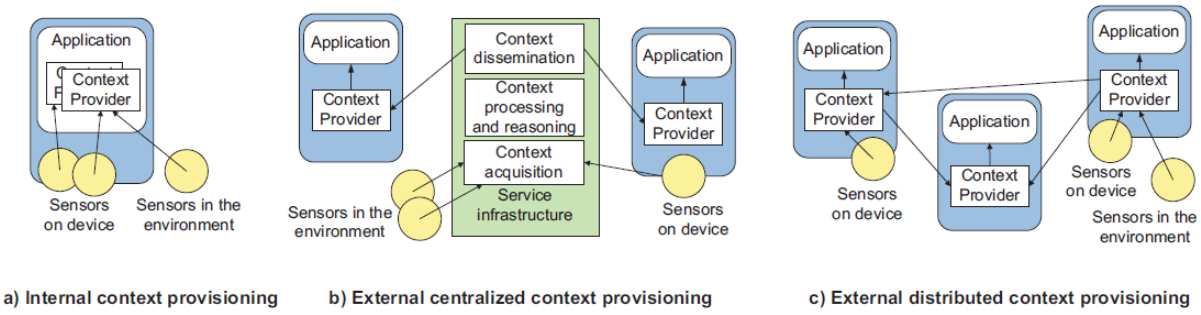


Figure 13: Context management and provisioning categories (from [268])

Many early Ubicomp projects such as those discussed in the last section relied upon internal context provisioning, where the application software is tightly coupled to local sensing hardware and device drivers. Although some early projects involved notions of generalized context management and processing [293], the tight coupling common to the underlying instrumentation prevented more generalized use. One of the first techniques for separating context management from the underlying acquisition approach was provided by Salber and Abowd [275] who suggested the notion of a generic *context-server*. According to Salber and Abowd, a context server "gathers raw local context data through sensors, stores it and provides context data access to local and remote applications. In addition, each context server runs services, called context synthesizers, which act on local or remote context data to generate context information at a higher level of abstraction." Further, the introduction of the context-server mechanism was intended to impart the following system properties: to allow networked applications to access local and remote context data in heterogeneous environments; to accommodate a variety of applications, sensors, and operations on context data; and to preserve a history of sensed contextual data.

The notion of the generic context server became widely adopted and extended by projects that utilized external centralized context provisioning techniques. For example, the Strathclyde Context Infrastructure (SCI) [121] provides dynamic composition and representation of contextual information through a layered architectural model. Similar to other middleware-based context-server approaches, SCI defines a distributed model whereby *multiple* context-servers are deployed throughout an environment to support a network overlay of partially connected nodes. Each context server is deployed within a specified operational area (called a "range") and is responsible for providing both local and global services to dependent applications (e.g. context event subscription and inter-range communications).

Other notable approaches investigated techniques for external distributed context provisioning. For example, the Hydrogen approach [152] extends the generic context-server to support the needs of mobile devices. Hydrogen addresses the increased dynamism of mobile computing environments and resource constraints of mobile devices through a three layer architecture that supports extensible context types, disconnected operation and peer-to-peer context sharing. Similar to Hydrogen, the

Solar project [63] provides a context management approach capable of discerning and delivering higher-order context to applications using sensor fusion techniques. Notably, Solar involves aspects of both service composition and context transformation. Solar defines a distributed context-server-based overlay architecture where each host (called a "planet") is responsible for a given operational area. Multiple planets manage access to connected sensing equipment and provide aggregation and fusion services (among much else). Notably, Planets may autonomously or cooperatively select sensors and aggregate (or fuse) incoming data streams to produce high-order context data for use by subscribing context-aware applications.

While context-server techniques have predominated in Ubicomp research throughout the last several years, most suffer from intractable challenges in large-scale deployments (see section 3.2). Hence, a variety of additional context management techniques have been proposed. For example, Hewlett-Packard's Cooltown project [179] takes a more general approach to context management by embedding context information directly into physical objects with the intention of bridging the Web and the physical world. Cooltown extracts context information using two principle mechanisms. First, *direct sensing* uses a dedicated short range wireless protocol to send URIs between devices using wireless technologies such as IR and Bluetooth (called eSquirt). Using direct sensing, objects provide Cooltown clients high-order contextual information without the need for context interpretation; however, participating objects must be retrofitted with Cooltown technology. Cooltown's second approach, termed *indirect sensing*, is based on an external context registry that supports associations between Web Resources and specific types of context data; allowing externally stored context information to serve as the bridging mechanism. Related, Intel's PlaceLab project [291] explored the extraction of existing context information by reappropriating common radio signal sources (termed *beacons*) such as 802.11 access points, Bluetooth radios and Global System for Mobile Communication (GSM) cell towers. As discussed in detail in section 7.2.2, PlaceLab proposes a community-centric acquisition model, whereby volunteers use specially outfitted laptops and "stumbler" software to *war-drive* large geographic areas (i.e. search for beacons using vehicles).

As richly described in [89] and [146], real-world contextual information is often provided by heterogeneous sources that may not be foreseen at design time. Accordingly, context management approaches have begun exploring mechanism for deploying context acquisition and modeling components to mobile devices *at runtime*. For example, our previous context-aware system, called Aladin [49, 50], provides a modular, client-centric approach for *cross-domain* context acquisition and modeling. Operating without the need for dedicated instrumentation, the Aladin Framework provides a plug-in-based architecture, whereby context modeling mechanisms are dynamically deployed to mobile devices based on the capabilities of the device and characteristics of the environment. Using plug-ins, Aladin exploits the computational capabilities and integrated sensing equipment of commodity hardware as a means of modeling context information from common environmental sources such as GPS, RF signals, imaging data and other sensor types. Other client-centric approaches have been proposed. For example Riva developed the CONTextfactORY (Contory) [268] as a "middleware specifically designed to accomplish efficient context provisioning on mobile devices." While not addressing runtime integration of context acquisition and modeling plug-ins, Contory does provide dynamic selection of inbuilt context acquisition strategies and allows for context sharing

between mobile devices. Other notable context management platforms for mobile phones include the context-aware Blackboard architecture from Korpipää et al. [185] and the ContextPhone [259] rapid prototyping platform from Raento et al.

2.3.5 Context-aware Component Interoperation

Based on the context management and provisioning techniques described in the previous section, many context-aware systems utilize modeled context information to discover, select and interoperate with contextually relevant components at runtime. Accordingly, context-aware systems are often classified as service composition frameworks, whereby an dependent application's structure and behavior derive from the integration of component constituents [208]. Accordingly, context-aware systems have much in common with the conventional adaptive systems discussed in section 2.2.3. For example, context-aware systems may dynamically adapt their runtime behavior and capabilities to fit the characteristics of a mobile user's current situation. Moreover, context-aware systems may extend the architectural principles common to compositionally adaptive systems such as separation of concerns, computational reflection and component-based design [73]. In this regard, context-aware systems face many of the same challenges as adaptive distributed systems, including automated checking of functional and nonfunctional system properties (i.e. assurance), security, interoperability and decision making [209]. However, as context-aware systems often operate across *multiple* organizational boundaries and may encounter heterogeneous networks and distributed components, additional adaptive approaches have been devised. This section outlines key adaptive techniques in this regard and discusses several representative approaches.

At their foundation, a primary task of many context-aware systems is the advertisement and discovery of relevant components for runtime composition [180]. Over the last decade, the number of networked services available on the local link (e.g. printers, routers and media systems) and across the enterprise and Internet (e.g. through Web services) has increased dramatically [24, 314]. Significant interest in developing techniques for advertising and discovering networked services has resulted in the development of various *service discovery protocols*, including the Service Location Protocol (SLP) [135]; Zero Configuration Networking (Zeroconf) [314]; Universal Plug and Play (UPnP) [169]; and JINI [342]. While differing in approach, service discovery protocols are broadly designed to facilitate service advertising (i.e. publishing a service's capabilities and interfaces) and service discovery, whereby appropriate services can be located and selected based on certain desirable characteristics. In terms of scope, UPnP and Zeroconf target home automation applications, whereas SLP and JINI are designed primarily for enterprise scenarios. Related, JINI provides support for the runtime delivery of executable software and UPnP promotes industry-wide standardization of specific service and device types. Notably, approaches such as Zeroconf include automatic network configuration mechanisms designed to support ad-hoc or isolated network scenarios. While such service discovery protocols are not primarily intended for Internet-based scenarios, the Web Services Dynamic Discovery protocol (WS-Discovery) specification [26] has been proposed as a mechanism for multicast advertisement and discovery of SOAP Web services; however, as of the time of writing, WS-Discovery remains unstandardized.

While some service discovery protocols have achieved limited commercial adoption (notably Zeroconf and UPnP), most are seen as insufficient for supporting wide-area context-aware scenarios due to inherent limitations in service density, network accessibility (i.e. local-link constraints) and protocol interoperability [378]. For example, UPnP's non-standard UDP message format faces interoperability challenges in wide-area scenarios; resulting in low service density [3]. While techniques for supporting multi-protocol and multi-network service discovery have been proposed (notably MSDA [261] and Haggle [318]), they are not widely deployed or supported. Further, most conventional service discovery techniques provide only limited support for contextual information and rarely accommodate sophisticated component descriptions [29]. For example, in many approaches service descriptions are expressed using simple string-based name/value tuples, which lack high-fidelity representations, constraint specifications and support for inexact service matching [59]. In contrast, context-aware scenarios require context mediation based on complex context information such as location, identity, device proximity, temperature and so forth [89]. In this regard, recent service discovery approaches for mobile scenarios (e.g. Splendor [378]) have begun introducing richer context semantics (e.g. location).

In addition to service advertisement and discovery, most context-aware systems require sophisticated distributed middleware support [145] as a means of supporting hardware abstraction, service provisioning, distributed communications and comprehensive context acquisition, management and dissemination facilities [228]. Early approaches for addressing such requirements took the form of *intelligent environments* (IEs), whereby specially instrumented physical spaces are outfitted with specialized computing infrastructure. For example, Microsoft's EasyLiving project [43] provides an architecture capable of aggregating diverse devices into a coherent user experience using heterogeneous sources of context data. Similar to related approaches such as iRoom [111] and iRos [255], EasyLiving adopts an *infrastructure-centric architecture*, whereby the environment itself provides orchestration of distributed components and related context data acquisition and modeling abstractions that rely on environmental instrumentation. In this regard, IEs typically utilize abstractions that encapsulate participating entities (e.g. sensing, input and output systems) in order to provide a unified development environment for the creation of context-aware services. As such, distributed communications and component interoperation are often proprietary and domain-specific. For example, iRoom and iRos are based on a custom designed distributed communication architecture called the EventHeap, which enables multicast-style groupware communication through a publish/subscribe model and extensible event system [111]. Similar to EasyLiving's utilization of the proprietary InConcert middleware (a predecessor of modern SOAP Web services), the EventHeap provides support for inter-process communication across heterogeneous middleware platforms. The complexity and domain-specificity of the EventHeap architecture is illustrated in Figure 14.

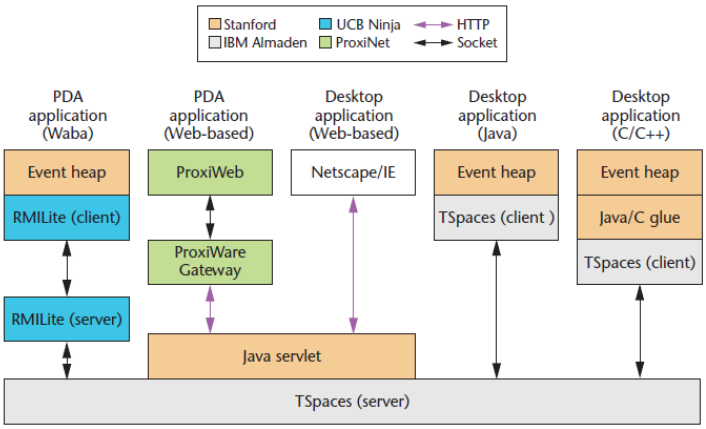


Figure 14: Overview of the EventHeap architecture (from [111])

Several other projects have extended the IE concept dramatically. For example, projects such as iQueue [70] attempt to extend the basic IE model to support more generalized Internet-based data sources that can be dynamically selected and composed at runtime. Specifically, iQueue provides mechanisms whereby data such as files, databases, newsfeeds, SOAP web services and sensor output can be described by a unified functional data specification provided by the project team. When described appropriately, iQueue is able to dynamically select data sources that satisfy an application's requested data types, reselect data sources as appropriate and mediate between incompatible data formats by transcoding. Notably, the iQueue approach is contingent on the definition of appropriate functional data specifications for each possible data source in combination with related support for data source syntax parsing. Next, the Gaia meta-operating system [272] extends conventional operating system concepts – e.g. program execution, I/O operations, file-system manipulation, communications, error detection, and resource allocation – to support distributed application development in "integrated programmable environments." Gaia is based on the notion of "Active Spaces," which constitute a "physical space coordinated by a responsive context-based software infrastructure that enhances mobile users' ability to interact with and configure their physical and digital environments seamlessly" [272]. Giaia applications are created using a set of predefined components that intercommunicate via the Gaia Kernel using a dedicated Corba-based communications middleware. An overview of Gaia's physical versus active space models is shown in Figure 15.

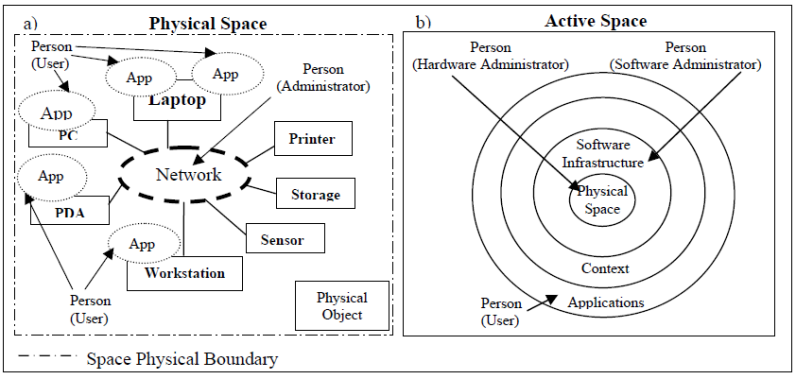


Figure 15: Gaia's (a) physical space versus (b) active space models (from [272])

Similar in scope to Gaia, the Aura project [116] focuses on minimizing user attention demands through a comprehensive middleware architecture that "spans every system level: from the hardware, through the operating system, to applications and end users." Accordingly, Aura monitors user and component activity transparently; attempting to anticipate and proactively support user needs – e.g. by proactively staging data on servers nearest to the user as a means of reducing network latency. The prototype version of Aura is based on a custom-designed Corba communications infrastructure; however, Aura also provides a connector abstraction that provides transport independence. (Notably, usage of the connector abstraction requires that wrappers be constructed for a given underlying communications protocol based on Aura's set of well-defined service interfaces.)

Unlike relatively small-scale deployments such as Gaia and Aura, the ActiveCampus project [129] proposes a more scalable context-aware system capable of "simultaneously supporting extensibility and tight integration." ActiveCampus is designed around a centralized, layered client/server architecture, whereby loosely-coupled set of interfaces provide "component contracts that are easy for implementers to satisfy (i.e., supporting innovation), yet whose behaviors are rather open (i.e., enabling integration)" [129]. ActiveCampus' interfaces are designed with the intention of providing extensibility along several dimensions, including service provisioning, context acquisition, modeled physical entities and end-user devices. Although ActiveCampus arguably provides significant benefits for larger deployments, the designers noted that its process-centric interface abstractions quickly led to increased interoperation complexity, performance problems, tight component coupling and an inability to effectively introduce new services. This observation lies at the core of Ocean's component interoperability approach that is introduced in section 3.2.8.

Recognizing the deployment limitations of conventional IE approaches, more recent efforts have focused on utilizing the capabilities of existing mobile devices and reducing reliance on infrastructure-centric architectures. For example, the Cortex middleware [310] utilizes computational reflection and related component based techniques to accommodate mobile and ad-hoc scenarios. Cortex extends the notion of autonomous *sentient objects*, which encapsulate service discovery and provide context acquisition, modeling and inferencing (i.e. high-order context quantization). Cortex also provides an event model called STEAM that allows context events to be published and disseminated without a centralized infrastructure. In contrast to IE approaches, Cortex provides architectural support for mobile devices through the adoption and extension of OpenCOM, which is a lightweight component model based on Microsoft's COM. Related approaches such as the Reflective Middleware for Mobile Computing (ReMMoC) [126] address issues related to heterogeneous mobile service platforms through the adoption of a lightweight middleware model that accommodates *multiple* service discovery protocols (e.g. SLP and JINI) and provides additional binding and interaction techniques intended to support the independent evolution of context-aware applications.

In terms of more generalized frameworks, several approaches have emerged. For example, the Java Context-Awareness Framework (JCAF) [23] provides a general-purpose, event-based runtime and related Java programming framework focused on the development of context-aware applications. Similar to peer-to-peer context management architectures such as Solar (see section 2.3.4), JCAF defines a set of extensible Context Services that are dedicated to modeling and managing particular types of context information. Context Services communicate via a related Entity Environment, which

handles context aggregation and transformation. The JCAF approach is based on the Java J2EE specifications[7] and relies on a long-lived component container for handling shared resources such as database connections and RMI stubs. A related Java-centric approach, called the Java ADhoc Application BootStrap (Jadabs) [112], extends the Java container model to include support for ad-hoc service composition on small, resource-constrained devices. Notably, Jadabs adapts conventional service-oriented computing techniques (see section 2.2.2) to the requirements of mobile devices utilizing the Java micro and standard edition runtimes.

To promote wide-area component interoperation, several context-aware projects have adopted the SOAP Web services infrastructure introduced in section 2.2.2. As a prototypical example, Keidl and Kemper [175] describe a framework for context-aware adaptable SOAP Web services. In their approach, automated Web service discovery is performed by first acquiring context information from the user's environment using extensible context modules capable of including context information within UDDI component Discovery Requests. Related, a specially designed UDDI registry is used to select appropriate Web services based on mappings between well-known context information and the associated WSDL documents describing registered Web service components. As noted by Belotti et al. [29], an important challenge for context-aware selection of Web service components is the application of descriptive semantics describing the contextual relevancy of a given component. Towards this end, context-aware Web-service registries such as SOPHIE [29] have been devised to facilitate discovery and binding of application constituents based on a semantic data model that integrates object-oriented and entity-relationship concepts. Other approaches address Web service component annotation through semantic description. For example, Manners [11] exploits notions introduced by grammar-oriented object design and Cobra [64] extends the Web Ontology Language (OWL)[8] as proposed within the Semantic Web initiative [368].

Although SOAP Web services have improved syntactical interoperability though open interface and the use of well-known Internet protocols, they rarely accommodate spontaneous, cross-domain discovery and interoperation without involving significant prior interface knowledge [336]. As noted by Edwards et al. in [96], "Interoperability among a group of devices, applications, and services is typically predicated on those entities having some degree of prior knowledge of one another." Thus, several context-aware projects have explored techniques for reducing the reliance on domain-specific interface definitions as a means of promoting widespread reuse. For example, the SpeakEasy project [96] from Xerox PARC described a new computing paradigm, called *recombinant computing*, that aims to allow arbitrary component interaction through the use of a fixed set of compact interface definitions and mobile code to allow dynamic extension of functionality at runtime. In this regard, SpeakEasy leverages conventional Web architecture techniques as a means of supporting "serendipitous interoperability – the ability for devices and services to use one another with only very restricted prior knowledge" [96]. Related projects such as SA-REST [192] have similarly demonstrated how conventional Web architecture can be used to promote cross-domain component interoperation through the Representational State Transfer (REST) architectural style (see section 3.2.8).

[7] http://java.sun.com/javaee/
[8] http://www.w3.org/2004/OWL/

2.4 Chapter Summary

This chapter presented background and related work regarding context-aware computing. It began by discussing how the rise of data communication networks led to the emergence of distributed computing. Next it described key aspects of distributed computing, including the characteristics of distributed software architectures and their associated challenges. Based on these challenges, it described several key distributed computational approaches, including ad-hoc, Structured Communication, Distributed Object Communications Middleware and Message-Oriented Middleware. Next, it described Service Oriented Computing as a generalized approach for combing self-contained, loosely coupled units of functionality (called services) and SOAP Web services as a popular mechanism for providing SOC across Internet-based infrastructure. We noted that SOAP Web services often rely on process-centric descriptions of a component's end-point addresses, supported methods and associated data types using technologies such as Corba IDL or WSDL. The background section concluded by describing the influence of mobile computing and wireless networks. Notably, it discussed how parameter and compositional adaptation are increasingly used to dynamically adapt software systems to better fit the characteristics of a mobile user's current situation.

The remainder of the chapter described context-aware computing as an approach for adapting mobile and embedded computing approaches to the complex dynamics of the physical world. The first section discussed the complexity of distributed mobile systems and how supportive middleware is well-recognized as an essential requirement for constructing non-trivial systems. Next, we introduced a conceptual framework intended to guide the discussion of context-aware computing techniques, middleware and projects. Using the conceptual framework as a guide, the next section began by defining key terms related to context and context-aware computing; noting the various important properties of context information. The next section described techniques for acquiring context information and noted that conventional acquisition approaches are often reliant on the widespread deployment of sensing instrumentation and infrastructure. The next section described context modeling and representation techniques. Importantly, this section discussed how effective context modeling often requires the participation of domain-experts to help capture the syntax and semantics of a given context domain. The next section introduced common mechanisms for context management and provisioning. Notably, this section described how existing context management approaches (e.g. context servers) often encounter scaling challenges that prevent large-scale deployments. Moreover, it noted how recent advances have begun exploring cross-domain context acquisition and modeling in real-world scenarios. The final section presented techniques for context-aware component interoperation, whereby context-aware systems utilize resultant context information to discover, select and interoperate with contextually relevant components at runtime. This section presented several service discovery protocols and related distributed communications infrastructures. Importantly, it noted how existing architectures require significant prior interface knowledge in order to support spontaneous component interoperation. The chapter concluded by mentioning several promising component integration techniques that closely resemble conventional Web architecture.

Based on the related work presented in this chapter, we note that current context-aware systems are typically devised with the assumption that the underlying network infrastructure, hardware devices, application components and context mechanisms are well-known a-priori and contained within a

limited and controlled administrative domain. Hence, many approaches mandate expensive and invasive deployment of context instrumentation; require domain-specific network configurations; rely on specially outfitted mobile devices; adopt enterprise-specific distributed middleware; and generally lack support for spontaneous cross-domain component interoperation. Further, the considerable expense and effort required to devise, implement and deploy such systems often promotes a top down development approach intended to address niche problem domains where the requisite support infrastructure can be readily provided, administrative access is available and return on investment is assured. Indeed, several recent surveys [22, 62, 106] indicate that existing systems generally fail to provide ubiquitous accessibility; resulting in a pronounced lack of developer adoption and end-user participation. The next chapter addresses these issues by introducing the foundations of the Ocean approach.

Chapter 3
Foundations of the Ocean Approach

3.1 Introduction

Throughout the last chapter, it was discussed how interrelated contributions from the domains of network engineering, distributed systems and mobile computing are rapidly converging to produce everyday environments comprised of powerful mobile and embedded devices, ubiquitous network connectivity and rich sources of networked computation. These advances are recognized as important foundations for the development of mobile distributed systems capable of supporting human-centric tasks and modes of interaction [353]. However, while many everyday environments provide ample opportunities for adaptive computing, mobility often introduces significant challenges related to heterogeneity, scalability, security, privacy, spontaneous interoperation, and so-forth [73]. Accordingly, context-aware computing has emerged to address these challenges through an evolutionary synthesis of distributed systems and mobile computing. As discussed in section 2.3, context-aware systems model environmental and situational information – e.g. location, identity or the proximity of nearby devices – as a means of orchestrating parameter and compositional adaptation. In this way, a context-aware system aims to dynamically optimize its runtime behavior and capabilities to fit the characteristics of a user's current situation and computing environment. While applicable to many application domains, context-awareness is recognized as particularly compelling for mobile computing, where users often encounter rapidly changing execution environments and sources of information and computation potentially not known a-priori [293]. However, despite decades of research effort, context-aware systems remain consigned to small-scale deployments and research prototypes [79, 346].

Based on the background and related work presented in the last chapter, this chapter begins a discussion of our novel context-aware computing approach, called Ocean, which aims to capture the entrepreneurial spirit of modern Web architecture as a means of supporting large-scale, real-world context-aware systems. The structure of this chapter is based on an abbreviated version of the IEEE Recommended Practice for Software Requirements Specifications (SRS) [156], which provides a approach for deriving software specification requirements based on the recommended practices of the IEEE. The structure of this chapter is as follows: Section 3.2 begins by introducing the key challenges facing large-scale context-aware systems and discusses related advances in cross-domain context modeling, network engineering, scalable middleware and component interoperability. Next, section 3.3 introduces the overall scope of the Ocean approach, which aims to address the challenges discussed in section 3.2. Section 3.4 introduces Ocean's non-functional requirements by presenting Ocean's major design principles in section 3.4.1 and related design constraints in section 3.4.2. Based on these non-functional requirements, section 3.5 derives the overall Ocean approach by describing its principle architectural abstractions, application model, component contextualization and discovery techniques, registry architecture and integrated support for community-based processes (e.g. open contribution, collaborative annotation, volunteer-based computing and recommender systems). Related, section 3.6 describes Ocean's principle stakeholders. The chapter concludes with a discussion of the Ocean Reference Implementation as a validation methodology in section 3.7.

3.2 Large-scale Context-aware Systems: Challenges and Foundations

Despite decades of research effort, current context-aware systems remain consigned to small-scale deployments and research prototypes; existing primarily within isolated islands of niche functionality that are far removed from everyday use [79, 288]. Indeed, several recent surveys [21, 62, 97] indicate that existing systems generally fail to provide ubiquitous accessibility and often incur intractable scaling costs that inhibit widespread reuse; resulting in a pronounced lack of developer adoption and end-user participation. As previously discussed, current context-aware systems are often designed with the assumption that the underlying network infrastructure, hardware devices, application components and context mechanisms are contained within a limited and controlled administrative domain and are well-known a-priori [97]. As such, many mandate expensive and invasive deployment of context instrumentation; require domain-specific network configuration; rely on specially outfitted mobile devices; adopt enterprise-specific distributed middleware; and lack support for spontaneous *cross-domain* component interoperation. Further, the considerable expense and effort required to devise, implement and deploy such systems often promotes a top-down development style intended to address niche problem domains where the requisite support infrastructure can be readily provided [62, 97]. However, while existing techniques have been moderately successful in supporting enterprise scale systems, they are often inappropriate for large-scale, real-world environments due to a number of interrelated challenges. In this section, we discuss these challenges and present related advances that provide the foundations for the Ocean approach.

3.2.1 Challenge: Ubiquitous Context Infrastructure

As previously discussed, context aware systems are founded upon mechanisms for acquiring, modeling and provisioning context information. Most systems utilize modeled context information to orchestrate parameter and compositional adaptation (see section 2.2.3). In sophisticated approaches, compositional adaptation techniques are used to discover, select and interoperate with relevant networked components at runtime. Notably, such systems often aim to provide adaptive features to nomadic users operating within heterogeneous, real-world environments. In this regard, most non-trivial context-aware systems are organized hierarchically; with context acquisition, context modeling and representation and context management and provisioning forming a generalized layered model (see section 2.3). Hence, at their foundation, large-scale context-aware systems presume the availability of ubiquitous context infrastructure; however, as discussed shortly, existing approaches often suffer from significant scalability issues in larger scenarios.

Context-aware systems presume a level of physical integration and network accessibility beyond that of conventional distributed and mobile systems [180]. Physical integration is most apparent at the boundary between the physical and virtual domains, where sensors are used to capture contextual attributes from the environment. In many context-aware systems, expensive and invasive context instrumentation is deployed throughout the system's operational area in order to support the context acquisition and modeling process. For example, the University of Karlsruhe's MediaCup project [28] explores techniques for augmenting physical objects with digital presence while preserving an object's original appearance and purpose. In this regard, the MediaCup project team outfitted conventional coffee cups with dedicated sensing equipment, wireless technologies and low-power

microcontrollers that broadcast the cups' physical state to interested applications (see Figure 16a). Similarly, the Smart Floor [242] from the Georgia Institute of Technology transparently identifies and localizes users based on footstep force profiles detected by specially engineered floor tile with integrated load sensors (see Figure 16b). Related, MIT's Cricket system [258] provides fine-grained position estimation by way of custom designed ultrasonic hardware that must be deployed throughout the intended operational environment. Although such infrastructure-centric projects are inarguably capable of providing high resolution contextual data, the augmentation of existing physical environments with custom-designed instrumentation and related infrastructure is often prohibitively costly outside of small-scale deployments [79]. Indeed, even with significant industry backing, the development and standardization of the UPnP service discovery protocols required almost a decade of effort [162].

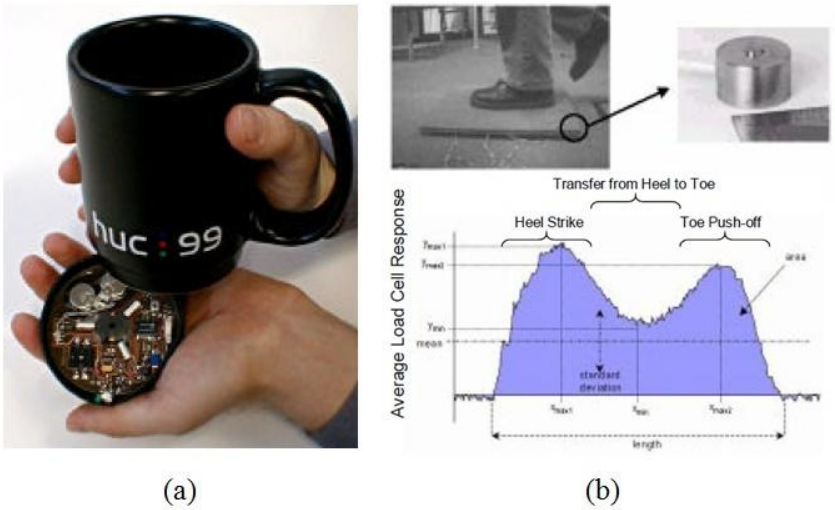


Figure 16: Dedicated context instrumentation: a) MediaCup and b) the Smart Floor

Aside from the costs associated with context instrumentation, other factors have inhibited the widespread deployment of context infrastructure. For example, many context-aware projects require dedicated context servers that impose significant scalability issues. Notably, systems such as Solar [63], SCI [121] and Hydrogen [152] require the widespread deployment and maintenance of *multiple* context servers; resulting in increased hardware costs, increased development time and necessitating administrative access across the operational area. Related, many context-aware systems rely on the related deployment of "heavyweight" middleware components such as Corba ORBs or Java RMI stubs, which imposes limitations on the devices, programming languages and communication protocols available for a given project. For example, the JINI architecture mandates use of the Java programming language, requires a device-compatible Java virtual machine, requires pre-deployment of runtime software components and constrains distributed communications to Java RMI [342]. Further, the adoption of complex distributed computing middleware often limits application development to experts [145]. As such, current context-aware systems are generally based on niche application models that lack adequate toolkits and programming models [147]. Moreover, niche context-aware infrastructures are often designed to accommodate preconceived application domains and may only support a limited set of context information types [73]. Notably, such systems rarely

support the dynamic integration of new context mechanisms at runtime or allow service provisioning by external developers; further limiting their use in heterogeneous, large scale scenarios [97].

3.2.2 Foundation: Aladin-based Context Acquisition and Modeling

Although the deployment of dedicated context infrastructure has proven to be infeasible for large-scale scenarios, many everyday environments are becoming increasingly saturated with *preexisting* sources of potential contextual information (e.g. GPS signals, GSM cell tower identifiers, MAC addresses, RFID tags, barcodes, accelerometer data, light intensity, etc.) As noted by Edwards and Grinter [95], suitable environments for context-awareness arise more or less "accidentally" from an "accretion of technological components embedded in an environment that has not benefited from a holistic, ground-up approach to design and integration." Based on such observation, we developed a context-aware computing approach, called Aladin [49], which explores how commodity devices can be dynamically adapted to acquire and model a diverse range of context information using local available hardware and software. Aladin is specifically designed to automatically acquire and model context information in large-scale, cross domain scenarios without requiring significant context instrumentation or related infrastructure. Aladin provides a domain-neutral, client-centric approach that dynamically extends the context modeling capabilities of commodity mobile devices through the use of runtime deployed plug-ins. Client-side Aladin software enables devices to adapt their context acquisition and modeling capabilities by performing ongoing capability analyses and integrating platform-specific context acquisition and modeling plug-ins on-the-fly. Related, we also developed an Internet-based plug-in repository whereby external parties can develop and integrate context plug-ins for use by the Aladin community.

The Aladin approach is based on the extensible, client-centric software architecture shown in Figure 17. In the Aladin approach, domain-specific software running on a commodity device hosts the Aladin Framework. Based on the Façade pattern [114], the Aladin Framework provides a context management API and related set of context events. During runtime, the Aladin Framework automatically analyses the capabilities of its host device and environment by way of dynamically installed capability analysis plug-ins. Based on the detected capabilities the host device and environment, Aladin then dynamically downloads and installs context acquisition and modeling plug-ins that are capable of rendering high-fidelity native context data (NCD). During runtime, low-level context preprocessing and quantization are provided by the installed context plug-ins; resulting in the generation of context events (containing NCD) that are received by the hosting application. The hosting application may react to incoming context events as needed, according to their local application logic; however, Aladin provides an additional context interpretation abstraction that can be implemented as a means of customizing Aladin with support for a particular application model. Aladin has been validated through the construction of three diverse application models, including a mobile interactive cinema platform [50]; a museum tour-guide system [49]; and a pervasive multiplayer tangible game [148]. Related work indicates that client-centric approaches, such as Aladin, can be effectively adapted to large-scale heterogeneous environments [139, 268]. Accordingly, the Aladin Framework is used to provide a foundation for the Ocean approach (see section 3.4.2).

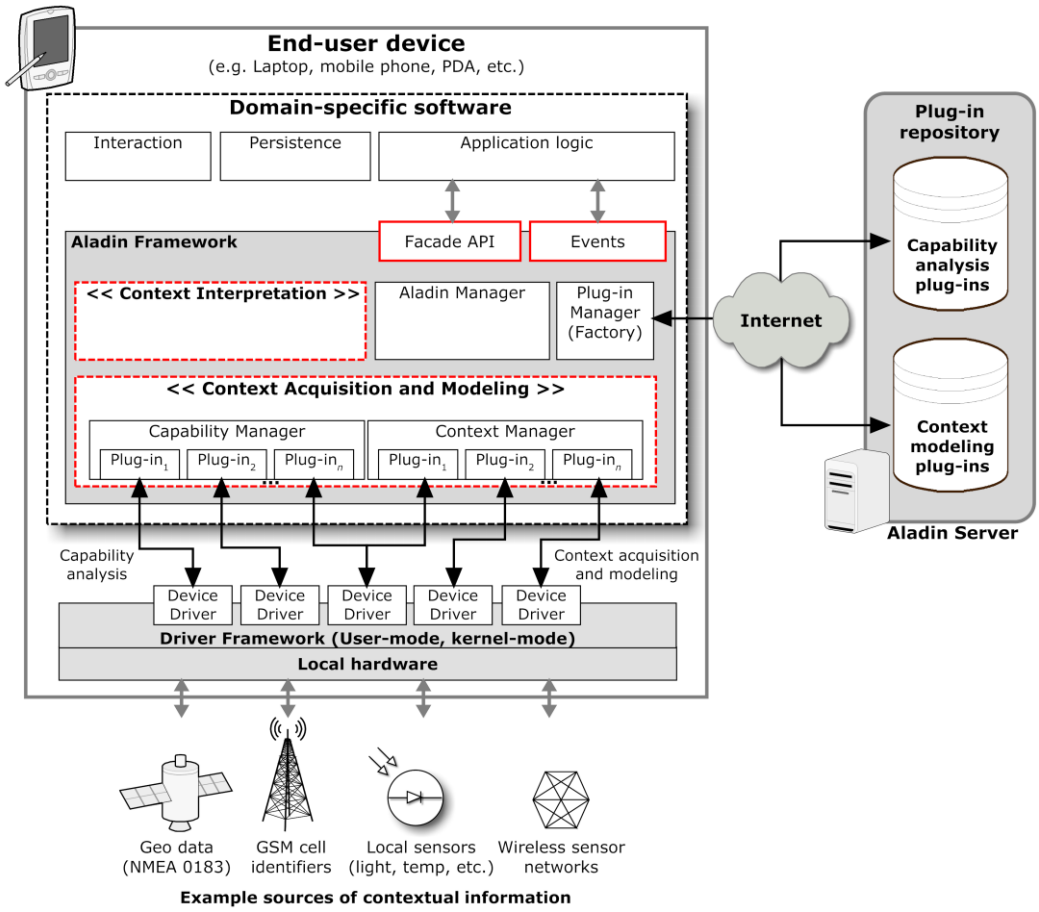


Figure 17: Overview of the Aladin Framework architecture

3.2.3 Challenge: Widespread Network Accessibility

Given effective cross-domain context acquisition and modeling, large-scale scenarios presuppose a foundation of network accessibility encompassing the intended operational area [73]. As discussed in sections 2.3.4 and 2.3.5, most existing context-aware infrastructures adopt conventional distributed object communication (DOC) infrastructure (e.g. Corba, Java RMI and SOAP Web services) to facilitate context provisioning and component interoperation. For example, the Gaia meta-operating system [272] adopts Corba as its principle communications middleware. As a result, the Gaia infrastructure inherits Corba's mechanisms for "object registration, location, and activation; request demultiplexing; framing and error-handling; parameter marshalling and demarshalling; and operation dispatching" [122]. Similarly, the design of many context-aware systems is significantly influenced by the underlying service discovery and distributed communications models.

As discussed in section 2.3.5, context-aware approaches that utilize ad-hoc, local-link service discovery protocols (e.g. SLP, UPnP and Zeroconf) are inherently limited to short-range operations with relatively homogenous peers. While ad-hoc integration techniques are inarguably important for context-aware computing, existing approaches provide impoverished mechanism for wide-area service discovery and lack protocol interoperability [194]. For example, in order for an UPnP AV MediaServer to discover and interoperate with an UPnP MediaRenderer, compatible devices must be capable of communicating via multicast. Although, the UPnP standards allow for service discovery

beyond the local link, most firewalls block multicast traffic beyond the local administrative scope; effectively constraining UPnP to a single administrative boundary. While some service discovery protocols provide external service registries and related network address translation (NAT) traversal, current registries have only achieved limited adoption [283] and many NAT traversal techniques remain unstandardized or poorly supported [323] (e.g. the Internet Gateway Device Protocol). Finally, although service discovery protocols share similar functionality, most are not interoperable [3]; hence, inter-protocol discovery and interoperation scenarios are generally precluded. While hybrid protocols have attempted to address wide-area service discovery and advertisement [194], such techniques have not yet gained traction.

In addition to service discovery limitations, the adoption of conventional DOC infrastructure imposes network accessibility challenges for many context-aware systems. For example, current DOC middleware often imply considerable implications regarding firewall traversal in cross-domain deployments [72]. Specifically, wide-area use of conventional DOC techniques may pose security risks (due to unencrypted protocols) and often requires the opening of specific network ports in an organization's firewall [233]. While some corporate network policies allow for the transport of DOC protocols, many organizations have proven reluctant to allow intercommunication between local services and distributed components outside their administrative control [144]. Further, additional cross-domain issues related to NAT also arise in situations where isolated IP address spaces conflict with a DOC middleware's object registration, location, and activation techniques (e.g. Corba's Interoperable Object Reference). In such cases, published private IP addresses may not be reachable by external entities, even with appropriate firewall security policies. In this regard, approaches such as Java RMI and Corba have attempted to accommodate firewall traversal through the use of HTTP tunneling techniques and dedicated firewall hardware; however, such techniques often result in performance limitations, increases security risks and are currently poorly supported due to technical shortcomings and lack of interest from firewall vendors [144].

3.2.4 Foundation: Public Internet Infrastructure

The development of large-scale context-aware systems will require support for ubiquitous network accessibility. In this regard, we suggest that public Internet infrastructure represents a compelling foundation for context-aware systems capable of cross-domain context modeling. In terms of adoption, the Internet remains unparalleled as an open data communications infrastructure. Recently, innovative census techniques [141] have been used to survey the Internet's ubiquity and global scope. As of the time of writing, the most recent census of Internet edge hosts describes an rapid and increasing allocation of IP addresses; ranging from 315 in 1982 [306] to over 2.7 billion in 2006. Other recent surveys [227] estimate global Internet usage at approximately 20.3% of the world's population (1.36 billion people) according to the following geographic distributions: North America (72.2%); Oceania/Australia (56.4%); Europe (46.8%); Latin America/Caribbean (22.1%); Middle East (17.1%); Asia (13.6 %); and Africa (4.1%). Related, recent Web server response surveys [222] and search engine indexes [189] estimate the total number of Web servers at 187 million and the total number of indexed Web pages at over 25.81 billion. While an accurate assessment of the Internet's growth dynamics remain challenging, modern Internet infrastructure inarguably represents an increasingly ubiquitous global phenomenon that spans continents and cultures.

Although the Internet provides ubiquitous network accessibility, its design has proven difficult to exploit by traditional context-aware systems [97]. As previously described, Internet architecture is often hostile to the service discovery and distributed communication techniques common to many context-aware systems. Moreover, by emphasizing connectivity and the end-to-end principle (see section 2.2), Internet infrastructure reveals very little of the rich contextual semantics often required by context-mediated adaptation strategies. As previously stated, aside from basic addressing and routing information, potential context information – such as underlying communications hardware, network topologies and physical location of components – is intentionally hidden from end-systems as a means of providing "the illusion of a single, seamlessly connected network where the fragmented nature of the underlying infrastructure and the many layers of protocols remain largely transparent to the user" [358]. As such, the Internet's architectural model often confounds traditional context-aware approaches that rely on domain-specific network configuration and well-known context sources. Indeed, the conspicuous lack of Internet-scale context-aware systems highlights the fundamental conflicts that arise between the scope of context-aware systems and the requirements of large-scale network architectures.

Although Internet-scale context-aware systems have yet to be devised, several interrelated advances are providing the foundations for larger deployments. First, as described in section 3.2.1, adaptive context-aware frameworks such as Aladin [49] and Contory [268] demonstrate techniques for cross-domain context acquisition and modeling that are well-adapted to conventional Internet infrastructure. Next, the widespread adoption of the Internet is providing an increasingly ubiquitous foundation of network communications across a broad range of environments. Importantly, the widespread adoption of the TCP/IP model has led to "constant innovation and entrepreneurial spirit at the physical substrate of the network as well as at the application layer" [358]. The resultant explosion of Internet-based services such as email and the Web have resulted in the development of broadly supported application-layer communication protocols such as HTTP [104]. Related, the pairing of open data-exchange mechanisms (e.g. XML) with Internet-friendly communication protocols has resulted in the emergence of SOAP Web services as a mechanism for constructing distributed computing systems using Internet technologies. While SOAP Web services face significant challenges in wide-area context-aware scenarios (see section 3.2.7), their current market dominance has promoted the widespread adoption of software support across a broad range of devices (e.g. XML parsing).

3.2.5 Challenge: Ubiquitous Middleware

Although Internet infrastructure provides widespread accessibility and architectural flexibility, cross-domain context-aware systems presuppose distributed middleware that is similarly accessible, highly scalable [38] and supportive of a wide variety of problem domains [323]. While definitions vary, *middleware* has been described as a "software layer between the operating system – including the basic communication protocols – and the distributed applications that interact via the network. This software infrastructure facilitates the interaction among distributed software modules" [117]. In contrast to enterprise-scale distributed systems, which are generally highly complex and contained within a limited administrative boundary, large-scale distributed architectures must address several additional requirements such as a low entry-barrier, extensibility, independent deployment of components and rapid evolution [105]. Importantly, large-scale distributed applications must be

capable of operating across multiple trust domains, and "continue operating when subjected to an unanticipated load, or when given malformed or maliciously constructed data, since they may be communicating with elements outside their organizational control" [106]. Table 6 presents an overview of the key requirement for context-aware middleware as identified by Henricksen et al. [147].

Middleware requirement	**Description**
Support for heterogeneity	Hardware components ranging from resource-poor sensors, actuators and mobile client devices to high-performance servers must be supported, as must a variety of networking interfaces and programming languages. Legacy components may be present.
Support for mobility	All components (especially sensors and applications) can be mobile, and the communication protocols that underpin the system must therefore support appropriately flexible forms of routing. Context information may need to migrate with context-aware components. Flexible component discovery mechanisms are required.
Scalability	Context processing components and communication protocols must perform adequately in systems ranging from few to many sensors, actuators and application components. Similarly, they must scale to many administrative domains.
Support for privacy	Flows of context information between the distributed components of a context-aware system must be controlled according to users' privacy needs and expectations.
Traceability and control	The state of the system components and information flows between components should be open to inspection - and, where relevant, manipulation - in order to provide adequate understanding and control of the system to users, and to facilitate debugging.
Tolerance for failures	Sensors and other components are likely to fail in the ordinary operation of a context-aware system. Disconnections may also component occur. The system must continue operation, without requiring excessive resources to detect and handle failures.
Ease of deployment and configuration	The distributed hardware and software components of a context-aware system must be easily deployed and configured to meet user and environmental requirements, potentially by non-experts (for example, in "smart home" environments).

Table 6: Requirements for context-aware middleware (from [147])

To accommodate the requirements presented above, current context-aware systems typically utilize "heavyweight" middleware infrastructure (see section 3.2.1) that impose significant restrictions on the

types of devices and computing environments that can be supported. Notably, Java RMI, Microsoft .NET and Corba all require specific local runtimes that may not be widely available across all device types. For example, Java RMI requires a Java virtual machine, mandates use of the Java programming language and is available only as an optional package for mobile devices running the Java 2 Mobile Edition (J2ME) [321]. Similarly, Corba requires the deployment of a compatible Object Request Broker across participating entities [233] and .NET remoting presumes widespread availability of a compatible .NET runtime library. Furthermore, even distributed applications based on SOAP Web services require a local SOAP stack to provide marshaling, serialization, transport and demarshaling of XML-encoded SOAP data. As noted by Garlan et al. [115], such techniques often result in tight-coupling between middleware components and dependant distributed applications; potentially limiting reuse or inhibiting a system's ability to accommodate new application scenarios without retooling.

3.2.6 Foundation: Conventional Web Architecture

Although rarely directly exploited by context-aware systems, conventional Web architecture is increasingly recognized as an Internet-scale middleware platform that fulfils many of the context-aware computing requirements presented in the last section [106, 139]. Modern Web architecture emerged from experiments at CERN by Tim Berners Lee between 1982 and 1988 [31]. During this period, Berner's Lee envisioned a novel application scenario whereby conventional Internet infrastructure could be used to support "a shared information space through which people and machines could communicate" [33]. Communication within this information space was seen as a way for participating users to independently structure and publish a variety of information such as research notes and contact details. The intended users of the original system were physics researchers who were broadly categorized as geographically dispersed, connected via the Internet and utilizing a heterogeneous collection of computing devices. Consequently, the "challenge was to build a system that would provide a universally consistent interface to this structured information, available on as many platforms as possible, and incrementally deployable as new people and organizations joined the project" [106]. The resultant architecture, called the World Wide Web (Web), was designed to extend existing hypertext techniques [31] with a data-centric distributed computing approach that emphasized non-centralization, remote access across multiple networks and heterogeneity of devices [105].

An idealized model of the interactions within a Web application has been formalized as the Representational State Transfer (REST) architectural style by Fielding in 2000 [105]. Although not specifically tied to Web architecture, REST was derived from the standardization of its first application-layer protocol, known as the Hypertext Transport Protocol (HTTP) [104]. REST was designed to provide "caching and reuse of interactions, dynamic substitutability of components, and processing of actions by intermediaries, in order to meet the needs of an Internet-scale distributed hypermedia system" [106]. The foundational architectural styles underlying REST include replicated repository, cache, client-server, layered system, stateless, virtual machine, code on demand, and uniform interface (for details see [105]). The constraints of these underlying styles differ from traditional distributed computing approaches that hide underlying components, network entities and data. In contrast, REST intentionally exposes the nature and state of the network and data elements that comprise a system [105]. On overview of the REST architectural style is shown in Figure 18.

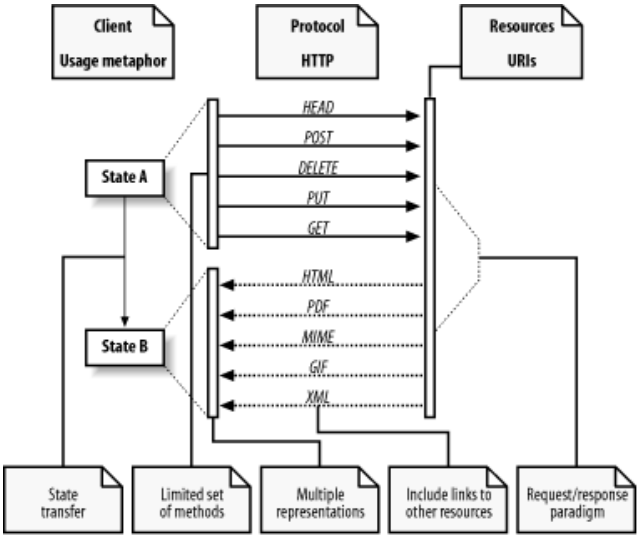


Figure 18: Overview of the REST architectural style (from [262])

REST is based on a key information abstraction known as a *Resource*, which represents "any concept that might be the target of an author's hypertext reference" [106]. As described in [105], any information that is important enough to be named can be modeled as a Resource (e.g. an image, newsfeed, software release, Web page, etc.) Resources comprise identity, state and behavior. Further, Resource naming is accomplished through the use of a globally adopted addressing scheme known as the Unified Resource Identifier (URI) [34], which provides a standardized mechanism for acting upon or obtaining information about Resources. As described in [341], "A resource should have an associated URI if another party might reasonably want to create a hypertext link to it, make or refute assertions about it, retrieve or cache a representation of it, include all or part of it by reference into another representation, annotate it, or perform other operations on it. Software developers should expect that sharing URIs across applications will be useful, even if that utility is not initially evident." In this regard, the use of URIs allows applications to independently expose "interesting" aspects of their internal data or functionality. Likewise, URIs allow clients to decide which parts of an application's data are important; allowing for component integration in ways perhaps not originally envisioned by the designer of the Resource [266].

On the Web, distributed entities called *Web Agents* observe and change Resource state by sending and receiving *Representations*. A Representation can be understood as a sequence of bits (generally in a standardized data format) that represents the current (or desired) state of a Resource. Web agents include software systems acting directly on behalf of a user (i.e. user agents) or intermediary entities such as proxies, browsers, spiders and multimedia players (i.e. software agents). As Representations are exchanged between Web agents, Resource state is managed by its hosting *origin server*, whereas the application state is managed by the clients. State transitions between application states are facilitated through the use of hypermedia, which enable clients change transition from one state to the next by examining and dereferencing hyperlinks embedded within received Representations [105]. In the REST model, interactions between Web agents are *stateless;* meaning that each request for a Representation occurs in isolation. Stateless interactions help improve a systems scalability because

servers do not need to cache state information between requests; allowing for improved resource allocation and simplified load balancing that does not require server affinity or state passing [266].

Perhaps most distinguishing feature of the REST model is its establishment of a *uniform interface* between distributed components, which helps improve interoperability across *multiple* organizational boundaries [336]. Rather than allowing components to expose unrestricted method vocabularies (i.e. process-centric distributed computing), all RESTful components implement the *same* minimal set of generic interface methods. By requiring that Resources to provide suitable implementations of the same generalized interface, requesting clients need only understand the semantics of a single interaction vocabulary. Further, the uniform interface is complimented by the use of self-describing message payloads, whereby the data passed between entities are well-known. On the Web, message payloads typically adhere to globally standardized formats (e.g. MIME content types [327]) and the uniform interface is provided by the set of HTTP methods shown in Table 7. As noted by Ray et al. [262], "It is precisely because HTTP has few methods that HTTP clients and servers can grow and be extended independently without confusing each other."

Method	Description
GET	Retrieve a representation of a Resource addressed by a URI.
POST	Creates a new Resource when directed to an existing URI.
PUT	Modifies an existing Resource when directed to an existing URI. Creates a new Resource if directed to a new URI.
DELETE	Deletes an existing Resource when directed to an existing URI.
HEAD	Retrieve a metadata-only representation of a Resource.
OPTIONS	Check which HTTP methods a particular Resource supports.

Table 7: Overview of the HTTP uniform interface (adapted from [262])

Based on characteristics of the REST architectural style, conventional Web architecture provides a promising middleware foundation for context-aware systems. The Web's design addresses scalability beyond geographic dispersion by incorporating techniques that accommodate multiple trust domains, unanticipated load and allow for independent component deployment [341]. Next, as evidenced by the tremendous number of Web-based applications, the hypermedia application model has proven remarkably capable of accommodating a variety of application domains. Moreover, the distributed architecture designed to support the Web's hypermedia model is sufficiently flexible to accommodate a variety of non-hypermedia application scenarios [266]. Additionally, the Web' low entry-barrier and non-proprietary standards have made its communication protocols, functional apparatus and device support ubiquitous. Related, the Web's increasing ubiquity has resulted in significant developer adoption that has resulted in the emergence of a broad array of development toolkits, application frameworks and related knowhow. Consequently, increasing developer adoption has resulted in an explosion of Web-based information and computation.

3.2.7 Challenge: Cross-domain Component Interoperation

Given a sufficiently ubiquitous middleware approach, cross-domain context-aware systems must be able to discover and interoperate with relevant distributed components at runtime without extensive prior knowledge [96, 180]. As described in section 2.2.1, traditional distributed component architectures rely on *process-centric* descriptions of a software component's end-point addresses, methods and associated data-types provided by technologies such as Corba IDL or WSDL. However, although modern development environments simplify the creation of remotely accessible methods, the resultant proliferation of domain-specific interfaces can reduce the probability of component interoperation [225, 334, 336]. Notably, several recent surveys [21, 62, 97] indicate that the majority of current context-aware systems adopt process-centric interoperation (PCI) styles. PCI techniques such as those epitomized by Corba and SOAP Web Services make the coupling between callers and components clear and unambiguous; however, in large-scale scenarios, interactions between distributed components may suffer from architectural mismatch [115], where domain specific method syntax, sequencing and semantics prevent widespread reuse due to a lack of widespread understanding of a given interface [334]. While dynamic interactions between specialized interfaces can be resolved in small-scale distributed systems (e.g. enterprise scenarios) they become problematic in larger scenarios where component interfaces cannot be known a-priori [334]. Moreover, PCI techniques rely on complex infrastructure, highly skilled developers, platform specific mobile code and significant tooling, which are all recognized as antithetical to widespread developer adoption [145].

3.2.8 Foundation: RESTful Component Interoperation

The Web's underlying REST architectural style, as described in section 3.2.6, provides several advantages for component interoperation in scenarios where a-priori interface knowledge is impossible or difficult to achieve [336]. Unlike conventional PCI approaches, which generally rely on domain-specific interface definitions, proprietary component addressing and opaque communication endpoints, REST employs a lightweight middleware model that has been rapidly gaining in popularity across a wide variety of cross-domain composition scenarios, including component architectures (see section 5.2.1), enterprise Web services [170, 262] and context-aware systems [96, 192]. As previously discussed, REST exploits a standardized endpoint addressing scheme that allows for the flexible organization of exposed component functionality through the use of URIs. Further, RESTful Resources are interconnected through hyperlinks that are embedded within their associated Representations; providing clients a set of available application states that can be provisioned at runtime. Finally, RESTful Web Resources adhere to the HTTP uniform interface, whose syntax and semantics are well-known by all participating entities [106]. In contrast to the specialized interfaces common to PCI techniques, the standardization of a limited set of generalized interface methods "enables entities to use new devices and services that appear in their environment without explicit rewriting, updates, or installation of drivers. In addition, it reduces the number of agreements that must be made among communicating entities, and allows for dynamic, runtime interoperation of devices and services on a network" [247]. Vinoski [336] describes the following beneficial data-coupling characteristic common to RESTful architectures:

- System resources adhere to the same semantics for each operation in the Uniform Interface, thus simplifying client applications by eliminating the need for custom code to support specialized interface semantics.
- Developing Resources means designing an implementation to fulfill the Uniform Interface and its expected semantics, essentially eliminating the development phase required for designing separate interfaces for each Resource, with their specialized semantics and implied workflow.
- Error handling is typically a source of significant variance between interfaces as interface designers individually cook up their own data structures and exceptions for reporting problems. Under the Uniform Interface constraint, however, error handling also gains uniformity.
- Intermediation becomes highly practical because intermediaries can understand the Uniform Interface semantics just as well as Resources and clients can. For example, a Uniform Interface can specify which calls are idempotent (that is, can be called repeatedly without side effects) and which aren't. Resources can include cache control information in responses to idempotent operations, so that developers can easily insert caches between a client and the resources it uses without breaking the client or needing to specialize the caches for the invoked resources.
- Without the presence of numerous specialized interfaces, overall system simplicity increases, which typically decreases the number of defects. Notably, interface versioning issues are significantly reduced, though not entirely eliminated. Moreover, the overall system becomes much more extensible.

In the traditional hypermedia model, user agents (e.g. Web browsers) discover, select and compose components (i.e. Resources such as Web pages or newsfeeds) at runtime using three basic steps: identification, interaction and message payload interpretation [341]. Briefly, in the *identification* phase, a discovered URI is used to address an abstract Resource in a standardized way. Next, in the *interaction* phase, the user agent interacts with a Resource using HTTP's uniform interface, which supports the exchange of messages according to a well-defined set of semantics (see [104]). During a typical interaction, a Web browser may provide supplemental information (e.g. the HTTP Accept request-header field) to help the server provide a suitable Resource Representation. Importantly, the well-known semantics of HTTP's uniform interface allows developers to predictably weigh the impact a give method call may have on a Resource's state (for example, the side effects of a single **`GET`**, **`HEAD`**, **`PUT`** or **`DELETE`** request are the same as $N > 0$ identical requests [104]). As an interaction completes, a server may return a Representation to the client that represents the current state of the Resource involved in the interaction (or an appropriate status code). Finally, during *message payload interpretation*, Representations are handled by the user agent according to the Representation's data type and the original method semantics. For example, after performing a **`GET`** request, Web browsers may render the resultant HTML document for the user. As per the hypermedia model, embedded hyperlinks within Representations allow Resources to suggest potential next application states to the user, who may continue the interaction cycle by dereferencing additional hyperlinks.

The basic hypermedia model described above provides insight as to how complex, machine-based interactions can be accomplished using REST. In this regard, an illustrative example proposed by Tilkov [330] is now summarized as a means of comparing typical REST versus SOAP Web service

implementations. Tilkov's example considers a simple procurement scenario where a Web-based application is designed to handle common tasks related to customer registration and order management. In typical service-oriented approaches (i.e. those common to SOAP Web services) distributed components are designed to represent high-level coordination entities (e.g. an `OrderManagementService`) and related domain-specific interfaces are defined to provide the requisite functionality. An example of the resultant PCI-based interface methods are shown in Figure 19.

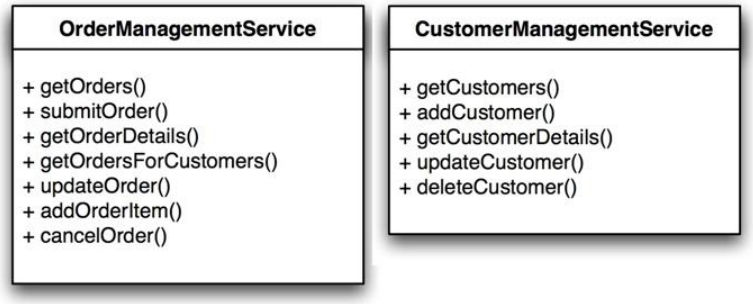


Figure 19: Implementing the procurement scenario using PCI principles (from [330])

With reference to Figure 19, the interface specifications are designed according to the *processes* a particular component supports (e.g. `getOrders()`, `addCustomer()` and `cancelOrder()`). In this case, the problem domain has been decomposed into two basic components, each with a specialized interface (i.e. `OrderManagementService` and `CustomerManagementService`). To utilize such components in a distributed application scenario, a client application would need to be specifically engineered to understand the syntax and semantics of each component's interface. In SOAP Web services, WSDL is used to provide these types of interface contracts (see section 2.2.2). However, although WSDL describes *how* a given method can be used, it does not inherently capture the *meaning* of each method [67]. As a result, process-centric interfaces such those shown in Figure 19 require significant prior domain knowledge to achieve component integration and reuse. As noted by Vinoski [336], "To invoke a service, a caller must incorporate details of each specific operation defined in the service's contract. In other words, the specialized interface forces each calling application to include custom code specific to the operations it wants to call. Calling applications must also be cognizant of each contract's 'implied workflow,' which is the order in which the service's operations were designed to be invoked."

In contrast to PCI-based interoperation techniques, recasting the aforementioned procurement example in terms of the Web's RESTful implementation yields dramatically different results. In terms of object oriented design, REST's Resource abstraction can be understood as an interface that all exposed component classes must implement. On the Web, this interface is provided by the HTTP specification, which defines the syntax and semantics of methods such as `GET`, `PUT`, `POST`, `DELETE`, etc. To model the previous example using REST, the domain-specific requirements of the procurement scenario are mapped to the HTTP uniform interface by defining an appropriate set of Resources (each providing a mapping between the example's functional requirements and a set of appropriate HTTP methods). For example, rather than defining an `OrderManagementService` component that exposes a specialized `getOrders()` method and `cancelOrder()` method, a single `/orders` Resource is created to provide domain-specific functionality through generalized HTTP methods. For example,

functionality provided by `getOrders()` can be encapsulated within the `/orders` Resource's `HTTP GET` method, whereas the functionality provided by `cancelOrder()` can be encapsulated within the `/orders` Resource's `HTTP DELETE` method. In this way, the functionality of the procurement scenario can be fully expressed using the HTTP's uniform interface as shown in Figure 20.

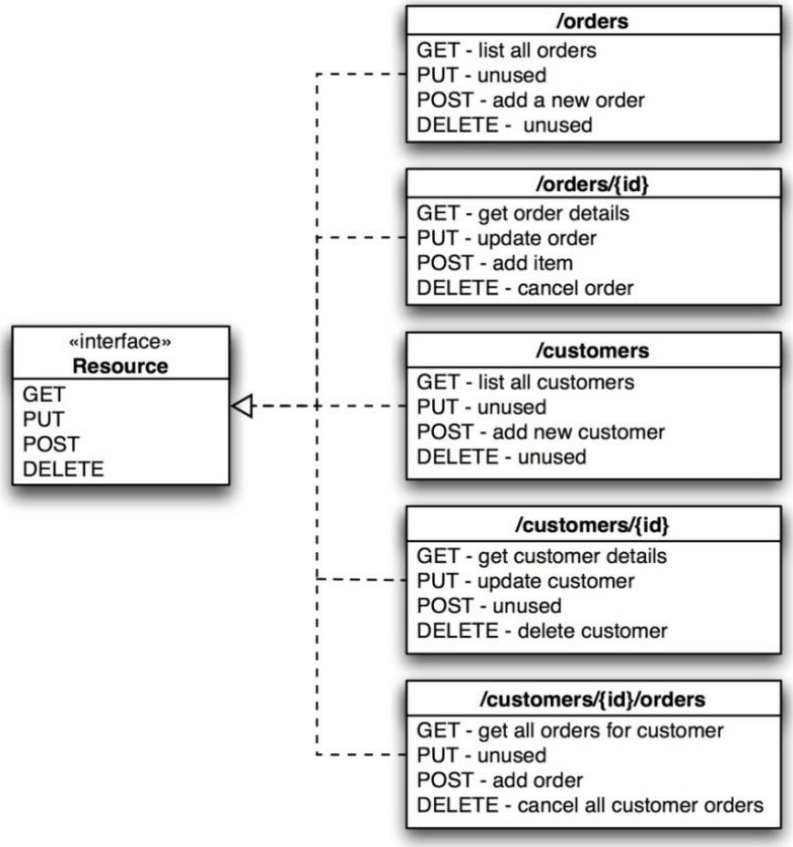


Figure 20: Re-implementing the procurement scenario using RESTful principles (from [330])

Once a problem domain is mapped to the HTTP uniform interface, the Web's RESTful architecture helps support wide-area integration and reuse. First, the use of URIs and standardized interface methods allows additional network entities, such as proxies and caches, to provide intermediary services that can help facilitate accessibility and scalability. Further, exposed URIs allow Web clients to selectively compose Resources in application-specific ways (perhaps not originally envisioned by the service developers). Importantly, Resource Representations are often connected to other related Resources through the use of embedded hyperlinks. This inherent *connectedness* [266] allows hypermedia to become the "engine of application state" [105] whereby the Resource's themselves provide descriptions of relevant components that are available for runtime composition. As noted by Prescod [257], "It is the client that knows what mission it needs to complete for the end-user. It is the client's responsibility to navigate from resource to resource, collecting the information it needs or triggering the state changes that it needs to trigger."

While modern Web architecture arguably supports several context-aware computing requirements, two important limitations have prevented its widespread use in this regard. First, while the human-based Web is highly interconnected through the use of embedded hyperlinks, REST-based Web

services often lack such connectedness as their Representation formats often consist of serialized data structures that lack embedded hyperlinks [266]. While some Resource representation formats support connectedness (e.g. the Atom Syndication Format [328]) many require external mechanisms for component discovery, which can inhibit ad-hoc integration [266]. Second, the current REST model provides no inherent support for component discovery and selection based on complex, real-world context information [105]. Recall that context-aware applications require pre-filtering of relevant application constituents based on domain-specific requirements [21, 29, 192, 231]. As such, additional wide-area discovery and selection mechanisms are required in order to support the spontaneous, cross-domain component integration capabilities required by large-scale context-aware systems.

3.3 Approach Scope

Based on the challenges and advances presented in the last section, we aim to develop a context-aware computing approach that captures the entrepreneurial spirit of modern Web architecture as a means of supporting the emergence of large-scale, real-world context-aware systems. Importantly, we suggest that much of the information and computation available on the Web has semantic associations to real-world contexts. This fundamental observation follows in the tradition of an increasing number of researchers who suggest that ubiquitous Web-based computation has much to offer context-aware computing [37, 174, 179]. As richly described in [179], "much of the information on the Web describes the world we physically inhabit, [however], there are few systematic linkages to real-world entities. This is unfortunate, because most of our activities concern physical objects other than computers." To provide a few familiar examples, digital images are often *associated with* physical locations [302]; real-world products may be *related to* Web-based reviews [200]; an organization may *publish* online calendar data [81] or news feeds [99] describing upcoming events; a person may be *linked with* digital business card data or specific Web pages [80]; and streaming media may be *preferred* in certain locations [339]. However, while examples of contextually-relevant Resources abound on the Web, most remain hidden from context-aware systems due to an inherent lack of Resource contextualization and discovery mechanisms within conventional Web architecture (see section 4.2).

The scope of our hybrid context-aware computing approach, called Ocean, addresses the large-scale computing challenges outlined in section 3.2. Unlike existing approaches, Ocean addresses these challenges by emphasizing user participation and community-based computation. Accordingly, Ocean defines a comprehensive conceptual model for augmenting existing Web-based software components (Resources) with expressive contextual metadata as a means of facilitating in-situ discovery and integration. Related, Ocean provides a complimentary software architecture that provides simple, accessible and scalable mechanisms for distributed applications to discover and compose contextually-relevant Resources at runtime. Towards these ends, Ocean extends emerging community-centric computing techniques such as collaborative annotation, open contribution, volunteer-based computing and recommender systems. By leveraging community participation, Ocean aims to support the emergence of a new class of hybrid context-aware Web applications capable of in-situ, context-mediated component discovery and composition.

3.4 Non-Functional Requirements

To guide Ocean's development, we now present several non-functional requirements that are intended to align our approach with the requirements of conventional Web architecture. The following sections include a presentation of Ocean's design principles and related approach constraints.

3.4.1 Design Principles

Based on the design principles underlying conventional Web architecture [105], the following design principles are proposed for the Ocean approach:

- **Widespread accessibility:** The Ocean approach should be widely accessible across a broad range of context situations, network infrastructure and device types. Deployment of supplemental context instrumentation and related infrastructure should be minimized or avoided. Related, Ocean should be directly supported on commodity end-user devices without requiring prohibitive runtime software such as complex distributed computing middleware.
- **Low entry-barrier:** The Ocean approach should be conceptually simple and amenable to a wide variety of developer skill-levels. The approach should be designed to accommodate developers who may not be expert in context-aware computing. Finally, Ocean should co-opt well-known application models in order to leverage existing knowhow and infrastructure.
- **Application independence:** The Ocean approach should accommodate a broad range of application types without modification of its underlying architecture. Ocean should limit domain preconceptions by relying on the end-to-end principle of system design as described in [279].
- **Extensibility:** The Ocean approach should support an extensible application model and a broad range of context data. In particular, context acquisition and modeling should be facilitated by external experts who understand the intricacies of a given context domain. Moreover, the Ocean application model should accommodate emerging data types and application scenarios without requiring changes to its underlying architecture.
- **Scalability:** The Ocean approach should be highly scalable in terms of application model, context mechanisms and component interoperation. Further, the Ocean application model should accommodate cross-domain component scalability and independent deployment.

3.4.2 Approach Constraints

Based on the scope and design principles previously discussed, Ocean's key approach constraints are now presented. This somewhat unconventional presentation is motivated by the significant challenges facing large-scale context-aware systems, which impose important practical considerations. Recall that section 3.2 identified these key challenges as ubiquitous context infrastructure; ubiquitous network infrastructure; ubiquitous middleware; and effective cross-domain component interoperation. Below we defined our primary approach constraints, which derive directly from the foundations of large-scale context-aware systems identified throughout section 3.2.

- **Constraint 1: Aladin-based context acquisition and modeling:** In order to accommodate cross-domain context-aware scenarios, Ocean directly subsumes Aladin's extensible, client-centric context modeling style. Accordingly, Ocean adopts the Aladin architecture as the

foundation of its application model. In this regard, Ocean applications inherit Aladin's client-centricity and overall approach; however, we impose no further constraints regarding how Aladin capabilities might be realized for a given platform.

- **Constraint 2: Internet-based network infrastructure:** Despite its inherent lack of rich contextual semantics, we suggest that the Internet's ubiquity, flexibility and core design principles can serve as a foundation for specific classes of context-aware systems. Hence, our second constraint limits Ocean's data communications to those that can be directly accommodated by the *public* Internet, without requiring adoption of domain-specific security configurations, supplemental infrastructure or requiring changes to the existing TCP/IP communications model.
- **Constraint 3: Web-based middleware:** Based on constraint 2, Ocean's subsumes conventional Web architecture as the foundation of its middleware approach. Hence, distributed communications within Ocean applications are bound to the Web's architectural styles as described in [104, 341]. By adopting the Web's architectural model, Ocean inherits its scalability, communication protocols, widely deployed functional apparatus and developer knowhow; however, Ocean must provide additional mechanisms for context-mediated Resource discovery and selection.
- **Constraint 4: REST-based component interoperation:** Based on our adoption of conventional Web-based middleware, Ocean constrains component interoperation to the REST architectural style as described in [105]. As such, participating distributed components must be addressed using standard URIs and resolved using conventional DNS mechanisms. Second, participating components must provide domain-specific implementations of the HTTP uniform interface as described in [104]. Third, message payloads exchanged between distributed components must adhere to self-describing, standardized data types (e.g. `MIME` types). Fourth, interoperation between components must adhere to the REST style as described [105].

3.5 Approach Derivation

Based on the nonfunctional requirements presented in the last section, this section derives the overall Ocean approach by describing its conceptual underpinnings and highlighting its key architectural aspects that are discussed in detail throughout the remainder of this dissertation. We note that by imposing the constraints presented in section 3.4.2, Ocean is strongly influenced by conventional Web architecture and, thus, inherits many of its capabilities and limitations. In this regard, the aim of Ocean is to provide a simple, accessible and scalable mechanism for mobile applications to discover, select and compose contextually-relevant Web Resources at runtime. To address this foundational aim, we first address the challenge of cross-domain operation by applying the Aladin-based context acquisition and modeling constraint described in section 3.4.2. According to this constraint, the Ocean approach directly subsumes the Aladin architecture as described in [49]. The extension of Aladin provides Ocean applications a means of operating in large-scale, cross domain scenarios. Notably, by subsuming the Aladin approach, the Ocean application model becomes architecturally client-centric, as shown in Figure 21.

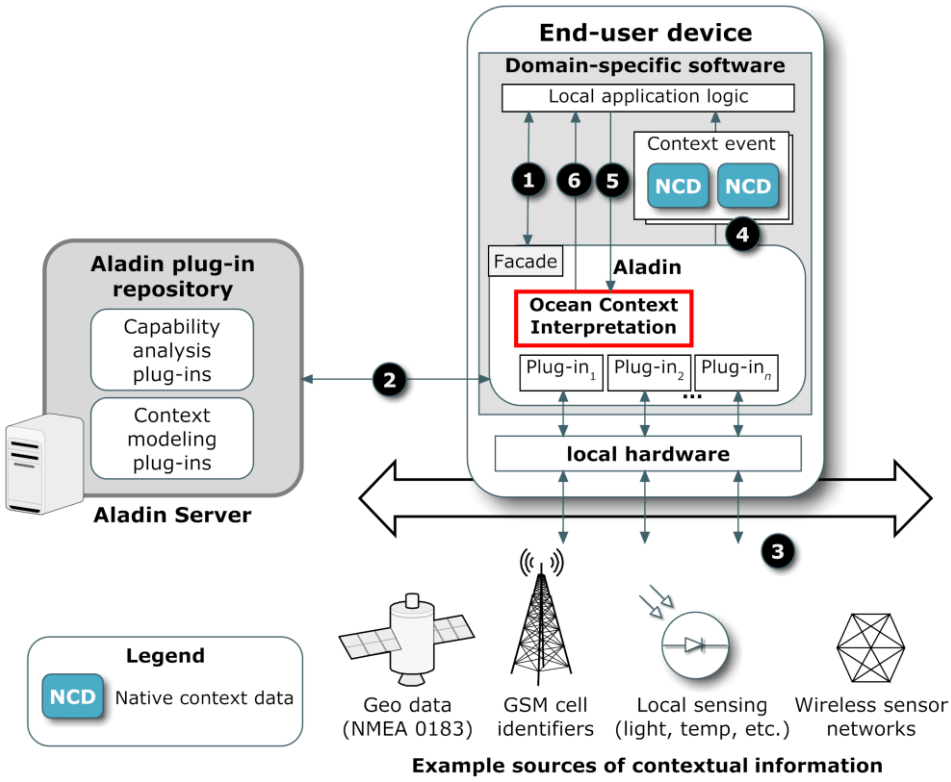


Figure 21: Ocean's extension of the Aladin architecture

With reference to Figure 21, Ocean extends the Aladin architecture in the following way:

1. Domain-specific software utilize Aladin's Façade to control high-level framework features such as context event subscription, resource allocation (e.g. thread priority), preference policies (e.g. privacy requirements), communications, etc. (If needed, context acquisition and modeling may be performed locally without Aladin using any appropriate means.)
2. The client-side Aladin Framework dynamically analyzes the capabilities of the end-user's device; downloading and integrating appropriate context acquisition and modeling plug-ins at runtime.
3. Using its installed plug-ins, Aladin continually acquires and models native context data (NCD) from the user's environment using local hardware and related device drivers.
4. Aladin notifies its host application of changes to the user's context situation through events that include NCD. Applications may react to context events as needed by parsing and interpreting extracted NCD internally.
5. Finally, the Aladin architecture is extended with Ocean-based context interpretation, which translates locally modeled NCD into a ranked list of contextually relevant Web Resources (described shortly).

To address the scalability and heterogeneity issues related to Ocean's Internet infrastructure constraint (see section 3.4.2), Ocean subsumes existing Web architecture by casting applications as conventional Web agents that "communicate using standardized protocols that enable interaction

through the exchange of messages which adhere to a defined syntax and semantics" [341]. Specifically, Ocean applications are defined as user agents that act on behalf of the user according to the interaction model elaborated throughout this chapter. As in conventional Web architecture, the Ocean application model makes no assumptions regarding the type of component interactions that might be implemented; requiring only that interoperation between distributed components adheres to the principles underlying Web architecture [32]. Hence, Ocean's application constituents (i.e. Resources) are addressed using standard URIs, resolved using DNS, adhere to the HTTP uniform interface, exchange standards-based message formats and provide suitable Representation formats. As in conventional Web architecture, Resource state management is governed by origin servers, which represent the "definitive source for representations of its resources and must be the ultimate recipient of any request that intends to modify the value of its resources" [106]. Accordingly, *any* RESTful Web Resource may become part of a dynamically assembled or adapted Ocean application; even those Resources not specifically designed for context-aware scenarios.

Next, Ocean extends the Web's hypermedia application model with additional aspects of dynamic, context-mediated Resource discovery and composition. Recall that in traditional hypermedia applications, Resource content and structure (e.g. Web page text) are used to provide the context mediation necessary for users to discover, select and compose Resources on-demand (e.g. dereference a link using a Web browser) [106]. Notably, conventional Web architecture defines URIs as having *global scope*; meaning that "the resource identified by a URI does not depend on the context in which the URI appears" [341]. However, while global scope supports the hypermedia application model, it does not inherently support Resource pre-filtering based on environmental context information such as location, proximate devices or activity (see section 4.1).

To overcome the Web's context-mediation limitations, Ocean defines a foundational architectural abstraction in section 4.3, called a *Contextualized Resource*, which provides an extensible semantic metadata model designed to constrain the *Discoverability Context* of conventional Web Resources. Extending the definition proposed by Dey and Abowd in [89], we define a Resource's Discoverability Context as:

> *Discoverability Context: The set of contextual criteria that must be fulfilled before a Resource is considered relevant to the interaction between a user and an Ocean application, including the user and application themselves.*

Definition 1: Discoverability Context

Further, as described in section 4.3.2, the Contextualized Resource is based on an extensible Context Metadata abstraction, which encapsulates the syntax and semantics of a given context domain; allowing domain-neutral processing by Ocean infrastructure and enabling non-experts to describe the Discoverability Context of Resources without domain expertise (see section 5.3.2). In this regard, a conventional Resource is *contextualized* (i.e. provided a Discoverability Context) by associating it with a specifically configured set of General and Context Metadata using the Contextualized Resource data model described in section 4.3.1.

Using the Contextualized Resource as a foundation, Ocean defines a Web-centric application model in section 5.2 that is based on the client-centric mashup (CS mashup) style. Briefly, in standard CS mashups, Resources are manually composed to create (relatively static) hybrid Web applications that combine data or computation from multiple sources. In the Ocean approach, we extend the conventional CS mashup style with support for dynamic, context-mediated component discovery and selection. Accordingly, Ocean allows mashups to be dynamically created and adapted in-situ using discovered contextually-relevant components (i.e. Resources). To promote wide-area Resource integration, we apply the REST-based component interoperation constraint described in section 3.4.2. Hence, Contextualized Resources in Ocean refer to Web Resources that adhere to the REST architectural style as defined in [105].

To maintain backwards compatibility with existing Web architecture, Contextualized Resource metadata are persisted within a context-aware component registry, called the Ocean Registry, which facilitates Resource contextualization and discovery (see section 5.3). To perform Resource discovery, Ocean applications aggregate NCD locally using the Aladin context modeling style and then query the Ocean Registry using a flexible search protocol that allows NCD to be included as query terms (see section 6.4.1.1). Notably, Ocean application developers may formulate Discovery Requests without detailed knowledge of the underlying context domains. In response, the Ocean Registry returns a ranked list of Descriptive Metadata regarding the relevant Contextualized Resources for a given Discovery Request (see section 6.4.1.2). Ocean applications can then select and compose appropriate Resources dynamically according to the extended CS mashup style described in section 5.2.2.

To support Contextualized Resource storage and lookup, the Ocean Registry provides integrated Persistence and Discovery Frameworks, which are described in Chapter 6. Briefly, the Persistence Framework allows Contextualized Resources to be efficiently stored and indexed for rapid retrieval. In real-world scenarios, Context Metadata are often represented by complex object types that resist classical database indexing techniques [29, 61]. As described in section 6.4, Ocean's Persistence Framework extends several similarity search techniques to address this issue. Further, the Persistence Framework accommodates a variety of domain-specific indexing techniques to support rapid query processing. Query processing is provided by Ocean's Discovery Framework, which operates in conjunction with the Persistence Framework; allowing NCD to be compared to persisted metadata contained within the Ocean Registry (see section 6.4). Related, domain-specific implementations of the aforementioned Context Metadata abstraction, called Context Handlers, are provided by external context domain experts who extend the functionality of the Ocean Registry using the open contribution process described in section 5.4. Further, in order to promote large-scale Resource contextualization, the Ocean Registry provides an additional open contribution process that allows community-based Resource contextualization using arbitrary combinations of Context Metadata (see section 5.5). For reference, a high-level overview of the Ocean approach is presented in Figure 22 (see section 5.2 for a complete description).

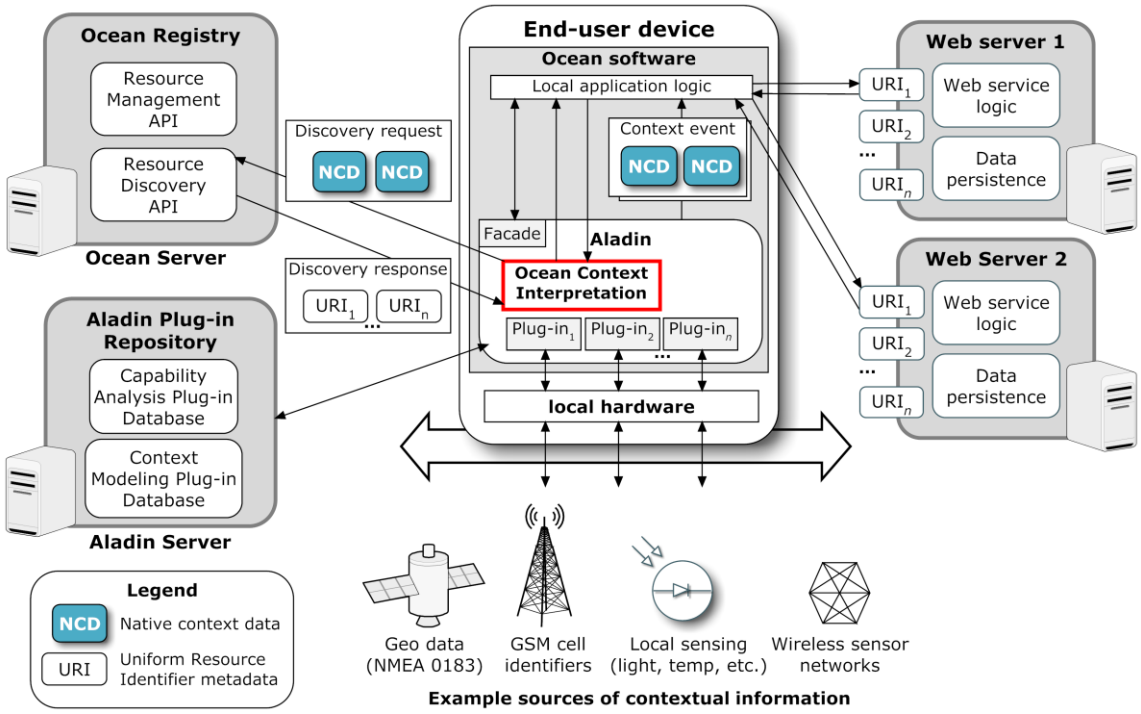


Figure 22: High-level overview of the Ocean approach

Ocean's Web-scale focus introduces two critical challenges for effective Resource discovery and selection, which are addressed in Chapter 7. The first challenge relates to context-mismatch, whereby Resource discovery performance is degraded due to mismatches between the Context Metadata used to describe a Web Resource and incoming query terms. Section 7.2 addresses context-mismatch by defining a query expansion mechanism that automatically supplements Discovery Requests with additional, contextually-relevant query terms extracted from query information shared by members of the Ocean end-user community. The second challenge relates to information overload, whereby Resource discovery performance is degraded due to extremely large numbers of undifferentiated query results. Section 7.3 addresses information overload by describing Ocean's Resource personalization approach that automatically predicts a user's affinity for a given Resource based on modeled preference information from similar Ocean end-users. Notably, both query expansion and Resource personalization are available as optional, privacy-aware enhancement features that can be used either alone or in combination to help improve discovery results.

3.6 Principle Ocean Stakeholders

The Ocean approach introduced in the last section implies participation by a diverse set of developers and end-users. In this section, we identify Ocean's principle stakeholders; providing a description of each along with an overview of related technical and business constraints. It should be noted that, while each stakeholder is presented in isolation, some roles may overlap to form hybrids. In addition, Ocean's adoption of existing architecture may involve ancillary or unintentional stakeholders; however, these are not investigated within this dissertation. Notably, the following stakeholder descriptions are not intended to be rigorous or complete. Rather, they are presented as a means of clarifying the principle users of Ocean as related to subsequent chapters of this dissertation.

3.6.1 Ocean Core Developers

Description: Ocean core developers (Core Developers) are responsible for the development and maintenance of the Ocean Registry, its software architecture and its Contextualized Resource data store. Key development aspects include: conceptualization of Ocean's application model; definition of related external APIs; development of associated communication protocols; definition of the Ocean Registry architecture and its related interface abstractions; development of Context Metadata and Contextualized Resource contribution mechanisms; development of the Contextualized Resource Persistence Framework; development of the Contextualized Resource Discovery Framework, including integration of appropriate similarity search algorithms; and development and maintenance of a suitable technical infrastructure. Key aspects of the Core Developer stakeholder are elaborated in section 5.3.

Key technical and business constraints: Core Developers are experts in the Ocean Registry software infrastructure and its related context-aware computing approach. Primary technical skills include: development and maintenance of large-scale software systems; knowledge of multi-feature similarity search algorithms; knowledge of domain-neutral indexing abstractions and associated data modeling; database development and optimization; and setup and maintenance of supportive technical infrastructure. Core Developers are not necessarily experts in any specific context or application domain. Core Developers devise and implement business models supporting the following key aspects: Ocean Registry design; technical infrastructure operations and maintenance; acquisition of required support software, tooling and training; and related costs associated with development and support staff.

3.6.2 Context Domain Experts

Description: In Ocean, we suggest that context modeling is best accommodated by external experts who understand the inherent complexities of a given context domain. In this regard, context domain experts (Context Experts) develop, test and contribute Context Handlers for use within the Ocean Registry. As Context Handlers encapsulate the syntax and semantics of a given context domain, Context Experts must be well versed in any associated native context data formats and related semantics. Context Experts also devise Context Metadata configuration options and related documentation intended to allow non-experts to sufficiently describe the Discoverability Context of conventional Resources. Finally, Context Experts participate in the Ocean Registry's Context Handler contribution process and manage contributed handlers throughout their lifecycle. Key aspects of the Context Expert stakeholder are further elaborated in section 4.3.2.

Key technical and business constraints: Context Experts are technically proficient within their given context domain. As such, Context Experts have technical knowledge related to developing Context Model implementations (i.e. Context Handlers) along with any associated configuration options and related documentation. Context Experts must be technically proficient in deriving, testing and optimizing the algorithms underlying a given context domain. Context Experts are not necessarily experts in the Ocean Registry architecture or any specific application domain. Context Experts devise and implement business models supporting the following key aspects: Context Handler design;

development and testing; acquisition of required support software, tooling and training; and related costs associated with any required development and support staff.

3.6.3 Resource Contextualizers

Description: Resource contextualizers (Contextualizers) create and maintain Contextualized Resources using the Ocean Registry's Resource Management API (see section 5.3.2). Contextualizers describe the Discoverability Context of Resources using the Context Handlers contributed by Context Experts. Contextualizers devise application scenarios that motivate and guide their contextualization efforts. Such application scenarios are domain-specific and may not be related to the semantic content of the Resources under consideration. Contextualizers may utilize the Resource Management API directly using its XML-based protocol; however, we envision the emergence of an ecosystem of stand-alone and Web-based tools designed to provide simplified contextualization services. In this regard, we suggest that many Web site developers may be motivated to provide integrated Ocean support; enabling end-users to transparently contextualize Resources using Ocean (e.g. a photo sharing Website that utilizes Ocean to automatically contextualizes shared digital photos on behalf of users). Finally, the Contextualizer stakeholder may be provided by software agents that automatically create Contextualized Resources based on specific semantic rules and strategies (e.g. context-aware Web crawlers – see section 8.4). Key aspects of the Contextualizer stakeholder are further elaborated in sections 5.3.2 and 5.5.

Key technical and business constraints: Contextualizers may vary widely in technical skill; however, most will have at least modest technical skills within a given contextualization domain. As such, Contextualizers may have technical knowledge related to the configuration of Context Metadata and an understanding of the basic Ocean application model. Some contextualization scenarios may require domain-specific technical skills, including Website design, web service interaction or stand-alone application development. Contextualizers devise and implement business models supporting the following key aspects: Contextualized Resource creation and maintenance; design, development and testing; acquisition of required support software, tooling and training; and associated costs associated with any required development and support staff.

3.6.4 Ocean Application Developers

Description: Ocean application developers (Ocean Developers) develop, deploy and maintain software applications that leverage Ocean functionality. In the Ocean approach, software applications created by Ocean Developers are known as Ocean applications. Broadly, Ocean Developers use the Ocean Registry's Discovery API to discover, select and compose contextually-relevant Web Resources at runtime (see section 6.4.1). As such, Ocean Developers are responsible for developing suitable application scenarios, providing domain-specific application logic and providing suitable technical infrastructure. Moreover, Ocean Developers are responsible for responding to Discovery Responses by appropriately selecting Resources based on Ocean's Descriptive Metadata. Once contextually-relevant Resources have been selected, Ocean Developers are responsible for orchestrating component composition and interoperation according to the REST architectural style. Key aspects of the Ocean Developer stakeholder are elaborated in section 5.2.

Key technical and business constraints: Ocean Developers are typically experts in a specific application domain. Accordingly, primary technical skills often include: mobile application development; Web application development; XML parsing and validation; REST-based component interoperation; human-computer interaction; and maintenance of supportive technical infrastructure. As previously described, Ocean context acquisition and modeling is accomplished using Aladin's client-centric approach. Accordingly, Ocean Developers are responsible for developing (or integrating) a suitable Aladin style context modeling mechanism (possibly through integration of third-party libraries). Ocean Developers are not necessarily experts in any specific context or contextualization domain. Ocean Developers devise and implement business models supporting the following key aspects: Application scenario development; software engineering and deployment; context modeling libraries; technical infrastructure operations; acquisition of required support software, tooling and training; and associated costs associated with any required development and support staff.

3.6.5 Ocean Application End-users

Description: Ocean application end-users (Ocean Users) utilize domain-specific Ocean software to accomplish tasks according to a specific set of user-specific needs, goals and modes of interaction. Typically, an Ocean User owns and manages the computing device executing Ocean software; however, software deployment and hardware maintenance may be handled by the user's organization. Ocean Users may be differentiated in terms of security and privacy considerations. Finally, Ocean Users may operate Ocean software in a variety of real-world computing environments.

Key technical and business constraints: Ocean Users vary widely in terms of technical capabilities; however, they typically have expertise in a given Ocean application domain. As such, Ocean Users may have no knowledge of Ocean infrastructure, context modeling, Resource contextualization or component interoperation techniques. Ocean Users are capable of operating and maintaining the hardware device upon which an Ocean application executes. Further, Ocean Users may rely on technical support provided by their organizations, hardware manufacturers or Ocean Developers. Ocean Users typically consider business models directly addressed by Ocean applications.

3.7 The Ocean Reference Implementation

Each aspect of the Ocean approach introduced in this chapter is described in detail throughout the remainder of this dissertation. Throughout this dissertation, Ocean is presented hierarchically by providing focused discussions of related work, formulating related theoretical contributions, and then validating each contribution with an ongoing discussion of the Ocean Reference Implementation (RI). The Ocean RI is a software-based implementation of core Ocean theoretical concepts and is used to validate that Ocean's various contributions are indeed realizable using existing techniques, technologies and infrastructure. Notably, the Ocean RI is intended to validate our contributions in terms of the design principles and approach constraints described in sections 3.4.1 and 3.4.2 respectively. Rather than presenting the Ocean RI within a dedicated chapter, we describe its implementation in conjunction with Ocean's theoretical development as a mechanism for clarifying core Ocean concepts. As such, our presentation of the Ocean RI is distributed over the next several

chapters. In addition, Chapter 8 presents an example Ocean application (based on the Ocean RI) that aims to draw together the theoretical and practical aspects discussed throughout the dissertation as a means of validating the Ocean approach in large-scale scenarios.

3.8 Chapter Summary

This chapter presented the foundations of the Ocean approach based on an adapted version of the IEEE Recommended Practice for Software Requirements Specifications. The chapter began with a description of the key challenges facing large-scale context-aware systems and described several related advances that address each challenge in isolation. Next, it described how Ocean aims to capture the entrepreneurial spirit of modern Web architecture as a means of supporting large-scale, real-world context-aware systems. Next, nonfunctional requirements were presented, including key design principles and approach constraints. Notably, in order to align Ocean with the requirements of conventional Web architecture, Ocean's key constraints include: Aladin-based context acquisition and modeling; Internet-based network infrastructure; Web-centric middleware; and REST-based component interoperation. Based on these constraints, the Ocean approach was then derived. Notably, the Ocean approach extends Aladin's client-centric context modeling style with an accessible and scalable mechanism for mobile applications to discover, select and compose contextually-relevant Web Resources at runtime. In order to facilitate wide-area component contextualization and discovery, several core Ocean concepts were introduced, including the Contextualized Resource abstraction; the Context Metadata abstraction; the Ocean Registry; community-based contribution models; mechanisms for Contextualized Resource storage and indexing; and community-based methods for overcoming context mismatch and information overload. Next, Ocean's principle stakeholders were described, including Core Developers, Ocean Developers, Contextualizers and Ocean Users. The chapter concluded with a discussion of the Ocean RI as an approach validation methodology.

Chapter 4
The Contextualized Resource

4.1 Introduction

The last chapter presented the foundations of the Ocean Web-centric context-aware computing approach. As previously introduced, Ocean investigates techniques for promoting the emergence of large-scale context-aware computing systems based on the integration of existing context sources, network infrastructure, application models and component interoperation styles. It was discussed how many of the adaptive techniques uncovered by isolated context-aware systems are increasingly relevant for everyday computing environments, which are rapidly becoming saturated with network connectivity and contextually-relevant distributed computation. Towards this end, the last chapter presented Ocean's scope, design principles, approach constraints, approach derivation, principle stakeholders and validation methodology. Unlike traditional context-aware systems, which are generally intended to address niche application scenarios, Ocean represents a generalized conceptual approach and complimentary technical infrastructure that intends to enable context-aware applications to arise in an "evolutionary fashion from modest beginnings, rather than from a Grand Plan" [223]. Towards this end, this chapter presents the Contextualized Resource as a mechanism for extending conventional Web architecture with extensible context-awareness features.

The structure of this chapter is as follows: Based on the Ocean approach derivation presented in the last chapter, relevant background and related work are discussed in section 4.2. Next, section 4.3 introduces the Contextualized Resource abstraction, which provides a mechanism for constraining the Discoverability Context of conventional Web Resources. Section 4.3.1 details Ocean's extension of the conventional Resource model with extensible semantic metadata indented to facilitate contextualization and wide-area component discovery. Next, section 4.3.2 details the Context Metadata abstraction underlying the Contextualized Resource by describing its architectural lineage and defining its data model. Section 4.3.3 provides a theoretical discussion of similarity modeling with regard to the syntax and semantics of a given context domain. Section 4.3.4 discusses validation of the Context Metadata interface. Ocean RI validation of the Contextualized Resource model and related XML schema are presented in section 4.3.5. The chapter concludes with a Contextualized Resource example in section 4.4.

4.2 Background and Related Work

Ocean's high-level approach derivation described in the last chapter draws inspiration from a broad range of related work. As is evident from the design constraints described in section 3.4.2, Ocean's foundation is rooted firmly in conventional Web architecture and Internet-based network communications. However, as described in section 3.5, the effectiveness of the Ocean approach is contingent upon the development of mechanisms for dynamically discovering, selecting and interoperating with contextually-relevant Resources in-situ. As previously introduced, Web architecture has been designed primarily to support the requirements of an Internet-scale distributed hypermedia system [105, 106, 341]. This section discusses the context-mediation approaches common to conventional Web architecture with regards to the Ocean approach.

In terms of runtime adaptation, conventional Web architecture addresses the late-binding of application constituents based on two principle mechanisms: Resource mediation and metadata mediation [341]. In the Resource mediation model, context information is delivered to users encoded within requested Representations as *inline context*, which typically takes the form of the informational and structural elements within a given Representation's data format. For example, Web pages typically provide inline information such as text, page layout and graphic elements that serve as mediators between users and the embedded hyperlinks within the page. As users navigate between application states (i.e. other Web pages) using discovered hyperlinks, additional contextual information is delivered progressively as the transaction unfolds.

During metadata mediation, supplemental descriptive information is provided alongside a Resource's Representation as a means of supporting services such as caching, content negotiation, cataloging, information retrieval, etc. Broadly, metadata has been defined as "machine understandable information about web resources or other things" [30]. Typically, metadata "consists of assertions about data, and such assertions typically, when represented in computer systems, take the form of a name or type of assertion and a set of parameters, just as in the natural language a sentence takes the form of a verb and a subject, an object and various clauses" [30]. In Web architecture, examples of metadata may include information regarding a Resource's language, owner, content-type and other arbitrary information [341].

Metadata are used to describe Resources in three principle ways. First, *embedded metadata* occur within the Resource's Representation itself (e.g. HTML metadata tags [365]). Second, *accompanying metadata* are provided separately from the Resource's representation during transmission (e.g. HTTP entity headers [104]). Third, *associative metadata* are used to provide semantic descriptions of Resources that can be stored externally (e.g. within another Resource or component registry). Broadly, embedded and accompanying metadata are used to guide interactions between Web agents and origin servers [104]. For example, during an **`HTTP GET`** request, HTTP headers are used to specify the requirements and preferences of the client (e.g. accepted content types or preferred languages). For example, the **`HTTP Accept`** header can be used to specify the media types that are acceptable as a response to a **`GET`** request. As described in [104], an **`Accept`** header may include **`"Accept: audio/*; q=0.2, audio/basic"`**, which indicates that the client prefers **`"audio/basic"`** but would accept any audio type if appropriate. Additional header examples include accepted encoding, charset, language and authorization credentials [104]. Notably, embedded and accompanying metadata have been classified as *transactional* context information due to their relevance *during* the Resource request/response phase [266].

While inline and transactional contextual information support the requirements of hypermedia applications, they are generally poorly suited for supporting the component interoperation requirements of many context-aware systems [29]. Problematically, Web-based hypermedia requires that Resources are requested and composed *before* context information can be extracted, which prevents component pre-filtering outside of the hypermedia transaction. In contrast, adaptive context-aware applications generally require pre-filtering of potential constituents based on domain-specific requirements [21, 29, 192, 231]. In Web-based context-aware systems, constituent pre-filtering is typically accommodated through the use of associative metadata, which are stored within a

component registry. Broadly, component registries are used to mediate dynamic binding and interaction between loosely-coupled elements of a distributed application [29, 308]. Within component registries, associative metadata are used to describe important attributes of distributed components such as addressing information, interface descriptions, data types and other semantic information [254]. Distributed applications discover suitable constituents by querying the component registry using a search protocol. The component registry uses incoming queries to perform a component lookup using the associated metadata as a filtering mechanism. Components that match the specified search parameters and query terms are returned to the application where they may be used for runtime composition.

Component registries are typified by the Universal Description, Discovery and Integration (UDDI) specification central to SOAP Web Services [231]. UDDI provides an XML-based metadata specification that provides structured information describing SOAP-based Web-services. UDDI is used in combination with the Web Services Description Language (WSDL) [67], which typically provides a *process-centric* description of a Web Service's method syntax, supported data types and endpoint addresses. UDDI supports Publish and Inquiry APIs that provide clients a means of storing, updating, querying and deleting Web Service metadata. As per the SOAP Web Services model, once a component has been discovered using UDDI, point-to-point communications between the discovering application and the selected distributed component occur without further interaction with the UDDI registry (unless initiated by the application). The query facilities of UDDI are based on a SOAP-based search protocol that provides basic interface matching and text-based keyword search. However, UDDI's lack of semantic metadata support has been viewed as serious limitation in more complex interaction scenarios such as mobile and context-aware systems [29, 308]. Indeed, while several open-source and commercial implementations of UDDI exist [155, 238], industry support has been waning. Notably, Microsoft, IBM and SAP discontinued public Internet-based UDDI registries in 2006 [283].

The lack of semantic metadata within the UDDI specifications has been addressed by more recent work. For example, the Electronic Business using eXtensible Markup Language (ebXML) [232] defines a registry service that incorporates a more advanced metadata model capable of supporting hierarchical classification and association descriptions of registered components. In addition, the ebXML registry provides a comprehensive query interface that supports SQL-like search constructions. Similarly, Pokraev et al. [254] developed an enhanced UDDI registry that incorporates a semantic model built on the Web Ontology Language for Services (OWL-S) [244]. Their approach integrates domain-specific metadata for semantic service description such as the Composite Capability/Preference Profiles (CC/PP) ontology [364]. Their approach also allows clients to include contextual information within a query in order to improve component discovery results for context-sensitive applications. Similarly, Song, et al. [308] use an ontology-enabled registry to address semantic interoperability. More recently, the SOPHIE architecture [29] was developed as a means of addressing the needs of context-aware scenarios through the association of semantic metadata with conventional WSDL using a hybrid registry approach. Seeing the need for improved semantic component mediation, the W3C developed a charter focused on creating Semantic Annotations for WSDL and XML Schema (SAWSDL) that intends to define a standards-based mechanism whereby semantic annotations can be added to WSDL components [289].

The inclusion of semantic metadata is increasingly recognized as important for supporting context-aware scenarios [29]. In this regard, the Multi Channel Adaptive Information Systems (MAIS) project [14] provides a component registry designed for mobile applications and intelligent environments. MAIS aims to improve upon competing registry techniques by accounting for both the context of the client and the service during query operations. Similarly, Chakraborty et al. [59] proposed a set of semantic extensions, called DReggie, which supplement the JINI lookup service (JLS) [342] with supplemental semantic information based on the DARPA Agent Markup Language (DAML)[9]. Additional work by Doulkeridis et al. [93] demonstrates the effectiveness of augmenting traditional component registries with a multidimensional Object Exchange Model (OEM) graph that models services as atomic nodes [313]. Notably, their approach demonstrates how contextual information can improve query results in dynamic service scenarios and how the type of context metadata model constrains an approach in terms of computational complexity and semantic expressiveness. Related, Blackstock, Lea and Krasic [37] suggest a "shared environment model" as a means of bridging divergent middleware using enhanced interface specifications. An additional registry approach has addressed search-space reduction using context values and semantic parameters [265].

While the addition of semantic metadata to component registries has been shown to improve constituent discovery in loosely-coupled distributed systems, existing techniques are largely incompatible with the Ocean approach. For example, context-enhanced registries define intermediary metadata models that may not be known be all participants or capable of expressing the fidelity of native context information [254]. Further, existing approaches provide only a restricted set of metadata types and do not provide inbuilt mechanisms for promoting contributions by external domain-experts [29]. Further, in order to accommodate component interoperation, current registry approaches focus on supporting *process-centric* interoperation (PCI) styles such as those typified by SOAP-Web services. As described in section 3.2.7, PCI-based interoperation relies on the definition of domain-specific interface methods and related data types using interface description languages such as WSDL or Corba IDL. As a result, PCI techniques result in an explosion specialized interfaces; each with complex method semantics and related sequencing requirements. Thus, in order for an application to effectively utilize a discovered process-centric component, it must possess significant prior component knowledge; reducing the chance of interoperation in large-scale, real-world environments [334].

As detailed in section 3.2.8, by standardizing component addressing schemes, method semantics and message payloads, the REST architectural style is increasingly recognized as well-suited for supporting *cross-domain* component interoperation. In this regard, several projects have begun exploring the application of semantic metadata to conventional RESTful Resources. One of the first approaches in this regard was the Platform for Internet Content Selection (PICS) [366] specification proposed by the W3C. Broadly, PICS "defines a language for describing rating services. Software programs will read service descriptions written in this language, in order to interpret content labels and assist end-users in configuring selection software" [366]. Accordingly, PICS metadata provide a set of extensible attribute-value pairs that are used to describe Resources according to name, subject, category and content rating. PICS metadata can be embedded within Resources or stored within an

[9] http://www.daml.org/

external rating registry where they can be used to mediate interactions between User agents and Resources. While PICS was integrated within the Microsoft Internet Explorer browser within its "approved sites" feature, its narrow application scope and lack of a coherent registry design ultimately led to the project's discontinuance.

A substantially richer approach for contextualizing Web Resources is the Dublin Core (DC) metadata approach [82]. As part of the W3C's semantic Web initiative [368], which endeavors to extend the Web with machine-based reasoning and processing capabilities, the DC represents an international metadata standard for describing the semantics of Web Resources. The DC provides a standardized set of metadata elements that are implemented using non-proprietary technologies such as XML and RDF [367]. As described in [83], the elements that comprise the DC are separated into two principle categories. The first category, termed *simple elements*, refers to generic metadata that are applicable across domains (e.g. title, description, date, language, etc.) The second category, termed *qualified elements*, provides domain-specific extensions of simple elements. Both simple and qualified elements incorporate established vocabularies and provide for extensions. For example, the simple element "Date" may be refined as "Date Submitted" and encoded as "W3C-DTF." Implementations of simple or qualified elements may be embedded within Resources or linked via associative metadata. While the DC and RDF do not define a specific application model that can be used directly by context-aware applications, they do provide insight into the benefits of non-proprietary technologies and community contribution. An example DC metadata element is provided below.

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:dc="http://purl.org/dc/elements/1.1/">

 <rdf:Description rdf:about="http://example.com/audio/guide.ra">

  <dc:creator>Rose Bush</dc:creator>
  <dc:title>A Guide to Growing Roses</dc:title>
  <dc:description>Description text<dc:description>
  <dc:date>2001-01-20</dc:date>

 </rdf:Description>
</rdf:RDF>
```

Figure 23: An example of Dublin Core metadata (from [83])

In terms of RESTful context-mediation in context-aware systems, many current approaches associate relatively simple contextual metadata to domain-specific Web URIs (generally under the control of the project). For example, Yarin and Ishii's TouchCounter project [375] provides Web-based information describing the usage patterns of physical storage containers and shelving surfaces through Resource mediation based on infrared tag identifiers. In their approach, a unique infrared tag identifier is used to mediate interactions with related Web content (e.g. updates or consuming). Similarly, the Lancaster GUIDE system [78] uses a Web-based infrastructure as a means of providing dynamic information within a mobile tour-guide scenario. User's of the GUIDE system carry a specially outfitted mobile device that runs the GUIDE application and related context acquisition mechanisms. As users encounter changing contextual information (e.g. transitioning between

WaveLAN cells) the system automatically applies an integrated filtering algorithm to discover relevant URIs from within a set of predetermined Resources. Rendering of selected Resources is provided by an integrated Web browser within the GUIDE application. Similar Web-centric tour-guide systems have been developed to explore the binding of various other context types with Web-based content [191, 303].

Hewlett-Packard's Cooltown project [179] takes a more general approach to Resource contextualization. In Cooltown, HP researchers developed a software system designed to augment people, places and things with Web URIs. Cooltown supports URI context mediation through two principle mechanisms. The first mechanism, termed *direct sensing*, involved the development of a short range wireless protocol (called eSquirt) that is intended for sending standard URIs across common technologies such as IR and Bluetooth. The second approach, termed *indirect sensing*, is based on an external context registry that supports associations between Web Resources and specific types of context data. Cooltown proposes that indirect sensing might be accomplished by linking information such as RFID tags or iButtons[10] to Web URIs using a registry; however, Cooltown's indirect sensing approach was only preliminarily elaborated and its initial design supports a limited set of predefined context information.

Cooltown can be understood (anachronistically) as an *object hyperlinking* technique refers to the association of physical objects with Web-based Resources using some mechanism for context-mediation. Similar to Cooltown, object hyperlinking systems may employ either direct or indirect sensing techniques to obtain URI information or resolvable identifiers from real-world objects. For example, Semapedia[11] have developed an approach for linking physical objects and locations to Wikipedia[12] articles through a dedicated two dimensional (2D) barcode system and related mobile-phone reader software. Using a Semapedia reader, images taken of discovered 2D barcodes are translated into valid URIs and used to open the associated Wikipedia page in a mobile browser. Related barcode techniques include Denso-Wave's Quick Response Code (QR Code)[13] and Cambridge University's ShotCode[14] system.

While object hyperlinking is becoming a commercial success in many markets [343], current approaches are generally proprietary and limited to a single application model. For example, Semapedia only allow association of 2D barcodes to Web Resources contained within the Wikipedia domain. Similarly, the Hardlink[15] system provides a mobile phone gateway that maps physical hyperlinks to Resources within the dedicated .mobi domain. Moreover, sophisticated contexts (e.g. profile, temperature and geo-location) are not directly supported by current approaches. Additionally, many current object hyperlinking approaches require specific hardware support (e.g. a compatible inbuilt camera) and domain-specific software for translating captures images into suitable identifiers. For example, QR Codes require a compatible onboard camera and ISO/IEC 18004 compliant translation software whereas ShotCode requires entirely different translation software. In this regard,

[10] http://www.maxim-ic.com/products/ibutton/
[11] http://www.semapedia.org/
[12] http://www.wikipedia.org/
[13] http://www.denso-wave.com/qrcode/aboutqr-e.html
[14] http://www.shotcode.com/
[15] http://hardlink.mobi/

the proprietary nature of these context mechanisms cannot be used to support *arbitrary* application domains or comprehensive pre-filtering and ranking of contextually relevant URIs. In response, the recently formed W3C UbiWeb[16] has been formed to promote more open contextualization approaches; however, to the best of our knowledge, no current object hyperlinking technique is directly capable of supporting the Ocean application model introduced in section 3.5 and elaborated in section 5.2.

The commercial success of existing object hyperlinking illustrates the effectiveness of RESTful component interoperation. In particular, existing approaches demonstrate how the translation of contextual information into standard Web URIs can be combined with the Web's uniform interface to provide cross-domain access to contextually-relevant Resources. Recent efforts for associating additional semantic information to REST resources includes OWL-S [244], WSMO [372] and WSDL-S [4]. These approaches provide a controlled vocabulary for semantically annotating REST Resources; however, they do not address the component mismatch that may occur between Resources that provide differently represented data. Towards this end, Lathem, Gomadam and Sheth have developed the SA-REST [192], which provides an approach for bridging incompatible component representations using ontology-based matching and data transcoding. SA-REST is derived from W3C SAWSDL recommendation [289] and provides a means of annotating the inputs and outputs of REST-based Resources using RDF descriptions. Using semantic annotations, SA-REST applications are able to combine the inputs and outputs of various Resource representations; however, SA-REST does not provide a wide-area component discovery approach based on arbitrary context metadata.

Based on the increasing popularity of REST-based component interoperation, Hansen et al. proposed a context-aware hypermedia system called the HyCon framework [139]. HyCon represents a "framework for context aware hypermedia" that is designed to facilitate context-aware browsing, annotation, searching and tour guidance. Of particular interest, HyCon provides integrated support for community-driven Resource annotation, whereby users of the systems can "tag" conventional Resources with context data such as RFID or Bluetooth identifiers. In this regard, HyCon supports the automatic collection of context information that can be used later to facilitate context-aware browsing. To overcome challenges related to heterogeneity, HyCon employs open representation formats such as XLink and SVG. Finally, the HyCon framework uses a data model intended to capture and associate contextually relevant information with Resources. While HyCon's dedicated application model is incapable of supporting Ocean's generalized approach, its abstract data model and community-focus provides insight into the benefits of extensible metadata and collaborative annotation respectively. An overview of the HyCon abstract data model is shown below in Figure 24.

[16] http://www.w3.org/UbiWeb/

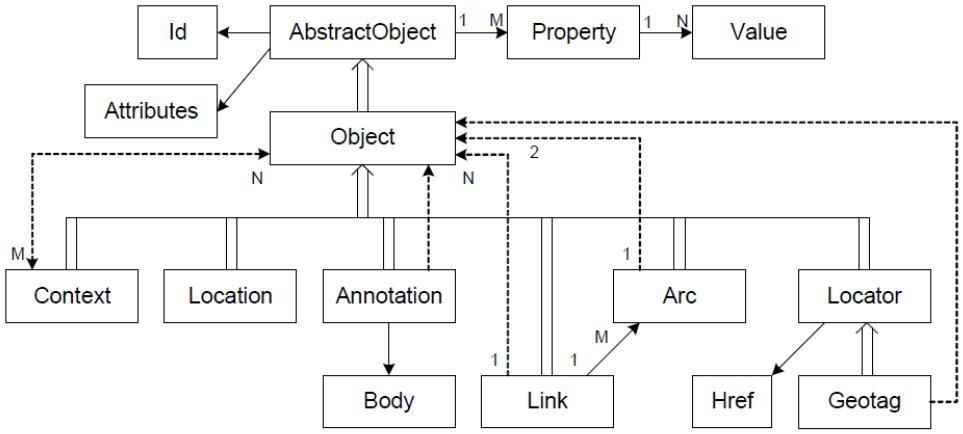


Figure 24: The HyCon abstract data model (from [139])

While the approaches presented in this section ostensibly support some features of the Ocean approach, several important challenges persist. First, many current techniques rely on intermediary context formats; requiring that locally acquired native context data be converted into a predetermined intermediary format for use with a given discovery protocol (e.g. mapping native geospatial data into a specific data structure using an OWL ontology). Such techniques errantly presuppose significant domain-expertise on the part of the application developer and may impose significant processing overhead on mobile devices. Second, although recent approaches have explored Resource mediation based on context information, existing techniques are limited to a predetermined set of context metadata, do not address wide-area scenarios and cannot be extended by external context domain experts. Third, existing approaches are generally application specific and cannot easily support a wide variety of application domains. Fourth, existing techniques do not address context mismatch scenarios whereby the query terms provided within a component Discovery Request do not sufficiently match the metadata used to contextualize Web Resources (see section 7.2). Finally, existing techniques are not well-suited for overcoming information overload in complex environments; requiring that developers (or users) manually filter and select appropriate Resources for runtime composition (see section 7.3).

4.3 The Contextualized Resource Abstraction

Based on the limitations presented above, this section derives the associative metadata model that provides a foundation for flexible Resource contextualization and discovery in Ocean. Notably, this model provides a mechanism whereby additional context handling support can be dynamically integrated into Ocean by external Context Experts. Further, it supports the development of an open contribution model whereby communities of Contextualizers can freely annotate Web Resources with arbitrary contextual metadata using an open set of APIs. Further, it provides a foundation for Ocean's Persistence and Discovery Frameworks, which enable Ocean applications to perform wide-area component discovery and selection using locally modeled NCD as query terms. Finally, it imposes a separation of concerns between conventional Web Resources and associated contextual metadata; providing developers a familiar Web-centric programming approach and enabling the independent evolution of both Resources and contextual metadata.

4.3.1 Extending the Web's Resource Model

Recall from section 3.2.6 that a *Resource* represents the primary informational abstraction upon which the Web's architectural style is based. As described in [105], any information that is important enough to be named can be modeled as a Resource (e.g. an image, newsfeed, software release, Web page, search result, etc.) In conventional Web architecture, Resources are addressed by Unified Resource Identifiers (URIs), which provide a globally standardized mechanism for component access. As described in [341], "A resource should have an associated URI if another party might reasonably want to create a hypertext link to it, make or refute assertions about it, retrieve or cache a representation of it, include all or part of it by reference into another representation, annotate it, or perform other operations on it. Software developers should expect that sharing URIs across applications will be useful, even if that utility is not initially evident." During Web-centric component interoperation, distributed clients observe and change Resource state by sending and receiving Resource Representations using HTTP's uniform interface methods (i.e. **`GET`**, **`POST`**, **`PUT`**, etc.) Briefly, a Representation can be understood as a sequence of bits (generally in a standardized data format) that represents the current *or desired* state of a Resource. As Representations are exchanged between client and server, Resource state is managed by the origin server while application state is managed by the client. An overview of the Web's conventional Resource model is shown in Figure 25.

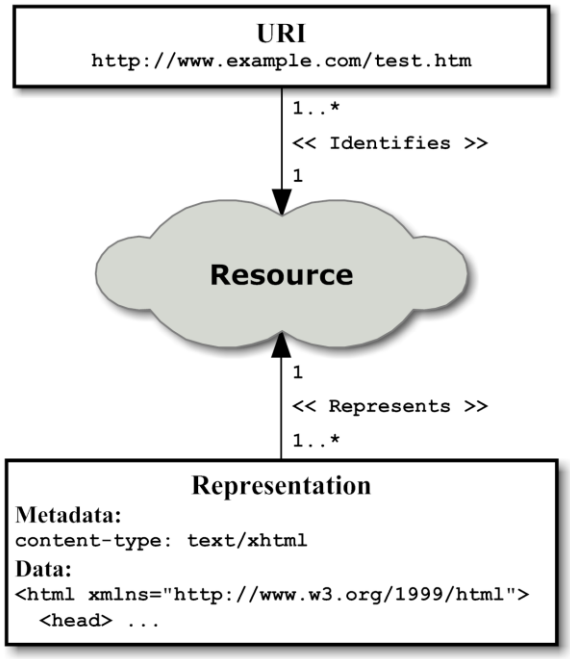


Figure 25: The Web's conventional Resource model

The Web's Resource model provides a foundation for many of the key features of modern Web architecture. As described in [106], "First, it provides generality by encompassing many sources of information without artificially distinguishing them by type or implementation. Second, it allows late binding of the reference to a representation, enabling content negotiation to take place based on characteristics of the request. Finally, it allows an author to reference the concept rather than some singular representation of that concept, thus removing the need to change all existing links whenever the representation changes." In order to retain these features, Ocean adopts the conventional Resource

model as the foundation of its contextual metadata approach (described shortly). Accordingly, Ocean application constituents are defined as standard Web Resources as described in [341].

Recall that in the REST application model, hyperlinks represent the "engine of application state" [105] whereby clients navigate between various application states by discovering, selecting and composing Resources at runtime. As previously described, inline and transactional context information are used to provide contextual mediation during an application's interactions with distributed Resources (e.g. Web page text, HTML metadata and HTTP headers). However, as introduced in section 4.2, the Web's hypermedia model precludes component pre-filtering based on real-world context data such as location, proximate devices, temperature, etc. Towards this end, we propose an extended Resource model, called the *Contextualized Resource (CR)*, which supplements the Web's conventional Resource model with an extensible set of Ocean Metadata intended to constrain a Resource's global scope through the establishment of a Discoverability Context. Recall from Definition 1 that a Resource's Discoverability Context is *the set of contextual criteria that must be fulfilled before a Resource is considered relevant to the interaction between a user and an Ocean application, including the user and applications themselves.* An overview of the Contextualized Resource model (CR model) is shown in Figure 26.

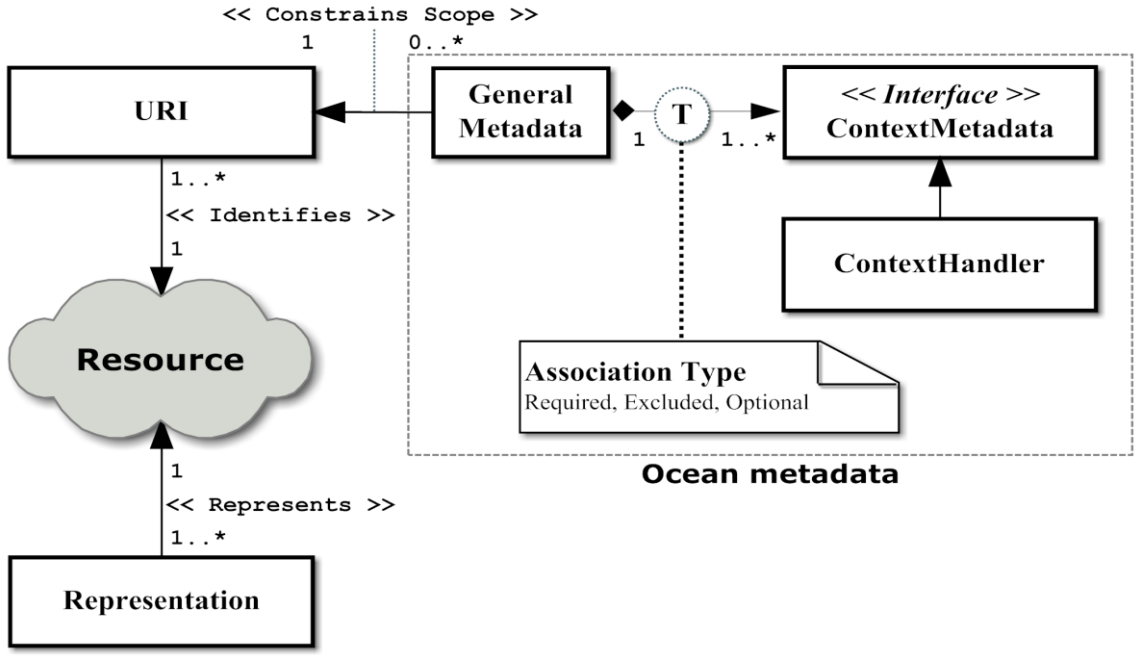


Figure 26: The Contextualized Resource model

The CR model extends the Web's conventional Resource model in the following ways. First, in order to maintain backwards compatibility with existing Web infrastructure, the conventional Resource model is retained entirely. Next, we define an associative semantic metadata model, called *Ocean Metadata*, which is used to describe the Discoverability Context of an associated Resource. Ocean Metadata consists of a single *General Metadata* entity and one or more *Context Metadata* entities. General Metadata include Resource-specific information such as the component's URI, data-type, title, description and an optional WADL document that can be used to provide a machine-processable description of the Resource's implementation of the HTTP uniform interface (see [137]

for details). Context Metadata abstract the syntax and semantics of a given context domain into an interface known as the *Context Metadata interface* (section 4.3.2). Briefly, the Context Metadata interface supports CR configuration (section 4.3.2), similarity modeling (section 4.3.3), persistence and indexing (section 6.3), discovery (section 6.4) and association modeling (section 7.2.4). Domain-specific implementations of the Context Metadata interface are termed *Context Handlers*. Notably, Context Handlers support instantiation from a given set of supported NCD (during query operations) and may be specifically configured when establishing a Discoverability Context (during contextualization). Finally, Context Metadata can be associated with General Metadata according to the three association types described briefly below:

- **Required**: Indicates that a given Contextualized Resource is only discoverable if an Ocean application provides NCD that is supported by at least one of the Context Handlers within the Ocean Metadata entity. For example, a Context Handler may support NCD based on the Geography Markup Language (GML) Encoding Standard [237]. If such a Context Handler is associated within Ocean Metadata using the *required* type, an Ocean application *must* provide compatible NCD in order for the Contextualized Resource to be considered discoverable (e.g. by providing NCD in the GML format).
- **Excluded***:* Indicates that that a given Contextualized Resource is not discoverable if an Ocean application provides NCD that is supported by a Context Handler within the Ocean Metadata entity. For example, a Contextualizer may wish to specify geo-locations where a given Contextualized Resource is *not* discoverable.
- **Optional**: Indicates that supported NCD may be used during discovery but are neither required nor excluded.

Importantly, the CR model introduced above is domain neutral regarding which Resources should be contextualized; the types of Ocean Metadata that may be included; and how Ocean applications might select and compose discovered Resources in-situ. Its use of the associative metadata approach (see section 4.2) allows for the contextualization of *existing* Web Resources without requiring changes to conventional Web architecture. In this regard, Ocean Metadata are persisted in a context-aware component registry, called the Ocean Registry, which is used to facilitate Resource contextualization and lookup (see section 5.3). Further, Contextualized Resources do not require domain control over a given Resource in order for Discoverability Contexts to be established. As such, *multiple* Ocean Metadata entities may exist for any given URI; providing the foundation for Ocean's community-based contextualization process, which is described in section 5.5.

4.3.2 The Context Metadata Abstraction

To address Ocean's Web-scale focus (see section 3.3), the CR model must support expressive contextualization across a diverse range of context domains. As detailed in section 2.3.3, native context data (NCD) represent the complex and often subtle semantics of a given context domain. Further, NCD are typically rendered using high-fidelity data formats that are specifically designed to express the specific details of a given context modeling technique. Recall that prominent context modeling techniques include key-value models, markup-scheme models, graphical models, object-oriented models and logic-based models. Hence, a key challenge for the CR model is to provide an

extensible means of encapsulating a variety of context modeling techniques for use within the Ocean approach. Towards this end, this section elaborates upon the Context Metadata abstraction introduced in the last section.

To illustrate the domain-centricity inherent to many context modeling techniques, we briefly consider a few aspects of the *location* context domain. As detailed by Hightower and Borriello [149], a variety of location sensing techniques have been devised, including several prominent triangulation approaches. Briefly, triangulation leverages the geometric properties of triangles to compute object locations by calculating distance (i.e. lateration) or by calculating angle or bearing measurements (i.e. angulation). Prominent lateration techniques include direct (e.g. physical action impediment), time-of-flight (e.g. ultrasonic pulse timing) and attenuation (e.g. radio signal power decrease). While similar to lateration, angulation techniques measure angles rather than distance (e.g. exploiting phased antenna arrays and receivers with a known geometry). Various approaches are used to represent such data. For example, the physical or symbolic location of a radio may be inferred from a set of received radio signals and associated signal characteristics (e.g. signal strength or time-of-flight) [19]. To express the details of a given context domain, native context data formats are generally devised. For example, one system may represent such signal values according to a proprietary key-value data structure where the keys represent signal source identifiers and the values represent associated signal strength values (according to a specific decibel normalization scheme [352]). Another system may provide additional quantization of low-level signal values into a higher-order context representation such as a geo-location encoded using a markup model [237]. While we do not elaborate further, we note that many context domains are similarity complex and domain-specific (see section 2.3.3).

In traditional context-aware systems, the effective use of complex NCD such as the triangulation examples described above requires that a system be capable of parsing and understanding the underlying context domain. While such domain knowledge can be safely assumed in small-scale or prototype systems, large-scale context-aware scenarios become quickly intractable as the heterogeneity of context data increases [79, 145]. To accommodate context heterogeneity, Ocean provides the Contextualized Resource abstraction (see section 4.3.1), which supports the contextualization and discovery of conventional Web Resources using Ocean Metadata. As previously described, the principle architectural abstraction of the CR model is the Context Metadata interface, which allows for the development of a broad range of Context Handler implementations. In the Ocean approach, the Context Handlers are used to support effective and efficient CR persistence and discovery (see sections 6.3 and 6.4 respectively). Relevant details of the Context Metadata interface are shown in Figure 27.

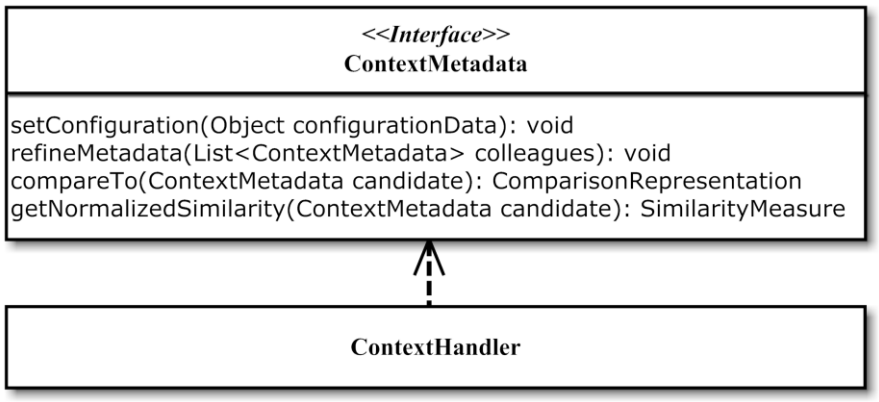


Figure 27: The Context Metadata interface

With regard to Figure 27, the Context Metadata interface methods are defined as follows:

- **`setConfiguration(Object configurationData): void`** – As Context Handlers encapsulate domain-specific context information, their configuration is provided by a similarly domain-specific configuration data object. Notable, the **`configurationData`** object is designed by Context Experts to provide configuration of the instance's internal state and comparison semantics by non-experts.

- **`refineMetadata(List<Context Metadata> colleagues): void`** – Provides a mechanism whereby Context Handlers may provide additional configuration refinement in response to the presence of other Context Metadata contained within a Contextualized Resource (termed *colleagues*). During instantiation, each Context Handler receives all of its colleagues via the **`refinemetadata`** method and may adapt its internal configuration as needed. For example, a Context Handler encapsulating physical positioning based on radio frequency signal strength values may alter its comparison semantics if it determines that the receiver's antenna is located indoors (i.e. potentially affected by physical obstructions such as walls and floors). In this example, an indoor location might be inferred from the presence of a colleague providing a well-known ontological description of an indoor state (e.g. **`Environment:Location:Building="indoors"`** as described in [185]).

- **`compareTo(ContextMetadata candidate): ComparisonRepresentation`** – Supports domain-specific comparison of instantiated Context Metadata as required for indexing via the Ocean Persistence Framework (see section 6.3). Implementations of **`compareTo`** provide domain-specific comparisons, which are represented by an extension of the **`ComparisonRepresentation`** class. The **`compareTo`** method is used to encapsulate various comparison models, such as geometric models, feature models, alignment-based models and transformational models [125]. Common implementations may include Quadratic Form Distance, Levenshtein distance (also known as edit distance) and Jaccard's Coefficient [376]. Additional details regarding Context Metadata similarity modeling are provided in section 4.3.3.

- **`getNormalizedSimilarity (ContextMetadata candidate): NormalizedSimilarity`** - As **`ComparisonRepresentations`** cannot generally be combined across diverse context domains, **`getNormalizedSimilarity`** is used to obtain a domain-independent similarity score. The output of the **`getNormalizedSimilarity`** method is a **`NormalizedSimilarity`** object whose value is a numerical score constrained to the unit interval ($\{0 \leq x \leq 1\}$, where $x \in \mathrm{R}$) where 0 equals no similarity and 1 equals perfect similarity. As **`ComparisonRepresentations`** may be based on the distance between multivariate points in vector space, they may need to be converted to an appropriate normalized similarity score in order to be combined. A typical example of how such a conversion might be accomplished using Euclidian distance is shown below (from [39]):

$$NormalizedSimilarity = \frac{1}{1 + EuclidianDistance}$$

4.3.3 Modeling Similarity

As described in the last section, the Context Metadata abstraction provides a means of encapsulating domain specific Contextualized Resource configuration and comparison. As such, Context Metadata implementations (i.e. Context Handlers) may be represented by complex objects that are difficult to compare, index and retrieve from a data store. In general, information systems that must contend with large amounts of complex and heterogeneous data require domain-specific mechanisms for differentiating between persisted objects [376]. As database systems have increased in complexity and scope, the types of information stored within them has changed dramatically. Moreover, the problem domains served by such systems now span a broad range of disciplines [61], including statistics, computational geometry, artificial intelligence, databases, pattern recognition, etc. Increasingly, application scenarios within these disciplines rely on the storage and retrieval of data types such as images, audio, video, unstructured text and object hierarchies. However, the persistence features and query mechanisms common to classical databases cannot often differentiate between such data types meaningfully [281]. Similarly, Contextualized Resources can be understood as arbitrarily complex data structures that may resist classical indexing and search techniques. Hence, this section provides background and related work regarding similarity modeling as a foundation for the upcoming discussions regarding Context Handler indexing, persistence and discovery within the Ocean approach.

Similarity modeling in common database approaches is often designed to accommodate simple structured data that can be easily compared. In many classical techniques, candidate objects are retrieved from a data store if a well-defined textual or numerical data entity present within the record (i.e. a key) matches a given set of search criteria (e.g. constraints expressed in a query grammar) [205]. Searching and indexing operations generally operate on simple fields represented by data-types that can be directly compared to related aspects of a posed query (e.g. strings or integers). As discussed in [61], classical search techniques include *key search*, where records are returned if a record key precisely matches the query; *range search*, where records are returned based on full or partial matches within specific fields or value ranges; and *proximity search*, where records are returned if they are considered statistically related to the search query.

Notably, in proximity search, statistical relevancy is often determined by modeling the search space as a hyper-rectangle of *k* dimensions, where each dimension is related to a specific field of interest within the database (numerical or alphabetical). Records are then projected as points within the hyper-rectangle and queries return records if they are contained within a sub-rectangle specified along a dimension of interest. This approach is common to many Web search engines and is exemplified by techniques such as grid file [229]. While such techniques may resolve issues related to text retrieval, they often lack the sophistication to model similarity between complex data types such as multimedia elements or object hierarchies [61]. For example, in many image search scenarios such as fingerprint analysis and facial recognition, it is derisible to search through a large image repository in order to extract images that are *similar in some meaningful way* to a query image [351].

To illustrate how similarity comparisons might be implemented using the Context Metadata's **`compareTo`** and **`getNormalizedSimilarity`** interface methods described in the last section, a brief overview of similarity modeling techniques is now presented. Importantly, this section is not intended to provide a complete treatment of similarity modeling; rather, it presents a sampling of approaches as a means of illustrating how specific Context Handlers might be implemented. Moreover, it also illustrates how similarity modeling techniques often imply significant domain-expertise that may be prohibitively complex for non-experts. As such, the Context Metadata interface is used to insulate the Ocean approach from complex similarity calculations during persistence and discovery operations, which are elaborated in sections 6.3 and 6.4 respectively.

We begin our overview of relevant approaches by introducing g*eometric models*, which have long been recognized as an influential notion of similarity [125]. Broadly, geometric approaches represent similarity relationships by modeling important aspects of a given object type as a set of points within a dimensionally organized metric space. The input to geometric models may consist of any measure of pair-wise proximity such as correlation coefficients, joint probabilities, similarity judgments, etc. Objects in the dataset are represented as points in an *n*-dimensional space, where the similarity between a pair of objects is inversely related to the distance between the object's points in the space. For example, as presented in [299, 300], distance metrics within Euclidian space are exemplified by non-metric multidimensional scaling (MDS) of the basic form:

$$dissimilarity\ (i,j) = \left[\sum_{k=1}^{n} |X_{ik} - X_{jk}|^r\right]^{\frac{1}{r}}$$

where *n* indicates the number of dimensions, X_{ik} represents the dimension *k* for item *i*, and *r* is a value that allows different metrics to be used. For example, if *r*=1, the distance between two points is computed by a city block metric (also known as the rectilinear distance or Manhattan distance), whereby the total distance is found by summing the distance between points for each dimension. As another example, when *r*=2, a standard Euclidian notion of distance is invoked, whereby the distance between two points is found by the length of a straight line connecting them. Notably, a Euclidian metric has been shown to be an effective model for some perceptually fused dimensions, whereas the city block metric has been shown as an effective model for separated similarity notions such as color and size. For reference, the Manhattan (or Minkowski L_1 distance) is typically given by*:*

$$d(x, y) = \sum_{i=1}^{n} |x_{i-} y_i|$$

While geometric models utilize a distance metric, other metric space approaches are possible. For example, information retrieval systems often model documents as vectors in Euclidian space; representing the similarity between documents as the cosine of the angle between the vectors [205]. As an illustrative example, we summarize an approach for evaluating the similarity of publications, as presented by Bani-Ahmad, Ali Cakmak, and Ozsoyoglu in [22]. In their approach, a vocabulary *T* of atomic terms *t* is generated from a collection of publication documents. An individual document can then be represented as vector of real numbers $v = R^{|T|}$, where each element is described by a term. Accordingly, v_t is used to denote an element of v that corresponds to the term $t, t \in T$. The value of v_t is related to the importance of t in the document represented by v. Using the Term Frequency-Inverse Document Frequency (TF-IDF) weighting scheme, as described in [277], v_t is defined as:

$$v_t = log(TF_{v,t} + 1) \times log(IDF_t)$$

where $TF_{v,t}$ represents the number of times the term t occurs in the document represented by v; $IDF_t = N/n_t$; N is the total number of documents in the database; and n_t represents the total number of documents in the database that contain the term t. The cosine similarity between two documents with vectors v and w is then computed as:

$$\cos(v, w) = \left(\sum_{i=1}^{|T|} f(v_i) \cdot f(w_i)\right) \Big/ \left(\sum_{i=1}^{|T|} f(v_i)^2 \cdot f(w_i)^2\right)$$

where $f()$ is a damping function, which is either the square-root or the logarithm function. It should be noted that such approaches remove stop-words (e.g. an, the and then) from the document and then apply the Porter's algorithm [256] to stem the terms.

While distance and angle similarity operate within domains that can be well-represented as a metric space, *feature metrics* are useful for detecting similarities and differences between sets of events [331]. For example, Microsoft's SensCam [47] is a multi-sensor camera (worn around the neck) that records a continuous stream of images as a means of enabling so-called "lifelog recording". Because SensCams take approximately 3,000 images per day, the volume of image data is often a barrier for effective image retrieval. To help address this issue, SensCam logs a variety of context information along with the collected images. One novel SensCam technique is the detection of *Bluetooth familiarity* as a means of inferring the people present during a particular event. In this technique, SensCam identifies nearby Bluetooth devices (by MAC address) logs the duration of the encounter; linking the presence of Bluetooth devices to the ongoing image stream. To provide a basic similarity score, the set of devices present during an event are compared using a Jaccard coefficient scheme [205]. The similarity score is then computed by calculating the intersection of devices co-present during two events as:

$$J(A, B) = |A \cap B| / |A \cup B|$$

which results in a similarity score in the range [0,1], with values of 0 indicating no similarity and values of 1 indicating strong similarity. Related, the researchers suggest that the longer a device is

present, the more significant the device's owner may be to the event. Hence, the duration of detected Bluetooth signals is used to weight the significance of proximate devices during an event. The duration weight is calculated as:

$$DurationWeight = \frac{|X \cap Y| - \sum_{i=0}^{|X \cup Y|} DiffDur(X_i - Y_i)}{|X \cup Y|}$$

where *X*=Event 1, *Y*=Event 2, *i* = devices present in *both* events and $DiffDur = |Durration(X_i) - Durration(Y_i)|$. Next, using the Jaccard co-efficient and the $DurationWeight$ value, events can be segmented in several ways. First, devices can be sorted by familiarity, where increased significance is given to devices that are present in events relative to devices that are not. Second, devices can be weighted by both duration and familiarity, where the $DurationWeight$ is used to enhance the significance of devices that are present *longer* during an event. Finally, events can be segmented by inverse familiarity, where strangers and outliers can be used to enhance image retrieval in certain situations (e.g. during a chance meeting with an unfamiliar person).

Feature metrics are useful for determining similarity between complex objects that vary along a large number of dimensions; however, feature models may not adequately capture the semantics of the structured information within a comparison [187]. For example, the feature "red" may be part of the description of an automobile; however feature models may not be capable of discerning between automobiles with red wheels versus red body paint [125]. Hence, refinements of the feature-based model have been proposed. Common approaches include modeling features as fuzzy predicates within a contrast model (e.g. in computer vision) [281] and the LogOdds approach, which models the probability of set membership based on the presence of an object in related communities [311]. Finally, transformational models such Levenshtein distance are used to represent the similarity of objects by calculating the number of operations required to transform one object into another (e.g. sequences of text in information retrieval scenarios) [205].

Notably, in large-scale information retrieval systems, indexing becomes a critical aspect of finding similar information in high dimensional space [376]. In a single dimension, naïve search algorithms exploiting linear scan strategies exhibit query times of $\Theta(dn)$, where *d* represents the number of dimensions and *n* represents the number of search items. However, as dimensionality increases, naïve strategies often encounter the so-called *curse of dimensionality*, which is characterized by exponential increases in storage space or search time requirements [39]. For example, the nearest-neighbor problem has a solution of $O(d^{O(1)} \log n)$ query time, but requires roughly $n^{O(d)}$ space [296]. Hence, while such algorithms can be effective for small datasets, they become quickly intractable when applied to larger datasets and are often too slow for many time-sensitive applications. To improve indexing operations in multi-feature scenarios, a common optimization approach includes the use of *approximate* nearest neighbor algorithms (ANN), which trade query precision for speed improvements or memory savings [69]. Moreover, the speed of ANN algorithms can be further improved through the use of efficient index data structures such as R-Trees [134] and M-trees [68]. Note that such issues are discussed further during the presentation of Ocean's Persistence Framework in section 6.3.

4.3.4 Validating the Context Metadata Interface

In order to validate the Context Metadata abstraction described in section 4.3.2, we implemented several real-world Context Handlers within the Ocean Reference Implementation (RI). We began by identifying ten heterogeneous context information types as a means of exploring the applicability of the Context Metadata abstraction to diverse context domains. We selected context domains according to the following criteria: (1) the domain used non-proprietary or open data formats; (2) the domain was applicable to real-world context-aware scenarios; (3) the domain was amenable to the Aladin approach described in section 3.2.2; and, (4) context domain expertise could be extracted from existing development efforts. Related, we implemented several customizable comparison functions intended to allow non-experts to configure Context Handlers for a specific contextualization use-case. To illustrate how such comparison functions might be integrated into development and end-user tools, we also created a dedicated contextualization application, known as Ocean Studio, which provides forms-based Contextualized Resource creation and Context Handler configuration (see section 8.2).

To validate the Context Metadata abstraction using the Ocean Reference Implementation (RI), we began by creating an **`IContextMetadata`** interface that expresses the methods described in section 4.3.2. Additionally, we created a **`ContextHandlerBase`** class as a means of consolidating common functionality across multiple Context Handler types (e.g. configuration data). Next, we implemented several concrete Context Handler classes using the **`IContextMetadata`** interface and **`ContextHandlerBase`** class as foundations. Further, we designed a complimentary plug-in framework that allows the dynamic integration of Context Handlers at runtime as a means of supporting the community-based contribution approach described in section 5.4. An overview of the implemented Context Handlers, the **`IContextMetadata`** interface and the **`ContextHandlerBase`** class is shown below in Figure 28. (Note that implementation-specific methods may be included in the figure below; however, in the interest of clarity, these are not described.) Related, Table 8 below provides a brief description of each Context Handler.

Figure 28: Implemented Context Handlers, the IContextMetadata interface and the ContextHandlerBase class within the Ocean RI

Context Handler	Description
`GEOPointHandler`	Preliminary support for the OpenGIS Geography Markup Language (GML) Encoding Standard as presented in [237]. Comparison functions include **`GPSPointDistanceBoolean`** and **`GPSPointDistanceLinear`** (see section 4.4 for details). Geographic distance calculations are provided by the dinopolis gpstool package[17].
`WIFIPlaceLabStumblerHandler`	Preliminary support for the PlaceLab stumbler format as described in [66]. Supports the 802.11a/b standards. A supplemental **`RFSignalPropagation`** comparison function provides handler tuning based on a variation of the log-normal shadowing signal propagation model as described in [202]. Provides automatic refinement of comparison functions based on the **`VTTOntologyHandler`**.
`ISO639LanguageHandler`	Preliminary support of the ISO 639 Language standard[18].
`ISO8601DateTimeHandler`	Preliminary support for the ISO 8601 date and time formats as described in [324]. Time and data evaluation are provided by the open-source Joda Time software package[19].
`RFIDHandler`	Preliminary support for RFID strings in both Electronic Product Code (EPC) [98] and RAW formats.
`SecurityTokenHandler`	Preliminary support for secure hash tokens per the SHA-1 standard described in [219].
`TextSearchHandler`	An Ocean-provided Context Handler that supports contextualization of Resources based on textual metadata within the title and description elements of a Contextualized Resource. Support for textual search is provided through the Apache Lucene search engine[20].
`VTTOntologyHandler`	Preliminary support of the Ontology for Mobile Device Sensor-Based Context Awareness, as described in [184].
`QRCodeHandler`	Preliminary support of the Quick Response two-dimensional bar code format (QR Code) developed by Denso-Wave Corporation[21].

Table 8: Overview of Context Handlers implemented within the Ocean RI

[17] http://gpsmap.sourceforge.net/
[18] http://www.loc.gov/standards/iso639-2/
[19] http://joda-time.sourceforge.net/
[20] http://lucene.apache.org/java/docs/
[21] http://www.denso-wave.com/qrcode/index-e.html

4.3.5 The Contextualized Resource XML Schema and Ocean RI Validation

To provide an open mechanism for describing Contextualized Resources, a flexible XML representation format is now defined. While the next chapter elaborates in detail on how the following XML schema is used within the Ocean approach, we introduce it here to help support the Contextualized Resource example presented at the end of this chapter. The Contextualized Resource XML schema is shown in Figure 29.

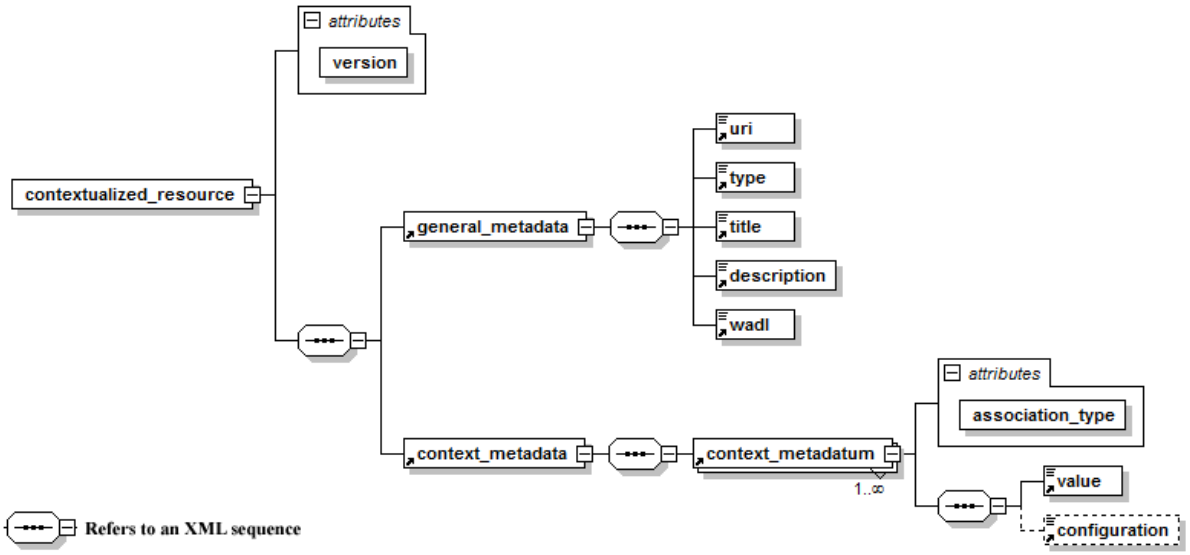


Figure 29: The Contextualized Resource XML schema

With reference to Figure 29, the Contextualized Resource XML scheme is defined as follows:

- The **`contextualized_resource`** root element provides a **`version`** attribute (used for finding a suitable parser) that includes a single **`general_metadata`** element and single **`context_metadata`** element.
- **`general_metadata`** provides Resource-specific information, including **`uri`** (as per RFC: 3986 [34]); **`type`** (e.g. MIME data types [327]); **`title`** (UTF-8 text); **`description`** (UTF-8 text); and an optional **`WADL`** document (as described in [137]).
- The **`context_metadata`** element includes one or more **`context_metadatum`** elements that have an **`association_type`** attribute with one of the following strings: **`required`**, **`excluded`** or **`optional`**.
- A **`context_metadatum`** element provides a **`value`** element that includes the native context data of the **`context_metadatum`**, plus an optional **`configuration`** element that may include domain-specific configuration data. Notably, configuration data is generally used to customize the comparison semantics of a given **`context_metadatum`**. Importantly, the information within **`context_metadatum`** sub-elements must be enclosed within an **`XML CDATA`** tag to allow for the inclusion of arbitrary configuration data (e.g. binary data structures and XML incompliant configuration strings).

With regards to the Contextualized Resource data model, Figure 30 shows the relevant classes and interfaces within the Ocean Reference Implementation (RI). For reference, the **`ContextualizedResource`** class provides an implementation of the Contextualized Resource data model described in section 4.3.1. Notably, both General and Context Metadata properties are provided by the class. Additionally, the **`IContextMetadata`** interface provides an implementation of the Context Metadata abstraction described in section 4.3.2. Finally, the **`Item`** interface is used to validate the Contextualized Resource in terms of the Resource personalization approach described in section 7.3. (Note that implementation-specific methods may be included in the figure below; however, in the interest of clarity, these are not described.)

Figure 30: The ContextualizedResource and IContextMetadata classes from the Ocean RI

4.4 A Contextualized Resource Example

To clarify the CR model and Context Metadata concepts introduced in this chapter, this section presents a simplified Contextualized Resource example. Importantly, the example is not intended to provide a complete overview of the Ocean approach; rather, it is intended to illustrate the foundational principles upon which Ocean is founded (upcoming chapters present the Ocean approach in detail). Hence, this section presents an intentionally simplified scenario whereby a Web-based calendar Resource is contextualized using Context Metadata consisting geo-location and time. In order to accommodate a broad range of application scenarios, the example CR references a Resource that provides a Representation in the well-known iCalendar data format (iCal) [81]. Briefly, iCal supports a variety of electronic calendaring functions (e.g. events, to-do and journal entry information) and is compatible with a broad range of applications. As per the CR model introduced in section 4.3, Resources are managed *outside* of the Ocean approach (i.e. iCal data may be created using tools such as Microsoft Outlook[22] and hosted on an organization's Web server). In this example, contextualized calendar data are used to support a simple conference scenario, whereby attendees receive time-sensitive event scheduling information when near the conference location. To develop this example scenario, we describe two related Context Handlers that were developed within the Ocean RI, including the `GEOPointHandler` and the `ISO8601DateTimeHandler`. Both Context Handler description sections provide an general overview of the related context domain, the associated comparison semantics and configuration options. The example concludes with a presentation of the example CR's UML diagram and associated XML configuration.

4.4.1 The GEOPointHandler

We now introduce the `GEOPointHandler` that was developed as a part of the Ocean RI. Briefly, the `GEOPointHandler` encapsulates the domain semantics of geo-location proximity based on key features of the OpenGIS Abstract Specification [236] and the OpenGIS Geography Markup Language (GML) Encoding Standard [237]. Based on these standards, the handler can be used to constrain the Discoverability Context of Resources to a geographic area that is represented by a defined center point described by latitude and longitude values. The handler supports instantiation based on native context data (NCD) in the Geography Markup Language simple profile format (GML simple) [237]; however, it could also be extended to support additional NCD formats such as NMEA 0183 [220] or the Microformat Geo standard [55]. (Note that we chose the GML simple specification initially because of its broad industry support and its ability to represent "variety of kinds of objects for describing geography including features, coordinate reference systems, geometry, topology, time, units of measure and generalized values" [237].)

To support contextualization and discovery within Ocean, the `GEOPointHandler` provides a domain-specific implementation of the Context Metadata interface described in section 4.3.3. In this example, we describe its practical implementation and several conceptual enhancements regarding persistence (see section 6.3). Accordingly, the `GEOPointHandler` implements the Context Metadata interface as presented in Table 9.

[22] http://www.microsoft.com/outlook/

Interface method	Implementation details
`setConfiguration`	Accepts XML configuration data based on the GML simple profile [237]. Additional configuration options include: **`GEOPointDistanceLinear`** and **`GEOPointDistanceBoolean`** comparison functions (described shortly).
`refineMetadata`	No implementation.
`compareTo`	Geo-location data are intended for storage within a 2-dimensional vector space model as per [183]. Intra-domain comparisons are represented by a Euclidean metric extension of the ComparisonRepresentation object.
`getNormalizedSimilarity`	Normalized similarity is computed by transforming the distances between points in 2-dimensional vector space using following equation: $NormalizedSimilarity = \frac{1}{1+EuclidianDistance}$

Table 9: The GEOPointHandler Context Metadata interface implementation

In its default configuration, the **`GEOPointHandler`** requires that two geo-locations represent exactly the same latitude and longitude values to provide a similarity of 1 (otherwise the similarity is determined to be 0). To improve the flexibility of the handler, we also developed two related comparison functions that allow Contextualizers to control how the handler performs similarity comparisons. The first comparison function, termed **`GEOPointDistanceLinear`**, allows a Contextualizer to define a circular area around a given geo-location point, whereby similarity is modeled as a linear falloff that is 1 at the center and fades to 0 at the perimeter. This comparison function represents a range query of the form *R(q,r)*, where *q* represents the center of the circle and *r* represents the radius of the search in meters [61]. Within the bounds of the circular area, similarity is represented by fractional numbers such as 0.8 or 0.5. The second function, termed **`GEOPointDistanceBoolean`**, allows a Contextualizer to define a circular area around a particular geo-location point that accommodates Boolean comparisons; yielding a similarity of 1 within the bounds of the circular area and 0 outside it. A visualization of both comparison functions is shown in Figure 31 (note that similarity values are normalized).

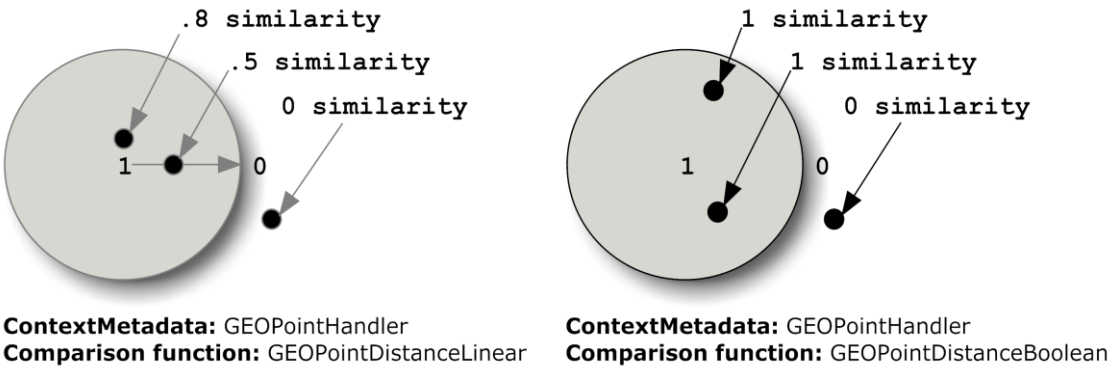


Figure 31: Visualization of two GEOPointHandler similarity comparison functions

Finally, based on the XML specification introduced in section 4.3.5, an example snippet of the **GEOPointHandler** configuration XML is provided in Figure 32.

```
...
<context_datum association_type="required">
 <value><![CDATA[
 <gml:Point>
  <gml:pos>53.874532,10.684183</gml:pos>
 </gml:Point>]]>
 </value>
 <configuration><![CDATA[
 <config>
  <comparison_function>GEOPointDistanceLinear</comparison_function>
  <max_meters>500</max_meters>
 </config>]]>
 </configuration>
</context_datum>
...
```

Figure 32: Example GEOPointHandler XML configuration

With regards to Figure 32, we note the following regarding the **GEOPointHandler** configuration data:

1. The Context Metadata is associated with the CR model according to the **required** type.
2. The value and configuration elements are enclosed with **XML CDATA** tags; allowing for arbitrary data to be passed in during instantiation.
3. The **value** element includes native GML Point data that is understood by the handler's **setConfiguration** implementation.
4. The configuration element includes a **GEOPointDistanceLinear** comparison function that is configured for a discoverability range of 500 meters around the handler's location.

4.4.2 The ISO8601DateTimeHandler

We now introduce the **ISO8601DateTimeHandler** Context Metadata implementation that was developed as a part of the Ocean RI. Briefly, the **ISO8601DateTimeHandler** encapsulates the semantics of date, time and interval information based on key features of the ISO8601 standard, as described in [324]. We selected the ISO8601 standard because of its non-proprietary nature and text-based representation format that is independent of communication medium. Notably, the ISO8601 standard can be used to express a variety of dates, times, time-zones and durations, intervals, repeating intervals, etc. Additionally, several safety and compactness features are built into the standard. For example, year representations are constrained to four digit representations to avoid millennium confusion (e.g. "YYYY") and time expressions utilize the twenty-four hour clock for compactness. Examples of the ISO8601 standard are shown in Figure 33.

B.1.3 Date and time of day

Combinations of calendar date and local time

Basic format	Extended format	Explanation
19850412T101530	1985-04-12T10:15:30	Complete

Combinations of ordinal date and UTC of day

Basic format	Extended format	Explanation
1985102T235030Z	1985-102T23:50:30Z	Complete

Combinations of week date and local time

Basic format	Extended format	Explanation
1985W155T235030	1985-W15-5T23:50:30	Complete

B.1.4 Time interval

Defined by start and end

A time interval starting at 20 minutes and 50 seconds past 23 hours on 12 April 1985 and ending at 30 minutes past 10 hours on 25 June 1985

Basic format	Extended format
19850412T232050/19850625T103000	1985-04-12T23:20:50/1985-06-25T10:30:00

Figure 33: ISO8601 format examples (from [324])

To support contextualization and discovery in Ocean, the `ISO8601DateTimeHandler` provides a domain-specific implementation of the Context Metadata interface described in section 4.3.3. Accordingly, its interface methods have been implemented as shown in Table 10:

Interface method	Implementation details
`setConfiguration`	Accepts XML configuration data based on the ISO8601 format.
`refineMetadata`	No implementation.
`compareTo`	The `GEOPointHandler` is intended for object-based persistence optimized for the ISO8601 standard.
`getNormalizedSimilarity`	Dependent on the configuration of the handler. Please see the description below for details. (Note that comparisons are provided for DateTime and Intervals only.)

Table 10: The ISO8601DateTimeHandler Context Metadata interface implementation

In terms of comparison semantics, the `ISO8601DateTimeHandler` is designed to encapsulate the `DateTime` and `Interval` specifications expressed by the ISO8601 standard; hence, comparisons are dependent on the configuration of the handler. For example, if an `ISO8601DateTimeHandler` is configured to support the date and time (i.e. `DateTime`), the comparison function computes the

similarity between two Context Metadata instances DT_1 and DT_2 based on the difference between the **`DateTime`** values (in minutes). The **`getNormalizedSimilarity`** value is computed as follows:

$$similarity = \frac{1}{1 + |DT_1 - DT_2|}$$

Note that **`DateTime`** similarity calculations are always based on the time resolution of the handler associated with the Contextualized Resource. For example, if such a handler has been configured to represent the year **`"2008"`** (without additional expressions for days, hours, etc.) normalized similarity comparisons will result in a value of 1 provided that candidates are configured with a **`"2008"`** year (regardless of day, hour, minute and second configurations). Likewise, an associated handler that has been configured using an extended format string (e.g. **`"2008-08-27T11:15:30")`** will result in a fractional normalized similarity values depending on the configuration of the candidate Context Metadata.

The **`ISO8601DateTimeHandler`** behaves differently if configured as an **`Interval`**. To support normalized similarity computation, an Interval-based **`ISO8601DateTimeHandler`** provides a Boolean comparison function that computes whether a given **`DateTime`** falls within the handler's **`Interval`**. If it does, **`getNormalizedSimilarity`** returns 1 (otherwise it returns 0). Note that Interval-based **`ISO8601DateTimeHandlers`** are only comparable to standard **`DateTime`** values (i.e. **`Interval`** to **`Interval`** comparisons are not currently supported due to time constraints). It should be noted that the default comparison semantics could be further enhanced through the use of a supplemental comparison functions.

Based on the XML specification introduced in section 4.3.5, an example snippet of the **`ISO8601DateTimeHandler`** configuration XML is provided in Figure 34:

```
...
<context_datum association_type="required">
 <value><![CDATA[2008-06-14T10:00:00/2008-06-14T18:00:00]]></value>
 <configuration/>
</context_datum>
...
```

Figure 34: Example ISO8601DateTimeHandler XML configuration snippet

With regards to Figure 34, we note the following regarding the Context Handler's configuration:

1. The **`context_datum`** is associated to the CR model with the **`required`** association type in order to constrain discovery to **`DateTime`** NCD that fall within the handler's **`Interval`**.
2. The value and configuration elements are enclosed with XML **`CDATA`** tags; allowing for arbitrary native data to be passed in during instantiation.
3. The value element includes a native ISO8601 extended format **`Interval`** string, which is understood by the handler's **`setConfiguration`** implementation.
4. The configuration element is not used by this handler, meaning that default comparison semantics are used.

4.4.3 Bringing it All Together

Using the **`GEOPointHandler`** and **`ISO8601DateTimeHandler`** Context Metadata implementations described in the previous sections, we now present a completed Contextualized Resource (CR). As previously introduced, the example CR is used to contextualize calendar data (in the iCal format) as a means of supporting a simple conference scenario, whereby attendees receive time-sensitive event scheduling information when near the conference location. With reference to the CR model introduced in section 4.3, the General Metadata for the example CR are presented in Table 11.

General Metadata	Value
`Type`	**`text/calendar`**
`URI`	**`http://isnm.de/2008/cal`**
`Title`	"ISNM Open House 2008"
`Description`	"Event calendar for the 2008 ISNM Open House"
`WADL`	None

Table 11: Example General Metadata

In addition to the General Metadata presented above, two Context Handlers are used to constrain the Discoverability Context of the iCal Resource. The Context Metadata associated with the example Resource are represented by the **`GEOPointHandler`** and **`ISO8601DateTimeHandler`** implementations previously introduced. An overview of the associated configuration specifications are shown in Table 12.

Context Metadata	Description
`ISO8601DateTimeHandler`	Configured as an ISO8601 interval encompassing the following time span: June 14th, 2008 (10:00) until June 14th, 2008 (18:00).
`GEOPointHandler`	Configured with a physical location of the Media Docks facility in Luebeck, Germany (Lat: 53.874532/ Lon:10.684183), which is represented using the GML simple profile. In addition, the handler is configured with a **`GEOPointDistanceLinear`** compare function with a maximum discovery range of 200 meters.

Table 12: Example Context Metadata configuration specifications

To form the completed CR, the General Metadata and Context Metadata presented above are constrained in their relationship according to the CR model described in section 4.3. The resultant Contextualized Resource data structure is shown in Figure 35.

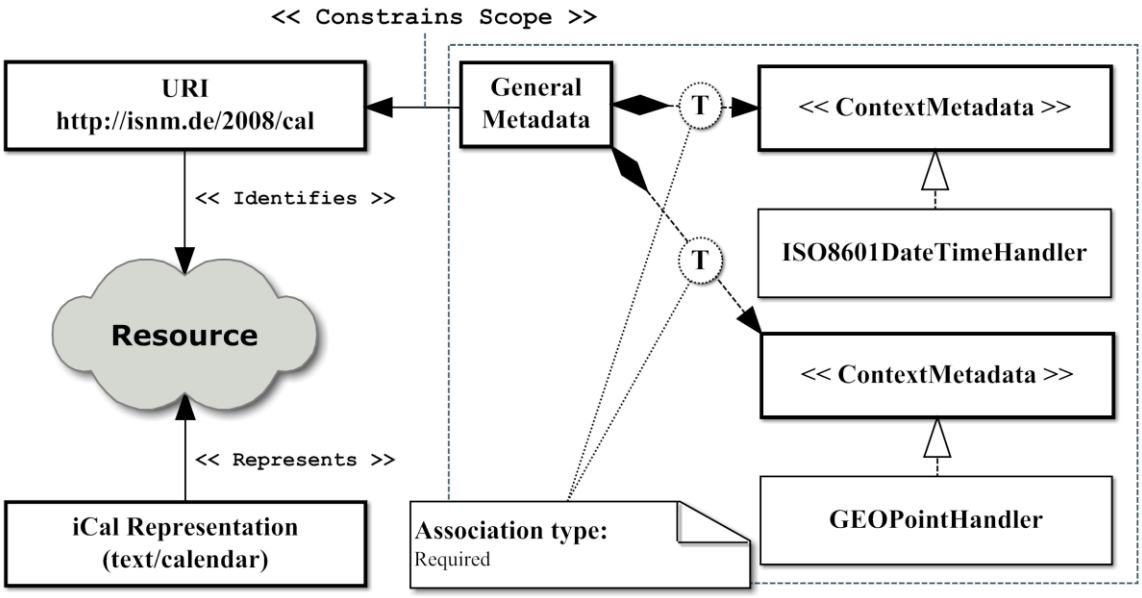


Figure 35: Diagram of the example Contextualized Resource

To realize the CR model shown above within the Ocean Registry, the Contextualized Resource specification is encoded according to the XML schema defined in section 4.3.5, as shown in Figure 36.

```
<?xml version="1.0" encoding="UTF-8"?>
<contextualized_resource version="1.0">
  <general_metadata>
    <type><![CDATA[text/calendar]]></type>
    <uri><![CDATA[http://isnm.de/2008/cal]]></uri>
    <title>2008 ISNM Open House</title>
    <description>Event calendar for the 2008 ISNM Open House</description>
    <wadl/>
  </general_metadata>
  <context_metadata>
    <context_metadatum association_type=”required”>
      <value><![CDATA[2008-06-14T10:00:00/2008-06-14T18:00:00]]></value>
      <configuration/>
    </context_metadatum>
    <context_metadatum association_type=”required”>
      <value><![CDATA[<gml:Point><gml:pos>53.874532,10.684183</gml:pos>
        </gml:Point>]]>
      </value>
      <configuration><![CDATA[<comparison_function><type>GEOPointDistanceLinear
                              </type><max_meters>200</max_meters>
                              </comparison_function]]>
      </configuration>
    </context_metadatum>
  </context_metadata>
</contextualized_resource>
```

Figure 36: Example Contextualized Resource configuration XML

4.5 Chapter Summary

This chapter presented Ocean's Contextualized Resource abstraction. It began by providing background and related work specific to context-mediation in RESTful distributed systems. It first described context-mediation in conventional Web architecture and then introduced several related approaches employed by current context-aware systems. It was noted that while many Web-centric component discovery techniques support features of the Ocean approach, several important challenges persist. These challenges were identified as: support for native context data; support for multi-feature similarity search; context mismatch handling; and techniques for overcoming information overload. Based on these limitations, the Contextualized Resource (CR) abstraction was introduced as a mechanism for supplementing the Web's conventional Resource model with an extensible set of Ocean Metadata intended to constrain a Resource's global scope through the establishment of a Discoverability Context. This section presented the Contextualized Resource data model, which included both General and Context Metadata entities. Briefly, General Metadata refers to Resource-specific information such as URI, title, description and an optional WADL document. In contrast, Context Metadata are intended to constrain the Discoverability Context of Web Resources through the encapsulation of the syntax and semantics of a given context domain. In this regard, the Context Metadata interface was presented as a means supporting Ocean's CR persistence and Discovery Frameworks described in Chapter 6. This section also included a theoretical discussion of similarity modeling techniques and their applicability to the Context Metadata interface. Next, the Contextualized Resource XML schema was presented and validation within the Ocean RI was discussed. The chapter concluded with a presentation of an example Contextualized Resource.

Chapter 5
Towards Web-scale Context-aware Computing

5.1 Introduction

The Contextualized Resource model (CR model) introduced in the last chapter provides an extensible and expressive mechanism for constraining the Discoverability Context of conventional Web Resources. However, the CR model is not sufficient for realizing the Ocean approach derived in section 3.5 as it provides only a *means* of differentiating between Web Resources based on native context data. This chapter builds upon the Contextualized Resource abstraction by defining Ocean's application model and deriving a complimentary support infrastructure that is further elaborated in subsequent chapters. First, section 5.2 details the overall Ocean approach. Related, section 5.2.1 discusses how the client-centric mashup model aligns well with Ocean's design principles and approach constraints introduced in sections 3.4.1 and 3.4.2 respectively. Next, section 5.2.2 defines the Ocean's application model by extending the client-centric mashup style to support in-situ, context-mediated Resource discovery, selection and composition. Based on the Ocean application model, we then define the necessary computing infrastructure required to support wide-area creation and discovery of Contextualized Resources. Related, section 5.3 describes Ocean's Contextualized Resource registry (Ocean Registry) and its related software architecture. Section 5.3.2 introduces the Ocean Registry's Resource Management API. Next, section 5.3.3 provides details regarding the instantiation of Contextualized Resources as a foundation for the CR persistence and discovery techniques presented in Chapter 6. The final sections of this chapter describe two preliminary community-centric techniques designed to support the overall Ocean approach. First, section 5.4 describes Ocean's Context Handler contribution model, which is based on an adaptation of the Java Community Process. Finally, section 5.5 introduces Ocean's open contextualization model, which is intended to promote the large-scale Resource contextualization through collaborative annotation.

5.2 The Ocean Application Model

Based on the large-scale context-aware computing challenges presented in section 3.2, we derived Ocean's Web-centric context-aware computing approach in section 3.5. As presented in section 3.4.2, Ocean adheres to the following design constraints: Aladin-based context acquisition and modeling; Internet-based network communications; Web-centric middleware; and REST-based component interoperation. Based on these constraints, Ocean inherits many of the capabilities and limitations of conventional Web architecture. Recall that in Web-based hypermedia, Resource content (e.g. Web page text) and transactional metadata (e.g. HTTP headers) provide the context mediation necessary for users to discover, select and compose Resources on-demand (see section 4.2). However, conventional Web architecture does not inherently support context-mediation based on real-world context information such as location, proximate devices and activity. Hence, Chapter 4 introduced the Contextualized Resource as a mechanism for constraining the Discoverability Context of conventional Web Resources based on extensible Ocean Metadata. While the Contextualized Resource abstraction provides a foundation for Web-centric context-aware computing, Ocean's wide-area focus presupposes a client-centric application model capable of supporting cross-domain component

interoperation using conventional Web technologies. Towards this end, the following section presents the client-centric mashup style as a promising application model that is well-aligned with Ocean's design constraints and approach derivation. Moreover, the following section describes an extension of the client-side mashup style as a means of supporting wide-area context-mediated component discovery and composition.

5.2.1 Introduction to the Client-side Mashup Style

As described in section 3.2.6, modern Web architecture has been increasingly used to support complex cross-domain component interoperation scenarios. Notably, hybrid web applications (or *mashups*) have emerged as a popular approach for combining Web-based information and computation in ways that add value beyond the individual application constituents [206]. For example, the popular mashup HousingMaps.com[23] combines house sale listings from CraigsList.com[24] with graphical map data from Google Maps[25] to provide a unique Web application that allows users to search for houses according to geographic location, price and number of rooms. Based on similar techniques, "Developers are now using various Web APIs to create a plethora of mashups to solve all types of problems, from esoteric mashups that record the location and availability of rare gaming consoles to those that create Sudoku games from Flickr photos" [206].

In contrast to enterprise-centric distributed computing techniques, which often require highly skilled developers, significant technical infrastructure and months of development time, mashups can be created by a broad range of developers using data-centric Web architecture and lightweight development tools such as Yahoo Pipes[26] and Marmite [361]. Component integration is typically coordinated by exploiting a shared surrogate key (e.g. address) as a means of joining disparate datasets along a semantically known dimension [170]. Examples include union, join and implicit searching along dimensions such as location, time, keyword, UPC/ISBN primary keys, etc. The underlying data-sources may involve screen scraping (where a mashup parses non-structured human-readable content), the inherent semantics of well-known Resource representation formats and specialized Web-based APIs, which expose an application's information and computation.

Based on data-centric interoperation, the adoption of mashup techniques and technologies has been rapidly increasing. Indeed, ProgrammableWeb.com[27], a comprehensive online compendium of established and emerging mashups, listed more than 3,478 mashups and 1,013 related Web service APIs as of November 2008. The popularity of mashups is often attributed to the simplicity and flexibility of RESTful Web architecture [170, 334, 336]. Although REST principles are not required for creating mashups, they align well with the data-centric, cross-domain interoperation style common to many implementations [336]. As a consequence, mashups are often based on a variety of Web-based information such as news feeds, mapping data and Web-based APIs. Due to the variety of available data sources and related application scenarios, several functional categories of mashups have emerged. These categories were described in a recent survey by Hong and Wong [362] as:

[23] http://www.housingmaps.com/
[24] http://www.craigslist.org/
[25] http://maps.google.com/
[26] http://pipes.yahoo.com/pipes/
[27] http://www.programmableweb.com/

1. **Aggregation:** Refers to mashups that aggregate data from several external sources; providing a summarized or application-specific view of the data. As a prototypical example, EveryBlock.com[28] uses a shared address key to aggregate geo-tagged images from Flickr[29], reviews of local businesses from Yelp[30], proximate CraigsList advertisements and civic information such as crime statistics.
2. **Alternate UI & In-situ Use:** Refers to mashups that provide an alternate user-interface or other application-specific adaptation of a Website or set of Web-based data. For example, Leaflets[31] provides specially adapted, low-bandwidth versions of popular Websites for use on the Apple iPhone[32].
3. **Personalization:** Refers to mashups that personalize the functionality of Websites based on user provided information. For example, the YES! mashup[33] calculates a year-end summary of sales and tax liabilities for a person's eBay account.
4. **Focused View of Data:** Refers to mashups that index and categorizes the content of large datasets according to a particular organizational scheme. For example, Youtorials[34] is a user-submitted compilation of tutorials that can be found on the YouTube[35] video service.
5. **Real-time Monitoring:** Refers to mashups that provide continually updated aggregations of rapidly changing and potentially large data sets. For example, Flickr real-time[36] dynamically updates an evolving term list of recently added tags from the Flickr photo sharing website.

The technologies underlying the majority of mashup applications can be divided into five main categories, as shown in Figure 37. As described in [128], these categories can be briefly summarized as follows: First, the *foundations* of many mashups are browser-based execution environments and HTTP-based network communications. As such, the associated *presentation* technologies leverage browser-based HTML/XHTML and CSS engines such as Webkit[37]. To support dynamic *interactivity* within browser-based applications, technologies such as JavaScript and Ajax are used to update user interface elements, provide network communications and provide application logic during runtime. Dynamic interactivity typically involves interactions with *Web service APIs*, using interoperation approaches such as XMLHTTP request, XML-RPC, SOAP and REST. Finally, dynamic interoperation in mashups typically involve the exchange of *data* between the mashup and Web service API using representation formats based on XML, Atom [328], JSON [76] and KML[38].

[28] http://www.everyblock.com/
[29] http://flickr.com/
[30] http://www.yelp.com/
[31] http://getleaflets.com/
[32] http://www.apple.com/iphone/
[33] http://yes.sagefire.com/
[34] http://youtorials.com/
[35] http://youtube.com
[36] http://www.pimpampum.net/rt/
[37] http://webkit.org/
[38] http://code.google.com/apis/kml/documentation/

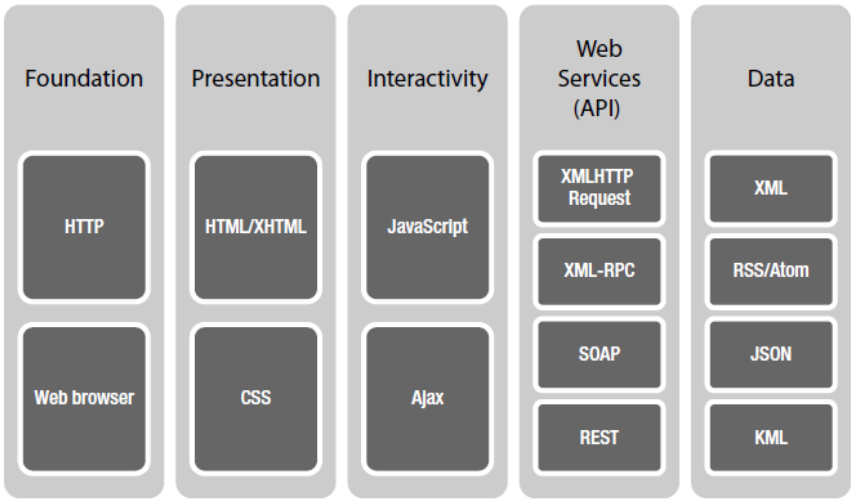


Figure 37: Key mashup technologies (from [128])

Currently, the *client-centric mashup style* represents one of the most common architectural approaches for devising hybrid Web applications [243]. In client-centric mashups (CS mashups), client devices serve as platforms for the orchestration and aggregation of distributed data and computation. CS mashups generally load presentation artifacts, application logic and related JavaScript from a remote server using conventional HTTP. Typically, the client-side application exists as a set of conventional Web pages that accommodate user interaction via JavaScript, which is deployed to clients using the code-on-demand (COD) style [105]. Briefly, the COD style allows clients to be extended with additional functionality through the deployment and execution of remote software. Data aggregation and related user interface updates are generally accomplished using asynchronous `XmlHttpRequests` that dynamically update the mashup's document object model (DOM); allowing Web pages to provide updated information without reloading. An overview of the CS mashup style is shown in Figure 38.

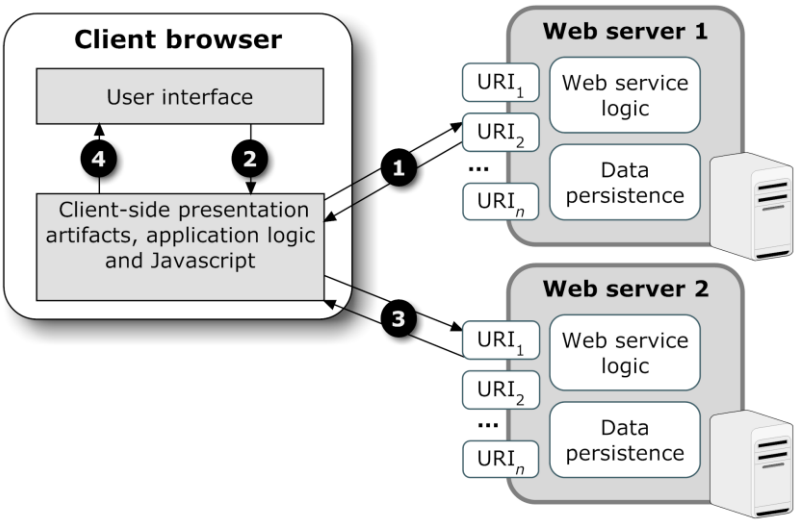


Figure 38: Overview of the client-side mashup style (adapted from [243])

With reference to Figure 38, the CS mashup style is described below:

1. The user navigates to an origin server using an appropriate Web agent (typically a Web browser). The mashup's presentation artifacts, application logic and JavaScript are downloaded to the client using conventional HTTP mechanisms and then rendered by the browser.
2. A DOM event (e.g. page loaded), user initiated event (e.g. mouse click) or other application-specific event triggers a request for Resource composition (e.g. data aggregation). Application logic and related JavaScript are typically used to request data or computation from a Web-based data source. In Web browser scenarios, the client typically uses an **`XmlHttpRequest`** to acquire external data.
3. At the remote server, Resource-level state management is provided and an appropriately encoded Resource Representation is sent to the client per HTTP content negotiation [104]. A lightweight data interchange format such as JSON [76] may be used to reduce communication overhead and interaction latency.
4. The client-side mashup receives the Resource's Representation from the remote server and applies data transformation processing as per local application logic. Application state is maintained at the client. Interface updates are typically applied dynamically using the DOM as to avoid page refresh.

Importantly, the CS mashup style is increasingly recognized as supporting *cross-domain* component interoperation through the application of RESTful principles [334, 336]. In particular, Resource-based problem modeling and extensible Representations help reduce client/server data-coupling as they are not bound to a specific underlying protocol [334]. Moreover, client-based application state and an increased URI surface area makes it easier for clients to extract "interesting" data from RESTful applications in ways perhaps not originally envisioned by the Resource's developer [266]. Indeed, the large number of URI entry points common to RESTful applications stands in stark contrast to process centric approaches (e.g. SOAP Web services), which typically present only a single service endpoint URI and require a shared understanding of domain-specific method semantics, sequencing issues and server-side state management requirements (see section 3.2.8). Hence, the CS mashup style has been increasingly used to support *situational applications*, where component-based constituents are rapidly assembled to solve an immediate business need [206]. Although current situational applications are generally assembled statically in days or weeks, we suggest that the addition of context-aware component discovery can provide an effective foundation for the *dynamic* assembly of such application types.

Based on the above observations, we suggest that the CS mashup style provides a foundation for extending the Web's conventional hypermedia model to support the Ocean approach. While current mashups often leverage Web browsers to support presentation and application logic execution, the CS mashup style does not preclude adaptation to other Web agent types or execution platforms. Furthermore, the CS mashup style aligns with Ocean's Internet-based communications and Web-centric middleware constrains described in section 3.4.2. Notably, CS mashups are based upon common Web technologies and are well supported by existing Internet infrastructure. Finally, the CS

mashup style aligns with Ocean's REST component interoperation constraint. Indeed, the majority of current mashup applications exploit RESTful Web APIs and other Resource types as a means of providing data-centric component interoperation. However, as discussed in section 4.2, conventional Web architecture does not inherently support context-mediated component discovery and selection. To address these issues, the next section describes Ocean's extension of the CS mashup style.

5.2.2 Contextualizing the Client-side Mashup Style

In this section, we extend the CS mashup style with context-mediation techniques intended to support the emergence of context-aware Web applications capable of dynamically composing contextually-relevant Resources at runtime. As discussed in the last section, current mashups are typically developed to address relatively static application scenarios and do not support context-mediated component discovery and selection beyond conventional hypermedia techniques (i.e. inline and transactional context information). In contrast, context-aware applications must often be rapidly adapted with relevant application constituents in-situ and may only exist in a particular configuration while a context situation remains valid. To support the development of Web-centric context-aware systems, we now define a Resource discovery and selection approach based on the Contextualized Resource model described in section 4.3.

As previously introduced, the Contextualized Resource extends the conventional Web Resource model with supplemental General and Context Metadata intended to constrain the *Discoverability Context* of the underlying Web Resource. Recall from Definition 1 that a Discoverability Context is defined as *the set of contextual criteria that must be fulfilled before a Resource is considered relevant to the interaction between a user and an Ocean application, including the user and application themselves.* In conventional hypermedia applications, inline and transactional context provide the necessary semantics for users to discover, select and compose Resources (i.e. dereference a link using a browser User agent).

In Ocean we introduce an extensible context-aware component registry, called the Ocean Registry, which maintains a shared data store of Contextualized Resources (described in section 5.3). To perform Resource discovery, autonomous Ocean applications acquire and model native context data (NCD) locally using the Aladin approach presented in section 3.2.2. Next, Ocean applications query the Ocean Registry using their locally modeled NCD as query terms (see section 6.4.1.1). The Ocean Registry uses the incoming discovery query to search for similar Contextualized Resources from within its shared data store (see section 6.4). Contextually relevant Resources are returned to the requesting client as a ranked and sorted list of Descriptive Metadata (see section 6.4.1.2). Notably, discovered Resources adhere to the REST architectural style as presented in [105]; hence, subsequent component interoperation is performed as per the CS mashup style discussed in the last section. An overview of the Ocean application model is shown in Figure 39.

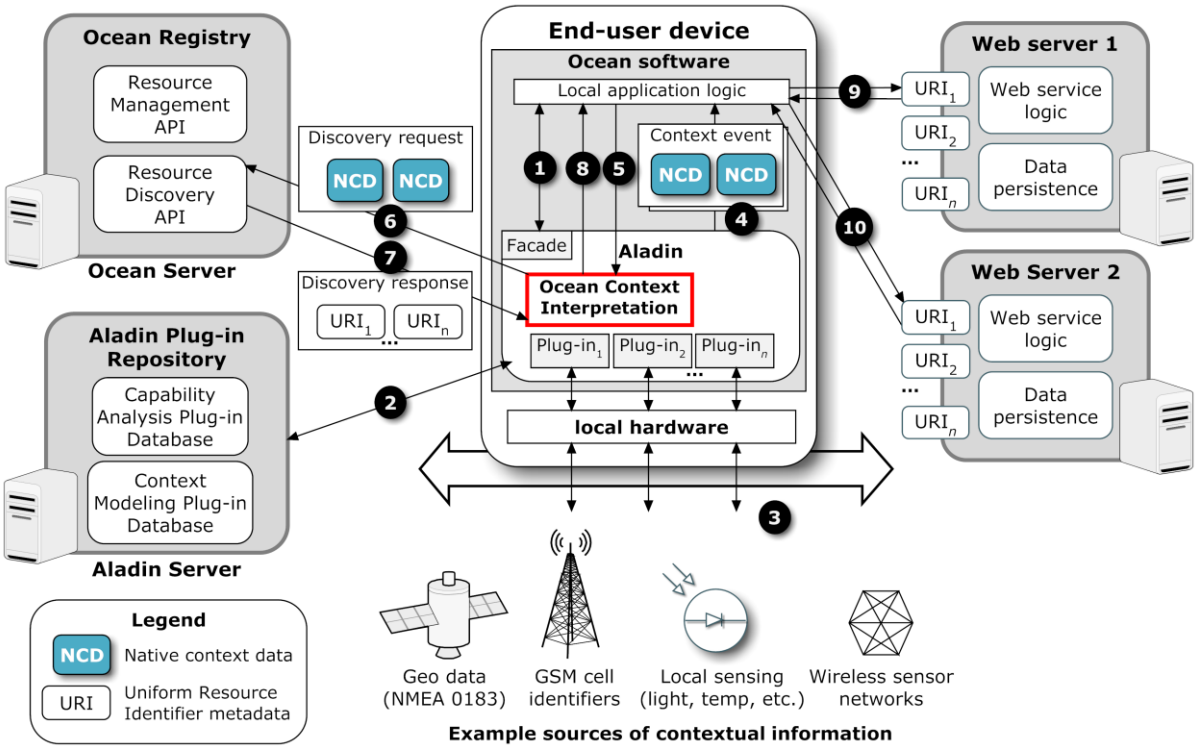


Figure 39: Overview of the Ocean application model

With reference to Figure 39, the Ocean application model is defined as follows:

1. An end-user device executes domain-specific software that adopts the Ocean approach. In the implementation shown above, Ocean extends the basic architectural framework of Aladin; hence, interaction between local application logic and the Ocean subsystem is provided by Aladin's façade API and related event mechanism (see section 3.2.2). However, applications may communicate directly with the Ocean Registry if necessary.

2. Aladin analyses the host device and environment capabilities; dynamically downloading and installing appropriate context acquisition and modeling plug-ins as necessary during runtime.

3. Context acquisition and modeling is provided using local device capabilities. Acquired low-level contextual information is quantized and formatted into native context data (NCD) that express the syntax and semantics of a given context domain.

4. NCD are passed to the Ocean application where they may be used to guide adaptation without the aid of Ocean context interpretation (provided that the local application logic is capable of parsing and understanding the incoming NCD).

5. Local application logic may utilize Ocean Context Interpretation (OCI) to perform a Resource Discovery query using the Ocean Registry. To Discovery queries, Ocean applications form Discovery Requests using an XML-based search grammar that comprises a set of search parameters and includes local NCD as query terms (discussed shortly).

6. The local OCI passes Discovery Requests to the Ocean Registry, which maintains a shared data store of persisted Contextualized Resources.

a. The shared data store is supported by the Context Metadata abstraction which encapsulates the syntax and semantics of a given context domain (see section 4.3.2).

b. The Ocean Registry provides an open contribution process whereby external Context Experts develop, contribute and manage Context Metadata implementations termed Context Handlers (see section 5.4).

c. The Ocean Registry provides an open contribution process whereby external Contextualizers contextualize conventional Web Resources using arbitrary Ocean Metadata (see section 5.5).

d. The Ocean Registry provides integrated Persistence and Discovery Frameworks that allows Contextualized Resources to be efficiently stored, indexed and queried (see Chapter 6).

7. Once contextually-relevant Resources have been discovered by the Ocean Registry, the results are marshaled into a Discovery Response, which contains a set of Descriptive Metadata (e.g. similarity score, personalization score, title, description, domain, etc.) The Discovery Response is returned to the calling client.

8. The Ocean OCI unmarshals the Discovery Response and passes the results to the Ocean application. The Ocean application uses domain-specific application logic and appropriate user interaction to select appropriate Resources for runtime composition.

9. The host application employs the REST architectural style to interoperate with selected Web Resources according to the CS mashup style described in section 5.2.1.

10. As additional contextually-relevant Resources are discovered, the Ocean application adapts its runtime configuration and composed constituents dynamically. (Note that arbitrary Web Resource may be discovered and integrated in this manner).

As described above, Ocean applications acquire NCD from their environments and then query the Ocean Registry to discover contextually relevant Resources using NCD as query terms. In this way, Ocean Developers may formulate Discovery Requests without requiring detailed knowledge of each modeled NCD. To support effective Resource Discovery, the Ocean Registry utilizes a multi-feature similarity search framework that allows Contextualized Resources to be discovered and ranked based on their similarity to the query terms contained within incoming Discovery Requests (see section 6.4). Related, the Discovery Framework also provides context-aware information filtering, automatic query expansion and Resource personalization to help improve search results for a broad range of device types and application scenarios (see Chapter 7). The next section describes the Ocean Registry in detail.

5.3 The Contextualized Resource Registry

As previously introduced, component registries and context-aware component registries are common approaches for mediating dynamic binding and interaction between the loosely-coupled application constituents in a distributed system [29]. However, as discussed in section 4.2, current component registries face significant challenges in large-scale context-aware scenarios. First, many current techniques rely on intermediary context formats; requiring that locally acquired native context data be converted into a predetermined intermediary format for use with a given discovery protocol (e.g. mapping native geospatial data into a specific data structure using an OWL ontology). Problematically, such techniques errantly presuppose significant domain-expertise on the part of the application developer and may impose significant processing overhead on mobile devices. Second, although recent approaches have explored Resource mediation based on context information, existing techniques are limited to a predetermined set of context metadata, do not address wide-area scenarios and cannot be extended by external context domain experts. Third, existing approaches are generally highly application specific and do not support a variety of application domains. Fourth, existing techniques do not address context mismatch scenarios, whereby the query terms provided within a Discovery Request do not sufficiently match the metadata used to contextualize Web Resources (for details see section 7.2). Finally, existing techniques are not well-suited for overcoming information overload in complex environments; requiring that developers (or users) manually filter and select appropriate Resources for runtime composition (for details see section 7.3). Based on these limitations, we have developed the Contextualized Resource Registry (Ocean Registry), which is introduced next.

5.3.1 Architecture Overview

The Ocean Registry is designed to mediate interactions between Ocean Core Developers, Context Experts, Contextualizers and Ocean Developers (see section 3.6). Accordingly, the architecture of the Ocean Registry is organized around a repository of Contextualized Resources (in the form of Ocean Metadata) that are persisted within a shared data store. This shared data store is intended to facilitate autonomous interactions in the following ways: First, the Ocean Registry is designed, developed and managed by Core Developers who are responsible for its software architecture and its Contextualized Resource data store. Second, the Ocean Registry allows Context Experts to contribute and manage Context Handlers, which implement the Context Metadata interface described in section 4.3.2 (using the Context Handler API). As previously introduced, Context Metadata are used to constrain the Discoverability Context of Web Resources according to the CR model presented in section 4.3. Third, the Ocean Registry allows Contextualizers to create, update and delete Contextualized Resources from the shared data store. Finally, the Ocean Registry allows Ocean Developers to discover, resolve and rate Contextualized Resources within their applications using a flexible discovery protocol that allows locally modeled NCD to be included as query terms (see section 6.4). In response to Resource Discovery Requests, the Ocean Registry discovers and returns contextually-relevant Resources from its shared data store. Finally, Ocean applications are then free to select and compose discovered Resources in-situ according to the Ocean application model described in section 5.2.2. An overview of the Ocean Registry is provided in Figure 40.

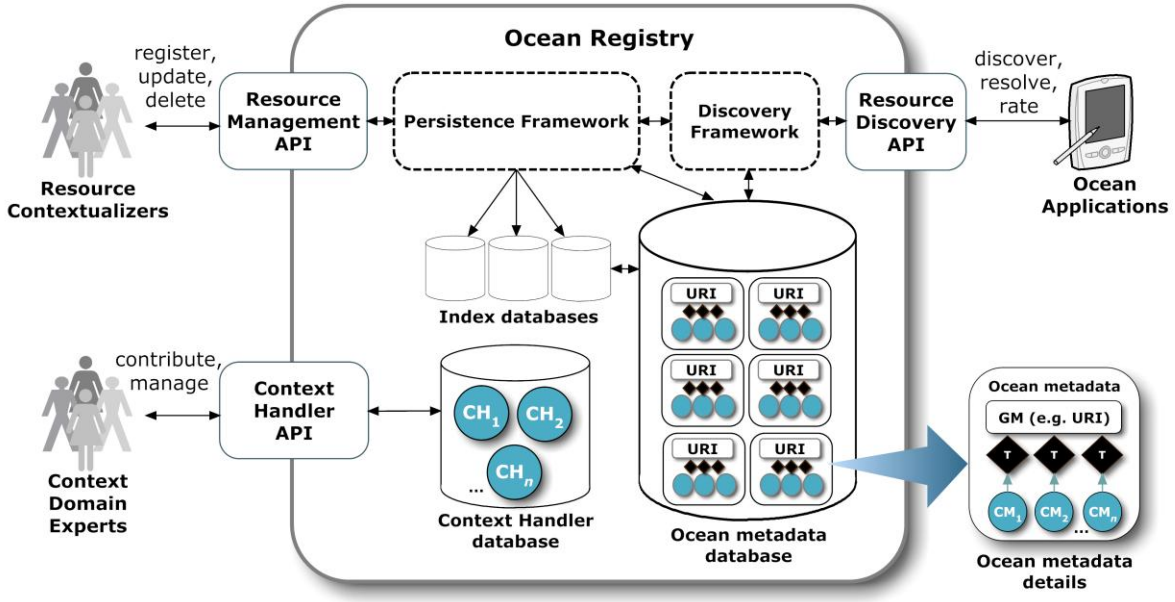


Figure 40: Overview of the Ocean Registry

The Ocean Registry is designed according to the blackboard architectural pattern [45]. Briefly, Blackboard architectures are useful for distributed systems where several independent entities work collectively on a common data structure. According to [45], "The Blackboard architectural pattern is useful for problems for which no deterministic solution strategies are known. In Blackboard several specialized subsystems assemble their knowledge to build a [possibly] partial or approximate solution." These specialized subsystems (e.g. distributed programs) typically have no explicit interdependencies (aside from the blackboard itself) and no predetermined activation sequence. Rather, the blackboard itself provides internal state management and coordinates interactions with distributed entities. This data-directed control approach makes "experimentation with different algorithms possible, and allows experimentally-derived heuristics to control processing" [266]. According to [45], the benefits of a blackboard architecture include flexibility through recombination; efficiency through parallel processing; and autonomous contribution.

To support Resource contextualization and discovery, the Ocean Registry provides an integrated Persistence Framework and complimentary Discovery Framework (see Chapter 6). Briefly, the Persistence Framework allows Contextualized Resources to be efficiently stored and indexed for rapid retrieval according to domain-specific indexing and similarity modeling techniques. As Ocean applications perform Discovery Requests, the Ocean Registry's Discovery Framework is used to rapidly discover contextually-relevant Resources based on the similarity of incoming query terms to the Ocean Metadata within the shared data store. Ocean similarity search operates in conjunction with the previously mentioned Persistence Framework; allowing native context data to be compared to persisted Context Metadata using the Context Metadata abstraction introduced in section 4.3.2.

To validate our approach, the Ocean Registry architecture shown above was implemented within the Ocean Reference Implementation (Ocean RI). Notably, the Ocean RI's implementation is organized around several key classes that provide unified access to request processing, query handling, personalization services, context management, plug-in handling and database access.

Related classes in the Ocean RI include: the **`OceanManager`**, which handles Ocean Registry state management; the **`PluginManager`**, which handles dynamic integration and instantiation of Context Handlers; the **`PersistanceFramework`**, which handles storage and indexing of Contextualized Resources (see section 6.3); the **`RequestFactory`**, which handles incoming Discovery Requests in combination with the **`ContextManager`** and **`QueryProcessor`** (see section 6.4.2); and the **`RecommendationEngine`** and **`AssociationDiscoveryFramework`**, which provide community-based discovery query enhancement (see Chapter 7). An overview of these classes is shown in Figure 41. (Note that implementation-specific methods may be included in the figure below; however, in the interest of clarity, these are not described.)

Figure 41: Overview of key Ocean Registry classes from the Ocean RI

5.3.2 The Resource Management API

Contextualizers contribute and manage Contextualized Resources using the Ocean Registry's Resource Management API (Resource API), which provides methods for creating, retrieving, updating and deleting Ocean Metadata from the Ocean Registry's shared data store. As described in section 4.3, Contextualized Resources are based on the CR model, which is used to constrain the Discoverability Context of conventional Web-based Resources. Notably, the CR model is agnostic regarding the Resources that might be contextualized and any associated Ocean Metadata; hence, Contextualizers are free to describe the Discoverability Context of Resources according to their individual requirements using the XML schema presented in section 4.3.5. The Ocean RI provides an implementation of the Resource API based on the RESTLet Web Services Framework[39]. Figure 42 provides an overview of the related classes within the Ocean Registry implementation (Note that

[39] http://www.restlet.org/

implementation-specific methods may be included in the figure below; however, in the interest of clarity, these are not described.)

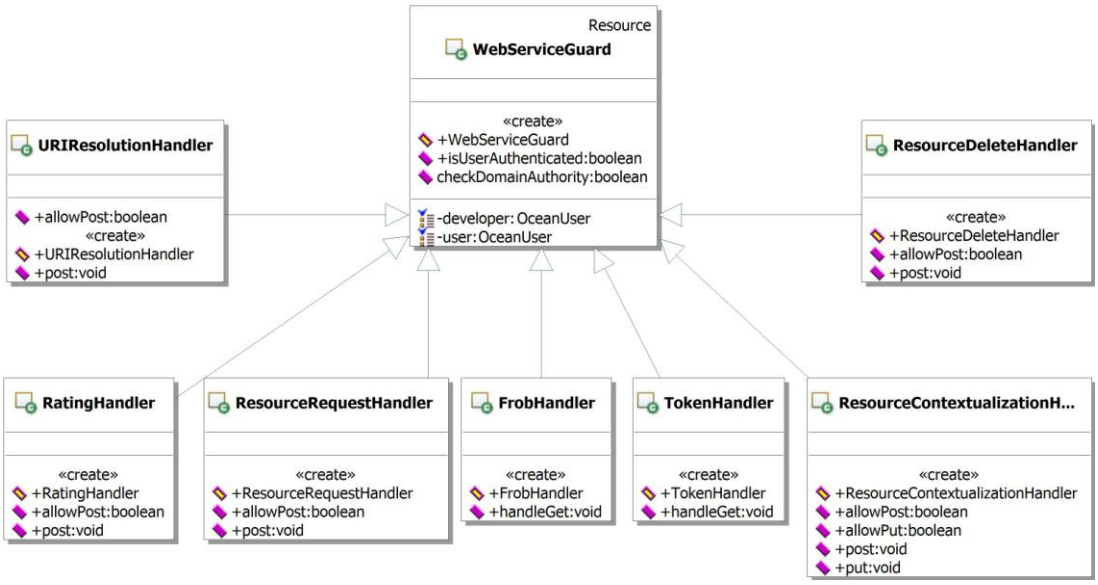


Figure 42: Web Service handler classes within the Ocean RI

To manage Ocean Metadata, Contextualizers are required to authenticate with the Ocean Registry. Notably, identity and authentication are core aspects of the Ocean Registry's security approach and are used in all access control decisions. Contextualizers are issued private API keys that must be included with all method calls made using the Ocean Registry. In the Ocean RI, all interactions with the Resource API are mediated by the **`WebServiceGuard`**, which throttles incoming connections (if needed) and performs authentication in combination with the **`FrobHander`** and **`TokenHandler`** using either HTTP Basic Authentication [104] or signed signature digest as per [7]. Requests arriving at the Ocean Registry without authentication receive a **`401 Unauthorized`** HTTP status code in reply.

Once authenticated, Contextualizers use the Resource API to create Contextualized Resources using the CR model XML description presented in section 4.3.5. Importantly, it remains the responsibility of the Contextualizer to properly describe the Discoverability Context of a given Resource using appropriate Ocean Metadata. Note that the creation of valid Contextualized Resource XML descriptions may be facilitated by software such as Web-based and stand-alone applications (see section 8.2 for details). Once an XML description has been created, it is sent to the Ocean Registry using **`HTTP POST`**. The URI for Contextualized Resource creation is **`http://oceanframework.org/create/`**. In the Ocean RI, create requests are directed to the **`ResourceContextualizationHandler`**, which provides request unmarshalling and validation. Once the request is validated, its data are unmarshaled and sent to the Request Factory, which attempts to instantiate a Contextualized Resource object (see section 5.3.3). If the creation process fails – e.g. no Context Handler could be found for a given context description – the method returns a **`400 Bad Request`** HTTP status code, along with an error message. If the creation process succeeds, the method returns a **`200 OK`** HTTP status code along with the globally unique identifier (GUID) assigned to the newly created Contextualized Resource. For security purposes, Contextualized Resources are

automatically associated to the Contextualizer developer account used to create it (e.g. to prevent unauthorized updates or deletions).

Contextualizers update existing Contextualized Resources by creating a new CR XML description and sending it to the Ocean Registry using **`HTTP PUT`**. The URI for Contextualized Resource updating is: **`http://oceanframework.org/update/{guid}`**. In the Ocean RI, update requests are directed to the **`ResourceContextualizationHandler`**, which provides request unmarshalling and validation. Once the request is validated, its data are unmarshaled and sent to the Request Factory, which attempts to instantiate a Contextualized Resource object. If a valid CR object can be instantiated, the new object overwrites the existing Contextualized Resource; however, its global identity is maintained. As in the creation process, updating existing Contextualized Resources may be facilitated by software tools. If the update process fails – e.g. no Context Handler could be found for a given context description – the method returns a **`400 Bad Request`** HTTP status code along with an error message. If the update process succeeds, the method returns a **`200 OK`** HTTP status code.

Contextualizers delete Contextualized Resources by calling the Resource API using **`HTTP DELETE`** using the following URI: **`http://oceanframework.org/delete/{guid}`**. In the Ocean RI, delete requests are directed to the **`ResourceDeleteHandler`**, which provides request handling. The deletion of a Contextualized Resource permanently removes its reference from the Ocean Registry's shared data store and makes it unavailable for subsequent Discovery Requests. If the delete process fails – e.g. no Contextualized Resources exists for a given GUID – the method returns a **`400 Bad Request`** HTTP status code along with an error message. If the delete process succeeds, the method returns a **`200 OK`** HTTP status code.

5.3.3 Contextualized Resource Instantiation

A factory approach [114] is used to instantiate CR objects according to the incoming XML descriptions provided by Contextualizers. To begin the instantiation process, a multi-threaded factory object is created and assigned to a request by the Ocean Registry. The factory unmarshals the XML data contained within the request and selects an appropriate instantiation strategy. The instantiation strategy is based on a modified version of the chain-of-responsibility pattern [114], which employs generalized command objects to perform a domain-specific actions. Specifically, contributed Context Handlers are utilized as command objects by way of a static **`create`** method that each Context Handler must provide. Similar to the Context Metadata's **`setConfiguration`** method described in section 4.3.2 (which accepts a generic object as an argument and uses domain-specific mechanisms to parse native description values), the **`create`** method accepts a generic initialization object that is used to prepare internal state. A diagram of the Contextualized Resource instantiation process is shown in Figure 43.

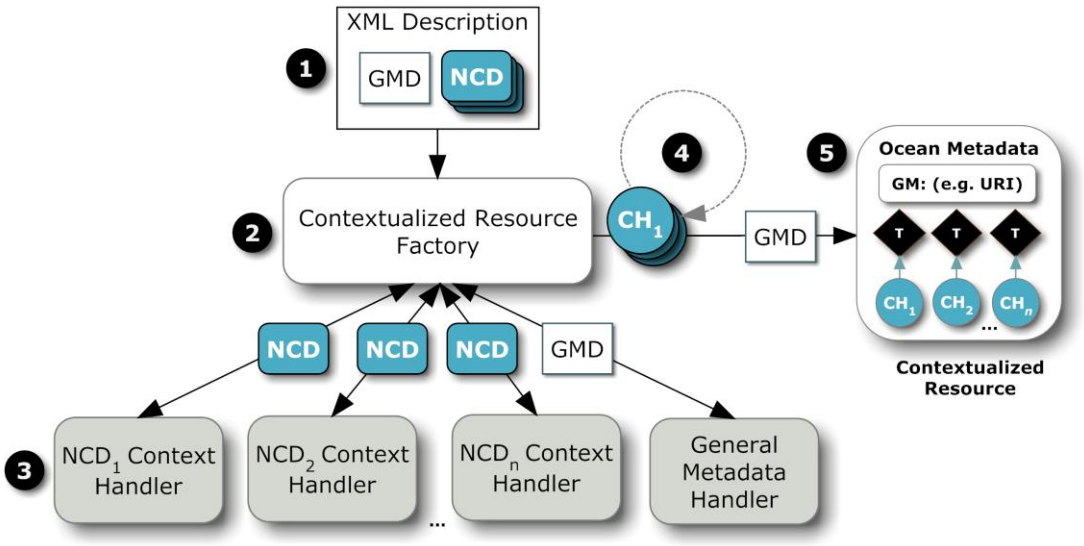


Figure 43: The Contextualized Resource instantiation process

Regarding Figure 43, the Contextualized Resource instantiation process proceeds as follows: First, a Contextualized Resource is described by a Contextualizer using the CR model XML schema presented in section 4.3.5 (XML Description). The completed XML Description is sent to the Ocean Registry's create or update API methods. Second, a multi-threaded factory object is created to handle the request (note that thread pooling techniques may be used to manage factory objects). Third, the assigned factory attempts to instantiate a compatible Context Handler for each NCD contained within the XML Description. Instantiation attempts are handled by calling each registered Context Handler within the Ocean Registry using its static **`create`** method and passing in the given NCD as the instantiation object. During this process, if a particular Context Handler cannot parse the provided NCD, it raises an exception; otherwise it returns a preliminarily configured Context Handler object. If an exception is raised, the factory recursively selects another Context Handler and repeats the instantiation process until suitable Context Handler is instantiated or the process fails (i.e. no compatible Context Handler could be found). This command process is repeated for each NCD (plus the GMD) contained within the XML Description. (To help improve efficiency, identifiable aspects of the request NCD may be cached as a means of providing mappings to compatible Context Handlers for future requests). Fourth, once all metadata are instantiated as Context Handler objects, each handler is provided all of the other handlers contained within the request using its **`refineMetadata`** method (see section 4.3.2). During refinement, Context Handlers may adapt their configuration to accommodate the presence of specific colleagues. Fifth, once the refinement process is complete, the factory creates a Contextualized Resource comprised of Ocean Metadata (i.e. General and Context Metadata).

5.4 Community-based Context Handler Contribution

A key challenge facing the Ocean approach is the contribution of a large set of Context Handlers for use in creating and discovering Contextualized Resources. As introduced in section 5.2.2, the effectiveness of the Contextualized Resource abstraction depends on the availability of diverse Context Handlers that encapsulate the syntax and semantics of complex context domains. As discussed in section 4.3.3, the development of Context Handlers is often highly complex and domain-specific. In this regard, the complexity inherent in many context domains *requires* participation by external Context Experts (see section 2.3.3). To support the dynamic extension of the Ocean Registry with additional Context Handlers, we introduced the Context Metadata abstraction (see section 4.3.2) and the associated Context Handler API (see section 5.3). As described in section 3.6.2, Context Experts may contribute Context Handlers based on a variety of motivations. As Context Handlers represent critical functional components of the Ocean Registry, implementations must not be allowed to adversely affect the performance of the overall registry architecture. Moreover, as Context Handlers are utilized by non-experts (i.e. Contextualizers) during CR creation and management, their intent, functionality and configuration options must be clearly described. Based on these requirements, we now propose a preliminary Context Handler contribution approach based on the Java Community Process (JCP) [167].

5.4.1 Overview of the Java Community Process

The JCP was established in 1998 to help guide the development and evolution of Java technologies. The JCP defined a formalized process whereby interested parties may propose new specifications and technologies for the Java platform [167]. Similar in spirit to the Context Metadata abstraction, the JCP suggests that "the best way to produce a technology specification is to gather a group of industry experts who have a deep understanding of the technology in question and then have a strong technical lead work with that group to create a first draft" [166]. Currently, there are over 300 JCP-developed technology specifications, called Java Specification Reviews (JSRs). Examples of notable JSRs include the Java API for XML Processing (JSR 5); Java Database Connectivity (JSR 221); and the Scalable 2D Vector Graphics API for J2ME (JSR 226). The diversity of current JSRs attest to the JCP's ability to attract external domain experts, build community consensus and promote the widespread adoption of Java technologies. A timeline of the JCP process is shown in Figure 44.

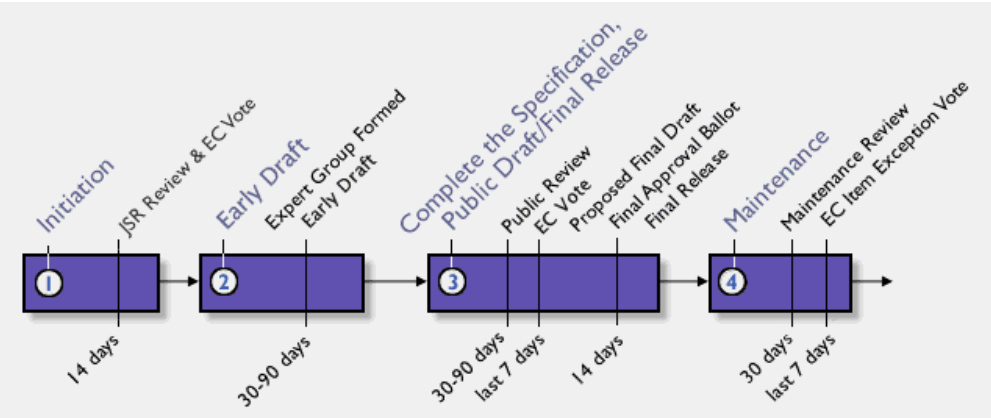


Figure 44: Timeline of the Java Community Process (from [166])

With reference to Figure 44, the contribution of new Java technologies follows the following four-phase process: In phase 1 (*Initiation*), any interested party may propose a new JSR specification or a revision of an existing specification. Once proposed, the Executive Committee (EC) reviews and votes on the JSR proposal. If approved, the JSR enters phase 2 (*Early Draft*), where an expert group is formed to write the initial specification. During this phase, JCP members (plus the general public) are allowed to comment on the draft. Next, the draft is revised according to the collected comments; resulting in an Early Draft specification. Once the Early Draft specification is produced, the JSR enters phase 3 (*Complete the Specification, Public Draft/Final Release*), where a combination of expert-group contribution, public review and EC voting result in the development of a Proposed Final Draft. If approved, the Proposed Final Draft is supplemented by the development of a Reference Implementation (RI) – intended to demonstrate that the JSR can indeed be implemented – and a Technology Compatibility Kit (TCK) – developed to test the impact of the JSR on existing Java technology and related public APIs. Once the RI and TCK are complete, a final round of member and public reviews culminates in a vote on the Final Release. If the Final Release is adopted, the JSR enters phase 4 (*Maintenance*), where a process of ongoing review and updates keep the specification current.

5.4.2 Towards Community-based Context Handler Contribution

We suggest that the JCP process introduced above provides a suitable conceptual framework for the controlled contribution of Context Handlers within the Ocean Registry. In particular, the JCP's combination of staged expert-group development, community and public reviews and executive committee oversight align well with the requirements of the Ocean Registry. Notably, the JCP's inclusion of member and public reviews provides a mechanism whereby problematic and structural issues can be identified and resolved in an open forum. In addition, broad participation in the review process can be used to generate consensus throughout the community. According to the JCP, consensus around the form and content of proposed JSRs is built by "using an iterative review process that allows an ever-widening audience to review and comment on the document" [167]. We suggest that a similar process could be integrated into Ocean, whereby Context Handler specifications are reviewed and commented before subsequent revision and executive committee voting. Recall that the JSP's emphasis on testing and verification are used to ensure that new Java technologies do not adversely affect existing Java platform. Similarly, we suggest that Ocean should adopt similar RI and TCK requirements as a means of ensuring such compatibility within the Ocean Registry. In particular, the Context Handler contribution process must carefully guard existing Context Handlers from potentially detrimental effects of new implementations. Furthermore, the contribution process must guard against functionality fragmentation such as duplication of Context Handler functionality or inconsistent implementations (e.g. support of the same NCD by different handlers). In this regard, Executive Committee oversight should be retained during the contribution process at key junctures. We have illustrated how the JCP process might be adapted for use in Ocean in Figure 45.

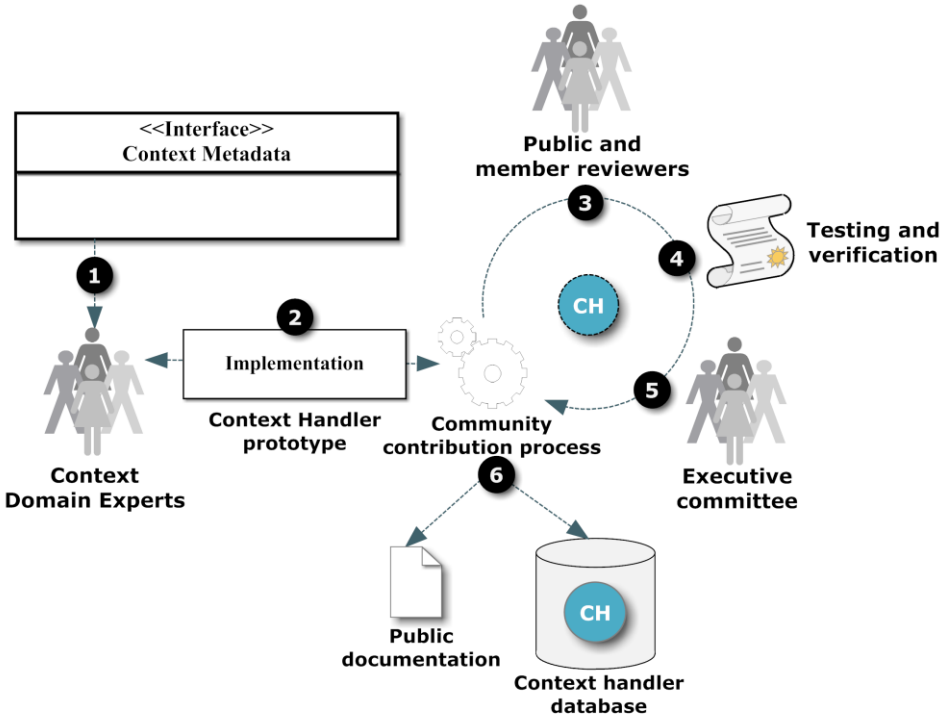


Figure 45: Preliminary community-based Context Handler contribution process

With reference to Figure 45, the preliminary Context Handler contribution process is defined as follows:

1. Context Experts review the Context Metadata interface descriptions, sample code and associated documentation. Context Experts prepare a request to develop a new Context Handler, which is submitted to the Ocean Registry's Executive Committee (EC) for approval.

2. If the request is approved, the Context Experts prepare an early Context Handler prototype implementation and related documentation. The prototype encapsulates the syntax and semantics of a given context domain as described in section 4.3.2. In addition, Context Experts may provide advanced configuration mechanisms (e.g. comparison functions), which allow non-experts to control Discoverability Context formation (see section 4.3.3).

3. The Context Handler prototype and related documentation enter the Ocean Community Process, which may include several rounds of development and review. During the process, public and member reviewers are able to comment on the Context Handler prototype. Oversight from the Executive Committee is integrated into the development process at key junctures. The cycle of prototype updates and community reviews continues until the prototype reaches a proposed final prototype (as decided by a vote of the EC.) If approved, the Context Experts prepare a reference implementation (RI) and related Technology Compatibility Kit (TCK), which are used to ensure proper operation of the Context Handler within the Ocean Registry.

4. Using the Context Handler's RI and TCK, the proposed final prototype undergoes a testing and verification phase, whereby the prototype is checked for compatibility with the existing Ocean infrastructure. If the proposed final prototype passes the testing and verification phases, a final Context Handler implementation is produced along with related documentation.
5. Once the final Context Handler implementation and documentation is complete, an additional round of public and community comments is allowed. The EC then rejects or accepts the final Context Handler release by ballet.
6. If adopted, the final Context Handler release is registered within the Ocean Registry and made available for use by Contextualizers and Ocean applications. Additionally, the Context Handler's official documentation package is made publically available.

The contribution process described above is preliminarily supported by the Ocean Reference Implementation (RI). Notably, we have developed an integrated plug-in framework that allows externally developed Context Handlers to be dynamically loaded into the Ocean Registry at runtime. The plug-in framework is managed by a **`PluginManager`**, which dynamically loads, caches and garbage-collects Context Handler implementations on-demand. Registered Context Handler plug-ins are described by **`PluginInfo`** objects and managed by **`PluginHolder`** objects, which are used to dynamically instantiate Context Handlers according to the request handling needs of the Ocean Registry. Further, the **`PluginManager`** insures that contributed Context Handlers properly implement the Context Metadata abstraction (i.e. the **`IContextMetadata`** interface) and adhere to appropriate security considerations. An overview of the **`PluginManager`** and related classes from the Ocean RI is shown in Figure 46. (Note that implementation-specific methods may be included in the figure below; however, in the interest of clarity, these are not described.)

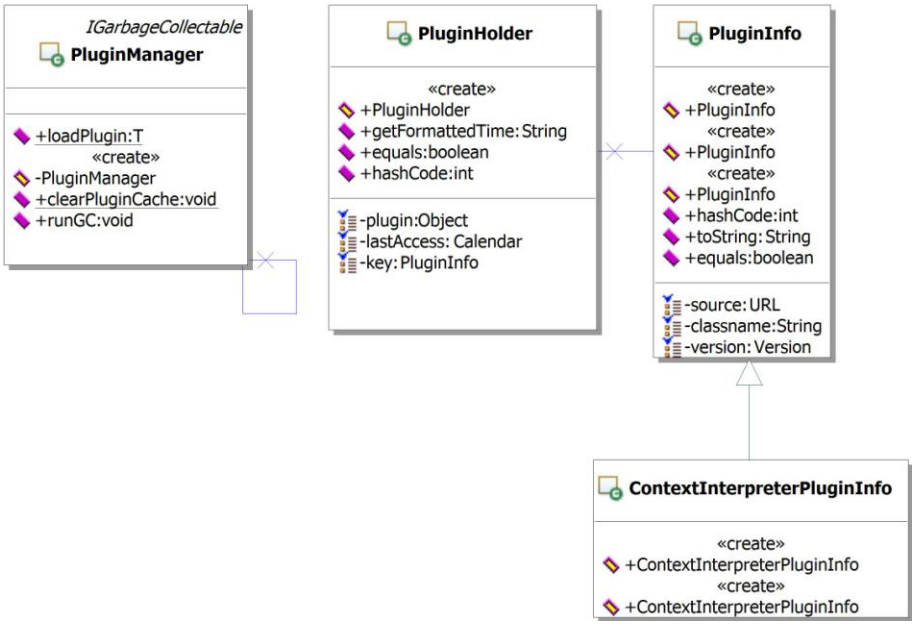


Figure 46: Overview of the PluginManager and related classes from the Ocean RI

5.5 Community-based Contextualized Resource Contribution

Given a sufficiently large set of contributed Context Handlers, a second key challenge facing the Ocean approach is the generation of a pool of Contextualized Resources that is large enough to support a broad range of Ocean applications. As described in section 4.3, Contextualized Resources express the Discoverability Context of conventional Web Resources by way of supplemental Ocean Metadata, which are stored separately from Resources within the Ocean Registry. As described in section 4.2, this associative metadata approach allows for the independent evolution of both Resources and Context Metadata by providing a separation of concerns between Resource providers, Contextualizers and Resource consumers; effectively allowing Ocean to integrate existing Resources without requiring changes to existing Web architecture. However, in order to overcome component sparsity in real-world environments, the Ocean application model presupposes a vast collection of Contextualized Resources. Given the tremendous volume of information and computation present on the Web, the creation and maintenance of such a collection represents a significant challenge. Recently, several community-based approaches have shown considerable promise in addressing large-scale contribution and classification tasks. In this regard, this section describes our preliminary community-based contribution approach that leverages collaborative annotation as a mechanism for promoting large-scale Contextualized Resource contribution.

5.5.1 Overview of Collaborative Annotation

The success of modern Web architecture has resulted in an explosion of Web-based information and computation; however, the scale, decentralization and dynamics of the Web present serious challenges for discovering information and computation that meet the requirements of users [205]. To address this issue, Web search and related information retrieval research has produced increasingly effective techniques for organizing and searching for Web content [205, 380]. Through the development of distributed indexing techniques [253], first generation search engines were capable of indexing large portions of the Web; however, the relatively unstructured nature of hypermedia Resources often resulted in low search effectiveness [205]. Hence, early contributors in Web search (e.g. Yahoo!) augmented full-text search offerings with the creation of Website taxonomies, whereby Resources are categorized into browseable classification hierarchies. Problematically, taxonomy-based approaches required significant human effort for performing the requisite editorial process. More recently, improvements such as large-scale distributed indexing [84], improved search algorithms [205], and the development of Resource ranking techniques (e.g. PageRank [246]) have made modern search engines invaluable tools for navigating the deluge of relatively unstructured Web content.

Based on the success of the human-centric Web, the semantic Web movement has endeavored to develop technologies that allow machine-based processing of Web-based information and computation [373]. Towards this end, Semantic Web techniques are used to augment the diverse and relatively unstructured nature of hypermedia with structured machine-readable metadata [367]. A primary approach in this regard is the development of formalized ontologies [377] that encapsulate a given knowledge domain and support reasoning models based on structured markup schemes such as RDF [367] and OWL [91]. As structured data is not an inherent aspect of conventional Web architecture, various approaches for annotating conventional Web resources with semantic metadata

have been explored. In several recent approaches, annotations are created through the automatic analysis of Web page content and structure using a combination of machine-learning algorithms and natural language processing (NLP) techniques [165]. However, while automated techniques have been gaining in popularity, their relative immaturity often necessitates manual human intervention using toolkits such as Protégé [230] and CREAM [138] or semi-automated disambiguation algorithms [309]. Notably, it is recognized that manual and semi-automatic classification techniques require human users who are familiar (or even expert) with a given knowledge domain and committed to annotating significant quantities of Resources [205]. Hence, while semantic Web approaches have found success in limited scenarios (e.g. bioinformatics [17] and knowledge management [309]) they are largely incapable of accommodating Web-scale environments due to a fundamental lack of semantic content [205].

Recently, *collaborative annotation* has emerged as a community-based approach for generating semantic annotations (often called tags) in large-scale scenarios. In this approach, *tags* have been defined as "an important subclass of annotations that comprise simple, unstructured labels or keywords assigned to digital resources to describe and classify the digital resource" [153]. Collaborative annotation systems rely on a community of users that create and freely associate tags with Resources such as Web pages, image data, and video content. The resultant tag collection provides a semi-structured, community-generated semantic vocabulary that is commonly known as a *folksonomy* [153]. In most cases, collaborative annotation is facilitated by an *open contribution model*, whereby tags can be freely defined and contributed by any member of the community [123].

Recent studies have shown that users are often willing to manually annotate Resources (such as photos) in order to help make them more discoverable [8]. For example, the popular photo sharing Website Flickr[40] uses collaborative annotation to help organize over 52 million publicly available images [302]. To achieve massive scalability, Flickr employs an open contribution model that allows the Flickr community to categorize photographs according to a multitude of preferences and perspectives. As illustrated in an example from [8], "a Flickr photo of La Sagrada Familia - a massive Roman Catholic basilica under construction in Barcelona - is described by its owner using the tags Sagrada Familia, and Barcelona. Using the collective knowledge that resides in Flickr community on this particular topic one can extend the description of the photo with the tags: Gaudi, Spain, Catalunya, architecture, and church. This extension provides a richer semantical description of the photo and can be used to retrieve the photo for a larger range of keyword queries."

Recently, collaborative annotation has become a central aspect of many large-scale Web applications, including blogging systems, digital library systems, and a variety of prominent Websites (for a detailed overview, see [123]). Within these diverse applications, a variety of techniques have been explored to help users create and maintain effective folksonomies. Common quality control techniques include image processing and machine-learning [199]; tag clouds to users help visualize popular keywords [304]; and various tag recommendation strategies, including co-occurrence detection, stability-promotion, descriptiveness-promotion and rank-promotion [302]. However, despite the emergence of such quality control techniques, recent studies have shown that a significant degree of inconsistency, contradiction and inaccuracy exists within many folksonomy-based systems

[40] http://flickr.com/

[94]. Common tagging problems include misspellings, confusing punctuation, non-descriptiveness and erroneousness [153]. Proposed solutions to these issues generally include constructing hybrid system architectures that combine freeform tagging with structured vocabularies. A common approach in this regard is an *ontology-directed-folksonomy*, where tags from a formalized ontology are suggested, but users retain the ability to define freeform tags [153, 337].

5.5.2 Towards Collaborative Contextualized Resource Contribution

In Ocean, we suggest that large-scale Contextualized Resource contribution can be facilitated by the application of collaborative annotation techniques. As described in section 4.3.2, the foundation of the CR model is a set of configurable Context Metadata which are associated to Web Resources as a means of describing their Discoverability Contexts. In this regard, Ocean Context Metadata can be understood as a controlled, yet configurable annotation vocabulary that supports machine-based processing and configuration options. For example, a Context Handler encapsulating the semantics of geo-location can be understood as roughly analogous to semantic metadata elements such as Dublin Core's DCMI Point scheme [75]; however, unlike Dublin Core metadata, which is itself an intermediary format, Ocean's metadata model supports component discovery based on native context data (NCD) such as raw NMEA sentences [220], GML encoded data [237], microformats [55], etc. Moreover, the Ocean Metadata model can be dynamically extended to support additional NCD formats as they become available.

In Ocean we adopt an open Contextualized Resource contribution model based on a modified version of the ontology-directed-folksonomy approach previously introduced. Specifically, we extend the basic contribution architecture introduced in section 5.3.2 to allow *any* Contextualizer to contextualize *any* Resource with *any* combination of Ocean Metadata. The open contribution architecture is facilitated by the Ocean Registry's Resource API (Resource API) described in section 5.3.2. Recall that interactions with the Resource API are accomplished using the Ocean Registry's Web services in combination with the Contextualized Resource XML schema described in section 4.3.5. Although interaction with the Resource API is straightforward, it is not generally intended for direct Contextualizer interaction. Rather, the Resource API is designed to support the emergence of a variety of software tools and Web-applications, which handle the underlying Web service interaction and provide simplified access to Context Metadata configuration. For example, Ocean's Resource API could be coupled with a photo sharing Website offering automatic Resource contextualization as part of its service offerings; making the user's photos discoverable by the community of Ocean applications.

In the Ocean Registry, the Context Metadata vocabulary is only limited to the available Context Handlers that have been previously contributed (see section 5.4). Unlike many semantic Web approaches, which require the encoding of metadata within Resource representations, Ocean's associative metadata model allows Context Metadata to exist independently. This approach allows Contextualized Resources to be created without Resource alteration (e.g. updating all Web-page headers with Dublin Core metadata). Moreover, the use of the associative model allows anyone (not only Resource owners) the ability to contextualize Resources; aligning with the best practices of many open contribution models [302].

In addition to the basic contribution model, Contextualizers may also establish *domain authority* over a given domain or sub-domain by demonstrating administrative access (e.g. by updating a given Web Resource with an Ocean-specified HTML metadata code). Once verified, any Contextualized Resources created by the authorized Contextualizer for that domain are marked as **`authoritative`**; meaning that they are considered to be created by the *domain owner*. This additional contextualization mechanism allows Ocean applications to further restrict Discovery Requests to **`authoritative`** results. In this way, Ocean applications can be constructed to interact specifically with a known set of Resources. For example, Flickr may develop an Ocean-based photo browser that is configured to discover digital images from the authoritative Flickr domain.

Finally, while Ocean's collaborative annotation model provides a community-based approach for scalable Contextualized Resource contribution, Ocean applications may still encounter constituent sparsity issues in less popular or newly established contexts (i.e. contexts without adequate Contextualized Resources). To address sparsity issues, we explored augmenting Ocean's context-aware collaborative annotation model with automated methods of Context Metadata extraction and annotation. Increasingly, Websites embed machine-readable context information within Resource collections such as geo-location [55], calendar data [81] or electronic business cards [80]. Several projects have explored automatic extraction of such semantic metadata. For example, Ding et al. [90] investigated geo-location extraction using various Websites; the TimesMine system [322] can automatically generate timeline-based views of date-tagged Web content; and Newsjunkie [99] mines online new sites to extract personally relevant content. As described in section 8.4, we developed a context-aware WebCrawler framework capable of automatically extracting Contextualized Resources from several popular Web applications.

5.6 Chapter Summary

This chapter provided an overview of wide-area context-aware computing techniques based on the Ocean approach. It began by describing the client-centric mashup (CS mashup) style, which aligns well with Ocean's approach presented in section 3.5 but lacks support for context-mediated adaptation. Accordingly, we introduced an extension to the CS mashup style that enables dynamic, context-aware component discovery and selection based on the Contextualized Resource model introduced in section 4.3. Related, we proposed the Ocean Registry, which represents an extensible mechanism intended for storing, managing and discovering Ocean Metadata. Notably, the Ocean Registry's Blackboard architecture provides a separation of concerns between Resource providers, Contextualizers and Resource consumers; effectively allowing Ocean to integrate existing Resources without requiring changes to conventional Web architecture. Related, several Ocean Registry processes were presented including a detailed discussion of Contextualized Resource management and an introduction to Context Handler management and Contextualized Resource discovery. The chapter concluded with the presentation of two preliminary community-based contribution mechanisms intended to promote the controlled contribution of Context Handlers and the open contribution of Contextualized Resources.

Chapter 6
Contextualized Resource Persistence and Discovery

6.1 Introduction

According to the Ocean application model described in section 5.2.2, the Ocean Registry must accommodate very large Contextualized Resource (CR) datasets while simultaneously supporting Discovery Requests that include native context data (NCD) as query terms. Importantly, Ocean applications make Discovery Requests at runtime (often under strict time constraints); hence, in addition to query effectiveness, query efficiency is a major aspect of the Ocean Registry. As introduced in section 4.3, Contextualized Resources represent arbitrarily complex data structures that cannot often be effectively indexed or queried using classical key, range or proximity techniques. Hence, in section 4.3.3, we introduced similarity modeling as a promising approach for discovering semantically related objects based on domain-specific features. In this regard, we adopted notions of similarity modeling within the Context Metadata interface described in section 4.3.2. Recall that implementations of the Context Metadata interface (i.e. Context Handlers) allow Context Experts to provide domain-specific configuration and comparison intended to facilitate complex information retrieval scenarios. However, as discussed in section 4.3.3, similarity modeling techniques often suffer from the "curse of dimensionality," whereby exponential search time or memory requirements may be encountered in high dimensional feature spaces [376]. This chapter discusses the Ocean Registry's approach for accommodating efficient storage, indexing and discovery of Contextualized Resources.

This chapter proceeds as follows: First, section 6.2 describes background and related work specific to similarity search. Next, section 6.3 discusses Contextualized Resource persistence. First, section 6.3.1 presents the IndexManager abstraction. Next, section 6.3.2 describes Ocean's persistence architecture. Finally, section 6.3.3 provides an indexing and persistence example. Next, section 6.4 discusses Contextualized Resource discovery. First, section 6.4.1 describes the Ocean Registry's Discovery API and associated query protocol. Next, section 6.4.2 describes Query Object instantiation. Next, section 6.4.3 provides an overview of multi-feature similarity search. Finally, section 6.4.4 presents Ocean's multi-feature similarity search approach, including the adoption of the Threshold Algorithm, search space reduction and the Discovery Framework software architecture.

6.2 Background and Related Work

The diversity of emerging data types and limitations of existing search techniques have motivated the development of *similarity search* approaches, whereby complex objects are retrieved from a data store if they are determined to be similar to a query object according to one or more domain-specific features [376]. As previously introduced, complex objects cannot often be meaningfully indexed or queried using classical means. Notably, complex objects may provide neither a natural ordering scheme nor a means of directly comparing equality. Related, similarity search algorithms model and compare search objects according to a *comparison function*, which analyses important aspects of the objects resulting in a domain-specific comparison metric. Common comparison functions include geometric models, feature models, alignment-based models and transformational models [125]. Common comparison metrics include geometric distance, cosine similarity, Jaccard coefficients,

Hamming distance, Levenshtein distance and so forth [281]. Similarity search algorithms are often developed to meet the requirements of a given domain. Example domains include computational biology, where researchers may be interested in searching for protein sequences that are similar to a representative sample (e.g. from slightly different animal species) [350]; unstructured text retrieval, where large repositories of information can be searched for objects containing similar "concepts of interest" [18]; and video compression, where objects of interest can be identified and described as vectors [163].

In many similarity search problems, prohibitive computational complexity derives from a *proximity problem*, whereby the distances between several points must be calculated in a metric space according to an exact nearest neighbor search algorithm [376]. Efficient solutions to such problems have been discovered when these points lie in a constant dimension [160]. For example, if the points in question lie within a plane, exact nearest neighbor algorithms achieve a query time of $O(log_n)$ and require only $O(n)$ storage [297]. However, as dimensionality increases, many algorithms' space or time requirements grow exponentially. For example, the exact nearest neighbor problem has a solution of $O(d^{O(1)} \log n)$ query time and requires $n^{O(d)}$storage [210]. Such performance issues may also be exacerbated in practical scenarios, where linear or near-linear storage causes exact nearest neighbor algorithms to exhibit linear query times for relatively small values of n [351]. Hence, it has been conjectured that no efficient solutions for the exact nearest neighbor problem exists when the dimensionality is sufficiently large [160].

Several approaches have been devised to address the performance challenges associated with exact nearest neighbor similarity search techniques. Often, it is possible to eliminate the exponential effect of increasing dimensionality by allowing solutions to be *approximate* rather than exact, using so-called approximate nearest neighbor (ANN) algorithms. Notably, several ANN approaches have shown that approximation can reduce exponential effects to polynomial [160]. In general, most ANN techniques allow for the specification of a precision parameter (ε), which is used to control the distance of neighboring points from the search point. ANN algorithms typically attempt to asymptotically approach the exact neighbor as ε goes to 0, and to increase the query performance (at the expense of precision) as ε becomes larger [61]. While a full treatment of ANN techniques is outside the scope of this dissertation, the interested reader may reference [356] for a comprehensive treatment. Table 13 presents the performance characteristics of selected ANN algorithms as described in [160].

Source	Query time	Storage	Update time
[190] Randomness: Monte Carlo	$d \log n / \min(\varepsilon^2, 1)$	$n^{O(1/\varepsilon^2 + \log(1+\varepsilon)/(1+\varepsilon))}$	$n^{O(1/\varepsilon^2 + \log(1+\varepsilon)/(1+\varepsilon))}$
[159] Randomness: Monte Carlo	$n^{O\left(\frac{1+\log(1+\varepsilon)}{1+\varepsilon}\right)}$	dn	$d\ log^{O(1)} n$
[161] Randomness: Monte Carlo	$dn^{1/(1+\varepsilon)}$	$n^{1+1/(1+\varepsilon)} + dn$	$dn^{1/(1+\varepsilon)}$
[158] Randomness: Las Vegas	$(d \log n/\varepsilon)^{O(1)}$	$n^{1/\varepsilon^{O(1)}}$	Static
[158] Randomness: Deterministic	$(d \log n/\varepsilon)^{O(1)}$	$n^{1/\varepsilon^{O(1)}}$	Static

Table 13: Selected approximate nearest neighbor performance characteristics (from [160])

In addition to approximation, *dimensionality reduction* techniques have also been proposed to increase query speed. The central idea of dimensionality reduction is to select the subset of the available features that best represents the data and then construct an index data structure for this reduced feature space [57]. Additionally, queries are also feature-reduced in order to match the search space of the constructed index. To help alleviate the curse of dimensionality with regards to the index data structure, datasets can be feature-reduced before indexing [164]. Importantly, results from dimensionally reduced queries are *lossy* in character (e.g. producing false positives); however, the amount of loss is dependent on both the dataset and feature-reduction techniques. For example, Principal Component Analysis (PCA) [172] is a widely used technique for dimensionality reduction that exploits detected variance in data as a means of determining an appropriate feature reduction space. However, while PCA results in little or no information loss when the data are globally correlated, many data types do not exhibit global correlation in practical scenarios; resulting in significant information loss [164]. Other approaches, such as Local Dimensionality Reduction (LDR), attempt to overcome these shortcomings by segmenting locally correlated data before performing dimension reduction [58].

Approximation and dimension reduction are typically combined with indexing techniques to facilitate sub-linear query performance. Unlike classical database systems, which often construct indexes based on specific table columns (e.g. record keys), similarity search approaches must index the complex features of the search objects in question. Towards this end, similarity-based indexing extracts domain-specific aspects of the information content into a data-structure supporting fast

lookup [376]. Importantly, indexing algorithms attempt to capture aspects of the data considered relevant to a user's information needs; however, most approaches introduce some measure of inexactness, which can be measured according to the approaches *effectiveness*. As informally defined in [380], "a system is effective if a good proportion of the first *r* matches returned are relevant. Also, different search mechanisms have different computational requirements and so measurement of system performance must thus consider both effectiveness and efficiency." Related, *precision* and *recall* are commonly used to quantify effectiveness [205]. Precision refers to the fraction of the retrieved objects that are relevant and recall refers to the fraction of relevant objects that are retrieved [380].

The determination and representation of important aspects of persisted data means that most indexing techniques rely on domain-specific similarity metrics. For example, many image databases provide spatial indexing techniques that must consider domain-specific aspects such as space partitioning, data partitioning and dynamic adjustment of overlapping image sub-regions within the index [217]. In contrast, indexing techniques have also been developed for large scientific datasets that are based on the structural aspects of self-describing file formats such as Planetary Data System and Hierarchical Data Format (see [218]). Such approaches rely on detailed knowledge of the application-specific metadata associated to these file formats. Similar domain-specificity can be observed across most similarity search domains, including unstructured text [205], audio data [360], fuzzy set values [143], Web service metadata [214] and so forth.

A variety of index data structures have been developed to address domain-specific similarity metrics [46]. Early multidimensional indexing data-structures included R-trees [134] and early variants such as the R+-tree [295]. These early approaches formed the basis of many subsequent techniques, but have been largely superseded due to inefficiencies in higher dimensional space [164]. For example, Weber et al. [351] developed a multidimensional index approach called VA-file, which represents index points as an array of compact geometric approximations. The VA-file has been shown to improve disk I/O performance in high dimensional space on uniform data, but can suffer from higher computational complexity and poor performance for skewed data. These types of performance issues have been explored by Ciaccia et al. [68], who proposed a height balanced M-tree where metric space point positioning relies only upon positivity, symmetry, and triangle inequality postulates. The M-tree demonstrates an ability to perform reasonably well in increasing dimensionality, scales well as file size increases and provides dynamic update capability. While M-trees provide a general approach, Filho et al. [107] proposed Omni-concept as a performance enhancement technique which can be applied when "the correlation behaviors of database are known beforehand and the intrinsic dimensionality d_2 is smaller than the embedded dimensionality d of database" [164]. Omni-concept can be realized using different index structures such as B+-trees and R-trees. Additional notable indexing data-structures include: P-Sphere trees [124], Slim-trees [48] and iDistance [164].

6.3 Contextualized Resource Persistence

The aforementioned indexing approaches and data-structures illustrate the domain-specificity of most similarity search algorithms. In particular, domain-specificity relates strongly to the underlying search objects, the associated comparison semantics and related persistence model of a given search technique [376]. As described in section 4.3, Contextualizers constrain the Discoverability Context of Resources by creating Contextualized Resources comprised of General Metadata and a set of configured Context Metadata (in the form of Context Handlers). As described in section 4.3.2, Context Handlers adhere to the Context Metadata interface as a means of encapsulating instantiation, configuration, similarity comparison for a given context domain. Using this interface, Contextualized Resources can be meaningfully compared to incoming Discovery Requests according to query terms containing native context data (NCD). However, significant variation exists regarding the persistence, indexing and query models that might be used to store and search for Context Metadata objects. Accordingly, this section describes Ocean's persistence approach, which supports domain-specific indexing and persistence techniques for each contributed Context Handler type.

6.3.1 The Index Manager Abstraction

In order to accommodate multiple indexing approaches, Ocean provides an architectural abstraction called the **`IndexManager`**, which is implemented by Context Experts for a given Context Handler using the contribution process described in section 5.4. The **`IndexManager`** provides a set of methods supporting domain-neutral indexing and query operations while encapsulating underlying data structures as required by the similarity mechanisms of a given context domain. We base our **`IndexManager`** interface definition on related work from Chaudhuri et al. as presented in [60]. The **`IndexManager`** interface is shown in Figure 47.

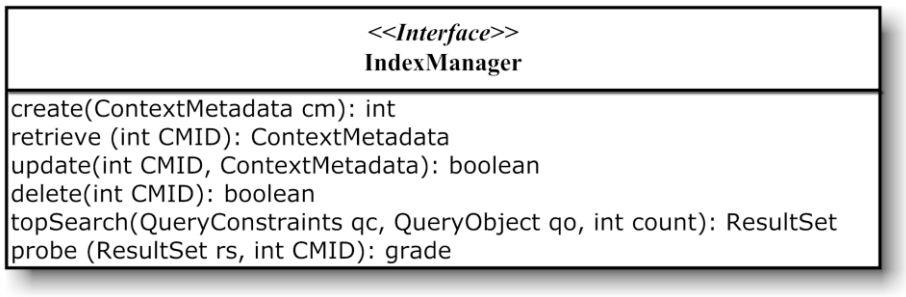


Figure 47: The IndexManager interface

Regarding Figure 47, the **`IndexManager`** interface methods are defined as follows:

- **`create`**: Indexes a Context Metadata object (of a specific type) in secondary storage according to a domain-specific indexing technique and related data structures. If successful, the method returns an integer representing the resultant Context Metadata identifier (CMID), which can be used to directly access the persisted Context Metadata object.
- **`retrieve`**: Returns the Context Metadata object referenced by a given CMID.
- **`update`**: Updates the previously persisted Context Metadata object (as identified by its CMID) with new values and updates related index data structures.

- **`delete`**: Permanently removes the previously persisted Context Metadata object identified by the given CMID.
- **`topSearch`**: For a given set of **`QueryConstraints`**, returns a sorted set of objects (**`ResultSet`**) from the underlying data store (bounded by **`count`**) that are considered most similar to the **`QueryObject`**. The **`QueryConstraints`** object indicates how the search space should be reduced according to the data type, URI domain or Contextualizer associated with the Context Metadata's parent (i.e. its associated Contextualized Resource). The **`QueryObject`** refers to a template Context Handler constructed from an incoming Discovery Request (described in section 6.4.2).
- **`probe`**: Provides random access to the similarity **`grade`** (in the interval [0,1]) of a specified Context Metadata entity (according to CMID) from within a given **`ResultSet`**. Returns **`null`** if the CMID is not found.

6.3.2 The Ocean Persistence Architecture

The **`IndexManager`** abstraction forms the foundation of the Ocean persistence architecture, which allows Ocean Metadata to be efficiently stored and indexed for rapid retrieval. As a given Ocean Metadata entity is stored within the Persistence Framework it may be referenced by *several* domain-specific indexes; each based on a different similarity mechanism and related Index Manager. Within the Ocean Registry, complete Ocean Metadata objects are persisted within a shared data store, called the Ocean Metadata Store, whereas associated Context Metadata objects are indexed and stored according to the associated Context Handler type. The persistence architecture is illustrated in Figure 48 using an example Ocean Metadata entity (note that other Context Metadata combinations are possible).

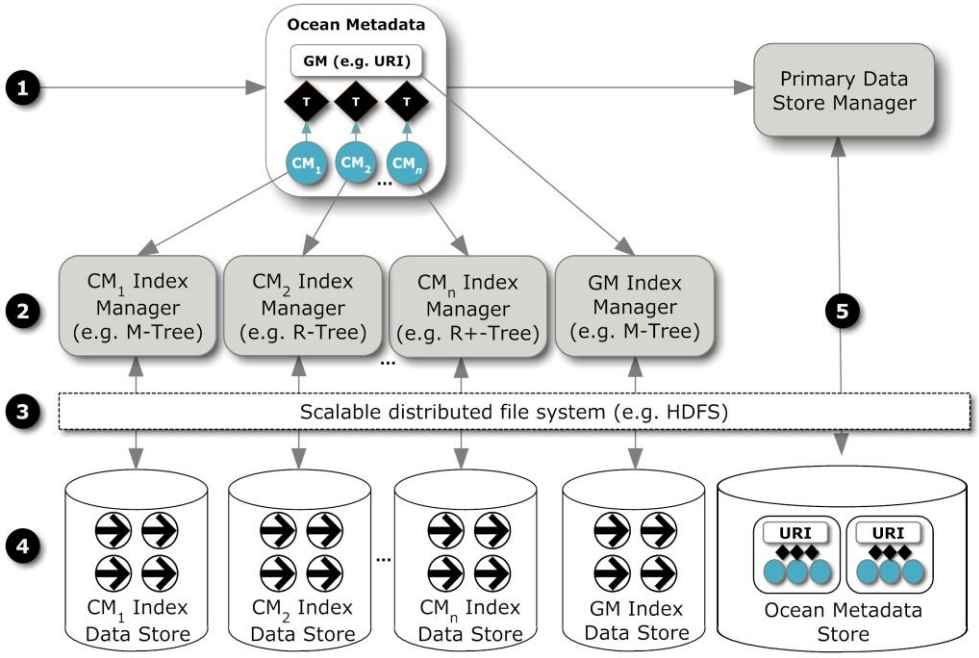


Figure 48: Overview of the Ocean Metadata persistence architecture

Regarding Figure 48, the Ocean Metadata persistence architecture is defined as follows:

1. At the conclusion of the instantiation process described in section 5.3.3, Contextualized Resources exist as complex Ocean Metadata objects comprised of General Metadata and appropriately configured Context Metadata (see section 4.3.3 for examples). The Ocean Metadata is sent to the Ocean Persistence Framework for indexing and storage.

2. The Persistence Framework checks the integrity of the Ocean Metadata object and ensures that no duplicate objects exist within the Ocean Metadata Store data store. Next, the Ocean Metadata object is assigned a globally unique identifier (GUID). Next, the General Metadata and associated Context Metadata are sent to their appropriate Index Managers (based on data type), where they are indexed and persisted according to the Index Manager abstraction described in section 6.3.1.

3. For large-scale scenarios, an optional scalable distributed file-system may be employed to support the storage of large datasets on commodity hardware (discussed shortly).

4. CM indexes are persisted in secondary storage using an appropriate underlying data model. Each Index Manager provides an appropriate file-system representation for its underlying index data structure (index store). Additional indexes are also established for each Context Metadata entity based on its parent's data type, URI domain and Contextualizer (see [380]).

5. Finally, the indexed Ocean Metadata object is sent to the Primary Data Store Manager (Storage Manager) where it is persisted within the Ocean Metadata Store as an object. The stored Ocean Metadata in combination with its associated Web Resource is referred to a Contextualized Resource.

While not discussed in this dissertation, recent approaches provide insight into real-world persistence mechanisms that might be adopted for large-scale data storage in Ocean. For example, Google have developed a distributed file-system called GoogleFS [118], which supports extremely large data sets and provides high I/O throughput by exploiting inexpensive commodity hardware. GoogleFS was developed to support large-scale distributed computing environments comprised of large numbers of commodity machines, vast amounts of very large files (i.e. multi-GB), data streaming, append-based file access. Google have demonstrated the feasibility of their approach throughout their enterprise using extremely large configurations (e.g. 300TB storage spread across 1000 notes). While GoogleFS is not available commercially, the Apache Software Foundation has developed an open-source implementation of GoogleFS, called the Hadoop File System (HDFS)[41], which provides similar capabilities. Notably, HDFS is designed to scale to petabytes of storage and can exploit a variety of underlying hardware architectures. HDFS' capabilities include streaming data access, large data files and a simple coherency model [178]. Related, a recent object-based database system has also demonstrated the ability to accommodate petabyte size datasets [27].

[41] http://hadoop.apache.org/core/

6.3.3 An Indexing and Persistence Example

Details regarding the Index Manager abstraction and related index storage are now discussed using an example based on the iDistance index approach presented in [164]. As described by Jagadish et al., iDistance is a modified B+-tree designed to address a class of approximate nearest neighbor search algorithms known as *K-nearest neighbor query*. In their paper, they formalize a *K-nearest neighbor query* as "Given a set of points *DB* in a *d*-dimensional space *DS*, and a query point $q \in DS$ find a set S which contains K points in DB such that, for any $p \in S$ and for any $p' \in DB - S$, $dist(q,p) < dist(q,p')$" [164]. To accommodate efficient queries, iDistance transforms high dimensional points into a single dimensional space using m data space partitions and a B+-tree to represent the transformed points (see [164] for details). In this regard, the iDistance approach is lossy as it trades the possibility of false positives for very fast single-dimensional range queries and the possibility of integration into commercial database management systems (which often support B+-tree indexing [205]). Given iDistance's inherent limitations, it may not be applicable for every type of Context Metadata; however, given some tolerance for error, the benefits of the approach may outweigh the costs (for details, see the cost benefit analysis in [164]).

As a real-world example, an Ocean Index Manager could be designed to map the iDistance algorithm directly into an underlying DBMS as per Figure 49. In this figure, an incoming Context Metadata (CM) object is processed by its associated Index Manager's underlying iDistance algorithm. During index processing, domain-specific features of the CM object are translated into data points within a *d*-dimensional space appropriate to the context domain. Next, as per the iDistance approach, high dimensional points are transformed into a partitioned single dimensional space according to the process described in [164]. The resultant partition information (denoted P_i below) and the related single-dimensional points are then stored within a commercial DBMS (configured to support B+-tree indexing). An additional database table is used to store iDistance partition data. Additionally, conventional approaches are also used to establish additional indexes based on attributes of the CM's parent Contextualized Resource, including data-type, URI domain and Contextualizer. To reference the parent Contextualized Resource, the persisted CM object retains the parent's GUID.

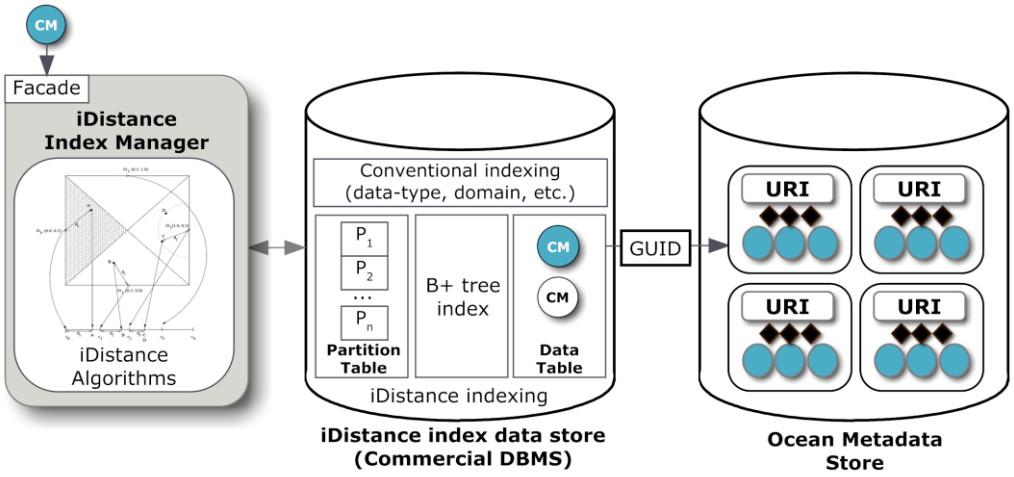


Figure 49: Example iDistance Index Manager and related persistence model

Similar to the iDistance example, Ocean Index Managers can be constructed for various Context Metadata indexing approaches using appropriate file system representations. For completeness we note that, while B+-trees are compatible with many commercial DBMSs, other tree structures may not be widely supported (e.g. M-trees). However, advanced indexing techniques may be provided by custom data store implementations that include specialized performance techniques to help improve indexing and query speed. Such techniques are described in [305] and include examples such as buffering strategies, which aim to reduce disk I/O using techniques such as LRU replacement [203]; dynamic layout rearrangement, which optimizes disk I/O for common access patterns by rearranging the physical layout of the data on secondary storage [54]; and physical designs, which exploit physical properties of modern disk drives using technique such as adjacent block utilization [305].

6.4 Contextualized Resource Discovery

The Ocean approach presupposes a similarity search mechanism whereby Ocean Metadata can be retrieved from the Ocean Registry's persistence model and returned to an Ocean application as Descriptive Metadata. In order to provide a foundation for discovering contextually-relevant Resources, we proposed the Contextualized Resource abstraction in section 4.3, which provides a means of constraining the discovery context of conventional Web Resources using Ocean Metadata. Notably, the Contextualized Resource abstraction adopts an associative metadata model, whereby Ocean Metadata are stored separately from the Resources themselves within the Ocean Registry. As described in section 5.5, the Ocean Registry provides an open contribution model whereby *any* Contextualizer can contextualize *any* Resource with *any* combination of Ocean Metadata. As described in section 6.3, Ocean Metadata are stored and indexed within the Ocean Registry using a persistence model that exploits the similarity modeling mechanisms of the Context Metadata abstraction. Recall that the Ocean persistence architecture is designed to support efficient Ocean Metadata storage and lookup based on *multiple* domain-specific indexing approaches. This section describes Ocean's Discovery Framework, which operates in conjunction with the previously introduced Persistence Framework to discover relevant Contextualized Resources based on arbitrarily complex Query Objects provided by Ocean applications.

Recall from section 5.2.2 that the Ocean application model is designed to support dynamic, in-situ composition of CS mashups. With regards to Figure 50, the Ocean application model can summarized as follows: First, domain-specific Ocean applications utilize Aladin techniques to acquire and model native context data (NCD) from their local environment (1, 2). Next, NCD are passed to the hosting application where they may be handled by local application logic as needed (3). In the Ocean approach discussed in this dissertation, the Aladin architecture is extended by an Ocean Context Interpreter (OCI), which supports communications with the Ocean Registry using the search protocol defined in section 6.4.1. Ocean applications formulate Discovery Requests (using the NCD as query terms), which are sent to the Ocean Registry's Resource Discovery API (4). The Ocean Registry receives the request and uses a set of information filtering techniques (see section 6.4) to discover contextually relevant Contextualized Resources (5). Next, discovered Contextualized Resources are formulated as a Discovery Response and returned to the Ocean application as Descriptive Metadata, termed Uniform Resource Identifier metadata in the figure below (6). The local OCI unmarshals the Discovery Response and sends the Descriptive Metadata to the Ocean application (7). Finally, the

Ocean application selects appropriate Contextualized Resources from the Discovery Response and composes the associated Resources in-situ using the REST interoperation mechanisms described in section 3.2.8 (8, 9).

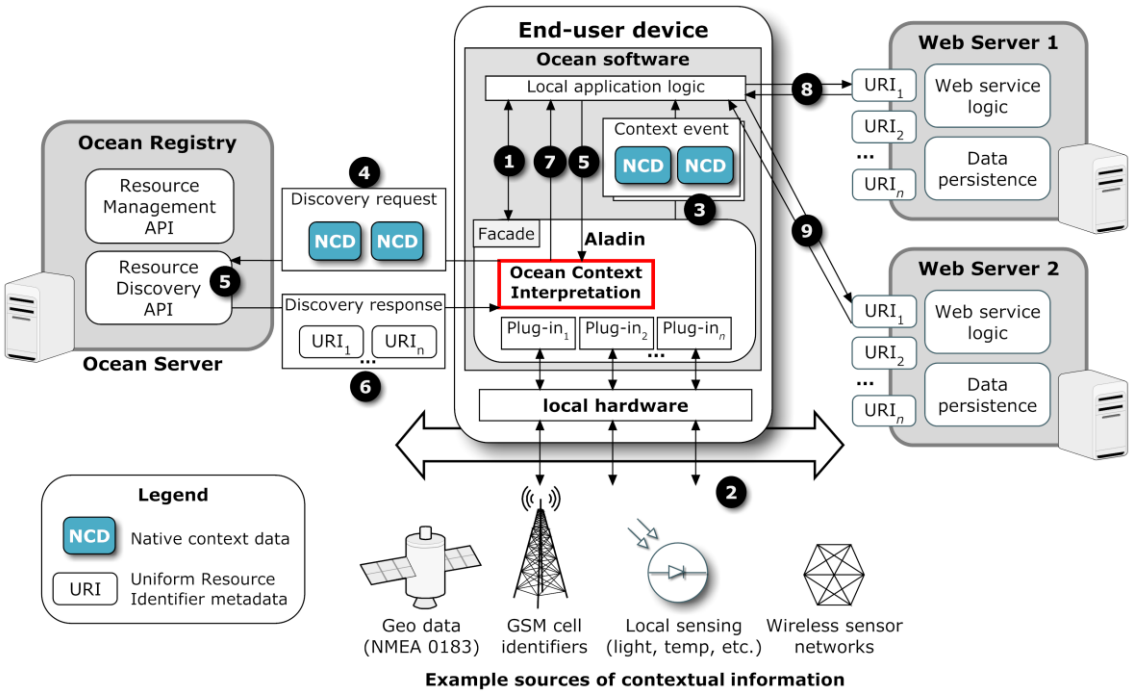


Figure 50: The simplified Ocean application model

6.4.1 The Contextualized Resource Discovery API

To support context-aware component discovery, the Ocean Registry provides a Contextualized Resource Discovery API (Discovery API), which is intended for use by Ocean applications. Interactions between Ocean applications and the Ocean Registry may occur directly or be facilitated by an Ocean Context Interpretation plug-in, which provides request marshaling/unmarshalling for Aladin-based Ocean applications. Interaction with the Discovery API is accomplished by sending an XML-based Discovery Request to the Discovery API's URI using **`HTTP POST`**. The Ocean Registry's Discovery URI is **`http://oceanframework.org/discovery`**. Importantly, Discovery Requests must adhere to the Ocean Discovery Request format XML schema described in section 6.4.1.1. The Discovery API responds to Discovery Requests synchronously; returning either a structured Discovery Response or an HTTP error code. The Discovery Response XML schema and associated error codes are described in section 6.4.1.2.

6.4.1.1 Discovery Request Format

Contextualized Resource Discovery Requests (Discovery Requests) consist of specially formulated XML requests, which allow Ocean applications to control various aspects of the Resource Discovery process. Importantly, before Discovery Requests may be initiated, Ocean application must first authenticate with the Ocean Registry using the mechanisms described in section 5.3.2. Once authenticated, Ocean applications may query the Discovery API as often as needed, provided that the request rate is not deemed excessive (e.g. requests in excess of 1 per second per IP may receive **`HTTP`**

response code 503 – Service Unavailable as a response). Discovery Requests must adhere to the Discovery Request XML schema shown in Figure 51.

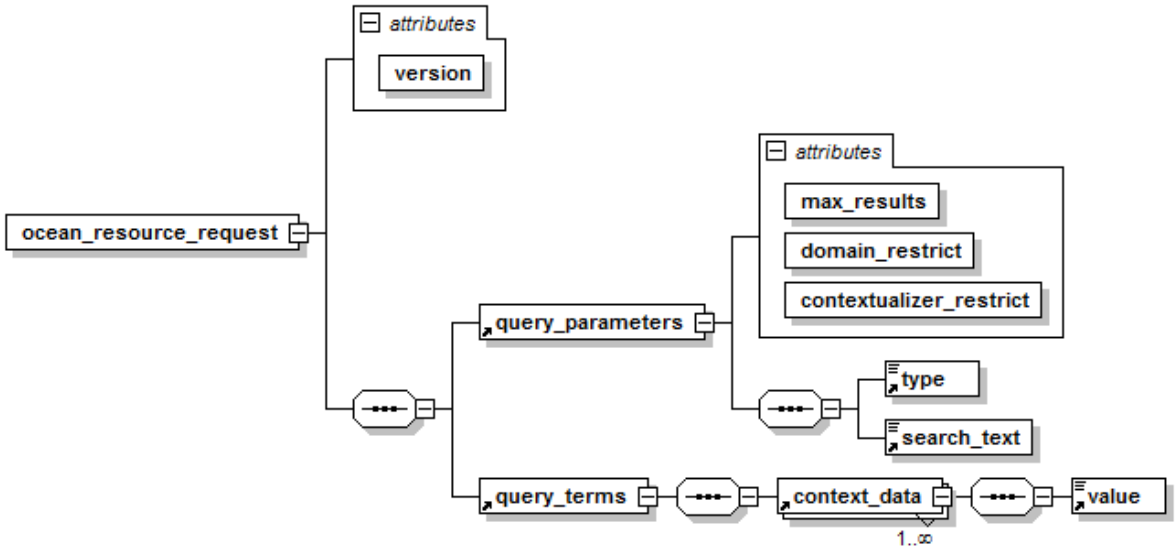


Figure 51: The Discovery Request XML schema

Regarding Figure 51, the Discovery Request schema is defined as follows:

1. The root **ocean_resource_request** element contains a single version attribute that is used by the Ocean Registry to select a proper request parser. Within the root element, two sub-elements represent the query's search parameters and query terms.

2. The **query_parameters** element provides several attributes that are used to control query processing: **max_results, domain_restrict and contextualizer_restrict** (described shortly). In addition, two sub-elements are used to describe fundamental aspects of the requested Contextualized Resources. The **type** indicates the data type of requested Resources (e.g. standard **MIME** types [327]) and **search_text** is used to match textual terms against the title and description contained within persisted Contextualized Resources (see section 4.3). Note that both **type** and **search_text** must be wrapped in **XML CDATA** tags to allow for the inclusion of arbitrary character values within the request.

3. Finally, the **query_terms** element includes one or more **context_data** sub-elements, which include native context data wrapped within **XML CDATA** tags to allow for the inclusion of arbitrary values within the request.

Regarding Figure 51, the query parameters for Discovery Requests are defined as follows:

Parameter	Description	Default
type	Indicates the desired data-type of discovered Resources (e.g. standard **IANA MIME** types). Only one data-type is allowed per request.	**text/html**
max_results	Specifies the maximum search results requested by the Ocean application.	**50**
domain_restrict	Indicates that the Query Processor should limit search results to a specific domain or sub-domain (as per [34]).	**Unrestricted**
contextualizer_restrict	Indicates if Query Processor should limit search results to a specific Contextualizer developer account.	**Unrestricted**

Table 14: Overview of available Resource Discovery search parameters

The Ocean Registry's information filtering algorithms perform similarity search according to the native context data contained within a Discovery Request's query terms. As such, query terms must be wrapped within an **XML CDATA** tag to allow for the inclusion of arbitrary context data. In this way, native data formats such as structured text and base64 encoded binary data may be provided. The number of **query_terms** that can be included within a Discovery Request is currently unbounded. An example Discovery Request is provided in Figure 52.

```
<?xml version="1.0" encoding="UTF-8"?>
 <ocean_resource_request version="1.0" >
 <query_parameters max_results="50"
      domain_restrict=""
      contextualizer_restrict="dcarlson">
  <type><![CDATA[mime:text/calendar]]></type>
  <search_text><![CDATA[Example search text]]></search_text>
 </query_parameters>
 <query_terms>
  <context_data>
    <value>
      <![CDATA[security_token:776958d554c987ae6d0b6f70826edfdaeb8670c8]]>
    </value>
  </context_data>
  <context_data>
    <value><![CDATA[<gml:Point><gml:pos>53.873488,10.686607</gml:pos>
    </gml:Point>]]></value>
  </context_data>
    <context_data>
    <value><![CDATA[TYPE=WIFI|TIME=1096470064731|ID=00:09:5b:de:fa:7a
    |NAME=NETGEAR|RSSI=-88|WEP=true|INFR=false]]></value>
   </context_data>
 </query_terms>
</ocean_resource_request>
```

Figure 52: An example Discovery Request

Several observations can be made regarding Figure 52. First, the example Discovery Request pre-filters Contextualized Resources that are associated to Web Resources of type **`text/calendar`**. Next, the request includes query parameters that restrict the maximum search results to 50; place no restrictions on the Resource's domain; and restrict results to those contextualized by the Ocean Contextualizer **`dcarlson`**. Finally, the request includes several native context data **`query_terms`**, including a security token as an SHA-1 hash-code [219], geo-location information in the GML-simple format [237] and Wireless LAN access point information in the NetStumbler format [212].

6.4.1.2 Discovery Response Format

Once query processing is complete for a given Discovery Request, the set of Ocean Metadata determined to be most similar to the request are returned to the calling Ocean application. Discovery Responses provide Ocean applications a set of Descriptive Metadata regarding contextually-relevant Resources for use in machine-based or user-based adaptation decision models. Recall that in the Ocean approach, applications are free to act upon Discovery Response metadata according to domain-specific application logic (see section 5.2.2). Note that each discovered Resource provides a composite **`similarity_score`** (in the interval [0,1]) indicating how similar the given Resource is to the Discovery Request (see section 4.3.2). The set of Descriptive Metadata is rank-ordered according to the associated score value and formulated according to the Discovery Response XML schema shown in Figure 53.

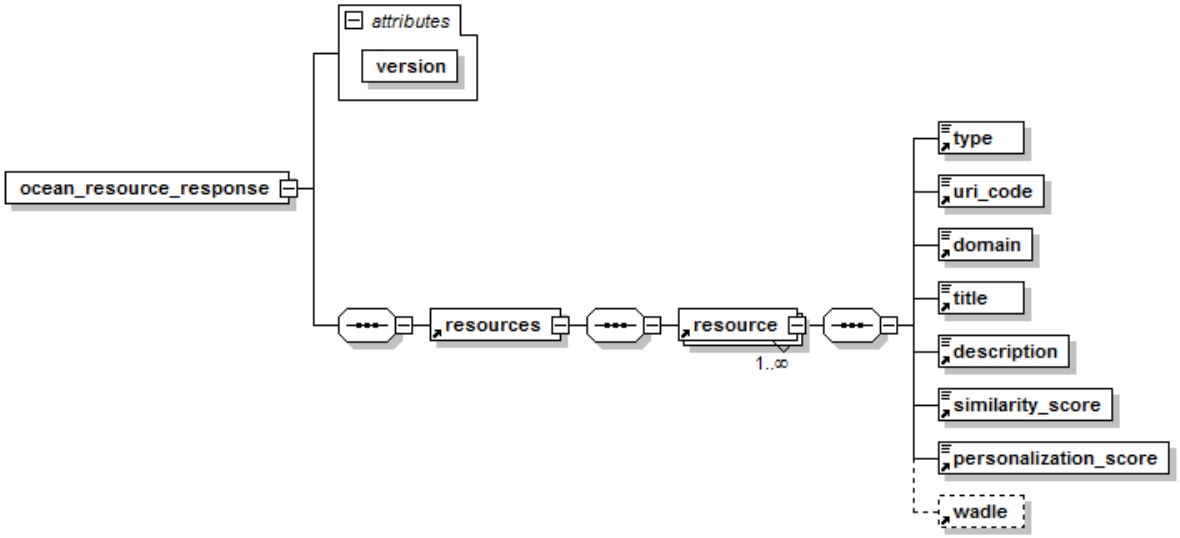


Figure 53: The Discovery Response XML schema

Regarding Figure 53, the Discovery Response XML schema is defined as follows:

1. The root **`ocean_resource_response`** element contains a single version attribute that can be used to select a response parser.
2. The **`resources`** element is the top-level container for all discovered Resource metadata and may contain from 0 to **`max_results`** sub-elements.

3. Each discovered **`resource`** element provides the following Descriptive Metadata:
    a. a **`type`** element describing the Resource's data-type (e.g. to its **`MIME`** type) as a means of assisting application state management;
    b. a **`title`** element providing the title text provided by the Contextualizer;
    c. a **`description`** element providing descriptive text provided by the Contextualizer;
    d. a **`uri_code`** element providing the URI of the Resource or a hash-code that can be used by the Ocean application to resolve the Resource's URI (described shortly);
    e. a d**`omain`** element indicating the domain of the URI (or sub-domain);
    f. a **`similarity_score`** element as a numerical score indicating how similar the associated Contextualized Resource is to the incoming Discovery Request (see section 6.4);
    g. a **`personalization_score`** element as a numerical score indicating the user's predicted affinity for the Resource (see section 7.3); and
    h. an optional **`wadle`** element providing an associated WADL document, if provided (as per [137]).

As previously introduced, the **`uri_code`** element within the Discovery Response can represent either a valid URI or a related hash-code. The presence of a hash-code indicates that the Ocean application requested Resource personalization, as described in section 7.3. In this case, the Descriptive Metadata are *intentionally insufficient* for fully resolving discovered Resources. Specifically, all URI references are removed from the Response metadata, including the removal of URI details from the base attribute of the **`<resources>`** element of associated WADL documents (see [137] for details). In place of URIs, a generated hash code is used to force Ocean applications to contact the Ocean Registry again to resolve true URIs. Notably, **`uri_codes`** are used to support the implicit rating of Resources as required by Ocean's recommender algorithms (see section 7.3). Ocean applications resolve true URIs by calling the Ocean Registry using **`HTTP GET`** using the following URI: **`http://oceanplatform.org/resolve/{uri_code}`**. If a given **`uri_code`** is found, the Ocean Registry returns a **`HTTP 303`** redirection status code, which provides the Resource's true URI (or a **`400 Bad Request`** status code if the **`uri_code`** was not found). An example Discovery Response is shown in Figure 54.

```
<?xml version="1.0" encoding="UTF-8"?>
<ocean_resource_response version="1.0" >
 <resources>
  <resource>
   <type><![CDATA[mime:text/html]]></type>
   <uri_code>10fb2d66a65335317b54c93b15edefebe62a19c9</uri_code>
   <domain><![CDATA[www.smugmug.com]]></domain>
   <title><![CDATA[Example photo website]]></title>
   <description><![CDATA[Example description]]></description>
   <similarity_score>.89</similarity_score>
   <personalization_score/>
   <wadle/>
  </resource>
  <resource>
   <type><![CDATA[mime:text/html]]></type>
   <uri_code>a0fc1e77a65235317b54a93c15edefebe45a19c8</uri_code>
   <domain><![CDATA[subdomain.example.com]]></domain>
   <title><![CDATA[Another example title]]></title>
   <description><![CDATA[Another example description]]></description>
   <similarity_score>.55</similarity_score>
   <personalization_score/>
   <wadle/>
  </resource>
 </resources>
</ocean_resource_response>
```

Figure 54: An example Ocean Discovery Response

6.4.2 Query Object Instantiation

Based on the results of local application logic and context modeling, Ocean applications attempt to discover contextually-relevant Resources using the Discovery API's request format (see section 6.4.1.1). Recall that query parameters allow Ocean application to control discovery results based on constraints such as data-type, domain and Contextualizer account. Further, provided query terms include native context data (NCD) that are wrapped by **`XML CDATA`** tags. As Discovery Requests arrive at the Discovery API, the Ocean Registry performs a multi-feature similarity search process based on the contents of the request. To begin the discovery process, an incoming Discovery Request is transformed into a Query Object by a multi-threaded *Query Processor* that controls query handling and coordinates the required Ocean Registry resources. Once a Query Processor is instantiated for a given request, it creates a Request Factory that is responsible for processing the request XML. The Request Factory first unmarshals all search parameters and query terms and then instantiates the necessary Context Handlers for each included NCD using the adapted chain of responsibility technique introduced in section 5.3.3. (Recall that Context Handlers represent implementations of the Context Metadata interface presented in section 4.3.2.) Resultant Query Object represents a machine-processable representation of the client's Discovery Request, which forms a central component of the Ocean Registry's similarity search approach (described shortly). An overview of the Query Object instantiation process is shown in Figure 55.

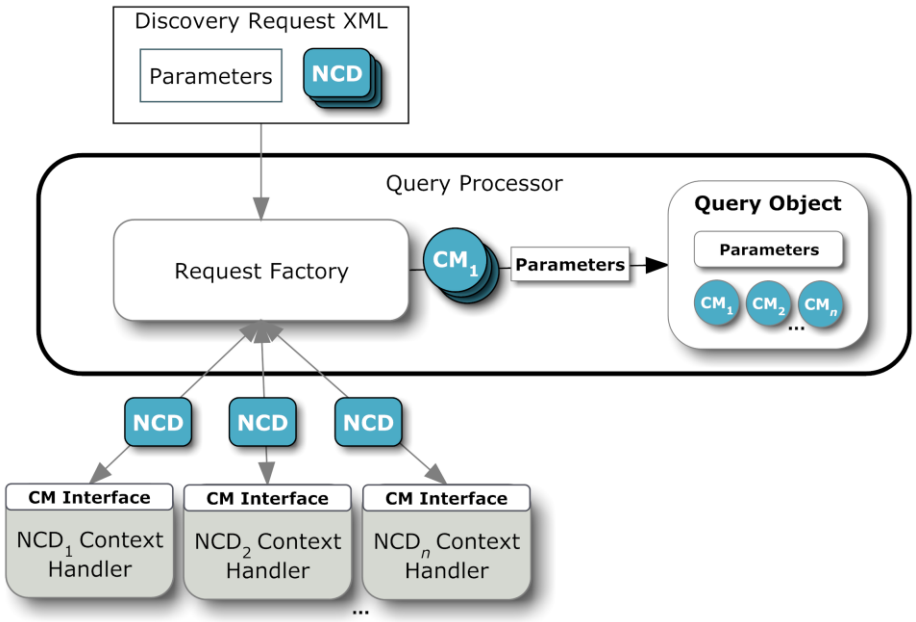


Figure 55: Overview of the Query Object instantiation process

With reference to Figure 55, the Ocean Reference Implementation (RI) provides the following realization of the `RequestFactory` and related classes as shown in Figure 56. (Note that implementation-specific methods may be included in the figure below; however, in the interest of clarity, these are not described.)

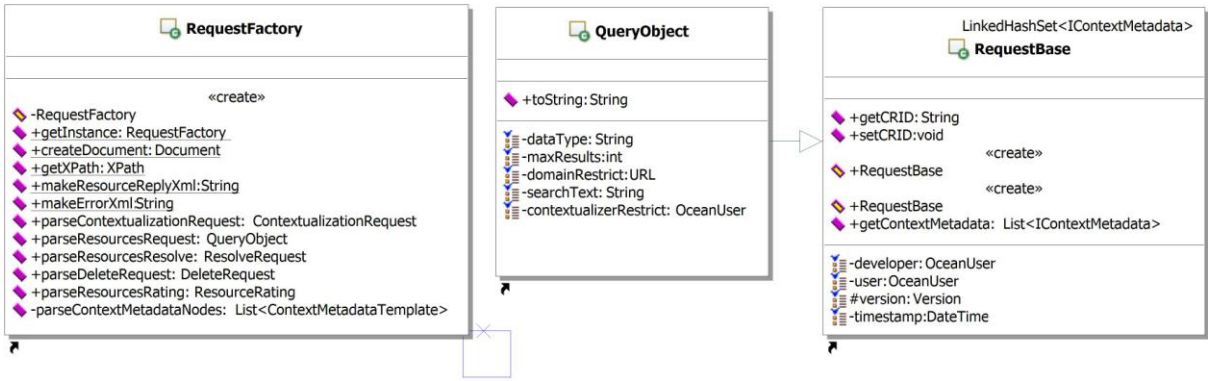


Figure 56: The RequestFactory and related classes within the Ocean RI

Once a Query Object has been instantiated for a given Discovery Request, the Query Processor dynamically formulates a search strategy based on the included Context Metadata. As persisted Ocean Metadata may be described by multiple Context Metadata entities, Ocean Contextualized Resource Discovery (Resource Discovery) supports a *multi-feature similarity search* (MFSS) approach, whereby multiple similarity metrics are considered simultaneously. Broadly, MFSS approaches employ multiple classifiers for a given Query Object and provide a ranked query model that scores the similarity of database object against the Query Object using multiple feature comparisons [376]. Resulting similarity scores are typically composed of several atomic similarity approximations (or *grades*) that are aggregated together. As presented in [102], "Assume that each object in a database

has *m* grades, or scores, one for each of *m* attributes. For example, an object can have a color grade, that tells how red it is, and a shape grade, that tells how round it is. For each attribute, there is a sorted list, which lists each object and its grade under that attribute, sorted by grade (highest grade first). Each object is assigned an overall grade that is obtained by combining the attribute grades using a fixed monotone aggregation function, or combining rule, such as **`min`** or **`average`**. To determine the top *k* objects, that is, *k* objects with the highest overall grades, the naive algorithm must access every object in the database, to find its grade under each attribute." Related, the next section presents important aspects of multi-feature search optimization.

6.4.3 Multi-Feature Search Optimization

At their foundation, most MFSS approaches define an aggregation function *t* where $t(x_1, \dots, x_m)$ is the overall score of an object *R*, where $x_1, \dots, x_m$ represent the set of *R's* grades of *m* attributes (also referred to as *features*). Common aggregation functions include min, average and sum (where min refers to a conjunction in fuzzy logic). As presented in [101], MFSS approaches utilize aggregation functions together with a method for modeling the similarity of object features by generating *m* sorted lists (1 per feature), where each list represents a graded set of pairs *(x,s)*, where *x* is the object and *s* is the real number grade for a given feature (typically in the interval [0,1]). Each list is generated in sorted order according to the object's feature grades. Most MFSS techniques are designed to discover the *top k objects*, where k is the number of requested query results and "top" refers to the highest ranked objects in the dataset as compared to a Query Object $\acute{R}$ (according to score).

Naïve search approaches exhibit linear efficiency and can become quickly intractable in large-scale scenarios [101]. To achieve sub-linear query performance, researchers have explored various optimization techniques. The first such optimization algorithm, called "Fagin's Algorithm" or FA, was proposed by Fagin in 1996 and was subsequently refined in 1999 [100]. Notably, the FA approach has been shown to be correct for monotone aggregation functions [100]. In terms of algorithmic efficiency, assuming that the items within each sorted list are probabilistically independent, FA exhibits a cost of $O(N^{(m-1)/m} k^{1/m})$ [100]. While FA can exhibit sub-linear performance, in many cases its performance guarantees are dependent on the aggregation function being "strict", where $t(x_1, \dots, x_m) = 1$ when $x_i = 1$ for every i. While strict aggregation functions are common, they are not always adequate in all search scenarios, resulting in reduced FA performance in many cases (e.g. the use of max is common; yet max is not strict). Moreover, even when the aggregation function is strict, common database structures have been identified that result in poor FA performance and its memory requirements grow arbitrarily large as the database grows [100].

Several related MFSS optimization approaches have also been explored. For example, Chaudhuri and Gravano [60] suggested methods of extending FA through the use of query constraints (for example compound searches where certain threshold conditions must be met, as per the **`IndexManager`** query constraints described in section 6.3.1). Additional middleware-based MFSS approaches such as IBM's GARLIC [274] and visual retrieval systems such as HERON [201] have also demonstrated techniques for aggregating search results from multiple underlying data stores. However, these techniques encounter performance problems in heterogeneous scenarios [131]. For

example, HERON relies on the Quick-Combine algorithm, which often deteriorates to linear scan (or worse) when processing heterogeneous data sets [130].

Based on the limitations of prior MFSS techniques, researchers sought to develop an algorithm that performs better than linear scan over *arbitrary* databases and requires less memory. The result of this research effort was the near simultaneous discovery of a new MFSS optimization algorithm, called the Threshold Algorithm (TA), by three independent research groups [102, 130, 221]. Unlike FA, which is only optimal in limited cases, the TA approach is considered to be optimal in a much stronger sense as it does not make any underlying probabilistic model assumptions [102]. Given *m* sorted lists (1 per feature), where each list represents a graded set of pairs *(x,s)*, where *x* is a database object and *s* is the real number grade for the given feature in the interval [0,1], a top k query under TA can be described as follows (as adapted from [100]):

1. Do sorted access in parallel to each of the *m* sorted lists L_i. As an object *R* is seen under sorted access in some list, do random access to the other lists to find the grade x_i of object *R* in every list L_i. Then compute the score $t(R) = t(x_1, \dots, x_m)$ of object *R*. If this score is one of the *k* highest we have seen, remember object *R* and its score *t(R)* (ties are broken arbitrarily, so that only *k* objects and their scores need to be remembered at any time).
2. For each list L_i, let $\underline{x_i}$ be the score of the last object seen under sorted access. Define the threshold value τ to be $t(\underline{x_1}, \dots, \underline{x_m})$. As soon as at least *k* objects have been seen whose score is at least equal to τ, then halt. (Note that $\underline{x_i}$ refers to the *aggregation* of individual feature grades from the last object seen under sorted access; hence, the last seen feature grades from each list do not need to be maintained.)
3. Let *Y* be a set containing the *k* objects that have been seen with the highest scores (ties are broken arbitrarily).The output is then the scored set $\{(R, t(R) | R \in Y)\}$.

6.4.4 The Ocean Multi-Feature Similarity Search Approach

This section draws together several threads of related work to derive Ocean's multi-feature similarity search approach. In order to find the *top k* most similar Contextualized Resources for a given Discovery Request, we propose a feature grade aggregation approach based on the Threshold algorithm (TA) introduced in section 6.4.3. We begin with the observation that the Ocean Registry's **`IndexManager`** abstraction described in section 6.3.1 provides domain-optimized, sorted access to collections of Context Metadata indexes that reference persisted Ocean Metadata. In response to an incoming Query Object, the set installed **`IndexManagers`** within the Ocean Registry are used to produce *m* sorted lists (1 per Context Metadata type), where each list represents a graded set of pairs *(x,s)*, where *x* is the object (i.e. a reference to the parent Ocean Metadata entity) and *s* is the real number grade for a given context feature (i.e. Context Metadata entity). List generation is accomplished using each **`IndexManager's topSearch`** method. Related, the **`IndexManager probe`** method is used to provide random access to similarity grades for each of the *m* sorted lists.

During list generation, the Query Processor first reduces each **`IndexManager's`** search space by introducing a set of **`QueryConstraints`** within each **`topSearch`** method. In the first stage of search space reduction, Context Metadata objects are excluded from an **`IndexManager's`** search space if their

parent Ocean Metadata entity does not match the specified data-type, domain or Contextualizer values contained within the query parameters. Additionally, each **`IndexManager's`** search space is further reduced by excluding objects whose parent's association types do not match the Context Metadata present within the Query Object (see section 4.3.1). For example, all Context Metadata indexes whose parent Ocean Metadata *require* **`GEOPointHandlers`** (through the "**`required`**" association type) will be excluded from the associated **`IndexManager's`** search space if the Query Object does not contain a **`GEOPointHandler`**. (Note that the "**`optional`**" association type has no affect on search space reduction but does have an effect on similarity scoring.) A simplified overview of search space reduction is shown in Figure 57 (for a *single* Context Metadata index type). In this example, the **`QueryConstraints`** specify a data-type "**`text/html`**" and a Contextualizer restriction of "**`dcarlson`**". (Note that only Context Handler indexes shown as dark circles will be included in the search space.)

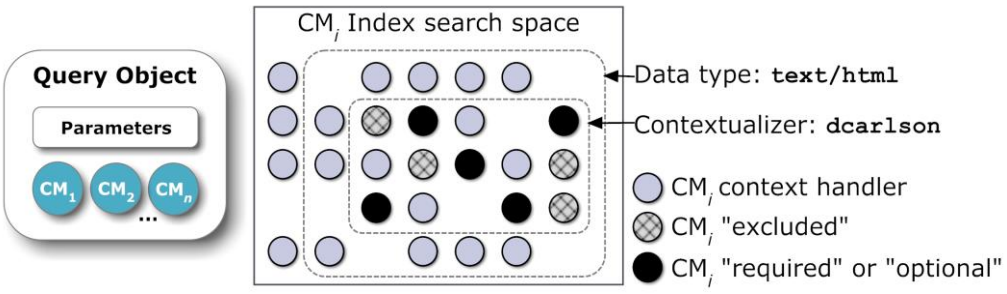


Figure 57: Example search space reduction for a single Context Metadata index type

Next, following [92], we introduce an aggregation function *t* that computes an overall score $t(R)$ of object *R* as:

$$t(R) = \frac{1}{n}\sum_{i=1}^{n} similarity(x_i, \acute{x}_i)$$

where *n* refers to the total number of Context Handlers associated to a Contextualized Resource, x_i refers to the Context Handlers associated with the Contextualized Resource and $\acute{x}_i$ refers to the matching Context Handlers within the Query Object. The similarity grade of x_i and $\acute{x}_i$ is calculated by passing the $\acute{x}_i$ Context Handler into the **`getNormalizedSimilarity`** method of Context Handler x_i (recall **`getNormalizedSimilarity`** returns domain-neutral similarity values in the interval [0,1]). Using this aggregation function, we adopt the TA algorithm as follows (adapted from [100]):

1. Do sorted access in parallel to each of the *m* **`IndexManagers`** using their associated **`topSearch`** methods. Importantly, the search space of each **`IndexManager`** should be reduced by passing in the set of **`QueryConstraints`** derived from the query parameters (i.e. data-type, domain and contextualizer restrictions).
2. As an object *R* is seen under sorted access in some **`IndexManager`**, use the **`probe`** method to find the grade x_i of object *R* in every **`IndexManager`** referred to by the Query Object. Then compute the score $t(R) = t(x_1, \ldots, x_m)$ of object *R* using the aggregation function defined above. If this score is one of the *k* highest we have seen, remember object *R* and its score

(ties are broken arbitrarily, so that only *k* objects and their scores need to be remembered at any time).

3. For each `IndexManager` referred to by the Query Object, let $\underline{x}_i$ be the score of the last object seen under sorted access. Define the threshold value τ to be $t(\underline{x}_1, \ldots, \underline{x}_m)$. As soon as at least *k* objects have been seen whose score is at least equal to τ, then halt. (Note that $\underline{x}_i$ refers to the *aggregation* of individual feature grades from the last object seen under sorted access; hence, the last seen feature grades from each list do not need to be maintained.)
4. Let *Y* be a set containing the *k* objects that have been seen with the highest scores (ties are broken arbitrarily). The output is then the scored set $\{(R, t(R) | R \in Y)\}$.

Using the adaptation of the Threshold Algorithm presented above, the *top k* Contextualized Resources are discovered for a given Query Object using the process shown in Figure 58.

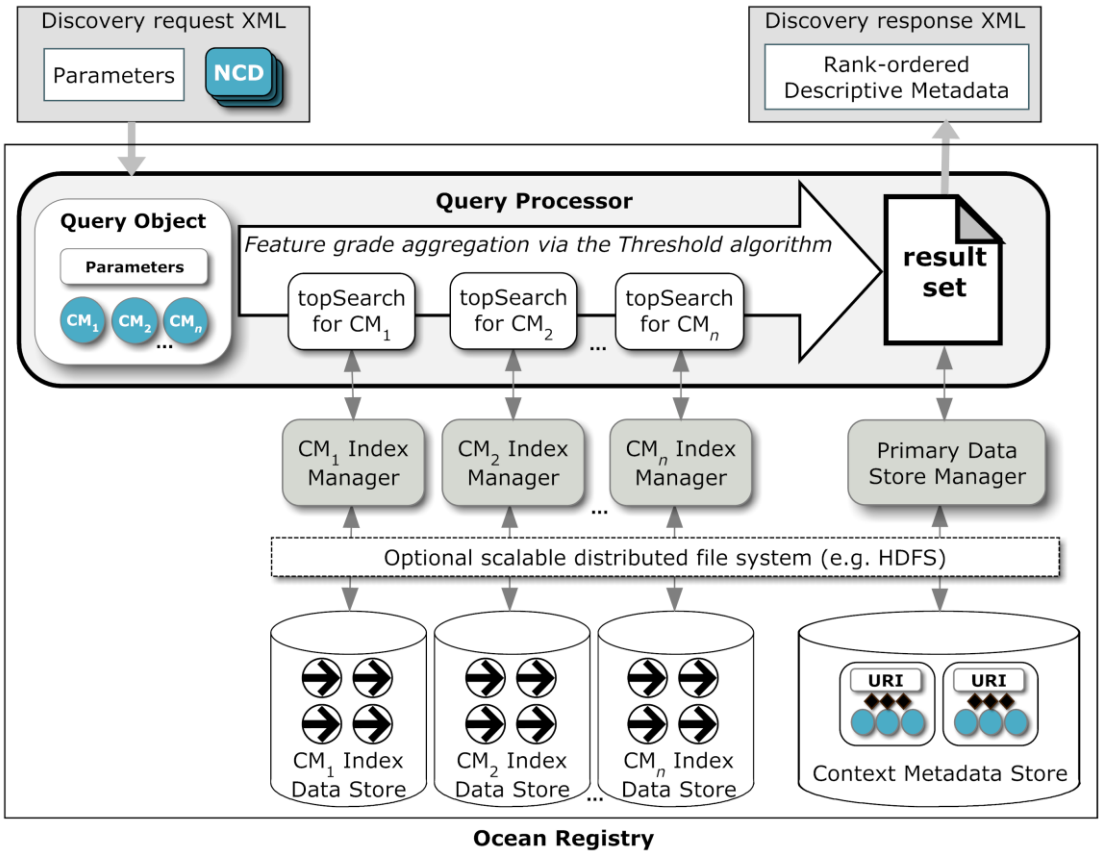


Figure 58: Overview of Ocean's multi-feature similarity search process

At the completion of query processing, Descriptive Metadata are generated for each discovered Contextualized Resource according to the following process: First, a Discovery Response object is created. Second, Descriptive Metadata for each Contextualized Resource are retrieved from the Ocean Metadata Store (using random access) and integrated into the response along with either a URI or a `uri_code` (generated using a time-based salted hash function that incorporates the true URI). Next, the Descriptive Metadata for each Contextualized Resource are updated with the composite similarity score generated during query processing. Next, the Descriptive Metadata are rank sorted according to

their associated similarity scores. Finally, the Descriptive Metadata objects are marshaled into a Discovery Response XML structure and returned to the Ocean application (see section 6.4.1.2). Once the Discovery Response is received by an Ocean application, its local application logic performs dynamic, in-situ component selection and composition as per the Ocean application model presented in section 5.2.

6.5 Chapter Summary

This chapter discussed the Ocean Registry's approach for indexing, storing and discovering Contextualized Resources. As discussed in section 4.3, Contextualized Resources represent arbitrarily complex data structures that cannot be effectively indexed or queried using classical key, range or proximity techniques. Towards this end, we leveraged the similarity modeling capabilities of the Context Metadata interface described in section 4.3.2. The chapter began with a discussion of similarity search algorithms and comparison metrics. Based on the related work discussion, we then defined the Index Manager abstraction, which encapsulates the indexing techniques and data models required to support efficient queries for a given context domain. Notably, Index Managers are provided by Context Experts using the Context Metadata contribution approach introduced in section 5.4. Next, we defined the Ocean persistence model, which allows Contextualized Resources to be efficiently stored and indexed for rapid retrieval. This section included a persistence example based on a B+-tree approximate nearest neighbor search approach called iDistance. Next, we developed Ocean's multi-feature similarity search approach, whereby contextually-relevant Ocean Metadata can be retrieved from the Ocean Registry and returned to Ocean applications as Descriptive Metadata. Related, this section presented the Ocean Registry's Discovery API and Discovery Protocol. The section concluded with a discussion of Query Object instantiation; an overview of multi-feature similarity search techniques; an Ocean-specific adaptation of the Threshold Algorithm; and a description of Ocean's multi-feature similarity search process (including Resource scoring and Descriptive Metadata generation).

Chapter 7
Leveraging the Ocean Community

7.1 Introduction

As described throughout this dissertation, the Ocean approach provides a conceptual and practical foundation for the development of Web-centric context-aware systems. By co-opting existing context sources, network infrastructure and distributed middleware, Ocean promotes the development of wide-area context-awareness techniques through significant developer adoption and user participation. However, the complexity and scale of many real-world environments impose additional challenges for effective Contextualized Resource discovery and selection. This chapter discusses two important challenges in this regard. First it addresses c*ontext mismatch*, which refers to the situation where an Ocean application may not be capable of generating the native context data necessary to discover Contextualized Resources in a given environment; reducing query effectiveness. Next, this chapter addresses *information overload*, which refers to the situation where a prohibitively large number of similarly scored Discovery Response results become difficult to differentiate based on Descriptive Metadata alone; resulting in ineffective Resource selection.

While context mismatch and information overload are not well-addressed by existing context-aware systems, these important topics have been explored in other large-scale networked systems. Notably, the emergence of the Internet and Web has given rise to powerful community-based computational models that have proven adept at solving large-scale information filtering challenges. To explore the application of such approaches in Ocean, this chapter presents two preliminary techniques for leveraging the participation of various Ocean stakeholders as a means of addressing the information filtering challenges facing large-scale context-aware systems. First, in section 7.2, we introduce a preliminary approach for overcoming context mismatch. This approach leverages the Discovery Requests provided by Ocean applications to automatically model context information in large-scale environments. Community-modeled context information is then used to automatically expand subsequent Discovery Requests with supplemental query terms to help improve query effectiveness. Next, in section 7.3, we introduce a preliminary approach for overcoming information overload. This approach captures both implicit and explicit Resource preference information from large numbers of Ocean Users as a means of personalizing discovery results. Resource personalization is facilitated by the development of an Ocean-specific Recommender Engine capable of integrating a variety of recommender algorithms.

7.2 Context-aware Query Expansion

As described in section 5.5, the Ocean Registry provides an open Contextualized Resource contribution model whereby *any* Contextualizer can contextualize *any* Resource with *any* combination of Ocean Metadata. For example, a Contextualizer may contextualize iCalendar data using Context Metadata such as geo-position and a valid time interval (see section 4.4); however, many other Contextualizers may contextualize the *same* Resource using a variety of other Context Metadata (e.g. device proximity, temperature or ambient light levels). As previously discussed, this community-based annotation technique is used to promote the contribution of vast amounts of Contextualized

Resources. For example, a context-aware crawler could automatically generate large numbers of Contextualized Resources from existing Web-based data sources (see section 8.4). While open contribution addresses large-scale Contextualized Resource generation and maintenance (see section 5.5), the scale of contribution, the diversity of available Context Metadata and the heterogeneity of Ocean devices may decrease query performance due to context mismatch. Briefly, context mismatch refers to the situation where an Ocean Discovery Request's query terms may not sufficiently match the Context Metadata used to create Contextualized Resources. In such cases, query effectiveness is diminished and contextually-relevant Resources may remain invisible to Ocean applications.

Recall that as Contextualized Resources are stored within the Ocean Registry, its Persistence Framework generates multiple indexes to facilitate fast lookup (using the `IndexManager` abstraction described in section 6.3.1). As described in section 6.4.1, Ocean applications formulate Discovery Requests based on application-specified search parameters and query terms that include locally derived native context data (NCD). In real-world scenarios, NCD are often complex, domain-specific, unsystematically organized and unpredictably available [21]. Further, while Aladin-based context modeling is capable of operating across multiple administrative domains (see section 3.2.2), client-centric context acquisition is inherently limited to the capabilities of a given host device. For example, while many devices are capable of position determination based on GPS hardware, this capability is far from universal. Further, even if such capabilities are present, the resultant NCD may be inaccurate or unavailable under certain conditions (e.g. indoors or in "urban valleys"). Given the device heterogeneity common to Web-scale scenarios, incoming Discovery Requests may not include the NCD necessary to produce effective query results using Ocean's multi-feature search approach described in section 6.4.4. This section presents our preliminary approach for addressing this issue by means of community-participation.

7.2.1 Background and Related Work

Important aspects of context mismatch have been studied in traditional information retrieval (IR) research in a related form known as *word mismatch*. Word mismatch "refers to the phenomenon that the users of IR systems often use different words to describe the concepts in their queries than the authors use to describe the same concepts in their documents" [374]. As Furnas et al. [113] noted in 1987, people use the same words to describe a search object less than 20% of the time. This so-called *vocabulary problem* gives rise to word mismatch. While the effect can be slightly mitigated through the use of longer queries, short queries are becoming increasingly common in Web-based search [205]. Moreover, large sets of complex search objects (e.g. long documents) can be difficult to meaningfully rank based on limited query terms [374]. Hence, the severity of the vocabulary problem has motivated the development of *query expansion* techniques that augment queries with supplemental search terms in order to improve query effectiveness [51].

Two general categories of query expansion have been studied. First, in *global query expansion*, various statistical analyses are applied to an entire collection of search objects (e.g. finding the co-occurrence of all possible pairs of terms). Common examples of global techniques include Boolean term decomposition [363], statistical factor analysis [85] and formal concept analysis [53]. Additional global techniques have been developed to help users select appropriate search terms by manually or

automatically deriving a thesaurus of related words (based on the initial query terms), which is presented to the user during an iterative search process [52, 71]. In a related technique, thesauri have also been used to supplement user queries with additional terms or to reweight existing terms [338]. While the effectiveness of thesauri-based query expansion has been questioned [51], more sophisticated query expansion techniques have been developed to exploit the content relationships between the various documents of a collection; however, as global techniques operate over the entire collection, they are often computationally expensive and have not demonstrated significant improvements in query results [51].

In contrast to global techniques, *local query expansion* demonstrates improvements in query effectiveness and offers lower computational complexity characteristics [51]. Instead of calculating term relevancy for all documents within a collection, local techniques derive supplemental query terms from the top ranked documents for a given query [12, 77]. The simplest local technique, called local feedback (or pseudo-feedback), assumes that the top ranked documents for a query are relevant and applies standard relevance feedback mechanisms to expand the query terms [374]. The relevance feedback algorithm simply adds common terms from highly ranked documents and then reweights the results based on term frequencies [278]. Work from Croft and Harper suggested a similar approach that uses the top-ranked documents to estimate the probabilities of the query terms, but does not add additional terms [77]. While local query expansion is often characterized by lower computational complexity, some techniques can suffer from erratic performance when the top-ranked documents are not relevant [374].

Based upon the foundation of local feedback, several recent proposals have been developed that demonstrate improved query effectiveness across a variety of problem domains [213]. In many approaches, a modified version of Rocchio's formula [271] serves as the foundation for term reweighting [278]. A common modification is described below, as presented in [51] :

$$Q_{new} = \alpha \cdot Q_{orig} + \frac{\beta}{|R|}\sum_{r \in R} r - \frac{\gamma}{|\acute{R}|}\sum_{\acute{r} \in \acute{R}} \acute{r}$$

where Q_{new} is a weighted term vector for the expanded query; Q_{orig} is a weighted term vector for the original unexpanded query; R and $\acute{R}$ represent the sets of relevant and non-relevant documents (respectively); r and $\acute{r}$ are term weighting factors extracted from the relevant and non-relevant document sets (respectively); and α, β and γ are multipliers used to tune the approach. Using this formula, the weights in each vector are computed by a weighting scheme applied to the whole collection. According to [51], if the query expansion technique is constrained to highly ranked documents, the aforementioned formula reduces to:

$$Q_{new} = \alpha \cdot Q_{orig} + \frac{\beta}{|R|}\sum_{r \in R} r$$

where R refers to the collection of top ranked documents that are considered relevant to the original query. While computationally efficient, the above approach has been criticized as being overly collection-centric rather than query-centric [51].

While a comprehensive treatment of query expansion is beyond the scope of this dissertation, we suggest that the addition and weighting of supplemental query terms can be used to improve query effectiveness for Ocean applications. Importantly, Ocean's vocabulary problem precludes the direct application of existing query expansion techniques as current approaches are designed for text-based scenarios, where the search objects (i.e. documents) often share identical terms (i.e. words and concepts) across the collection [213]. In contrast, even though a large number of Ocean's search objects (i.e. Contextualized Resources) may be relevant to a given Discovery Request, they may not share any terms with the Context Metadata within the query. Recall that Ocean's basic multi-feature similarity search approach considers only Ocean Metadata that have *at least one* element of Context Metadata in common with a given Query Object (see section 6.4.4). Without a minimal overlap between a given query's Context Metadata and the Context Metadata of persisted Ocean Metadata, contextually-relevant Resources will remain hidden from Ocean applications and unavailable for in-situ composition.

7.2.2 The Emergence of Community-centric Context Modeling

Recently, a multi-organizational initiative, called PlaceLab, has demonstrated a highly scalable approach for modeling context information in large-scale, real-world environments [291]. In contrast to many context-aware infrastructures, PlaceLab is designed to: (1) work over a wide area, indoors and out; (2) run on commodity devices; (3) observe privacy needs; (4) support standard programming interfaces; and (5) make accuracy a secondary goal. Towards these ends, PlaceLab aims to collect and model information regarding the geo-location and response-rate histogram patterns of commonly occurring radio signals in real-world environments (termed *beacons*). In PlaceLab, information regarding 802.11 access points, Bluetooth devices and GSM cell towers is captured by a community of volunteers that use specially developed "Stumbler" software to search for radio signals in large geographic areas using vehicles (termed *war-driving*). During field trials, PlaceLab volunteers constructed a database of over 35,000 Wi-Fi and 7,000 GSM beacons by war-driving 4,350 kilometers throughout the Seattle metropolitan area [151]. Using these results, application developers are able to perform position estimations using a PlaceLab client library and related Web service that provides access to the beacon information. Additionally, the PlaceLab client supports access to the wigle.net[42] beacon database, which provides an additional six-million beacon mappings worldwide. By using the PlaceLab client software, the position of mobile devices can be estimated without the need for GPS hardware or in conditions where GPS signal quality is poor or unavailable.

PlaceLab researchers evaluated their infrastructure with nine detailed user studies and several empirical evaluations that included the collection of comments from a public developer forum [66]. The researchers studied the effects of beacon density within three geographic regions, including urban, suburban and residential. Notably, their experiments demonstrated that a median positioning error of 15 – 20 meters could be achieved in urban and suburban areas, provided that 3 distinct beacons could be detected within 10 seconds (with 100 percent coverage). In more sparsely populated residential areas, where beacon density is lower, PlaceLab clients demonstrated a median positioning error of approximately 30 meters (with 100 percent coverage). Several additional experiments were

[42] http://wigle.net/

conducted using GSM beacons alone. The results from these studies show that PlaceLab could achieve a median positioning error of 94 meters in downtown areas and 196 meters in residential areas using a single GSM network. Using multiple GSM networks, the median positioning error was 65-134 meters in downtown areas. The GSM results were characterized using three positioning algorithms, including a centroid algorithm that does not model radio propagation; fingerprinting; and Monte Carlo localization with a Gaussian Processes signal propagation model. An overview of the PlaceLab GSM trace collection for Seattle is shown below in Figure 59.

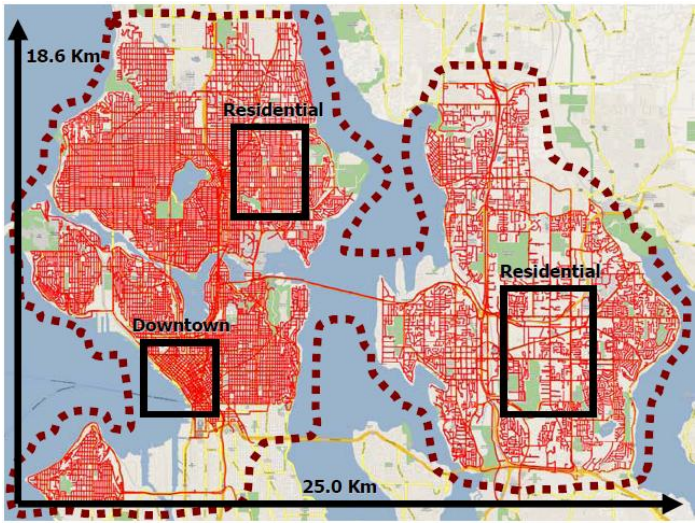


Figure 59: Example PlaceLab GSM trace collection for Seattle (from [66])

In addition to geo-positioning, PlaceLab researchers also studied methods of detecting *places* by means of beacon signatures. Towards this end, the researchers developed an algorithm, called BeaconPrint, which identifies previously encountered places (e.g. home or work) using the response-rate histograms of previously detected beacon patterns [150]. Experiments conducted using BeaconPrint demonstrated a 90 percent recognition rate for frequently visited places. For less frequently visited places, BeaconPrint achieves a 63 percent recognition rate for places visited once and an 80 percent recognition rate for places visited twice. While BeaconPrint does not provide geo-location information, it does provide relatively accurate symbolic location information and does not rely on a connection to the PlaceLab server.

Perhaps one of the most interesting aspects of the PlaceLab approach is its focus on leveraging *community participation* as a means of solving a computationally and logistically difficult problem. Specifically, PlaceLab addresses the daunting challenge of large-scale beacon discovery and maintenance by promoting a grass-roots, volunteer-based war-driving community [291]. In this regard, the PlaceLab contribution architecture has been characterized as an "open source approach to content generation" [181] that distributes context acquisition tasks to geographically dispersed volunteers. Individual beacon trace contributions are then aggregated by the centralized PlaceLab server and made available to clients via an open API. Mediation between the producers of PlaceLab data (i.e. the war-drivers) and the consumers of that data (i.e. the mobile clients) is provided by the PlaceLab infrastructure; however, similar to the Ocean application model, clients are free to use derived positioning information as needed.

The PlaceLab initiative can be understood as a type of *volunteer computing* system that provides collection and processing of beacon data using shared computing resources. Broadly, volunteer computing "allows high-performance parallel computing networks to be formed easily, quickly, and inexpensively by enabling ordinary Internet users to share their computers' idle processing power without needing expert help" [284]. A canonical example of volunteer computing is the SETI@home project [10], which leverages shared computing resources to analyze radio signals from space. Before the SETI@home project, the analysis of the data output from modern radio telescopes required specialized, expensive and dedicated computing systems. As a means of improving analysis power and decreasing the associated computing costs, Berkley researchers developed the SETI@home project as an experimental distributed computing architecture that divides space telescope data into fixed work units that can be distributed via the Internet to a related client program running on volunteers' computers. The distributed clients calculate and return results to the U.C. Berkeley server infrastructure, where they are collected and new tasks are assigned. Notably, the general public can participate by simply downloading and installing a client program, which exploits unused computing cycles by running as a screensaver or continuously in the background.

The results of the SETI@home project have been remarkable in terms of participation and computational performance. Within weeks of the public project announcement in 1998, over 400,000 volunteers had preregistered to participate [10]. Since its release in May 1999, the project has quickly garnered participants; growing from an initial user base of 200,000 to over 3.91 million users in 2002. At its peak, the original SETI@home infrastructure was capable of generating an average throughput of over 27.36 TeraFLOPS [10]. This success prompted SETI@home researchers to develop a generalized architecture for public-resource computing, called the Berkeley Open Infrastructure for Network Computing (BOINC) [9]. As of November 2008, BOINC generates an average processing throughput of 1,487 TeraFLOPS derived from the shared computing resources of 320,372 volunteers. Prominent BOINC projects include Rosetta@home[43] (biology), Climateprediction.net[44] (earth sciences) and Einstein@Home[45] (physics).

While differing in overall approach, PlaceLab shares some similarities with public-resource computing approaches such as BOINC. Similar to BOINC, PlaceLab leverages the shared computing resources of a community as a means of solving difficult computational problems. In PlaceLab, shared resources include mobile devices (i.e. Stumbler-equipped laptops), context detection hardware (e.g. GPS receivers), a means of mobility (e.g. cars and fuel) and volunteer time. Moreover, similar to SETI@home, PlaceLab provides a centralized computing infrastructure that aggregates results and provides a related public API. However, unlike the BOINC contribution model, which places few demands on participants beyond the installation of an unobtrusive software client, the PlaceLab contribution model required a significant investment of time, effort and money on the part its volunteers.

[43] http://boinc.bakerlab.org/rosetta/
[44] http://climateprediction.net/
[45] http://einstein.phys.uwm.edu/

Although the PlaceLab architecture demonstrated promising initial results, its contribution model has been criticized [181]. In PlaceLab, war-driver volunteers are required to provide a laptop running specially developed software along with appropriate radio hardware. Notably, meaningful beacon detection often necessitated the purchase of additional equipment such as GPS devices, Bluetooth radios and multiple GSM modems (see Figure 60). In addition, volunteers faced considerable time requirements and fuel costs when war-driving large geographic areas [66]. The cost and effort associated with collecting beacon traces is further exacerbated by issues related to the ownership, licensing, and copyright of collected data [181]. For example, the licensing terms of the PlaceLab Stumbler software state that "The access point data you submit becomes the property of Intel Corporation for use in this research"[46]. Licensing restrictions, combined with a general distrust of Intel's corporate sponsorship of the project, resulted in PlaceLab being "unable to both attract enough users and obtain the quality and breadth of data they desired to make the database widely useful" [181].

Figure 60: A PlaceLab war-driving laptop outfitted with specialized radio equipment (including one WiFi card, two GPS units, and three Sony Ericsson GM28 GSM modems)

7.2.3 Towards Volunteer-based Context Modeling in Ocean

In Ocean, we propose that community-generated context information can be used to expand Discovery Requests to help overcoming context mismatch. As discussed in section 7.2.1, various query expansion techniques have been developed to generate and ensure the relevancy of supplemental search terms; however, existing approaches are designed for text-based information retrieval and do not accommodate context-aware query term modeling. In this section, we discuss our preliminary approach for using the Ocean User community to acquire and model context information for use in automatic query expansion. We base our approach on an extension of the PlaceLab infrastructure previously introduced. We suggest that PlaceLab's community-based approach demonstrates how large amounts of context information can be generated and maintained for real-world environments. Related, we suggest that volunteer-based computing infrastructures such as BOINC illustrate how a low-cost contribution model can result in self-sustaining volunteerism and formidable computational capabilities.

[46] http://placelab.org/data/disclaimer.html

As discussed in section 6.4.1, Ocean applications discover contextually-relevant Resources using the Ocean Registry's Discovery API. In order to form Discovery Requests, Ocean applications first acquire and model native context data (NCD) from their local environment using the techniques described in section 3.2.2. In Ocean's client-centric approach, the type of NCD modeled depends on the capabilities of a given host device and the available sources of environmental context information. Notably, host devices may provide *several* context acquisition sensors; hence, multiple NCD may be generated during the context modeling process. For example, a given mobile device may simultaneously detect multiple types of nearby radio transceivers, an ambient light value, a temperature reading and geographic positioning information. In order to improve Resource Discovery effectiveness, an Ocean application may provide *multiple* NCD as query terms within Discovery Requests. Importantly, Discovery Requests are performed by *all* Ocean applications using the Ocean Registry; resulting in an incoming stream of heterogeneous contextual data from diverse device types and real-world environments. Notably, the query terms within individual Discovery Requests can be understood as "snapshots" of the context information encountered by each distributed Ocean application. In this regard, each Ocean User behaves similar to a PlaceLab war-driver; proactively modeling context information in-situ and then providing resultant NCD to the Ocean Registry in Discovery Requests. However, unlike the PlaceLab Stumbler software, which can detect only a few well-known context types (e.g. 802.11 beacons and GPS coordinates), heterogeneous Ocean applications include a variety of NCD that are transformed into *arbitrary* Context Handlers during Resource Discovery (see section 6.4).

In PlaceLab, establishing location information for discovered beacons is accomplished by domain-specific algorithms that take into account the semantics of the underlying context data [66]. For example, the PlaceLab Stumbler trace shown in Figure 43 provides the following domain-specific information: a beacon of type "**`WIFI`**" that is identified by a MAC address ("**`00:0f:b5:27:2e:a6`**") and name ("**`KVK NETGEAR`**"); and location information that includes latitude ("**`47.67006963632125`**") and longitude ("**`-122.29616917072389`**"). In order to provide meaningful associations between the discovered beacon and its related geo-coordinates, PlaceLab provides support for parsing the Stumber file format [212] and provides domain-specific mechanisms for associating geo-positioning information using the radio signal propagation characteristics of a given beacon type (e.g. WIFI or GSM) [66].

```
TYPE=WIFI|ID=00:0f:b5:27:2e:a6|NAME=KVK NETGEAR|
LAT=47.67006963632125|LON=-122.29616917072389|
VCNT=0|ACNT=0|ANCHORED=1|RAD=75.0|GCNT=1
```

Figure 43: An example PlaceLab Stumber trace (from the dataset discussed in [66])

As previously discussed, sophisticated proximity algorithms utilize radio signal strength values to infer a distance to the signal's source [188]. In the case of common omni-directional antennas (which many 802.11 transceivers employ) a single signal strength value may not indicate precisely where an associated geo-coordinate should be assigned. In this regard, several recent signal triangulation approaches have suggested to help improve localization precision by using *multiple* radio signal

sources [19, 188, 195]. However, even with a single 802.11 beacon, its relatively short range (e.g. 70 to 100 meters outdoors) results in relatively accurate position estimations [151]. In contrast, GSM signals exhibit a comparatively stable deployment configuration; however, the propagation range of GSM signals exceeds that of 802.11 transceivers by as much as 70 times [66]. Hence, PlaceLab's association algorithms rely on multiple tower "sightings," exploit signal strength information and involve multiple GSM providers where possible. In the next section, we describe Ocean's *generalized* approach for domain-specific context association discovery and modeling.

7.2.4 Context Association Discovery and Modeling

The previous section discussed PlaceLab's use domain-specific algorithms as a mechanism for creating meaningful associations between well-understood context information. To support query expansion within the Ocean Registry using *arbitrary* context information, we have developed several extensions to the PlaceLab approach. First, we enhanced the Discovery Request XML schema to include a mechanism for controlling context sharing. Next, to support domain-independent discovery of context associations, we developed an **`AssociationDiscoverer`** interface that Context Handlers may implement as a means of encapsulating context association discovery in accordance with the similarity model described in section 4.3.3. Next, we extended the Ocean's Index Manager interface to support the persistence and discovery of **`ContextAssociation`** objects, which represent discovered associations between incoming native context data (NCD). Finally, we developed a related Association Discovery Framework (ADF) that supports the creation, discovery and management of **`ContextAssociation`** objects within the Ocean Registry. This section describes each of these extensions in detail.

The foundations of Ocean's context association approach are the **`AssociationDiscoverer`** interface and **`ContextAssociation`** object shown in Figure 61. In order to support domain-independent discovery of **`ContextAssociations`**, the **`AssociationDiscoverer`** interface provides the **`discoverAssociations`** method, which takes a set of candidate Context Metadata objects and (potentially) returns a set of related **`ContextAssociation`** objects. The association semantics resulting from the **`discoverAssociation`** method are encapsulated by the **`ContextAssociation`** object, which is used to model domain-specific association information that is used during query expansion (described shortly). These foundations allow Context Experts to implement arbitrarily complex mechanisms for discovering and modeling associations using the Context Handlers contained within a given Query Object (see section 6.4.2).

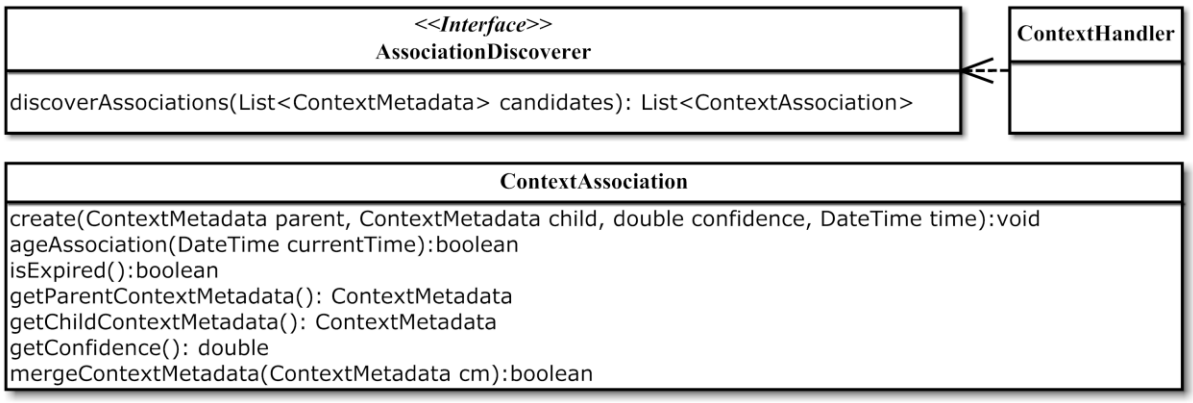


Figure 61: The AssociationDiscoverer interface and ContextAssociation object

Regarding Figure 61, the **`ContextAssociation`** object is defined as follows:

- **`create`**: Instantiates a **`ContextAssociation`** between the *parent* **`ContextMetadata`** object (i.e. the Context Handler whose **`discoverAssociation`** was called) and a *child* **`ContextMetadata`** object (i.e. a candidate Context Handler determined to be associated to the parent in some domain-specific way) using an initial confidence interval and creation **`DateTime`**.
- **`ageAssociation`**: Ages the **`ContextAssociation`** according to the incoming **`DateTime`**. The **`ContextAssociation`** uses the **`DateTime`** to calculate a new confidence interval or expire the association (expiration indicates that the **`ContextAssociation`** is no longer valid). If the **`ContextAssociation`** expires, the **`ageAssociation`** method returns true; otherwise it returns false.
- **`isExpired`**: Checks if the **`ContextAssociation`** is expired. If so, this method returns true; false otherwise.
- **`getParentContextMetadata`**: Returns the parent **`ContextMetadata`** object contained within the **`ContextAssociation`**.
- **`getChildContextMetadata`**: Returns the child **`ContextMetadata`** object contained within the **`ContextAssociation`**.
- **`getConfidence`**: Returns a double representing the confidence interval existing between the parent and child Context Metadata objects (note that confidence intervals are always expressed in the unit interval [0,1]).
- **`mergeContextMetadata`**: Merges the incoming **`ContextMetadata`** object with the **`ContextAssoication`**, resulting in a new confidence interval value, a modification of the underlying child **`ContextMetadata`** entity or expiration (discussed shortly). If the association expires during the **`mergeContextMetadata`** call the method returns true; otherwise it returns false.

To validate the above approach, the Ocean Reference Implementation (Ocean RI) provides a realization of the **`AssociationDiscoverer`** interface and **`ContextAssociation`** objects as shown in Figure 62. (Note that implementation-specific methods may be included in the figure below; however, in the interest of clarity, these are not described.) Notably, the Ocean RI provides a post processing framework capable of handling a variety of background processing tasks after Resource Discovery has been completed for a given request. Post processing tasks implement the **`IPostProcessor`** interface, which allows for generic task handling and scheduling. Related, the Ocean RI provides a prototype **`AssociationDiscoveryPostProcessor`**, which utilizes the **`IAssociationDiscoverer`** interface implemented by Context Handlers to discover **`ContextAssociation`** objects using the Ocean Registry's Association Discovery Framework (described next).

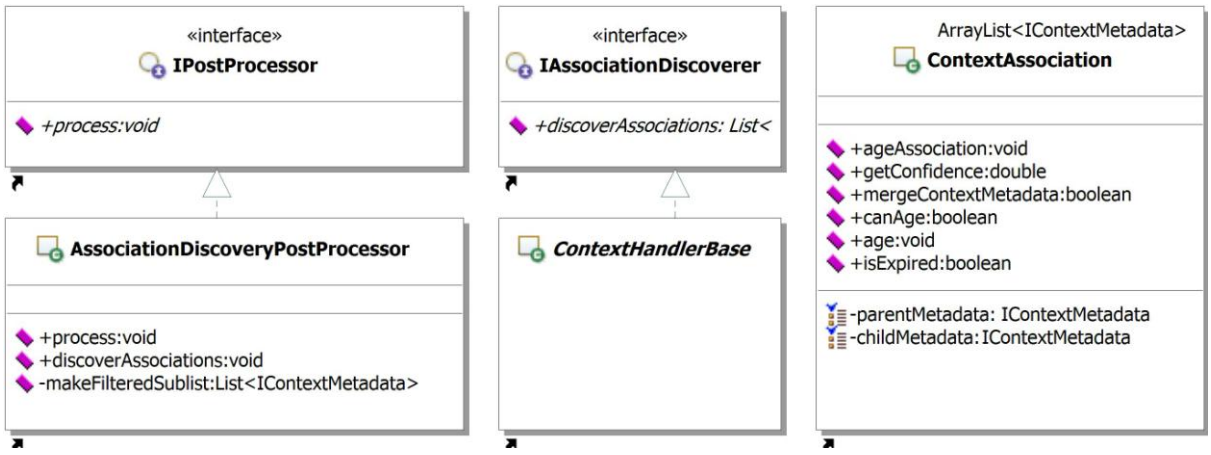


Figure 62: Association discovery support within the Ocean RI

As the NCD contained within Discovery Requests may invoke privacy concerns and dissuade participation, several privacy mechanisms are employed. First, as described in section 5.2, Ocean applications are responsible for modeling context and deciding which NCD to provide during Discovery Requests (allowing clients to control which context information is sent). Next, Ocean Users are not required to provide any personally identifiable information in order to use the Ocean Registry (i.e. anonymous access is allows). Next, Ocean applications may access the Ocean Registry using privacy mechanisms such as anonymous proxies, virtual private networks or onion routing[47]. Next, Ocean Users must explicitly state they wish share their context information with the Ocean community (i.e. context sharing is inactive by default). Finally, following the context obfuscation work by Wishar et al. [359], all shared context information is anonymized before association discovery is performed. Anonymized information includes the originating IP address, developer tokens and user identifiers (if present). While such privacy mechanisms may not be sufficient for all users, the benefits of providing detailed context information within Discovery Requests (to help improve query effectiveness) may outweigh the privacy costs for some users [171].

To support the instantiation, persistence and management of **`ContextAssociation`** objects within the Ocean Registry, we have developed an Association Discovery Framework (ADF). Broadly, the ADF attempts to discover meaningful associations between the native context data (NCD) contained

[47] http://www.torproject.org/

within incoming Discovery Requests using `ContextHandlers` that implement the previously described `AssociationDiscoverer` interface. For completeness, we note that our preliminary approach does not include indexing support; however, we suggest that the `IndexManager` model described in section 6.3.1 can be adapted to provide efficient `ContextAssociation` indexing and discovery using techniques similar to those discussed in section 6.4.4. An overview of the ADF is shown in Figure 63.

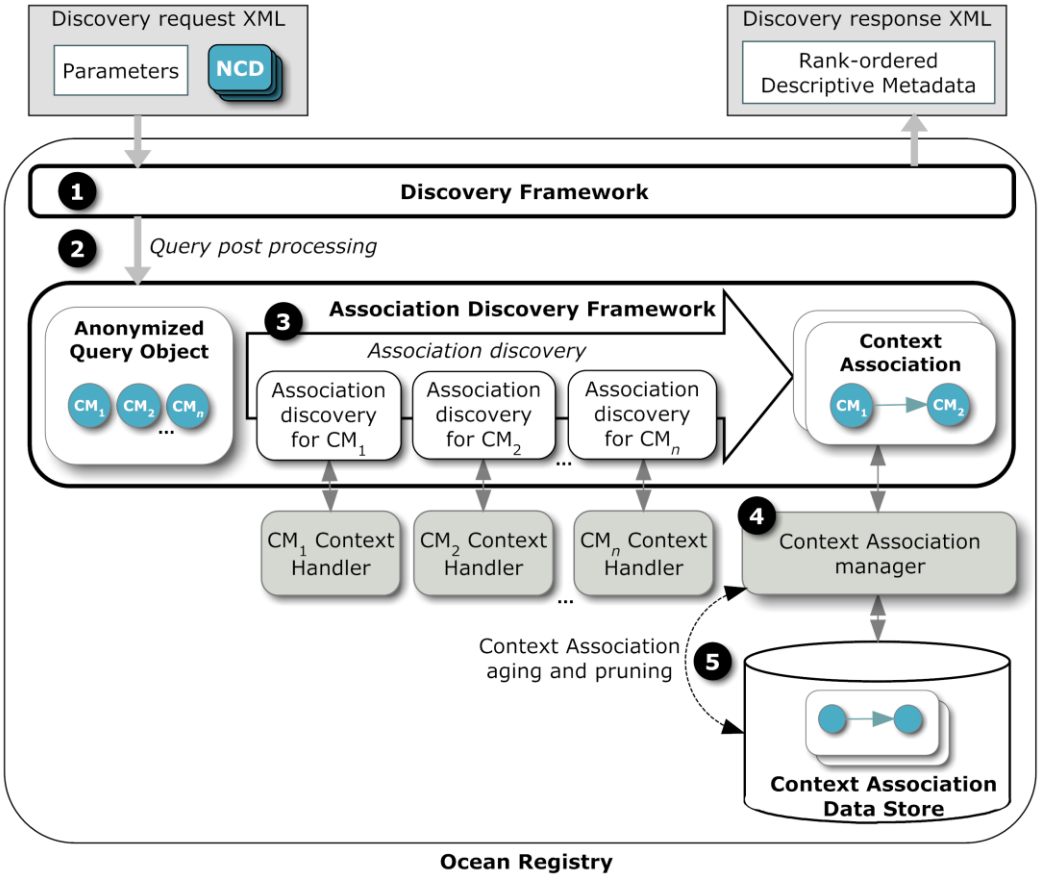


Figure 63: Overview of the Association Discovery Framework

Regarding Figure 63, the ADF operates as follows:

1. Resource Discovery processing occurs using the Discovery Framework described in section 6.4. Discovery results are returned to the requesting Ocean application as rank-ordered Descriptive Metadata.
2. Once discovery processing is complete, the Query Processor determines if context sharing is allowed. If so, the Query Object is anonymized and passed to the ADF for post processing (discussed shortly).
3. During post processing, the ADF performs "association discovery" for each Context Metadata present within the Query Object. During association discovery, each associated Context Handler implementing the `AssociationDiscoverer` interface receives all other Context Handlers contained within the Query Object through the `discoverAssociation` method. Resultant `ContextAssociation` objects are returned to the ADF.

4. The collection of discovered **ContextAssociation** objects is passed to the Context Association Manager (CAM), which queries its associated data store to determine if the given **ContextAssociation** has been previously persisted. If so, the **ContextAssociation's** child Context Metadata object is merged with the previously persisted **ContextAssociation** using the **mergeContextMetadata** method. If not, the newly discovered **ContextAssociation** is persisted within the data store as a new object.
5. In addition, the CAM continually ages and prunes the **ContextAssociation** objects persisted within the Context Association Data Store. Aging informs each persisted **ContextAssociation** object of the passage of time; possibly resulting in **ContextAssociation** expiration (meaning that the **ContextAssociation** is no longer valid). Pruning refers to the removal of expired **ContextAssociations** from the data store.

To illustrate the ADF concepts introduced above, a simple association discovery example is now presented. As per the Ocean discovery approach described in section 6.4.1, Discovery Requests are converted into Query Objects using instantiation mechanism similar to those described in section 5.3.3. Recall that instantiated Query Objects include search parameters and a set of Context Metadata representing the NCD extracted from the request. For example, the **query_term** elements in the sample Discovery Request show in Figure 64 will be converted into two Context Handlers, including a **GEOPointHandler** representing geographic location; and a **WIFIPlaceLabStumblerHandler** representing an 802.11 radio beacon identified by MAC address, name, RSSI value, etc.

```
<?xml version="1.0" encoding="UTF-8"?>
 <ocean_resource_request version="1.0" >
 <query_parameters max_results="50"
      domain_restrict=""
      annotator_restrict="dcarlson"
      share_context="true">
  <type><![CDATA[mime:text/calendar]]></type>
  <search_text><![CDATA[Example search text]]></search_text>
 </query_parameters>
 <query_terms>
  <context_data>
  <value><![CDATA[
   <gml:Point>
   <gml:pos>53.873488,10.686607</gml:pos>
   </gml:Point>]]>
  </value>
  </context_data>
   <context_data>
   <value><![CDATA[
    TYPE=WIFI|TIME=1096470064731|ID=00:09:5b:de:fa:7a
    |NAME=NETGEAR|RSSI=-88|WEP=true|INFR=false]]>
   </value>
   </context_data>
 </query_terms>
</ocean_resource_request>
```

Figure 64: A sample Discovery Request with context sharing enabled

Once query processing is complete for the sample Discovery Request, it is checked for context sharing permission and passed to the ADF. As the Query Object enters the ADF, it is anonymized by removing all personally identifiable information such as search parameters, user IDs, security tokens, IP addresses, etc. Next, Context Handlers implementing the **AssociationDiscoverer** interface are

identified for each **ContextMetadata** object within the Query Object. Once identified, the **discoverAssociation** method is called for each of the Context Handlers as previously identified. For example, the **WIFIPlaceLabStumblerHandler** will have its **discoverAssociation** method called using the **GEOPointHandler** as a parameter. Likewise, the **GEOPointHandler** will have its **discoverAssociation** method called using the **WIFIPlaceLabStumblerHandler** as a parameter. In this example, the **WIFIPlaceLabStumblerHandler** returns an instantiated **ContextAssociation** object with the **WIFIPlaceLabStumblerHandler** as the parent and the **GEOPointHandler** object as the child (with an initial confidence interval set by the **WIFIPlaceLabStumblerHandler's** domain logic). In this example no **ContextAssociation** is discovered by the **GEOPointHandler**; however, such functionality could be implemented.

Once a **ContextAssociation** object is discovered for a given Discovery Request, the Context Association Manager (CAM) checks its data store to determine if the **ContextAssociation** has been previously discovered and persisted (based on the parent Context Handler). In this example, no preexisting **ContextAssociation** exists; hence, the newly instantiated **ContextAssociation** is persisted (using its default configuration) within the CAM's data store. The initial configuration of a given **ContextAssociation** represents a validity duration and confidence interval appropriate for a single sighting. For example, given the relatively low deployment stability characteristics of 802.11 access points (as compared to a GSM towers) a **ContextAssociation** generated for a single sighting of an 802.11 signal and related geo-position may be configured with a short validity duration (e.g. 2 days) and a low confidence level (e.g. .25). Additional sightings of the same **ContextAssociation** will be integrated into the CAM's data store using the **mergeContextMetadata** method, whereby the **ContextAssociation** may increase its confidence interval, increase its validity time or improve its association accuracy in response. For example, if hundreds of Ocean applications encounter the same 802.11 access point near the same geographic location over several months, the related **ContextAssociation** will be continually merged with similar Context Handlers using its **mergeContextMetadata** method. In response, the **ContextAssociation** may determine that its confidence level should increase (e.g. .75) and that its validity time should be extended (e.g. 3 weeks). Similarly, various domain-specific implementations of the **AssociationDiscoverer** and **ContextAssociation** interfaces can provide association discovery and management implementations suited to the specifics of a given context domain.

7.2.5 Context-aware Query Expansion

The ADF described in the last section provides the foundation for Ocean's context-aware query expansion approach. As communities of autonomous Ocean applications make Discovery Requests using the Ocean Registry's Discovery API, the ADF maintains a continually evolving data model of **ContextAssociations** that can be used to automatically expand incoming Discovery Requests with supplemental query terms. In order to improve the quality of supplemental search terms, the ADF exploits the domain knowledge encapsulated within Context Handlers implementing the **AssociationDiscoverer** interface. This section describes an extension to the basic Ocean multi-feature similarity search (MFSS) approach (see section 6.4.4), which aims to help Ocean applications overcome context mismatch by using the ADF to discover supplemental Context Metadata.

Recall from section 6.4 that the Ocean MFSS approach is based on an adaptation of the Threshold Algorithm [101], which exploits the `IndexManager` abstraction presented in section 6.3.1. As previously discussed, the `IndexManager` abstraction provides domain-optimized, sorted access to collections of Context Handler indexes; each representing a context feature associated with a persisted Ocean Metadata entity. To process a given Query Object, set installed `IndexManagers` within the Ocean Registry are used to produce *m* sorted lists (1 per Context Metadata type), where each list represents a graded set of pairs *(x,s)*, where *x* is the object (i.e. a reference to the parent Contextualized Resource) and *s* is the real number grade for a given context feature (i.e. Context Metadata entity). List generation is accomplished using each `IndexManager's topSearch` method. Related, the `IndexManager probe` method is used to provide random access to Contextualized Resource similarity grades for each of the *m* sorted lists. (Note that search space reduction is performed as per section 6.4.4.)

To augment Ocean MFSS with context-aware query expansion, we extend the functionality of the previously introduced ADF by introducing a `QueryExpander` object within the Query Processor introduced in section 6.4. (Note that query expansion is understood as a pre-processing function; requiring only superficial changes to the Query Processor and Query Object.) Next, we posit a mechanism whereby the *top k* `ContextAssociation` objects for a given Context Metadata object can be discovered from the Context Association Manager's (CAM's) data store. The `QueryExpander` object is integrated within the Query Processor after the Request Factory, which is responsible for decomposing XML-based requests into Ocean-compatible objects as described in section 6.4.2. After the Query Object is instantiated for a given Discovery Request, the `QueryExpander` searches the ADF to discover the *top k* `ContextAssociation` objects for each of the Context Metadata objects that were derived from the NCD within the Discovery Request. The resulting Supplemental Metadata (SM) are integrated into the Query Object to form an Expanded Query Object as shown in Figure 65.

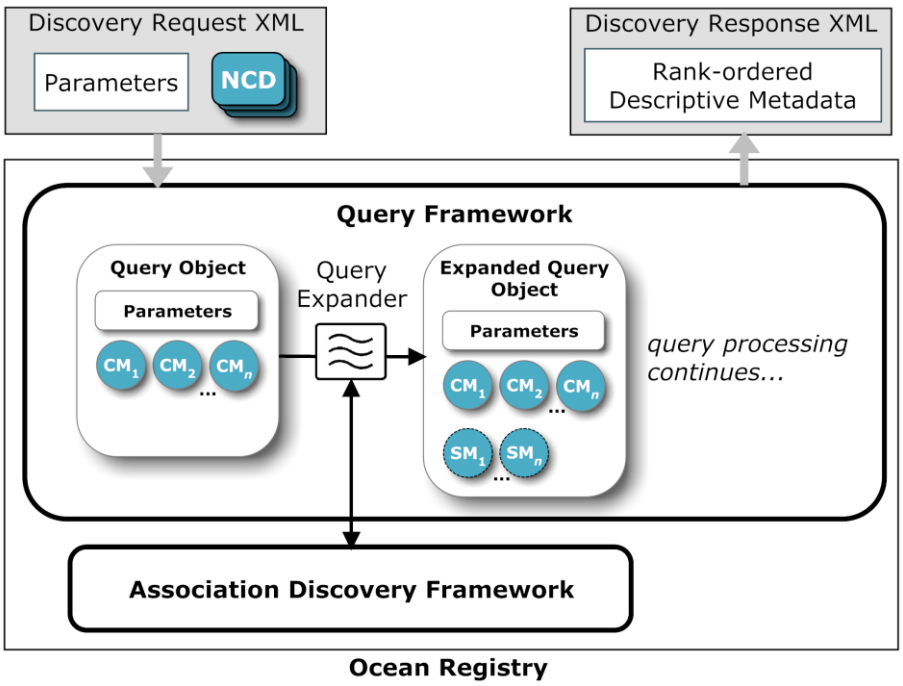


Figure 65: Overview of context-aware query expansion

Using the Expanded Query Object, we then compute the *top k* most similar Contextualized Resources using a modification of the Ocean MFSS approach presented in section 6.4.4. Recall that Ocean's foundational MFSS technique is based on an aggregation function *t* that computes an overall *base score* $t(R)$ of object *R* as:

$$t(R) = \frac{1}{n}\sum_{i=1}^{n} similarity(x_i, \acute{x}_i)$$

where *n* refers to the total number of Context Handlers associated to a Contextualized Resource, x_i refers to the Context Handlers associated with the Contextualized Resource and $\acute{x}_i$ refers to the matching Context Handlers within the Query Object. The similarity between x_i and $\acute{x}_i$ is calculated by passing the $\acute{x}_i$ Context Handler into the **`getNormalizedSimilarity`** method of Context Handler x_i (recall **`getNormalizedSimilarity`** returns domain-neutral similarity values in the interval [0,1]).

Next, the foundational MFSS approach is extended through the definition of a second aggregation function that is used to combine similarity grades for the Supplemental Metadata present within the Expanded Query Object. Hence, we introduce a second aggregation function $\acute{t}$ that computes an *expansion score* $\acute{t}(R)$ of object *R* as:

$$\acute{t}(R) = \frac{1}{n}\sum_{i=1}^{n} C_i \times (similarity(y_i, \acute{y}_i))$$

where *n* refers to the total number of Context Handlers associated to a Contextualized Resource; C_i refers to the confidence interval of the **`ContextAssociation`** providing the Supplemental Metadata (i.e. the Supplemental Context Handler); y_i refers to the Context Handlers associated with the Contextualized Resource and $\acute{y}_i$ refers to the matching Supplemental Context Handler within the Expanded Query Object. The similarity between y_i and $\acute{y}_i$ is calculated by passing the Supplemental Context Handler $\acute{y}_i$ into the **`getNormalizedSimilarity`** method of Context Handler y_i (recall **`getNormalizedSimilarity`** returns domain-neutral similarity values in the interval [0,1]).

Using the aggregation functions described above, query processing is completed using the adapted Threshold Algorithm presented in section 6.4.4. During query processing, the *overall score os(R)* of each object *R* is calculated by combining the results of the base score (BS) and the expansion score (SC) using an algebraic sum as per:

$$os(R) = (BS + SC) - (BS \times SC)$$

As previously described, at the completion of the query processing, Descriptive Metadata are generated for each discovered Contextualized Resource according to the following process: First, a Discovery Response object is created. Second, Descriptive Metadata for each Contextualized Resource are retrieved from the Ocean Metadata Store (using random access) and integrated into the response along with either a true URI or a **`uri_code`**, which is generated using a time-based salted hash that incorporates the true URI. Next, the Descriptive Metadata for each Contextualized Resource are updated with the overall score *os(R)* generated during the query process. Next, the Descriptive Metadata are rank sorted according to the associated overall scores. Next, the Descriptive Metadata objects are marshaled into a Discovery Response XML structure and returned to the Ocean

application (see section 6.4.1.2). Once the Discovery Response is received by an Ocean application, local application logic performs dynamic, in-situ component selection and composition as per the Ocean application model presented in section 5.2.

7.3 Discovery Personalization

Information overload represents another critical challenge for Ocean applications operating in complex, real-world environments. Similar to Web Search scenarios, where common query terms may result in an overwhelming number of search results [51, 205], Ocean Discovery Requests may result in an overwhelming number of component discoveries; making effective selection and composition difficult or impossible. Although Ocean constrains Resource Discovery results based on the client's modeled native context data (NCD), many environments may quickly accrue a large numbers of similarly contextualized Resources. In such scenarios, search enhancement techniques such context-aware query expansion (described in the last section), may actually worsen discovery results by supplementing queries with commonly encountered context information; increasing the number of contextually-relevant results. Hence, resultant Resource Discovery results may be overwhelming in quantity and difficult to distinguish based on similarity scores alone. As described in section 6.4, similarity scores provide a quantitative measure of the similarity between discovered Contextualized Resources and a given Discovery Request. However, such scores do not reflect qualitative measures such as the relevancy of the results to a user's personal interests and past preferences. Towards this end, this section defines Ocean's context-aware Resource personalization approach that enhances Contextualized Resource Discovery with community-based affinity predictions.

7.3.1 Background and Related Work

Due of its architectural lineage, the Ocean approach cannot escape the information overload challenges inherent in modern Web architecture. Ironically, the same Web Resource model that help Ocean overcome the component sparsity challenges can pose a significant hindrance to effective Resource selection and composition. Recently, *recommender systems* have emerged as promising approaches for reducing information overload in complex, information saturated scenarios where choice differentiation is difficult or impossible [264]. The principle objective of a recommendation system is to help users select relevant items from among a large set of similar items by generating suggestions or predicting the utility of specific items [339]. For example, Amazon.com have reported significant improvements in click-through and conversion rates for their personalized storefronts as compared to untargeted content such as banner advertisements and top-seller lists [200]. In most systems, the principle entities include *users* and *items*, where a user is a person who utilizes a recommender system to provide opinions (or *ratings*) about items that have been consumed and receive recommendations about new items that may be of interest. Importantly, recommender systems are based on the underlying assumption that users who have similar preferences in the past will probably have similar preferences in the future; allowing for an extrapolation of user history as a means of improving item suggestions (such as products) [379]. In this regard, a recommendation system can be defined as "personalized information filtering technology used to either predict whether a particular user will like a particular item (prediction problem) or to identify a set of *N* items that will be of interest to a certain user (top-*N* recommendation problem)" [86].

Recommendation systems can be broadly classified into two general approaches, including *content-based filtering* and *collaborative filtering* [379]. In content-based filtering, also known as *cognitive filtering*, algorithms compute the similarity between a user's collection of appreciated items and the universe of items still unknown to the user [379]. The computation of item similarity is based on a selection of domain-specific features such as plain-text terms or machine-readable metadata. Common content-based similarity approaches include naïve Bayesian classification of content features (e.g. product labels) [307] and nearest-neighbor vector-space queries (e.g. for keyword frequencies) [251]. Notably, content-based filtering is only appropriate for domains where feature extraction is feasible and attribute information readily available [379].

In contrast to content-based approaches, collaborative filtering (CF) does not rely on the availability of feature extraction or attribute information [263]. Rather, CF techniques compute similarities between users based on each user's known preferences; recommending items that are preferred by *similar users* [177]. Among the first systems to integrate CF techniques were Ringo [298] and GroupLens [263]. CF approaches compute user similarity based on their past ratings of the same items; generating a so-called "rating profile" for each user in order to identify potential *advisors* whose highly rated items are aggregated and used as recommendations. Due to their minimal information requirements and high quality recommendations, CF techniques have become dominant in commercial recommender systems such as those employed by enterprises such as Amazon.com and NetFlix. Notably, advanced recommender systems may employ a combination of both content and collaborative filtering in an attempt to mitigate the drawbacks of each [20].

To perform item filtering, CF systems first collect user *preference information*, which may include purchased products, click-stream data, demographic data (e.g. the age, gender and education of the users), content data (i.e. item features, such as text elements) and dedicated ratings [200]. In this regard, the two principle types of ratings are *explicit* or *implicit*. In an explicit rating, a user intentionally expresses appreciation for a given item using a numerical score such as a 5-point *likert scale* or *binary scoring*. Numerical rating scores are generally converted into a continuous range $[-1, +1]$, where negative numbers indicate dislike and positive numbers indicate fondness. However, while explicit ratings are generally considered accurate predictors of appreciation [264], the user-effort required for generating explicit ratings often dissuades participation and may promote "free riding" where users may not actively participate and rely on the results of others [15]. Therefore, many systems make use of implicit ratings [226], where preference information is inferred by observing key interactions between the user and the system such as purchase data and browsing behavior [211]. While implicit ratings have been shown to lower user-effort and increase participation, they are often less accurate predictors of affinity than explicit ratings [379].

The output of a recommender system is typically in the form of a *prediction* or *recommendation*. A prediction refers to the anticipated opinion of a user regarding a specific item according to the same numerical scale used to collect preference information from the user (e.g. a 5 point likert scale). Individually predictive preference values are categorized as *Individual Scoring*. Using the formalisms defined in [339], a recommendation refers to a ranked list of *N* items that are considered to be the most preferable for a given user ($N < n$ where n refers to the total number of items $I = \{i_1, i_2 \dots i_n\}$). Such outputs are typically categorized as a *Top-N Recommendation* or *Ranked Scoring* [86]. In most

cases, ranked scoring systems only include items for recommendation that have not been previously purchased or viewed (i.e. *consumed*) by the user.

The mediation between a recommender system's input (i.e. user opinions) and output (i.e. predictions or recommendations), is now briefly summarize (as presented in [339]). First, filtering algorithms consider input collected from m users $U = \{u_1, u_2, \dots, u_m\}$ regarding a set of n items $I = \{i_1, i_2 \dots i_n\}$. Hence, each user u_i has a collection of items i_{u_i} for which their opinions have been expressed and collected ($i_{u_i} \subseteq I$). It should be noted that i_{u_i} may include the *null set* (i.e. users have not provided opinions regarding each item). As previously mentioned, user opinions regarding items are expressed in the form of a rating score where a given user u_i expresses a rating score of item i_j as denoted by r_{ij} where the rating value is a real number or "no rating" (denoted by the symbol $\perp$). To facilitate filtering, ratings are collected into a $m \times n$ user-item matrix denoted by R. Filtering algorithms operate either on the rows of matrix R (corresponding to the ratings of a single user regarding different items) or on the columns of matrix R (corresponding to different users' ratings about a single item). An important distinction is made between the set of users U and the active user ($u_a \in U$), which refers to the single user for which recommendations or predictions are made. Broadly, two main classes of collaborative filtering algorithm can be discerned, including *user-based* and *item-based*. The next section introduces user-based collaborative filtering (user-based CF); item-based systems are introduced in section 7.3.1.2.

7.3.1.1 User-based Collaborative Filtering

User-based CF algorithms are designed to make item predictions and recommendations based on rating similarities that exist *between the users* in the collection U. User-based CF algorithms compute user similarities in a two step process. In the first step, the similarity values for all users in R are calculated according to a specific similarity metric. In most cases, rating similarities are computed using a Pearson Correlation or a Cosine Similarity Measure. The Pearson Correlation was proposed by the GroupLens project [263] as a means of computing the degree of linear relationship which exists between two users u_i and u_k and is given by:

$$sim_{ik} = corr_{ik} = \frac{\sum_{j=1}^{l}(r_{ij} - \bar{r}_i)(r_{kj} - \bar{r}_k)}{\sqrt{\sum_{j=1}^{l}(r_{ij} - \bar{r}_i)^2 (r_{ij} - \bar{r}_i)^2}}$$

where n is the total number of items in the user-item matrix and $l < n$; summations are calculated for l items for which both users u_i and u_k have provided ratings; and $\bar{r}_i$ and $\bar{r}_k$ represent the average ratings of the respective users.

Another method for calculating user similarity considers the user-item matrix to be an n-dimensional item space (or k-dimensional item space if dimensionality reduction has been applied). In these approaches, users represent feature vectors, where the vector consists of n feature slots (one for each item). The slots are filled with the rating r_{ij} provided by a user u_i for a corresponding item i_j (or filled with zero when "no rating" has been provided). Next, the similarity between users is calculated according to the Cosine Similarity Measure, which computes the similarity between each user's feature vector as the cosine of the angle between them. Cosine Similarity is given by:

$$sim_{ik} = cos_{ik} = \sum_{j=1}^{l} \frac{r_{ij}\ r_{jk}}{\sqrt{\sum_j r_{ij}^2}\sqrt{\sum_j r_{kj}^2}}$$

where *n* is the total number of items in the user-item matrix and $l < n$ (summations are calculated for l items for which both users u_i and u_k have provided ratings).

Once rating similarities have been computed for each user, the next step is neighborhood formation. Using one of the previously described similarity metrics, an $m \times m$ similarity matrix S can be generated for all users. A straightforward approach for neighborhood formation is the Center-based scheme, which establishes a neighborhood by selecting the users who have the highest similarity values in common with the active user based on the row of the similarity matrix *S* that corresponds to the active user. While simple to implement, the center-based scheme introduces significant computational complexity as the number of users increases [339].

To reduce computational complexity, the aggregate neighborhood scheme generates a neighborhood by picking users that are closest to the *centroid* of the current neighborhood. As described in [339], the aggregate neighborhood scheme operates as follows: First, the closest user to the active user u_a is selected from the similarity matrix; forming an initial neighborhood. Next, in order to select the remaining neighbors, the centroid $\vec{C}$ of the current neighborhood *N* is computed as:

$$\vec{C} = \frac{1}{h}\sum_{j=1}^{h} u_j$$

where *N* consists of h users and $h < l$. According to the aggregate neighborhood scheme, a user u_k not contained within the current neighborhood $u_k \notin N$ will only be selected for inclusion if it is closest to the centroid $\vec{C}$. Based on the similarity matrix, a neighborhood of similar users can then be generated for the active user.

Problematically, data within the $m \times n$ user-item are often sparsely populated due to large numbers of "no ratings"; resulting in significant space requirements and computational complexity during operations. To address this issue, several techniques for overcoming *rating sparsity* have been developed, including default voting [41], preprocessing using averages [285] and filterbots [286]. Broadly, these techniques attempt to automatically fill "no rating" entries with appropriate values; however, their significant complexity and the potential for inappropriate values are recognized as a serious challenge [41]. Consequently, other sparsity reduction techniques have also been explored, including capturing latent relationships among users through dimensionality reduction techniques (e.g. Singular Value Decomposition [35]).

Based on the computed neighborhood of *N* users considered most similar to the active user, predictions and recommendations can then be generated. A prediction refers to the expected opinion of the active user for a specific item (expressed as a numerical value), whereas a recommendation refers to a list of the top-*N* items expected to be most appreciated by the active user (based on predictions). The first step in the recommendation process is the formation of a set of predictions for all rated items in the active user's neighbors that are so-far unknown to the active user. As described in [339], a prediction may be expressed as:

$$pr_{aj} = \bar{r}_a + \frac{\sum_{i=1}^{l}(r_{ij} - \bar{r}_i) \times sim_{ai}}{\sum_{i=1}^{l}|sim_{ai}|}$$

where pr_{aj} represents a numerical prediction score for item i_j for the active user u_a; sim_{ai} refers to the similarity value obtained from the similarity matrix S, for all users u_i within the active user's neighborhood $i = 1,2, \dots, l$ (for users that have provided ratings).

As predictions are constructed, several performance enhancement techniques may also be applied to help improve prediction accuracy. Examples of such techniques include Inverse User Frequency [41], Significance Weighting [286] and Case Amplification [41]. Once a neighborhood of similar users has been constructed, a list of *N* recommended items can be generated for the active user. A common approach for generating recommendations is the Most-Frequent Item Recommendations *(top-N)* where a list $R_{pr} : \{1,2, \dots, N\} \rightarrow I$ is calculated based on the predictions pr_{aj} [86]. The list contains the items considered to be most appreciated by the active user in descending rank-order (i.e. highest predicted items listed first).

7.3.1.2 Item-based Collaborative Filtering

Although user-based algorithms underlie many popular recommender systems, they suffer from some serious drawbacks. As noted by Badrul in [285], user-based approaches generally perform poorly when rating data are sparse because it is difficult to generate a neighborhood of similar users. Sparsity issues are also difficult to overcome in many real-world scenarios where the total number of ratings for a given user is very small compared to the collection of available items (e.g. e-commerce). In addition, user-based algorithms generally suffer from scalability problems as they calculate predictions using the *entire database* of users and items. As user-based approaches generate user neighborhoods based on the entire set of users, their computational complexity grows linearly with the number of users and can become quickly intractable in large-scale systems. Hence, while such systems demonstrate good predictive characteristics and are relatively simple to implement, they are generally not well-suited for fast, online filtering operations.

Due to the aforementioned challenges, several *item-based* collaborative filtering approaches have been developed (item-based CF) [86]. Item-based CF exploits historical rating (or consumption) information to identify useful relationships between the various *items* under consideration. For example, the purchase of a given item (or set of items) is often shown to precede the purchase of second item (or set of items) with a high degree of probability [86]. Because the set of items remain relatively consistent and stable over time, similarity models can be pre-computed; leading to improved computational characteristics when the set of users is significantly larger than the set of items (i.e. $|U| \gg |I|$) [379]. Similar to user-based approaches, item-based CF approaches are based on the ratings r_{ij} provided by users $u_i \in U$ for items $i_j \in I$. However, unlike user-based CF techniques, item-based CF similarity values are calculated for items rather than users; where two items are considered similar if the users rating one item rate another item similarly. Item-based CF techniques analyze the set of items rated by the active user u_a and compute how similar each is to the target item i_j; selecting the k most similar items $\{i_1, i_2, \dots, i_k\}$ based on their corresponding similarities $\{s_{i1}, s_{i2}, \dots s_{ik}\}$. Predictions are generated by taking a weighted average of the active user's ratings on these similar items.

Importantly, item-based recommender techniques are probabilistic rather than deterministic; typically trading a time-intensive model building process (and often lower quality predictions) for improved computational complexity characteristics and faster query speeds [379]. In contrast to user-based CF techniques that operate over the entire database of users and items, item-based CF techniques improve performance by constructing a model of rating information that is initially based on the user-item matrix *R*; however the model becomes less reliant on the full matrix as the system is trained. Once training is complete, recommendations can be generated by interactions using the compact model; resulting in performance improvements [339]. Indeed, a recent survey of item-based recommenders demonstrated techniques that performed up to 28 times faster than user-based methods [86].

As described in [339], the general item-based CF approach operates as follows: First, the rating data are represented in an $m \times n$ user-item matrix *R*, where each element r_{ij} is a rating provided by a user u_i (i.e. row *i*) for item i_j (i.e. column *j*). Using matrix *R*, item similarity is calculated by isolating users that have *both* rated two items i_j and i_k and applying a similarity evaluation such as the Pearson Correlation Similarity and Cosine Similarity techniques described in the last section (exchanging item similarity for user similarity in the calculations). Once the similarities between all items in matrix R have been calculated, a neighborhood is created using the l most similar items i_k, with $k = \{1,2 \dots, l\}$, with regard to a specific item i_j of the active user u_a. Computing predictions is often accomplished using:

$$pr_{aj} = \frac{\sum_{k=1}^{l} sim_{jk} \cdot r_{ak}}{\sum_{k=1}^{l} |sim_{ak}|}$$

where ratings are weighted by a corresponding similarity sim_{ak} between the active user's item i_j and that of another item i_k. Given a set of predictions, a top-*N* list of recommendation items can be generated by following the user-based recommendation approach described in the last section.

7.3.2 The Ocean Recommendation Engine

To address the information overload that may affect Ocean applications during Resource Discovery, we have developed a hybrid recommender approach that supplements Resource Discovery Results with personalized affinity predictions based on the captured preferences of the Ocean User community. To support the integration of a variety of recommender techniques, we have developed a Recommendation Engine that extends the functionality of Ocean's Discovery Framework with generalized support for various recommendation algorithms. As Resource discovery results are intended to be composed into Ocean applications at runtime, query speed is a critical factor for application performance. Recall from section 7.3.1.1 that user-based recommender techniques provide good predictive quality but are often too slow for online query scenarios [285]. Hence, Ocean adopts the item-based class of recommenders as the foundation of its Resource personalization approach.

Extending the privacy mechanism introduced in section 7.2.5, Ocean does not provide Resource personalization by default (meaning that the Ocean Registry may be used anonymously). To enable Resource personalization, Ocean Users create an Ocean personalization account and obtain a private

key that is used to track preferences and support affinity predictions. To activate Resource personalization, Discovery Requests include personalization credentials as shown in Figure 66.

```
<?xml version="1.0" encoding="UTF-8"?>
<ocean_resource_request version="1.0" >
 <query_parameters max_results="50"
     domain_restrict=""
     contextualizer_restrict=""
     share_context="true"
     personalization_key ="45d20b1d2cc2d52e74b3cbf1750a2e31">
 (continued...)
```

Figure 66: An example Discovery Request with personalization credentials

When the Ocean Registry receives a Discovery Request with integrated personalization credentials, it first generates a subset of contextually-relevant Resources according to the basic Ocean MFSS approach described in section 6.4 (computing similarity scores for contextually relevant Resources as described in section 6.4.4). If query expansion is requested, supplemental query terms are generated and used during query processing as described in section 7.2.5. As the Query Processor completes Resource discovery for a given request, it passes its preliminary results to the Ocean Recommendation Engine as shown in Figure 67. To compute Resource predictions, a suitable recommendation algorithm is used to estimate a user's predicted appreciation of each Resource contained within the preliminary result set (for en example see section 7.3.3). Resulting personalization scores are added to each Resource's Descriptive Metadata and returned to the Ocean application at the conclusion of the discovery process. Ocean applications may then utilize *both* similarity scores and personalization scores to help select appropriate Resources for in-situ composition.

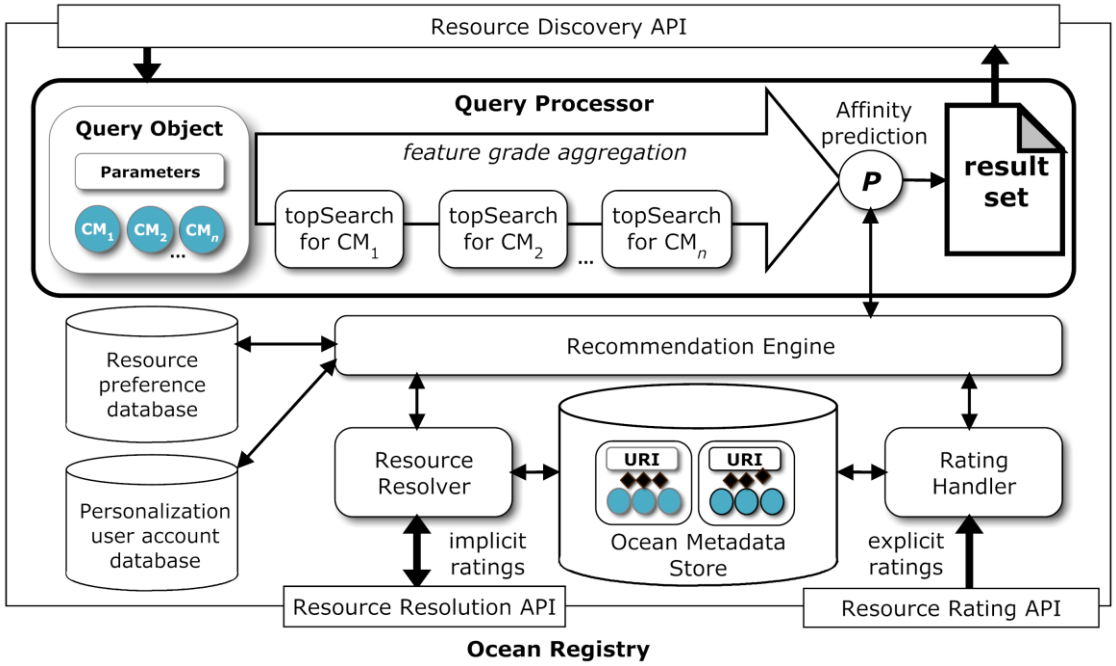


Figure 67: Overview of the Ocean Recommendation Engine

Within the Ocean Recommendation Engine, Resource affinity predictions are based on Resource ratings provided by the community of Ocean Users who have volunteered to share preference

information. To capture Resource ratings, the Recommendation Engine relies on two rating mechanisms, including *explicit* and *implicit*. To capture explicit ratings, the Ocean Registry provides a dedicated Resource Rating API, which is managed by a **`RatingHandler`** object. To explicitly rate a given Resource, an Ocean application constructs a Resource Rating XML document (defined shortly), which provides a reference to a given Resource's URI and an associated rating value (e.g. using a 5-point likert scale). To perform ratings, a user provides his or her personalization credentials to an Ocean application, which then establishes a unique user session with the Ocean Registry. Once a user-session is established, the **`RatingHandler`** ensures that incoming Resource references are valid and that rating limits are enforced (Ocean Users may provide one rating per Resource). As explicit ratings are created, the Recommendation Engine mediates interactions with the Resource preference model database, which stores rating information for use by recommendation algorithms. The Ocean Resource Rating XML schema is shown below in Figure 68.

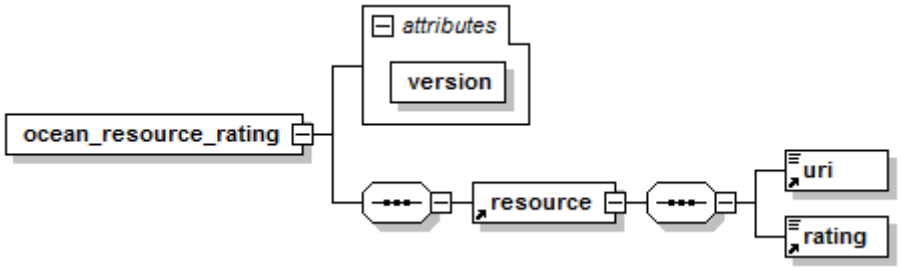


Figure 68: The Resource rating XML schema

To capture implicit ratings, the Ocean Registry provides the Resource Resolution API, which is managed by a **`ResourceResolver`** object. As introduced in section 7.3.1, the user-effort required to generate adequate numbers of explicit ratings often dissuades participation and promotes "free riding." To counteract this phenomenon, Ocean employs an implicit rating scheme whereby ratings are inferred by observing the Resource selections made by the Ocean Users. To facilitate implicit rating capture, the Descriptive Metadata within personalized Discovery Responses do not provide true URI information (see section 6.4.1.2). Rather, each Descriptive Metadata entity includes a unique **`uri_code`** that must be resolved to a valid URI using the Ocean Registry. As previously introduced, a **`uri_code`** represents a single-use, time-limited hash key that is generated by the Ocean Registry Query Processor for each Contextualized Resource referenced by a given Discovery Response. If an Ocean application wishes to resolve a particular Resource URI, it calls the Ocean Registry's Resource Resolution API using **`HTTP GET`** as **`http://oceanplatform.org/resolve/{uri_code}`**. As a resource resolution request arrives at the Ocean Registry, the **`ResourceResolver`** extracts the **`uri_code`** from the request and searches the given user's session hashtable using the **`uri_code`** as a key. If the **`uri_code`** is found, the Ocean Registry returns a **`HTTP 303`** redirection to the Ocean application, which provides the Contextualized Resource's true URI. The Ocean Resource Resolution XML schema is shown in Figure 69:

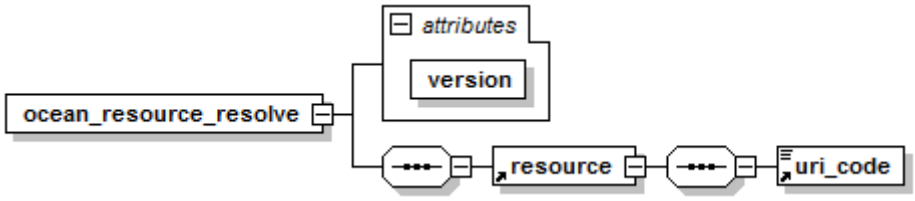


Figure 69: The Resource resolution XML schema

As URI resolution is completed, the `ResourceResolver` notifies the Recommendation Engine of the Resource selection. If Resource personalization is authorized by the user, the Recommendation Engine rates the Resource on behalf of the user using an appropriate preference value (if the user has not already provided an explicit rating). Depending on the recommendation algorithm employed, implicit rating for a given Discovery Response may be deactivated after a specific period of time has elapsed. As previously described, implicit ratings have been shown to be less accurate than explicit ratings when predicting a user's affinity for items [379]; hence, implicit ratings must be weighted accordingly. Finally, explicit rating should always override any implicit ratings already inferred for a given user.

7.3.3 Approach Validation Using the Weighted Slope One Algorithm

Recently, Lemire and Maclachlan [197] proposed a family of item-based recommenders, called Slope One, that demonstrate similar predictive accuracy to user-based recommenders yet are "easy to implement, dynamically updateable, efficient at query time, and expect little from first visitors while having a comparable accuracy to other commonly reported schemes." They observe that many item-based approaches rely on predictors that use weighted averages in a regression of the form $f(x) = ax + b$ [340]. As presented in [197], the Slope One approach suggests that a simpler regression scheme of the form $f(x) = x + b$ (where b is a constant and x represents rating values), can produce effective predictions while being:

1. easy to implement and maintain: all aggregated data should be easily interpreted by the average engineer and algorithms should be easy to implement and test;
2. updateable on the fly: the addition of a new rating should change all predictions instantaneously;
3. efficient at query time: queries should be fast, possibly at the expense of storage;
4. expect little from first visitors: a user with few ratings should receive valid recommendations;
5. accurate within reason: the schemes should be competitive with the most accurate schemes, but a minor gain in accuracy is not always worth a major sacrifice in simplicity or scalability.

As described by Lemire and Maclachlan, Slope One operates on an intuitive "popularity differential" principle, which indicates in a pair-wise fashion *how much better* items are liked by the various users in the complete set of users *U*. They suggest that simple way to measure the differential

between two given items is to simply subtract the ratings from users that have rated *both*. The resultant differential can then be used to predict a user's affinity for an *unrated* item.

For example, consider the two users A and B and the two items I and J presented in Table 15. In this example, user A rates item I with a value of 1 and item J with a value of 1.5. Next, the popularity differential between item I and item J is calculated by subtracting the ratings given for both items. Next, user B's affinity for the *unrated* item J can then be estimated by adding the previously calculated popularity differential (0.5) to user B's rating for the co-rated item J. In this case, the user B's predicted rating for item J is calculated as $2 + (1.5 - 1) = 2.5$. It should be noted that in realistic scenarios, the Slope One scheme computes averaged popularity differentials based on co-ratings from multiple users (discussed shortly).

	Rating for item I	**Rating for item J**	**Popularity differential**
User A	1	1.5	$1 - 1.5 = \mathbf{0.5}$
User B	2	**Unrated** Prediction: $2 + (1.5 - 1) = \mathbf{2.5}$	

Table 15: A simple rating profile and popularity differential calculation for Slope One

Lemire and Maclachlan use the following notation when describing Slope One schemes: First, all ratings of a given user, referred to as an *evaluation*, are represented as an incomplete array *u* where u_i represents the rating provided by the user for item *i*. The subset of all rated items for a given user array *u* is denoted S_u and the values of all evaluations for a given item pair (i.e. the *training set*) is denoted χ. The number of all elements in set *S* is denoted $card(S)$ (deferring to the notation provided by Lemire and Maclachlan rather than the common notation $|S|$). The average ratings in an evaluation u is denoted $\bar{u}$. The set of all evaluations $u \in \chi$ such that they contain item $i (i \in S(u))$ is denoted $S_i(\chi)$. Given two evaluations *u* and *v* their scalar product $\langle u, v \rangle$ is defined as $\sum_{i \in S(u) \cap S(v)} u_i v_i$. Hence predictions, which are denoted $P(u)$, represent a vector where each component is the prediction corresponding to one item (depending implicitly on the training set χ).

Using the above notion, Lemire and Maclachlan describe the baseline Slope One scheme as follows: First, given two evaluation arrays v_i and w_i $(i = 1, \dots, n)$, the best predictor of w with respect to v (in the form $f(x) = x + b$) is searched by minimizing $\sum_{i=1}^{n} (v_i - b - w_i)^2$. Next, deriving with respect to b and setting the derivative to 0 results in $b = \frac{\sum_{i=1}^{n} v_i - w_i}{n}$ (meaning that b is always chosen to be the *average difference* between the two evaluation arrays).

Next, given a training set χ and any two items *j* and *i* with associated ratings u_j and u_i from a user evaluation u $(u \in S_{j,i}(\chi))$, the average deviation of item *i* with respect to *j* is given by:

$$dev_{j,i} = \sum_{u \in S_{i,j}(\chi)} \frac{u_j - u_i}{card(u \in S_{i,j}(\chi))}$$

where the summation *does not* include any evaluation *u* which does contain ratings for both u_j and u_i. Using the above formula, a symmetric matrix can be computed once and quickly updated as new ratings are provided.

Given that $dev_{j,i} + u_i$ is a predictor of u_j when given u_i, a more meaningful predictor can be defined as the average of all such individual predictions as:

$$P(u)_j = \frac{1}{card(R_j)} \sum_{i \in R_j} dev_{j,i} + u_i$$

where $R_j = \{i | i \in S(u), i \neq j, card\left(S_{j,i}(\chi)\right) > 0\}$ represents the set of all relevant items.

In dense datasets (i.e. where most item pairs have ratings) $\bar{u} \cong \sum_{i \in R_j} \frac{u_j}{card\,(R_j)}$. Thus, the previous predictor can be simplified to:

$$P^{S1}(u)_j = \bar{u} + \frac{1}{card(R_j)} \sum_{i \in R_j} dev_{j,i}$$

Lemire and Maclachlan note that the Slope One scheme takes advantage of information from other users who have rated the same item as well as the items rated by the active user; emphasizing that the strength of the approach is the estimation of data points that are not specified (i.e. data not in the item array or user array). However, a drawback of the baseline Slope One scheme is that the total number of ratings for an item is not accounted for; meaning that items with a large number of ratings are treated the same as items with only a few ratings. Therefore, in order to place additional significance on those items with a larger number of ratings, the Weighted Slope One predictor is defined as:

$$P^{wS1}(u)_j = \frac{\sum_{i \in S(u) - \{j\}} (dev_{j,i} + u_i) c_{j,i}}{\sum_{i \in S(u) - \{j\}} c_{j,i}}$$

where $c_{j,i} = card(S_{j,i}(\chi))$.

To validate the Ocean Recommendation Engine previously introduced, we adopted a recommender algorithm based on the Slope One predictor presented above. Following the implementation described in [198], we devised a Weighted Slope One predictor and integrated it within the Ocean Recommendation Engine as shown in Figure 70. Resource predictions are generated as follows: First, the rating data for Ocean Users are collected using the explicit and implicit rating schemes described in section 7.3.2. These rating data are stored in a Rating table that is part of a Slope One database within the Ocean Registry. As ratings are captured by the Ocean Registry, popularity differentials between co-rated Contextualized Resources are pre-computed and inserted into a item-to-item matrix that is stored within a Differential table (according to the baseline SlopeOne approach). To support fast, online updating of the preference matrix, each item in the Differential Table includes both a sum value (representing the sum of all rating differentials for the item) and a count value (representing the total number of differentials comprising the sum) [198]. From the sum and count values in the Differential Table an average differential (AvDif) can be computed for each item pair.

If an Ocean User requests Resource personalization, the Query Processor sends its preliminary Discovery results to the Recommendation Engine at the conclusion of the multi-feature similarity

search process described in 6.4.4. The Recommendation Engine computes a set of affinity predictions for each discovered Contextualized Resource using the Slope One algorithm. After affinity prediction is complete, the Query Processor updates the `personalization_score` property of each Descriptive Metadata entity using the predicted affinity scores provided by the Recommendation Engine. Finally, the Descriptive Metadata are marshaled into a response XML structure and returned to the Ocean application as discussed in section 6.4.2. Once a Discovery Response is received by an Ocean application, local application logic performs dynamic, in-situ component selection and interoperation as per the Ocean application model presented in section 5.2. Notably, Ocean applications may use *both* `similarity_score` values and `personalization_score` values to select appropriate Resources for runtime composition.

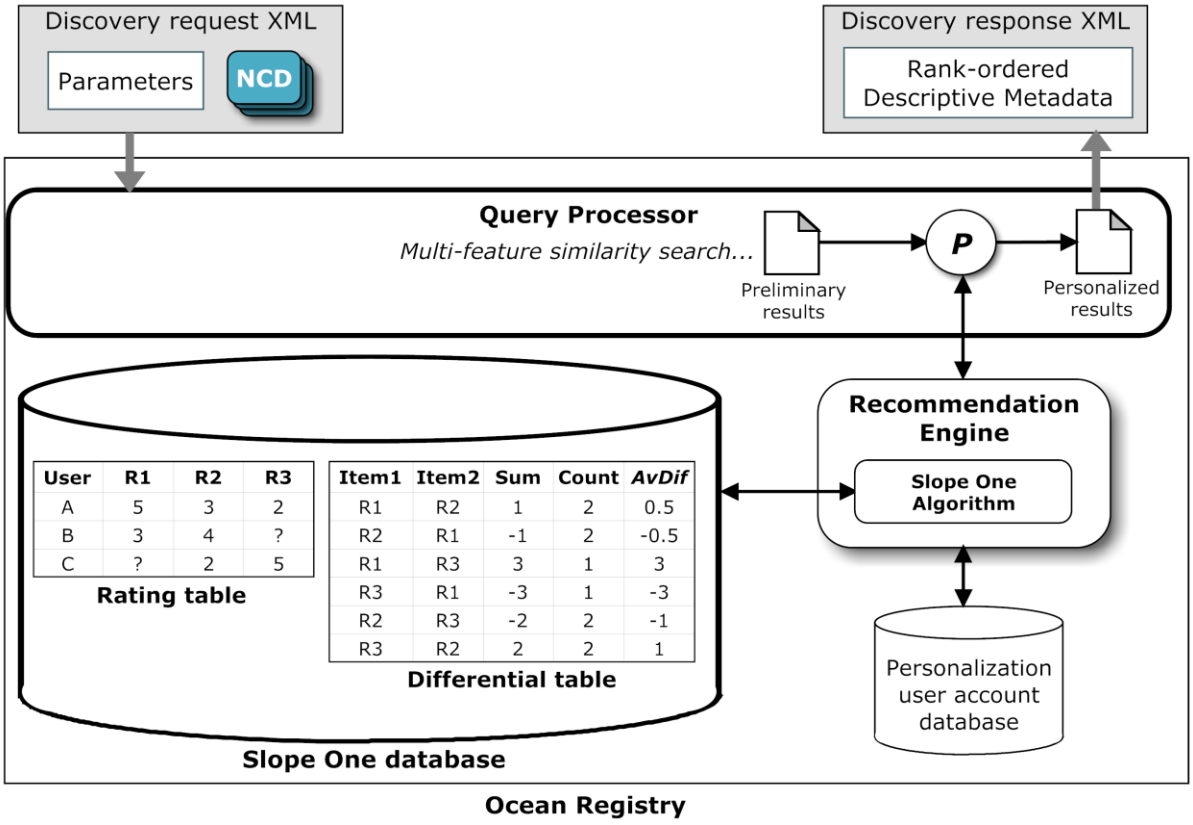


Figure 70: Integrating Slope One into the Ocean Recommender Engine

With reference to Figure 70, a Resource personalization example is briefly described (based on [357]). First, we posit a collection of Resource ratings that is stored in the Rating table. As per the Slope One approach, averaged popularity differentials are computed for each item pair as shown in Table 16.

Pair	Sum	Count	AvDif
R1/R2	$(5-3)+(3-4)=1$	2	$\frac{1}{2}=0.5$
R2/R1	$(3-5)+(4-3)=-1$	2	$\frac{-1}{2}=-0.5$
R1/R3	$(5-2)=3$	1	$\frac{3}{1}=3$
R3/R1	$(2-5)=-3$	1	$\frac{-3}{1}=-3$
R2/R3	$(3-2)+(2-5)=-2$	2	$\frac{-2}{2}=-1$
R3/R2	$(2-3)+(5-2)=2$	2	$\frac{2}{2}=1$

Table 16: Differential table calculations for Slope One

Next, we posit a preliminary Discovery result for User C, which includes Resource R1 with a `similarity_score` of 0.89 (note that User C has not yet rated R1 but has rated R2 and R3). Since User C requested Resource personalization, the Query Processor passes R1 to the Recommendation Engine for affinity prediction. Using the baseline Slope One approach, the average rating differential (AvDif) between R1 and R2 has been pre-computed as $\frac{(5-3)+(3-4)}{2}=0.5$ (note this value is stored within the Differential table for improved prediction speed). Using this average differential, user C's preference for R1 can be predicted by adding the averaged popularity differential existing between R1 and R2 to User C's rating for R2 as $2+0.5=2.5$. Similarly, affinity prediction for R1 can also be predicted for User C based on the average rating differential (AvDif) between R1 and R3, which has been pre-computed as $(5-2)=3$ (note this value is stored within the Differential table for improved prediction speed). Using this average differential, user C's preference for R1 can be predicted by adding the averaged popularity differential existing between R1 and R3 to User C's rating for R3 as $5+3=8$. Since User C has *multiple* co-ratings, a Weighted Slope One scheme is used to combine the predictions as $\frac{2\times2.5+1\times8}{2+1}=\frac{13}{3}=4.33$, where the weight is the total number of users contributing to the given popularity differentials (i.e. rating both items). Using this final value, the Query Processor updates the Descriptive Metadata's `personalization_score` for R1 using the weighted Slope One affinity result of 4.33. Finally, the Descriptive Metadata are marshaled into a response XML structure and returned to the Ocean application. An example personalized Discovery Response is shown in Figure 71.

```
<?xml version="1.0" encoding="UTF-8"?>
<ocean_resource_response version="1.0" >
  <resources>
    <resource>
      <type><![CDATA[mime:text/html]]></type>
        <uri_code><![CDATA[10fb2d66a65335317b54c93b15edefebe62a]]></uri_code>
        <domain><![CDATA[example_user.smugmug.com]]></domain>
        <title><![CDATA[Example's photo website]]></title>
        <description><![CDATA[Example description]]></description>
        <similarity_score>.89</similarity_score>
        <personalization_score>4.33</personalization_score>
        <wadle/>
     </resource>
(continued...)
```

Figure 71: Example personalized Discovery Response

As a validation of the Resource personalization approach described throughout this section, we created a prototype **`RecommendationEngine`** as part the Ocean Reference Implementation (Ocean RI). As a means of evaluating our approach using real-world algorithms, we adapted the Ocean RI for use with the Taste collaborative filtering engine (Taste Engine) [48], which provides open-source implementations of several popular recommender algorithms, including SlopeOne. Accordingly, we devised an Ocean-specific implementation of the Taste **`DataModel`** interface, called the **`OceanDataModel`**, which provides the Taste Engine access to the Ocean Persistence Framework described in section 6.3. Further, we enhanced Taste's basic SlopeOne approach by providing context-aware search-space reduction using the Resource discovery process described in section 6.4. Figure 72 illustrates how the Taste Engine was integrated within the Ocean RI. (Note that implementation-specific methods may be included in the figure below; however, in the interest of clarity, these are not described.)

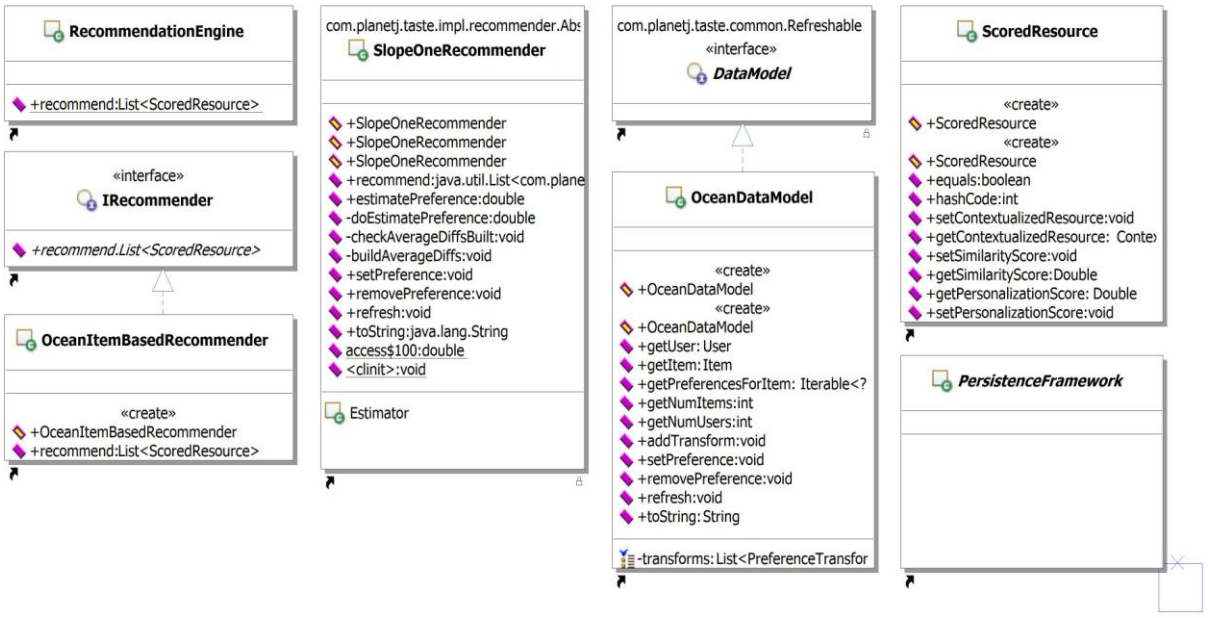


Figure 72: Integration of the Taste recommender within the Ocean RI

[48] http://taste.sourceforge.net/

7.4 Chapter Summary

This chapter discussed Ocean's use of community-based computation as a means of overcoming two key challenges inherent in large-scale networked systems. It began by describing the issue of context-mismatch, which refers to the situation where an Ocean application may not be capable of generating the native context data necessary to discover contextually-relevant Resources in a given environment. Notably, it discussed context mismatch with regards to a similar problem from the domain of Information Retrieval (IR), known as word mismatch. Briefly, word mismatch refers to the situation where the textual information within a IR search query may not adequately match the terms within a given document set; resulting in a vocabulary problem that reduces query effectiveness. It was noted how word mismatch has been effectively addressed through the use of query expansion techniques that augment queries with supplemental search terms in an effort to improve query results. Using query expansion as a foundation, we defined our preliminary mechanism for enhancing Ocean Discovery Requests with context-aware query expansion based on context information modeled from real-world environments. We founded our context modeling approach on a low-effort community contribution model (inspired by PlaceLab and BOINC), which exploits incoming Discovery Requests provided by large communities of heterogeneous Ocean applications.

Next, we described our preliminary approach for overcoming information overload in situations where Resource Discovery produces an overwhelming number of results; making effective component selection difficult or impossible. This section began by presenting background and related work regarding recommender systems, which have emerged as a promising technique for reducing information overload in information saturated environments. This section highlighted various recommender algorithms and discussed their applicability to Ocean's Discovery Framework. Based on this related work, we proposed the Ocean Recommender Engine as a mechanism for supplementing Discovery results with Resource affinity predictions based on captured preference information from the Ocean User community. Related, we also suggested techniques for capturing explicit and implicit rating information. To validate the Ocean Recommender Engine, an example recommender algorithm, called Slope One, was introduced and adapted for use within Ocean. The chapter concluded with a Resource personalization example and a discussion regarding the integration of the Taste collaborative filtering engine within the Ocean RI.

Chapter 8
Example Scenario

8.1 Introduction

The preceding chapters have detailed the theoretical aspects and practical infrastructure necessary to realize the Ocean approach for Web-scale context-aware computing. As previously discussed, the Ocean approach addresses the challenges facing large-scale context-aware systems presented in section 3.2. Notably, each of Ocean's related contributions has been detailed within a dedicated chapter in order to provide sufficient space for related work presentation, theoretical development and approach verification using the Ocean Reference Implementation (RI). As a method of illustrating how Ocean's various contributions form an integrated whole, this chapter presents a real-world Ocean application scenario that integrates various aspects of the Ocean approach. In addition, the example scenario provides further validation of Ocean's large-scale focus (see section 3.3) by integrating real-world sources of native context sources, significant amounts of Contextualized Resource information and more realistic models of Ocean community behavior. The intentionally simplified example scenario developed throughout this chapter aims to clarify core Ocean concepts and serve as a foundation for the development of more sophisticated Ocean applications.

The structure of this chapter is as follows: First, section 8.2 describes the experimental setup, including the development of an Ocean application development environment, called Ocean Studio, and an embedded version of the Ocean Reference Implementation (Ocean RI) designed for rapid prototyping. Next, section 8.3 provides an overview of the LinkFlow application, which forms the conceptual foundation for the chapter. Section 8.4 describes our data acquisition methodology and related toolset designed to capture and integrate large numbers of native context sources and real-world Contextualized Resources into Ocean Studio. Section 8.5 validates the basic LinkFlow scenario using the Ocean Studio development environment. Section 8.6 describes the integration of the previously acquired native context sources and Contextualized Resource information and discusses the resultant query performance reduction due to context mismatch and information overload. Related, a complimentary agent system is also presented as a means of approximating the behavior of a large Ocean application community operating within Ocean Studio. Finally, section 8.7 discusses how the aforementioned query challenges can be addressed by applying the context-aware query expansion and Resource personalization techniques presented in the last chapter.

8.2 Experimental Setup

To devise Ocean application scenarios we created a comprehensive Ocean development environment called Ocean Studio, which includes several interrelated components. First, it provides a device emulation framework that provides support for modeling, configuring and controlling device emulators that are capable of running Ocean-based applications. Device emulators provide a platform-specific set of context acquisition hardware – such as an onboard camera, language preferences, radio transceivers and GPS equipment – which is capable of receiving real-world context information from multiple data sources. Context acquisition and modeling within each emulator is based on the client-centric Aladin approach described in section 3.2.2. Using the Aladin style, emulators gather context

information using locally available hardware and provide the resultant native context data (NCD) to Ocean applications through events. Emulators are capable of running a single Ocean application, which provides its own internal application logic and is presented full screen within the active emulator's display. Ocean applications receive NCD as events from the emulator framework and receive user input from screen-clicks, hardware buttons and a settings profile. Using the emulator framework, we modeled two separate devices: (1) a Nokia N95 mobile phone[49] with inbuilt GPS hardware, Bluetooth and 802.11 transceivers and an onboard camera; and (2) a Dell Axim x50v PDA[50] with inbuilt Bluetooth and 802.11 transceivers and an attached camera. Within Ocean Studio, the active emulator can be changed at runtime by selecting from a drop-down list of installed emulators. An overview of Ocean Studio is shown in Figure 73.

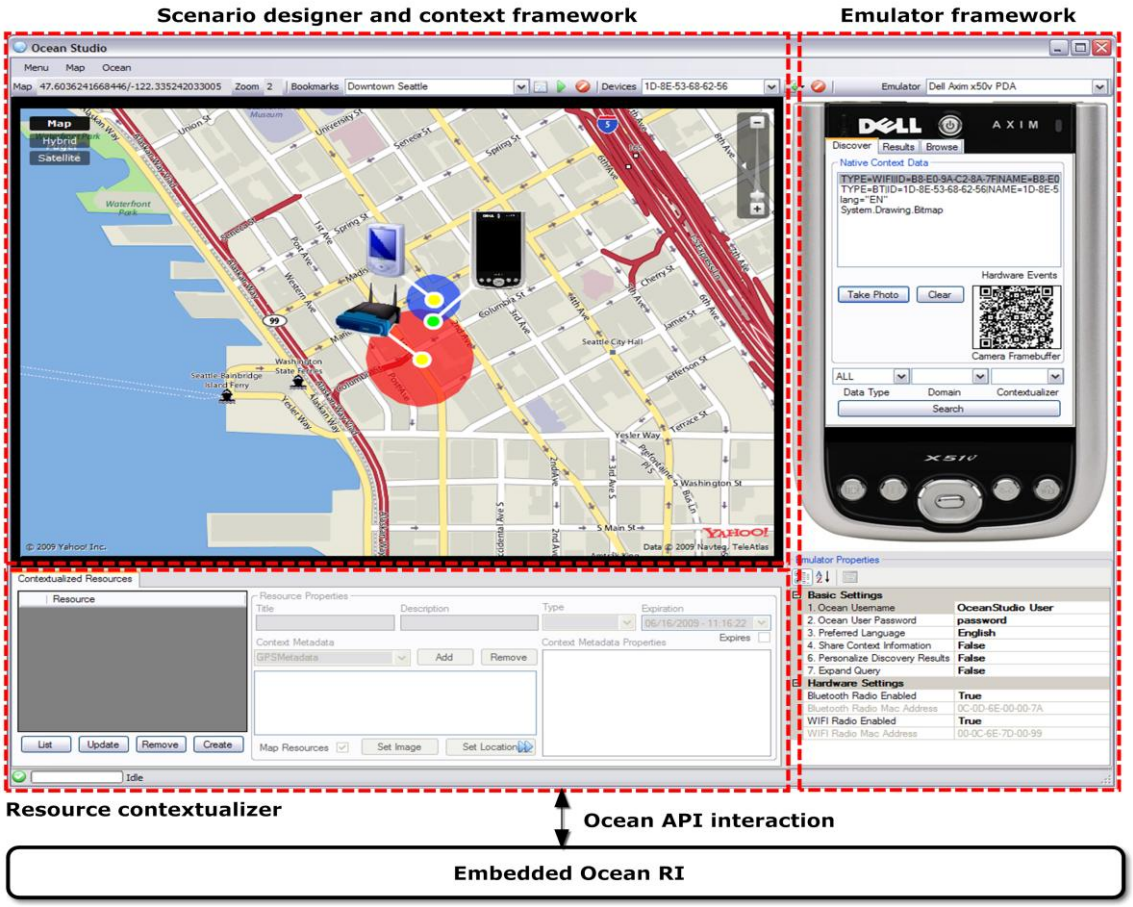


Figure 73: Overview of Ocean Studio

As shown in Figure 73, Ocean Studio provides an integrated scenario designer whereby context sources, emulator representations (i.e. the active emulator's icon) and Contextualized Resources can be placed, configured and visualized on a graphical map component. The map component is based on the Yahoo Mapping API[51], which provides support for visualizing mapping data, satellite imagery and overlay graphics. The mapping component is configured to allow location dragging and zooming and

[49] http://europe.nokia.com/find-products/devices/nokia-n95
[50] http://www.dell.com/content/topics/segtopic.aspx/brand/axim_x50
[51] http://developer.yahoo.com/maps/

provides support for drag-and-drop placement of 802.11 and Bluetooth transceivers. Additionally, the scenario designer automatically renders available Contextualized Resources that include a **`GeoPointHandler`** as Context Metadata (described in section 4.4.1). The active emulator appears within the scenario designer as a graphical icon that can be moved within the map component by dragging and dropping its location pointer, which appears as a green circle connected to the emulator by a white line. An overview of two context sources and the active emulator is shown rendered within the scenario designer in Figure 74.

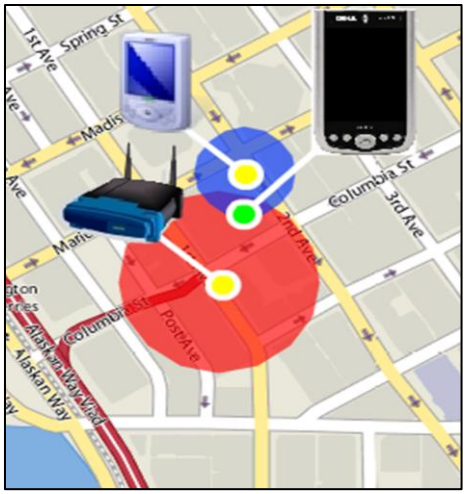


Figure 74: Example context sources and the active emulator rendered within the scenario designer

The scenario designer also provides an integrated context framework capable of aggregating and provisioning multiple sources of context information. The physical location of the active emulator is determined by querying the Yahoo Map API with the coordinates of the emulator's location pointer as it is moved within the map component. Physical positioning information for the active emulator is provisioned using the native NMEA 0183 ASCII sentence format [220] common to most GPS hardware. The physical positions of 802.11 and Bluetooth transceivers that have been manually created within the scenario designer are automatically managed by the context framework, which provisions transceiver data in the native NetStumbler format [212] common to popular "war-driving" software libraries [291] and wireless network diagnostic utilities[52]. The radio transmission characteristics of several transceiver types (e.g. Bluetooth class 1 and 802.11b) are estimated using radio frequency (RF) propagation models [312] that account for common transmission characteristics such as free space loss, signal power attenuation and signal scattering (for the example scenario we used a propagation model appropriate for outdoor environments with building obstructions). Additionally, the context framework is capable of importing bulk transceiver geo-location information from external data sources such as the beacon collection results of the PlaceLab project (see section 7.2.2). Finally, the context framework is capable of provisioning bitmap data to the active emulator's camera frame-buffer using an imported image that can be set using the Ocean Studio preferences. As an overview, Figure 75 shows an emulator acquiring and modeling native context data from the

[52] http://www.stumbler.net/

context framework. Specifically, this figure shows the active emulator modeling GPS positioning information in the NMEA data format, several nearby radio sources in the NetStumbler format, a language preference string according to RFC 3066 [6] and the raw image data from the emulator's camera frame-buffer as a device-independent bitmap.

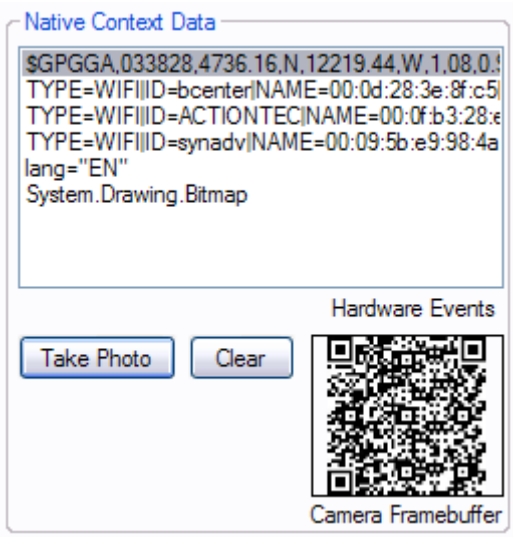


Figure 75: Native context data as modeled by the active emulator

Finally, Ocean Studio includes an integrated Resource contextualizer tool that allows Contextualized Resources to be created and managed using a form-based visual designer. The tool allows Contextualizers to create, retrieve (list), update and delete Contextualized Resources using both General Metadata and Context Metadata, as described in section 4.3. Notably, the Resource contextualizer provides form-based configuration options for each Context Handler installed in the embedded Ocean RI. Recall that, Context Experts may provide a set of arbitrarily complex configuration options that can be controlled by non-experts through relatively simple external interfaces (see section 4.3.2). Related, the Resource contextualizer automatically discovers and presents such configuration options using reflection mechanisms of the .NET Framework.

Based on the server-based Ocean RI presented throughout this dissertation, we developed an embedded version of the Ocean RI as a means of supporting Ocean Studio scenarios. The embedded Ocean RI includes the same core functionality as the server-based version previously discussed; however, the embedded version is designed specifically for rapid prototyping and includes support for server-less operation, high-speed integration of preconfigured data-sets, in-memory component communication, and rapid integration of test components such as prototype Context Handlers. Further, to support Discovery Request/Response debugging, the embedded Ocean RI operates without the Web service interfaces and security mechanisms discussed in section 5.3.1.

Ocean Studio facilitates communication between the active emulator's Ocean application and the embedded Ocean RI using an implementation of the Ocean discovery protocol described in section 6.4.1. Hence, Ocean applications running within the emulator construct and pose Discovery Requests to the Ocean RI as required by their internal application logic. If deemed advantageous by the application, the NCD acquired and modeled by the emulator may be included within Discovery

Requests as query terms. As such, Ocean application developers do not need to possess context domain expertise in order to utilize the Ocean approach. Figure 76 shows how the active emulator's profile settings are used to control various aspects of the Resource discovery process. The available profile settings include an Ocean username and password; a context sharing Boolean; a discovery result personalization Boolean; and a context-aware query expansion Boolean. (Note that each of these options is discussed in detail later in this chapter.)

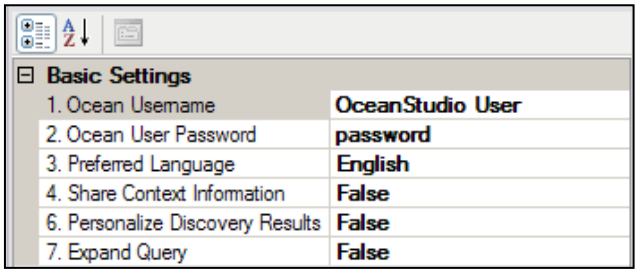


Figure 76: Ocean preference settings as shown in the active emulator's user profile

8.3 Scenario Overview

As a means of demonstrating various aspects of the Ocean approach, we devised a simple context-aware Web browser application called LinkFlow, which is intended to be executed within the Ocean Studio emulator framework. We created LinkFlow based on the results of other well-known context-aware hypermedia applications such as Guide [78] and Cooltown [179]. As discussed in Chapter 2, location-aware hyperemia applications often aim to extend the Web's conventional model to include Resource discovery and selection using context information such as physical or symbolic positioning data. Similarly, LinkFlow extends the traditional Web agent model (see [341]) and operates much like a conventional Web browser in traditional hypermedia scenarios. However, unlike a conventional hypermedia application, LinkFlow's set of available URIs flow from Discovery Requests made against the Ocean RI. As such, LinkFlow users are provided a continually evolving set of *contextually relevant* Resources in the form of Descriptive Metadata, which include a similarity score, personalization score, title, description, etc.

As detailed later in this chapter, the LinkFlow application involves the following workflow: context detection; Discovery Request formation; Discovery Response selection; and context-aware browsing. As detailed in section 3.2.8, component interoperation with discovered Resources takes place according to the REST architectural style as presented in [105]. However, while the LinkFlow scenario involves only simple context-aware browsing, more sophisticated application scenarios involving additional Resource types (e.g. structured XML) and various control flow scenarios (e.g. CS mashups) are also possible (see section 5.2.2).

LinkFlow differs from other context-aware browser approaches in that its application model and support infrastructure are based on the Ocean approach. With regards to LinkFlow, the Ocean approach can be summarized as follows: First, because the Ocean approach extends the Aladin context modeling style described in section 3.2.2, the LinkFlow application is capable of modeling real-world environments without the need for dedicated context instrumentation and related infrastructure. Second, as described in section 5.5, Ocean's open APIs for the creation and

management of Contextualized Resources can be used to promote the contextualization of large numbers of existing Web Resources. Third, based on Ocean's Context Handler contribution model described in section 5.4, external Context Experts are able to integrate complex domain semantics into the Ocean Registry; allowing LinkFlow developers use the Ocean Registry without requiring complex context domain knowledge (see section 6.4.1). Fourth, as described in section 6.3, the Ocean RI supports Contextualized Resource persistence and indexing and provides a multi-feature similarity search approach that generates search results based on incoming query NCD and domain-specific similarity features contained within persisted Contextualized Resources. Fifth, as described in section 7.2, LinkFlow applications running on resource constrained devices (e.g. without GPS hardware) may request that Ocean automatically expand Discovery Requests with supplemental context-relevant query terms that have been modeled from the shared context information provided by the Ocean User community. Finally, as described in section 7.3, LinkFlow applications may request Resource personalization, which allows the Ocean Registry to automatically rank Discovery Results based on privacy-aware user profiling, community generated preference information, and Ocean's integrated recommender algorithms.

8.4 Data Acquisition

As a means of acquiring significant quantities of real-world context information and Contextualized Resources data, we integrated two external datasets into Ocean Studio. First, we obtained the publicly available beacon capture datasets provided by the PlaceLab research team [151]. These datasets include the results from several large "war-driving" sessions completed using the data acquisition process described in section 7.2.2. Notably, the public PlaceLab datasets contain significant quantities of real-world beacon information (e.g. over 50,000 trace samples) and include detailed information regarding the identification and characteristics of various 802.11, Bluetooth and GSM radio transceivers located throughout the Seattle Washington metropolitan area. To support the LinkFlow scenario, we utilized a subset of the PlaceLab beacon capture data that was localized to a geographic region encompassing the *downtown* Seattle area (the resulting subset included 7,943 individual beacons).

Next, we obtained real-world Contextualized Resource data by creating a multithreaded Contextualized Resource crawler framework (crawler framework) capable of extracting domain-specific context information from several large, publicly available Web Resource datasets. As shown in Figure 77, the crawler framework provides several high-level features, including crawler controls (e.g. starting, stopping and crawl interval); crawler management tools (e.g. maximum results, start offset, etc.); and database interaction tools (e.g. database management, crawl totals and result clearing).

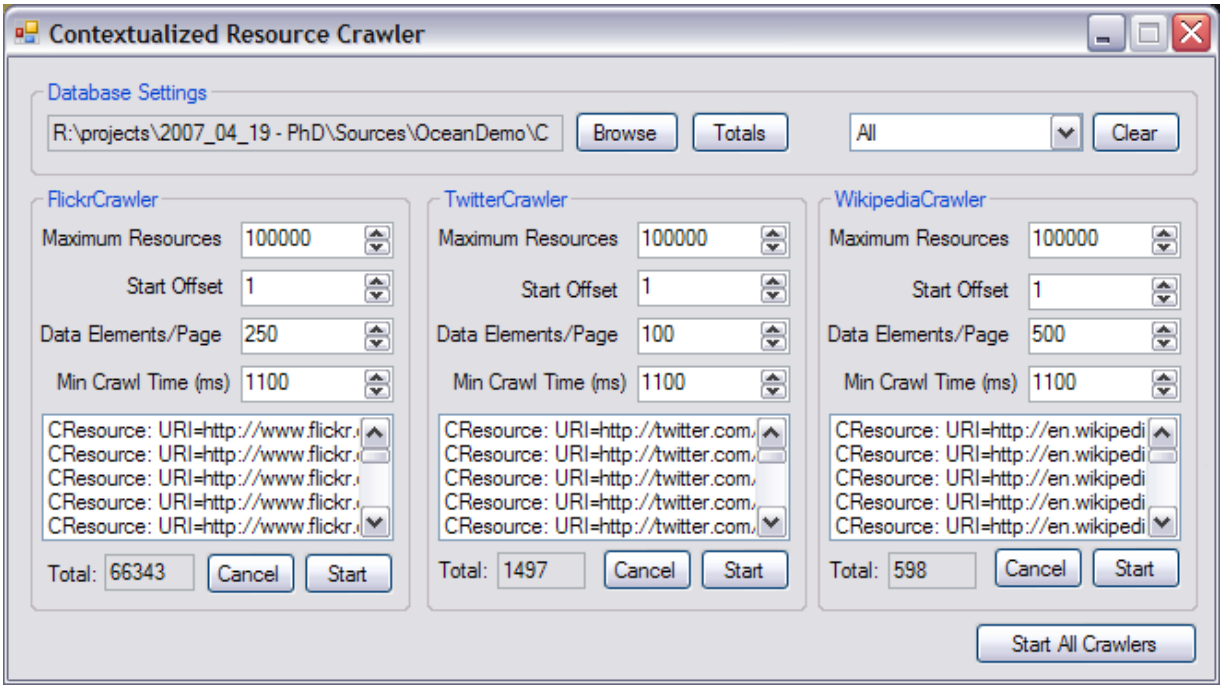


Figure 77: Overview of the Contextualized Resource crawler framework interface

Using the crawler framework, we developed three independent context-aware crawlers. First, a crawler was devised to extract image-based Contextualized Resources from the photo sharing Website Flickr[53]. The Flickr crawler utilizes Flickr's REST-based Web service API[54] to query for structured information related to shared photos that have been geo-coded using latitude and longitude information. The Flickr crawler can be customized to allow for additional query constraints based on specific geographic coordinates or text-based query terms. The Flickr crawler creates Contextualized Resources by parsing the resultant XML result data (in Flickr's native format) in order to obtain General Metadata information such as a title, description and URI. Additionally, Context Metadata is generated from Flickr results by encapsulating positioning data using the `GeoPointHandler` described in section 4.4.1.

Next, a crawler was devised to extract Contextualized Resources from the micro-blogging Website Twitter[55]. The Twitter crawler utilizes Twitter's REST-based Web service API[56] to query for XML-based feed data that have been geo-coded within a customizable geographic area. The Twitter crawler creates Contextualized Resource's by parsing the resultant Atom feed data [328] in order to obtain General Metadata information such as a title, description and URI. Additionally, Context Metadata is generated from Twitter results by encapsulating positioning data using the `GeoPointHandler` mentioned above. If precise positioning data is not present within the resultant Atom feed data, the Yahoo geo-coding API[57] is used to translate address information into geographic coordinates suitable for use by the `GeoPointHandler`.

[53] http://www.flickr.com/
[54] http://www.flickr.com/services/api/
[55] http://twitter.com/
[56] http://apiwiki.twitter.com/
[57] http://developer.yahoo.com/maps/rest/V1/geocode.html

Finally, a crawler was devised to extract Contextualized Resources from the community-based online encyclopedia Wikipedia[58]. As not all Wikipedia articles contain geo-coded information, we utilized the GeoNames geographical database[59], which aggregates multiple sources of geographical data and provides a unified Web-service query interface. The Wikipedia crawler constraints GeoNames query results to Wikipedia articles that have been geo-coded within a specific geographic area. The Wikipedia crawler creates Contextualized Resource's by parsing the resultant XML result data (in the GeoNames native format) in order to obtain General Metadata information such as a title, description and URI. Additionally, Context Metadata is generated from GeoName results by encapsulating positioning data using the **`GeoPointHandler`** mentioned above. Finally, the Wikipedia crawler creates an additional **`ISO639LanguageHandler`** based a supplemental query of the Wikipedia site, which checks for articles in both English and German.

Using crawler implementations described above, we performed several crawling sessions over a 13 day period. Crawling session duration ranged from approximately 2.5 minutes to over 17 hours and care was taken to adhere to Web service rates limits (e.g. the Flickr API limits Web service calls to 1 per second per IP). In order to improve crawling performance, each individual crawler operates in parallel. The Contextualized Resource results from each crawling session were stored in separate database files in order to organize the results for integration into Ocean Studio. We began by performing small initial test crawls and continued to perform additional crawls as the LinkFlow application was finalized. The longest crawling session took over 17 hours and resulted in the generation of 68,438 Contextualized Resources. To align our crawling results with the PlaceLab beacon datasets previously described, we configured each crawler to extract Contextualized Resources using geographic coordinates and search terms specific to the Seattle metropolitan area. Table 17 provides an overview of several Contextualized Resource crawl sessions and provides related information, including duration, description and total Contextualized Resources extracted (i.e. CR Totals).

[58] http://www.wikipedia.org/
[59] http://www.geonames.org/

Date	Crawl Duration (minutes)	Session description	CR Totals
February 22nd, 2009	22.5	Initial test crawl session with simultaneous crawler start.	**All:** 1490 **Flickr:** 500 **Twitter:** 500 **Wikipedia:** 490
February 22nd, 2009	105.9	Stability testing crawl session with a longer duration. Staged crawler start to test multithreaded operation.	**All:** 6990 **Flickr:** 5000 **Twitter:** 1500 **Wikipedia:** 490
March 4th, 2009	1036.9	Large crawl session using simultaneous crawler start and supplemental ISO639LanguageHandler generation.	**All:** 68,438 **Flickr:** 66,343 **Twitter:** 1497 **Wikipedia:** 598
March 6th, 2009	24.1	Sample extraction for Ocean Studio with start offset for use in test rendering.	**All:** 1594 **Flickr:** 500 **Twitter:** 497 **Wikipedia:** 597

Table 17: Overview of notable Contextualized Resource crawling sessions

8.5 Validating the Basic LinkFlow Scenario

To validate the basic LinkFlow scenario introduced in section 8.3 we used Ocean Studio's Resource contextualizer to manually create and test an example Contextualized Resource (CR). The example CR constrains the Discoverability Context of an HTML-based image slideshow using a specific geographic area definition and language preference. Recall from section 4.3 that Ocean Metadata are stored separately from the associated Resources within the Ocean Registry in order to impose the separation of concerns necessary for supporting context-aware compositional adaptation (see section 2.2.3). CRs created within Ocean Studio are persisted within the embedded Ocean RI using a version of the Ocean persistence model presented in section 6.3.2. The example CR's General Metadata include a title, description, data-type and URI (the URI references a Resource hosted on the SmugMug photo sharing Website[60]). Additionally, the example CR contains two Context Metadata entities, including a `GeoPointHandler` configured with a "Required" association type, a validity radius of 100 meters and a linear comparison function (see section 4.4.1); and a `ISO639LanguageHandler` entity configured with an "Optional" association type and a language code of "English." Figure 78 shows the example CR rendered within Ocean Studio's scenario designer.

[60] http://smugmug.com/

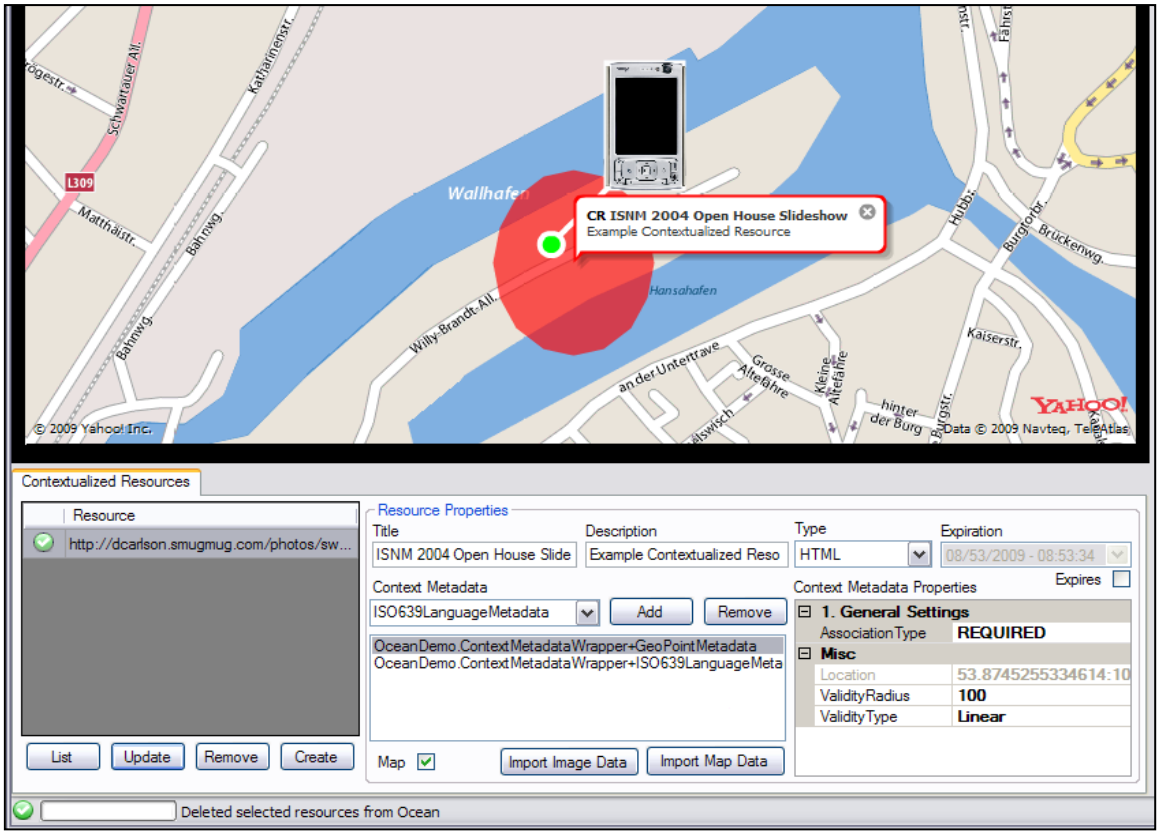


Figure 78: Example Contextualized Resource rendered within Ocean Studio's scenario designer

To validate the basic LinkFlow scenario, we began by positioning the active emulator within the validity radius of the example CR's `GeoPointHandler` (using the emulator's location pointer) and setting the emulator's preferred language to English (using the emulator's property controls). As described in section 8.2, the active emulator automatically models relevant NCD. In this scenario, the emulator models geo-location information (provided by local GPS hardware in the NMEA format), a language preference string in the ISO639 format, and a bitmap from the device's camera frame-buffer.

While methods of configuring and performing Discovery Requests are entirely application-specific (see section 6.4.1), the LinkFlow application presents several options as a means of illustrating the Ocean approach. As shown in Figure 79, the LinkFlow application provides both an NCD view (containing raw context data values) and several query controls that allow the user to constrain Discovery Requests by data-type, domain and Contextualizer. While some Ocean application types may shield details of the Resource Discovery process from users (e.g. by automatically performing Discovery Requests when certain criteria are met), LinkFlow provides an explicit "Search" button that allows users to initiate Discovery Requests directly. An overview of LinkFlow's Discover View is shown in Figure 79.

Figure 79: The Discover View showing locally modeled NCD and several search constraints

Using the Discover View controls shown above, Discovery Requests can be performed by choosing appropriate query constraints and clicking the Search button. Recall that when the Search button is clicked the LinkFlow application formulates a Discovery Request, which includes the emulator's modeled NCD as query terms. Discovery Requests arriving at the embedded Ocean RI are processed by a naïve implementation of the Ocean multi-feature similarity search (MFSS) approach described in chapter 6.4.4. Recall that Ocean's full MFSS framework utilizes the Context Metadata interface implemented by contributed Context Handlers to provide Contextualized Resource indexing, persistence and similarity determination.

At the completion of Ocean query processing, resultant Descriptive Metadata results are returned to the LinkFlow application using the Discovery Response format described in section 6.4.1.2. Recall that Descriptive Metadata include a URI (or `uri_code`), similarity score, personalization score (if requested), title, description, domain and an optional WADL document (see [137] for details). Several of these metadata are integrated into LinkFlow's Results View, which provides a multi-column ListBox component capable of rendering ranked results for the user. Next, LinkFlow users survey the discovered metadata as a means of making suitable Resource selections (according to similarity score (Sim), the personalization score (Pref) and the title).

As expected, Discovery Requests made within the example CR's Discoverability Context resulted in proper CR detection and the generation of appropriate similarity scores. Furthermore, double-clicking the CR's metadata entry in the Result View ListBox resulted in runtime composition according to the Web-browser model described in [341] (i.e. the CR's URI is rendered in the Browse View using an embedded Web browser). An overview of the basic LinkFlow application scenario is shown in Figure 80.

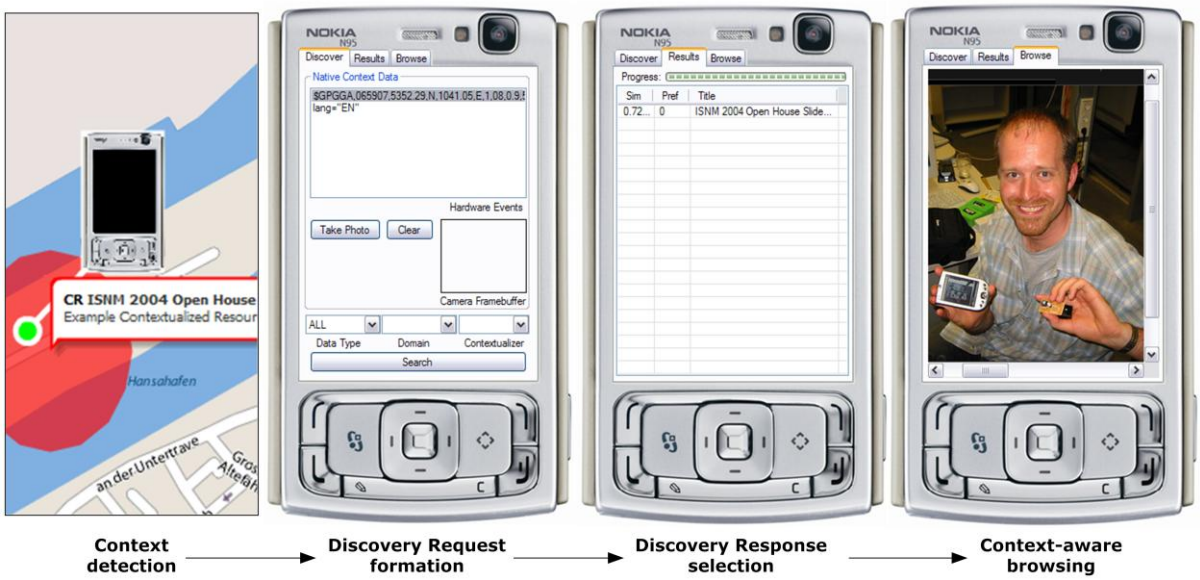


Figure 80: Overview of the basic LinkFlow application scenario

As described at the beginning of this chapter, LinkFlow is designed to closely mirror the Ocean application model as a means of clarifying Ocean concepts; however, other Ocean application types may use more sophisticated application logic to address specific problem domains. For example, Ocean applications may react in domain-specific ways to well-known NCD types while using Ocean to interpret unknown NCD types. As another example, Ocean applications may model Discovery Results themselves as NCD; using these data to form new Discovery Requests (e.g. using discovered image from Ocean data as NCD). Ocean Developers may also extend prepackaged control flows that may not mirror the Ocean application model directly. Briefly, Richardson and Ruby [266] describe control flow as "a set of instructions about what to do when you get certain kinds of requests" and identify two example types, including database-backed (where applications adapt based on information or state contained within a database) and the Atom publishing protocol [224] (which defines a set of Resources that capture the process of publishing feed data using the Atom XML format [328]). While we do not speculate on the types of applications that might be constructed using such techniques, we suggest that many control flow scenarios may be amenable for the integration of contextually relevant computation. Notably, as discussed in section 5.2.1, we view the contextualization of client-centric mashups styles (e.g. Aggregation, Personalization and Real-time Monitoring) as the foundation for new classes of context-aware Web applications capable of spontaneous cross-domain component discovery and interoperation.

8.6 Importing Crawled CR Data and Acquired Context-Sources

Using the basic scenario described in the previous section, we next validated the LinkFlow application using the crawled CR data and real-world context-sources described in section 8.4. First, the CR crawling results were imported into the embedded Ocean RI and rendered using the Ocean Studio scenario designer. During testing, it was discovered that the Yahoo Map component performs well for visualizing small CR datasets (i.e. less than 100); however, rendering larger CR datasets resulted in poor visualization quality (due to overcrowding) and map unresponsiveness (due to performance

limitations of the Yahoo Map component and API). To compensate for these limitations, three strategies were employed. First, unlike manually created CRs, which are rendered with Discoverability Contexts, crawled CR data are visualized using only red flags, which are placed on the map component according to their geo-coordinates (if available). Second, Ocean Studio is configured to utilize a localized subset of the CR crawl data (see section 8.4). Third, the scenario designer only renders CRs located within a customizable distance of the active emulator.

Next, Ocean Studio's import tool was used to import the 7,943 beacons from the PlaceLab dataset presented in section 8.4. In order to minimize visual overcrowding and maintain map responsiveness, imported beacon data are not rendered within the scenario designer; however, the context provisioning framework continually provisions imported context information to the active emulator, which appears as NCD within the LinkView Discover View. To further improve Ocean Studio performance, only beacons located within a theoretically detectable range of the active emulator are considered during context provisioning. An overview of Ocean Studio rendering crawled CRs and provisioning PlaceLab beacon data is shown in Figure 81.

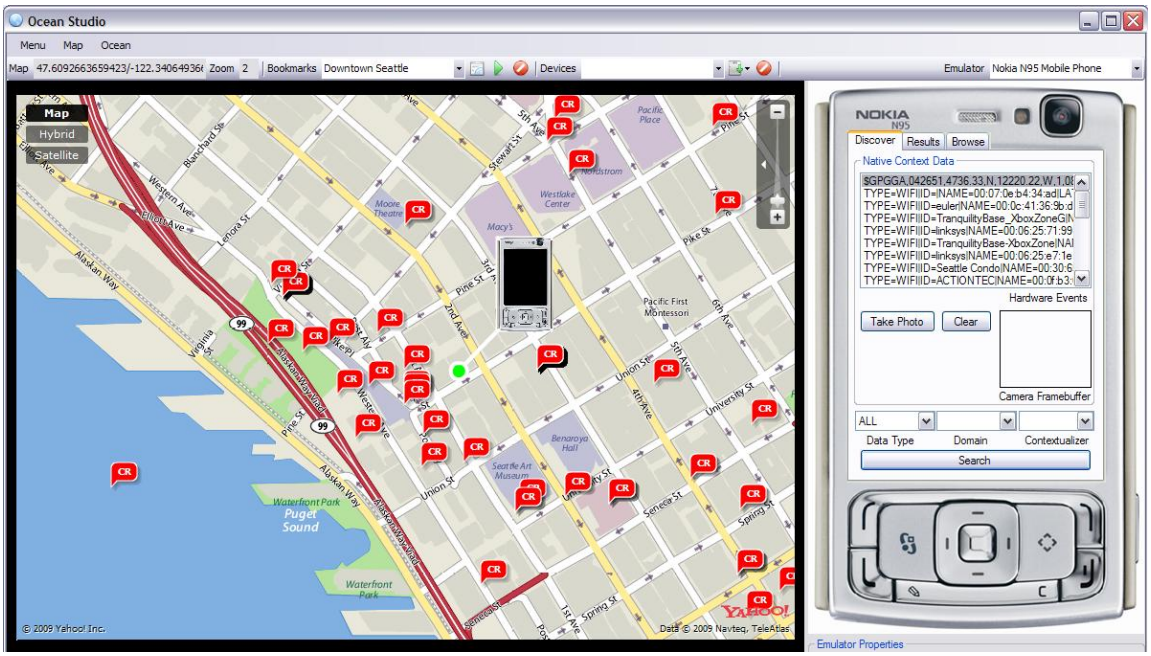


Figure 81: Ocean Studio rendering crawled CRs and provisioning PlaceLab beacon data

A Sony VAIO VGN-FZ21Z laptop with 2GB of RAM and a 2.2GHz Centrino Duo processor was used to run both the Ocean Studio application and embedded Ocean RI during LinkFlow validation. Although Ocean Studio is not performance optimized, it proved capable of consistently provisioning PlaceLab context information to the active emulator in less than 500ms. Similarly, although the embedded Ocean RI prototype relies on naïve query processing, it proved capable of processing Discovery Requests within 1000ms to 3000ms (depending on the amount of NCD provided as query terms). Although not discussed further, we note that many real-world applications must be designed to accommodate additional query handling delays (e.g. due to variations in network conditions). During the evaluations, each active emulator's local hardware devices were activated and deactivated to check the effect on context acquisition. As expected, when a given hardware device was turned off,

the active emulator stopped acquiring the associated NCD type. Moreover, switching between the Nokia N95 and Dell x50v emulators resulted in different sets of detected NCD, due to the differences in the device's capabilities (e.g. the N95 has inbuilt GPS hardware unlike the Dell x50v).

During the initial evaluations of the advanced LinkFlow scenarios, the active emulator was positioned in multiple locations throughout the downtown Seattle area to validate context provisioning and modeling. The LinkFlow application acquired and modeled NCD appropriately as the emulator was moved within the scenario designer (i.e. reasonable NCD results appeared within the emulator's NCD list). As the emulator was moved, multiple Discovery Requests were performed using a variety of query constraints and modeled NCD as query terms. Discovery Results accurately matched the scenario configurations rendered within the scenario designer according to both qualitative and quantitative metrics. For example, moving the emulator physically closer to a given CR resulted in increasing similarity scores within LinkFlow's Results View. Moreover, logging output from the Ocean RI provided quantitative validation by outputting expanded similarity score metrics that corresponded to the visual results (according to the **`GeoPointHandler`** linear similarity model described in section 4.4.1).

Query constrains were also evaluated. For example, selecting the WikipediaCrawler as a Contextualizer and changing the emulator's preferred language (resulting in a change in modeled NCD) correctly constrained Discovery Results to Wikipedia CR data in the specified language (e.g. German or English). Similarly, alterations to the Data Type query constraint resulted in appropriate changes to subsequent discovery results. For example, changing the Data Type constraint to "Image" isolated results from Flickr, whereas changing the Data Type constraint to "Feed" isolated CR data from Twitter (note that in realistic Ocean scenarios real-world data types would be used – e.g. **`MIME`** types as per [327]).

Discovery Results accurately rendered within the LinkFlow Results View, allowing users to survey contextually relevant CR metadata (i.e. similarity score, personalization score and title). Selection of CR metadata from the Results View (by double-clicking) resulted in the proper rendering of the Descriptive Metadata's associated URI within LinkView's Results View using the embedded Web browser previously described. As the aforementioned context-aware crawlers extracted Contextualized Resource data from Flickr, Twitter and Wikipedia, rendered CR URIs reflected real-world user-generated content. Moreover, as the crawlers formulated mobile content URIs for each CR, rendered content appeared well-proportioned within LinkView's resource constrained browser. As an example, a discovered Contextualized Resource from Flickr[61] is shown rendered within the active emulator in Figure 82.

[61] Photo credit: http://m.flickr.com/photos/a_ninjamonkey/3306601531 (published under the Attribution-Noncommercial-No Derivative Works 2.0 Generic license).

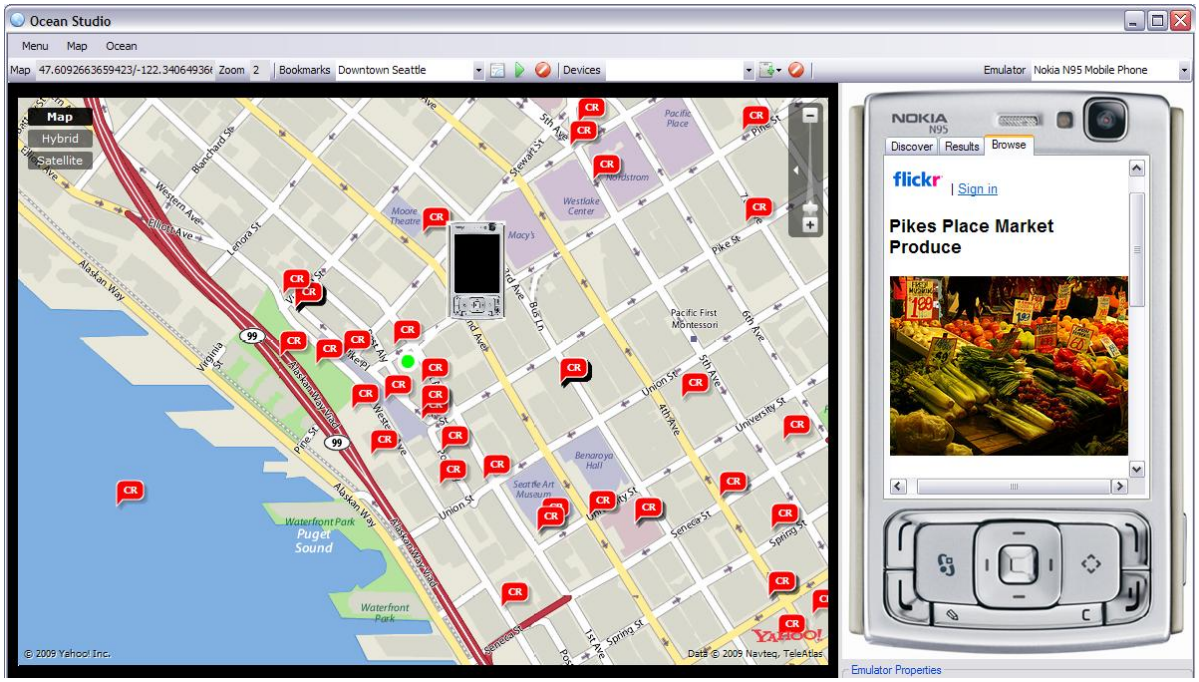


Figure 82: A discovered Contextualized Resource rendered within the active emulator

8.7 Validating Community-enhanced LinkView Query Processing

In the basic LinkFlow scenario described in section 8.5, switching from the Nokia N95 emulator to the Dell x50v emulator prevented CR discoveries due to context mismatch arising from the lack of GPS hardware within the Dell emulator. As discussed in section 7.2, context mismatch refers to the situation where an Ocean Discovery Request's query terms do not sufficiently match the Context Metadata used to create persisted CRs. In such cases, query effectiveness is diminished and contextually-relevant Resources may remain invisible to Ocean application. As per the Ocean discovery approach described in section 6.4, persisted CRs from the crawler framework require similar geo-positioning NCDs for discovery. To help improve query results for the Dell emulator, we activated the context-aware query expansion mechanisms built into the embedded Ocean RI.

As described in section 7.2.4, the Ocean Registry provides an Association Discovery Framework (ADF) that forms the foundation for Ocean's context-aware query expansion techniques. Briefly, as communities of autonomous Ocean applications make Discovery Requests using the Ocean Registry's Resource Discovery API, the ADF automatically models and maintains a continually evolving data model of **`ContextAssociation`** objects, which are used to automatically expand incoming Discovery Requests with supplemental search terms (see section 7.2.5). In order to improve the quality of supplemental search terms, the Ocean ADF exploits the domain knowledge encapsulated within implementations of the **`AssociationDiscoverer`** interface provided by Context Experts for a given Context Handler (see section 7.2.4).

To validate context-aware query expansion approach using Ocean Studio, we developed a PlaceLab agent system capable of parsing the NetStumbler data contained within the PlaceLab beacon dataset and making standard Ocean Discovery Requests using the beacon data as query terms (simulating a large number of real-world Ocean requests). The agent's Discovery Requests were formulated such

that context sharing was enabled; allowing the Ocean RI to anonymize and pass the query NCD to the ADF for processing. As PlaceLab agents make Discovery Requests, the ADF automatically extracts, models and persists **`ContextAssociation`** data using the process described in section 7.2. In our evaluation scenario, PlaceLab agent requests consisted of both transceiver information and geo-location information; hence, the resultant **`ContextAssociation`** data were created using an **`RFPositionHandler`** as the parent and the **`GeoPointHandler`** as the child (using an arbitrary initial confidence of 0.5). Additional **`ContextAssociation`** sightings (e.g. the same 802.11 access point sighted by another agent) were merged with the existing **`ContextAssociation`** using a simple step-wise confidence function that increased confidence to a maximum of 0.8. While **`ContextAssociation`** creation and merging functionality were implemented in both versions of the Ocean RI, advanced functionality such as **`ContextAssociation`** ageing and pruning were left for future work.

Context-aware query expansion was validated as follows: First, we used the agent system to process all 7,943 beacons contained within the PlaceLab dataset described in section 8.4. To help improve map rendering performance, agent requests were constrained to within 500 meters of the active emulator. Next, the Dell emulator was selected and positioned within the downtown Seattle area. Importantly, while the Dell emulator does not have inbuilt GPS hardware, it is capable of acquiring and modeling nearby 802.11 and Bluetooth transceivers using local hardware and appropriate Aladin plug-ins. Initially we validated that the Dell emulator was *incapable* of discovering persisted CR data using its locally modeled NCD by performing several Discovery Requests with "Expand Query" set to "False" in the emulator's query preferences. As expected, no CRs were discovered. Next, we set "Expand query" to "True" and performed several additional Discovery Requests. As expected, in locations where the PlaceLab agents had previously shared context information, the Dell emulator was capable of discovering previously invisible CRs persisted using **`GeoPointHandlers`**. Notably, expanded discovery results accurately matched the scenario configurations displayed within the scenario designer according to both qualitative and quantitative metrics. For example, moving the emulator physically closer to a given CR resulted in increasing similarity scores within LinkView's Results View (according to the nearby **`ContextAssociations`**). Logging output from the embedded Ocean RI provided quantitative validation of the visual results by presenting corresponding expanded similarity score values.

An overview of the expanded Resource Discovery results obtained by the Dell emulator is shown in Figure 83. In this figure, **`ContextAssociation`** data modeled by the ADF (in response to previous PlaceLab agent requests) are shown as semi-transparent circles, whose radius values correspond to the theoretical maximum range of the parent **`RFPositionHandler`** (according to transceiver type). Second, as described in section 7.2.5, supplemental query terms added to the Dell's Discovery Requests are weighted using associated **`ContextAssociations`** confidence values when performing similarity calculations. Accordingly, the similarity scores presented in the Results View are lower than those discovered by the Nokia emulator with inbuilt GPS hardware. Finally, while the embedded Ocean RI does not provide an implementation of the Resource personalization approach described in section 7.3, the Results View indicates how personalized query results could appear (using preference values in the unit interval ($\{0 \leq x \leq 1\}$). In this hypothetical example, the emulator's user is

predicted to have higher affinity for Resources from Wikipedia, which can be used by the LinkFlow application logic (or the user) to help improve Resource selection.

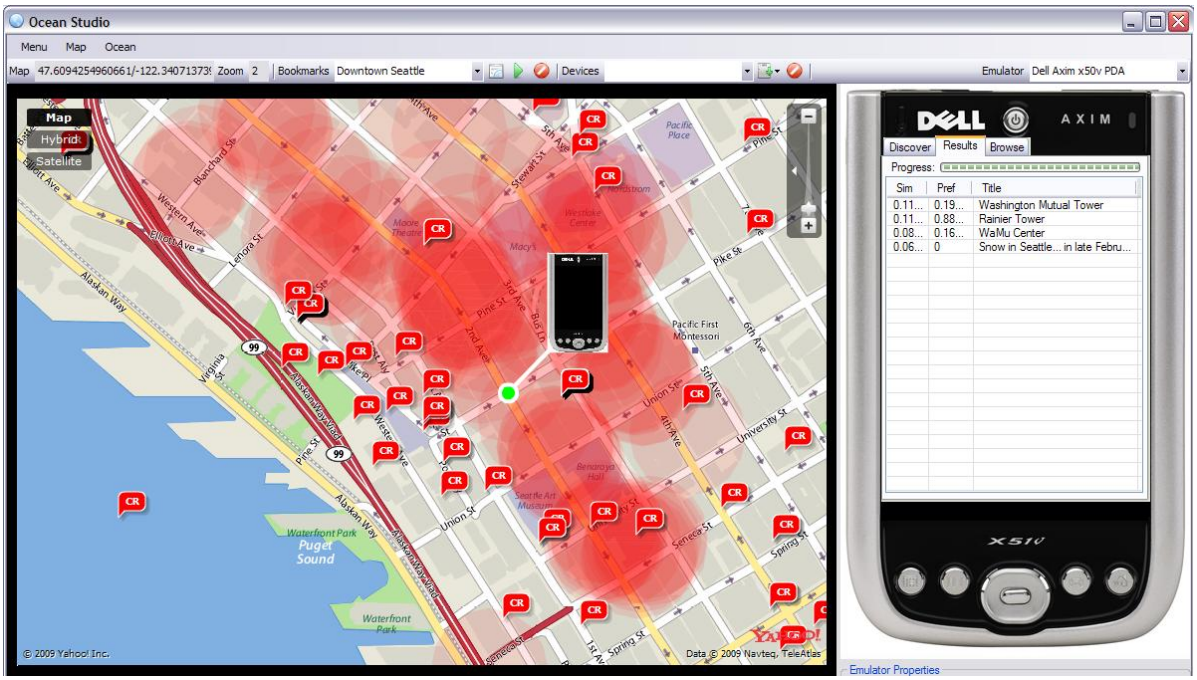


Figure 83: Resource discovery results obtained using context-aware query expansion

As a means of further validating the Ocean ADF, we implemented an additional Context Metadata type called the **`QRCodeHandler`**, which is capable of parsing the Quick Response two-dimensional bar code format (QR Code) developed by Denso-Wave Corporation[62]. Notably, the QR Code format supports comparatively large capacity data storage (e.g. 7,089 numerals, 4,296 alphanumeric characters or 2,953 bytes) and provides Reed–Solomon error correction for use in decoding low-quality or partial barcode images. Moreover, the QR Code format is supported by a broad range of mobile devices (e.g. the Open Source QR Code Library[63] supports most Java-based mobile devices). Within the embedded Ocean RI, QR Code handling is encapsulated within a **`QRCodeHandler`**, which utilizes the ThoughtWorks QRCode library[64] for low-level image processing and data extraction. In addition, Ocean Studio's Resource contextualizer tool provides form-based support for creating Contextualized Resources based on imported QR Code image data. Importantly, the **`QRCodeHandler`** implements the **`AssociationDiscoverer`** interface (see section 7.2.4), which allows the Ocean ADF to automatically discover, persist and manage **`ContextAssociations`** based on Discovery Requests containing both a QR Code data and geo-position data.

To evaluate the **`QRCodeHandler`**, we used Ocean Studio to design a scenario whereby a Wikipedia article about the University of Luebeck was contextualized using a single **`GeoPointHandler`**. Next, we selected the Nokia N95 as the active emulator and positioned it within the Discoverability Context of the previously created CR. The Nokia emulator's integrated camera was then used to obtain a QR

[62] http://www.denso-wave.com/qrcode/index-e.html
[63] http://qrcode.sourceforge.jp/
[64] http://www.twit88.com/home/opensource/qrcode

Code image from the device's environment (e.g. image data from a stationary sign). The Nokia emulator's context framework properly acquired and modeled local NCD, which included the QR Code image as a raw bitmap and geo-position data in the NMEA format. Next, context sharing was activated and a single Ocean Discovery Request was performed using the locally acquired NCD as query terms. As context sharing was enabled, the embedded Ocean RI performed association discovery after query processing was completed. In this case, the ADF modeled and persisted a single **`ContextAssociation`**, which included the Discovery Request's **`QRCodeHandler`** as the parent Context Metadata entity and its colleague **`GeoPositionHandler`** as the child Context Metadata entity.

Once the Discovery Request was completed for the Nokia emulator, the Dell emulator was then selected (its position was maintained within the Discoverability Context of the manually created CR previously introduced). Since the Dell emulator lacks inbuilt GPS hardware, its NCD list did not contain geo-positioning data. As expected, Discovery Requests performed by the Dell emulator failed to discover the manually created CR due to context mismatch. Next, the Dell emulator's attached camera was used to obtain the *same* QR Code image previously encountered by the Nokia emulator. The Dell emulator's context framework properly acquired and modeled the QR Code image, which appeared within its NCD list as a raw bitmap. Next, query expansion was enabled using the Dell emulator's Ocean preference settings and a single Discovery Request was performed using the embedded Ocean RI. Using the Association discovery approach described in section 7.2.5, the embedded Ocean RI's ADF automatically discovered the **`ContextAssociation`** previously created from the Nokia emulator's request. In this case, the QR Code bitmap within the Dell emulator's NCD list was used to match the **`QRCodeHandler`** parent of the previously persisted **`ContextAssociation`**. Next, the discovered **`ContextAssociation's`** child Context Metadata entity (i.e. the **`GeoPositionHandler`** provided by the Nokia emulator) was added to the Dell's query as a Supplemental Metadata query term. As described in section 7.2.5, supplemental query terms are weighted using the **`ContextAssociation's`** confidence value when performing similarity calculations. Accordingly, the Dell emulator was capable of discovering the previously invisible CR; however, the similarity value within the Discovery Results was lower than the value discovered by the Nokia emulator with inbuilt GPS hardware.

8.8 Chapter Summary

This chapter presented an example scenario as a means of clarifying core Ocean concepts and validating the Ocean approach using significant quantities of real-world data. It began by describing the experimental setup, which included the development of an Ocean application development environment, called Ocean Studio. Ocean Studio provides an integrated suite of development and evaluation tools, including a scenario designer with an integrated context provisioning framework; an emulator framework with two implemented emulator devices; a Resource contextualizer tool; and an embedded version of the Ocean RI designed for rapid prototyping. The next section introduced the example application scenario, called LinkView, which extends conventional context-aware Web browsing with core features of the Ocean approach. The next section described the LinkView data acquisition methodology and presented two related tools, including Ocean Studio support for PlaceLab beacon importing and a context-aware crawler framework, which included three example crawlers. The crawling framework was used to extract over 68,000 Contextualized Resources from

popular Web-based data sources, including the Flickr photo sharing Website; the Twitter micro-blogging platform; and the community-based Wikipedia encyclopedia project.

Next, the LinkFlow application was evaluated using several approaches. First, the basic LinkFlow scenario was validated by manually creating Contextualized Resources within Ocean Studio and testing the LinkFlow application model using emulator framework and scenario designer. Next, the crawled CR data and acquired PlaceLab context-sources were imported into OceanStudio. As expected, the addition of significant real-world data led to Resource Discovery challenges in the form of context mismatch and information overload. Next, Ocean Studio's PlaceLab agent system was presented as a mechanism for simulating the behavior of large Ocean application communities. Using the agent system, PlaceLab beacon data were used to perform large numbers of Ocean Discovery Requests (with context sharing enabled); allowing Ocean's Association Discovery Framework to automatically model **`ContextAssociation`** data for use in context-aware query expansion. Finally, Ocean's community-enhanced query approaches were validated by performing a series of Discovery Requests using a resource constrained emulator. Notably, the resource constrained emulator was capable of effectively overcoming context mismatch and information overload by leveraging Ocean's context-aware expansion and Resource personalization techniques.

Chapter 9
Conclusion

9.1 Summary of Contributions

This chapter concludes the dissertation by presenting a summary of contributions and a discussion of directions for future research. As discussed in Chapter 1, the central thesis of this dissertation is that traditional context-aware computing approaches are ill-suited for building truly ubiquitous, large-scale networked systems and generally fail to promote significant developer adoption and end-user participation. As a result, the considerable wealth of intriguing and innovating context-aware computing techniques remain consigned to small-scale deployments and research prototypes; existing as isolated islands of niche functionality that are far removed from everyday use [79, 288]. Given the rapidly increasing capabilities and sophistication of many everyday environments, the lack of large-scale context-aware computing systems was introduced as the key motivation for this research. The following paragraphs provide a summary of key contributions in this regard.

Chapter 2 included a survey of background material and a presentation of related work. Related work was presented using a layered conceptual framework that addressed context acquisition; context modeling and representation; context management and provisioning; and context-aware component interoperation. Based on a representative sampling of state-of-the-art approaches, it was observed that current context-aware systems are typically created with the assumption that the underlying network infrastructure, hardware devices, application components and context mechanisms are well-known a-priori and contained within a limited and controlled administrative domain. Hence, many approaches mandate expensive and invasive deployment of context instrumentation; require domain-specific network configurations; rely on specially outfitted mobile devices; adopt enterprise-specific distributed middleware; and generally lack support for spontaneous cross-domain component interoperation. Further, the considerable expense and effort required to devise, implement and deploy such systems often promotes a top down development approach intended to address niche problem domains where the requisite support infrastructure can be readily provided, administrative access is available and return on investment is assured. Indeed, several recent surveys [22, 62, 106] indicate that existing systems generally fail to provide ubiquitous accessibility; resulting in a pronounced lack of developer adoption and end-user participation.

Based on the above observations, Chapter 3 began by identifying several key challenges that have prevented the emergence of large-scale context-aware systems. These challenges included the deployment of ubiquitous context infrastructure; the widespread availability of suitable data communication networks; the scalability of underlying middleware; and techniques for spontaneous cross-domain component interoperation. Related, this section also identified several “foundations” for addressing each of the aforementioned challenges. These foundations included (respectively) Aladin-based context acquisition and modeling; public Internet infrastructure; conventional Web architecture; and RESTful component interoperation. Based on these foundations, Chapter 3 derived our novel context-aware computing approach, called Ocean, which aims to capture the entrepreneurial spirit of modern Web architecture as a means of supporting large-scale context-aware systems. The derivation

of the Ocean approach was based on the application of the aforementioned foundations as a coordinated set of design constraints that are intended to restrict the Ocean application model and resultant infrastructure. Related, Ocean's principle stakeholders were also described, including Ocean Developers, Context Experts, Contextualizers and Application Developers. Finally, the Ocean Reference Implementation (Ocean RI) was introduced as a means of validating the Ocean approach in terms of the design principles and approach constraints described in sections 3.4.1 and 3.4.2 respectively.

To overcome the Web's inherent context-mediation limitations, Chapter 4 presented Ocean's foundational architectural abstraction, called the *Contextualized Resource* (CR). Briefly, the CR abstraction extends the conventional Web Resource model with supplemental General and Context Metadata intended to constrain the *Discoverability Context* of an associated Web Resource. Recall from Definition 1 that a Discoverability Context is defined as *the set of contextual criteria that must be fulfilled before a Resource is considered relevant to the interaction between a user and an Ocean application, including the user and application themselves.* A secondary contribution from Chapter 4 included the definition of the Context Metadata abstraction, which provides a domain-neutral interface whereby the syntax and semantics of a given context domain can be encapsulated by Context Experts (concrete implementations of the ContextMetadata interface are termed Context Handlers). To clarify the CR model, a detailed interface description was provided along with a survey of relevant similarity modeling techniques. The chapter concluded with the presentation of a CR example that included two ContextHandler implementations.

Building on the CR abstraction, the primary contribution from Chapter 5 included the definition of key approaches for supporting large-scale context-aware computing using Ocean. First, the basic Ocean philosophy was defined as *a simple, accessible and scalable mechanism for mobile applications to discover, select and compose contextually-relevant Web Resources in-situ at runtime*. Based on this philosophy, Ocean's Web-centric application model was defined as an extension of the client-centric mashup style (which aligns well with Ocean's approach constraints defined in Chapter 3). In order to adapt the Ocean application model to the requirements of conventional Web architecture, a complimentary Contextualized Resource Registry (Ocean Registry) was presented as a means of facilitating wide-area Resource contextualization and discovery. Next, two related community-based contribution models were presented. First, a preliminary Context Handler contribution approach was proposed as a means of facilitating the extension of the Ocean Registry by external Context Experts (based on an adaptation of the Java Community Process). Second, an open Contextualization API was proposed as a mechanism for promoting large-scale Resource contextualization (allowing *any* Contextualizer to contextualize *any* Resource with *any* combination of Ocean Metadata).

Chapter 6 contributed techniques for persisting, indexing and discovering CRs from within the Ocean Registry. As CRs represent complex data structures that cannot be effectively indexed or queried using classical database techniques, related work regarding similarity search mechanisms was first presented. Using the related work as a foundation, the Ocean Registry's Persistence Framework was derived. Briefly, the Persistence Framework allows CRs to be efficiently stored and indexed for rapid retrieval according to domain-specific indexing and data modeling techniques. Notable

contributions in this regard included the IndexManager interface and its related software architecture. Another contribution from Chapter 6 included the development of the Ocean Discovery Framework, whereby contextually-relevant CRs can be retrieved from the Ocean Registry's Persistence Framework and returned to requesting applications. Related, the Ocean Registry's Discovery API and Discovery protocol were defined as a means of allowing applications to discover contextually-relevant Resources using native context data (NCD) as query terms. The Discovery Framework is based on a Query Object abstraction and complimentary multi-feature similarity search (MFSS) model. Importantly, Ocean MFSS operates in conjunction with the Persistence Framework and Context Metadata interfaces previously described; allowing NCD to be translated into appropriate Context Handlers and compared to persisted CRs within the Ocean Registry.

The primary contributions from Chapter 7 included two community-based computational models designed to address the significant challenges arising from Ocean's Web-scale focus. First, the challenge of context mismatch was introduced, whereby an Ocean application may not be capable of generating the requisite native context data necessary to discover contextually-relevant Resources in a given environment; reducing query effectiveness. To address this challenge, a context-aware query expansion technique was developed. Briefly, context-aware query expansion supplements incoming Discovery Requests with additional, contextually-relevant query terms, which have been extracted and modeled from previous Discovery Requests made by diverse Ocean applications. Related, Ocean's Association Discovery Framework (ADF) was defined as the entity responsible for automatically extracting domain-specific `ContextAssociation` data from incoming Discovery Requests (using an extension of the Context Metadata interface previously described). Next the challenge of information overload was introduced, whereby Discovery Results that include a large number of similarly scored Resources may become difficult to differentiate based on the Descriptive Metadata alone; resulting in ineffective Resource selection. To address this challenge, Ocean's Resource personalization approach was described. Briefly, Resource personalization estimates a user's affinity for discovered contextually-relevant Resources based on the captured preferences of the Ocean User community. Related Ocean's Recommendation Engine and preference modeling techniques were described in detail. Finally, the Recommendation Engine was validated using an implementation of the Slope One recommender algorithm.

Finally, the primary contribution from Chapter 8 was the presentation an example Ocean application scenario as a means of clarifying core Ocean concepts and validating the overall Ocean approach. Notably, the example scenario utilized real-world context-sources, large-scale CR datasets and more realistic models of Ocean community behavior. The chapter began with a description of the experimental setup, which included the development of an Ocean application development environment, called Ocean Studio, and an embedded Ocean reference implementation (RI) designed for rapid prototyping. Next, the example application scenario, called LinkFlow, was introduced as the conceptual foundation for the remainder of the chapter. Next, LinkFlow's data acquisition methodology and related toolset were described. First, Ocean Studio was enhanced with the ability to import beacon capture datasets from the PlaceLab project. Next, a context-aware Crawler framework was developed to extract large numbers of CRs from popular Web applications such as Flickr, Twitter and Wikipedia. The LinkFlow scenario was then validated within the Ocean Studio development

environment using both basic and enhanced scenarios. Notably, the advanced scenarios included an integration of the previously acquired native context sources and crawled CR data as a means of inducing context mismatch and information overload. Based on the community-based computational models introduced in Chapter 7, it was demonstrated how discovery query performance could be improved in real-world scenarios by applying Ocean's context-aware query expansion and Resource personalization techniques. Related, qualitative and quantitative query improvements were demonstrated using Ocean Studio's scenario designer and debugging output from the embedded Ocean RI.

9.2 Directions for Future Research

As described throughout this dissertation, Ocean represents a *preliminary approach* for enabling Web-scale context-aware computing. Notably, Ocean's conceptual foundations and practical infrastructure design co-opt *existing* context-sources, network infrastructure and distributed middleware. As such, Ocean's intentionally broad focus provides a variety of opportunities for future research. First, as described in section 5.3, the Ocean Registry requires a highly scalable software architecture to support Ocean's Web-scale focus. Although Ocean's application model addresses distributed component scalability and independent deployment through an extension of conventional Web architecture, Resource contextualization and discovery are currently reliant on the federated Ocean Registry architecture presented in section 5.3.1. During the presentation of this architecture (see section 6.3.2), we identified several promising technical foundations for developing large-scale version of the Ocean infrastructure. Examples of these foundations include the Hadoop Map Reduce implementation, HDFS, Amazon EC2 architecture, etc. However, aside from the two Ocean reference implementations created to validate core aspects of the Ocean approach, large-scale Ocean architecture designs have not yet been developed or validated. Towards this end, an exploration of high performance and alternative architectures (e.g. those based on peer-to-peer models) represents an important area for future investigations.

Related to the Ocean Registry, effective Resource contextualization and discovery are contingent upon the contribution of a broad range of real-world Context Handlers. As discussed in section 4.3.3, the development of Context Handlers is often highly complex and domain-specific. Importantly, such complexity often necessitates participation by external Context Experts as a means of capturing the subtle semantics of a given context domain. Hence, the development and integration of a variety of real-world Context Handlers remains an important area of exploration in Ocean. Related, the JCP was identified as a suitable conceptual framework for the controlled contribution of Context Handlers within the Ocean Registry. Towards this end, a preliminary community-based Context Handler contribution approach was presented in section 5.5; however, this contribution model has not been fully implemented or evaluated in real-world scenarios. Hence, an important area of future work is the elaboration of the Context Handler contribution process and the development of related infrastructure. Importantly, developed contribution models must promote Context Expert participation while simultaneously shielding the Ocean Registry from poorly engineered Context Handler implementations.

As described in Chapter 6, Context Handler indexing, persistence and similarity modeling represent critical aspects of the Ocean Registry. In our current Ocean RI versions, object-based database techniques are used to accommodate CR persistence and indexing based on General Metadata; however, complex indexing based on Context Metadata is largely unexplored at present. As such, Ocean's Persistence and Discovery Frameworks represent important areas for future work. Regarding the Persistence Framework, notable focus areas include the development of real-world indexing techniques for Context Metadata; investigations into suitable data models; and an exploration of efficient means for processing large numbers of resource-intensive indexing tasks. Regarding the Discovery Framework, notable focus areas include the identification and extension of additional similarity search algorithms; further elaboration of the Context Metadata abstraction with regard to MFSS; validation using heterogeneous Context Handler implementations; and the development of query optimization techniques.

As discussed in section 7.2, Ocean's context-aware query expansion approach is designed to improve Discovery Results using supplemental query terms modeled from incoming Discovery Requests provided by the Ocean application community. Ocean's Association Discovery Framework (ADF) was used to validate techniques for automatic **`ContextAssociation`** discovery using domain-specific mechanisms provided by Context Experts. In both the server-based and embedded Ocean RIs, preliminary versions of the ADF included basic support for **`ContextAssociation`** extraction, persistence, merging and discovery; however, ADF functionality such as **`ContextAssociation`** pruning and complex merging have not been explored. Further, as the number of **`ContextAssociation`** objects increases within the Context Association Manager's data store, efficient **`ContextAssociation`** discovery becomes a concern for rapid query processing. In this regard, section 7.2.4 suggested that **`ContextAssociation`** indexing could take advantage of the IndexManager interface previously described; however, this approach remains unexplored at present. Related, Ocean provides a Recommender Engine designed to support Resource personalization based on a variety of recommender algorithms. At present, Ocean's Recommender Engine has been validated using the SlopeOne algorithm (see section 7.3.3); however, investigations of other recommender algorithms are still needed (along with analyses of related performance characteristics and persistence requirements).

Finally, the development of Ocean application scenarios represents a broad area of future research. As described, Ocean is designed to support the emergence of new classes of context-aware Web applications capable of in-situ, cross-domain component discovery and interoperation. As with any generalized computing approach, forecasting successful application domains is often a futile exercise from the perspective of the infrastructure. Indeed, the most innovative applications will likely be uncovered by intrepid developers who understand the possibilities of the generalized Ocean approach with regard to the subtle details of a specific problem domain. As previously discussed, the challenge of addressing such application domains is predicated on interrelated contributions from a wide variety of Context Experts, Contextualizers and Application Developers; however, we suggest that Ocean's community-centric design provides a first step towards Web-scale context-aware computing.

References

1. Abowd, G.D., Atkeson, C.G., Hong, J., Long, S., Kooper, R. and Pinkerton, M. Cyberguide: A Mobile Context-Aware Tour Guide. *Wireless Networks*, 3 (5), 421-433, 1997.
2. Adams, N., Gold, R., Schilit, B.N., Tso, M.M. and Want, R. An Infrared Network for Mobile Computers. Mobile & Location-Independent Computing Symposium on Mobile & Location-Independent Computing Symposium, Cambridge, Massachusetts, USA, USENIX Association, 1993.
3. Ahmed, R., Boutaba, R., Iraqi, Y., Li, T., Limam, N., Xiao, J. and Ziembicki, J. Resource and Service Discovery in Large-Scale Multi-Domain Networks. *IEEE Communications Surveys & Tutorials*, 9 (4), 2-30, 2007.
4. Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M.-T., Sheth, A. and Verma, K. Web Service Semantics (WSDL-S). Retrieved from http://www.w3.org/Submission/WSDL-S/
5. Akman, V. and Surav, M. The Use of Situation Theory in Context Modeling. *Computational Intelligence*, 13, 427-438, 1997.
6. Alvestrand, H. Tags for the Identification of Languages (RFC: 3066). Retrieved from http://www.ietf.org/rfc/rfc3066.txt
7. Amazon Web Services. Authenticating REST Requests. Retrieved from http://docs.amazonwebservices.com/AmazonS3/latest/index.html?RESTAuthentication.html
8. Ames, M. and Naaman, M. Why We Tag: Motivations for Annotation in Mobile and Online Media. In *Proceeding of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 971-980, ACM Press, New York, NY, USA, 2007.
9. Anderson, D.P. Boinc: A System for Public-Resource Computing and Storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pp. 4-10, IEEE Computer Society, 2004.
10. Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M. and Werthimer, D. Seti@Home an Experiment in Public-Resource Computing. *Communications of the ACM*, 45 (11), 56-61, 2002.
11. Arsanjani, A., Curbera, F. and Mukhi, N. Manners Externalize Semantics for on-Demand Composition of Context-Aware Services. In *Proceedings of the IEEE International Conference on Web Services, ICWS'04*, p. 583, IEEE Computer Society, 2004.
12. Attar, R. and Fraenkel, A.S. Local Feedback in Full-Text Retrieval Systems. *Journal of the ACM*, 24 (3), 397-417, 1977.
13. Attiya, H. and Welch, J. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics.* John Wiley & Sons Inc., Hoboken, New Jersey, USA, 2004.
14. Atzeni, P., Catarci, T. and Pernici, B. Mais: Multichannel Adaptive Information Systems. *World Wide Web*, 10 (4), 345-347, 2007.
15. Avery, C. and Zeckhauser, R. Recommender Systems for Evaluating Computer Messages. *Communications of the ACM*, 40 (3), 88-89, 1997.
16. Bacon, J., Bates, J. and Halls, D. Location-Oriented Multimedia. *IEEE Personal Communications*, 4 (5), 48-57, 1997.
17. Bada, M., Turi, D., McEntire, R. and Stevens, R. Using Reasoning to Guide Annotation with Gene Ontology Terms in Goat. *ACM SIGMOD Record*, 33 (2), 27-32, 2004.
18. Baeza-Yates, R.A., Cunto, W., Manber, U. and Wu, S. Proximity Matching Using Fixed-Queries Trees. 5th Annual Symposium on Combinatorial Pattern Matching, Springer-Verlag, 1994.
19. Bahl, P. and Padmanabhan, V.N. Radar: An in-Building Rf-Based User Location and Tracking System. In *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 2000*, pp. 775-784, IEEE Computer Society, 2000.
20. Balabanovi, M. and Shoham, Y. Fab: Content-Based, Collaborative Recommendation. *Communications of the ACM*, 40 (3), 66-72, 1997.

21. Baldauf, M., Dustdar, S. and Rosenberg, F. A Survey on Context-Aware Systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2 (4), 263-277, 2007.
22. Bani-Ahmad, S., Cakmak, A., Özsoyoglu, G. and Al-Hamdani, A. Evaluating Publication Similarity Measures. *IEEE Data Engineering Bulletin*, 28 (4), 21-28, 2005.
23. Bardram, J.E. The Java Context Awareness Framework (Jcaf) – a Service Infrastructure and Programming Framework for Context-Aware Applications. Pervasive 2005, Munich, Germany, 2005.
24. Barros, A.P. and Dumas, M. The Rise of Web Service Ecosystems. *IT Professional*, 8 (5), 31-37, 2006.
25. Bauer, J. *Identification and Modeling of Contexts for Different Information Scenarios in Air Traffic*. Diplomarbeit, 2003.
26. Beatty, J., Kakivaya, G., Kemp, D., Kuehnel, T., Lovering, B., Roe, B., John, C.S., Simonnet, G., Walter, D., Weast, J., Yarmosh, Y. and Yendluri, P. Web Services Dynamic Discovery (Ws-Discovery Draft). Schlimmer, J. (Ed.), 2005.
27. Becla, J. and Wang, D.L. Lessons Learned from Managing a Petabyte. In *Second Biennial Conference on Innovative Data Systems Research (CDIR 2005)*, pp. 70-83, 2005.
28. Beigl, M., Gellersen, H.W. and Schmidt, A. Mediacups: Experience with Design and Use of Computer-Augmented Everyday Artefacts. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 35 (4), 401-409, 2001.
29. Belotti, R., Decurtins, C., Norrie, M.C. and Python, E. A Context-Aware Component Registry for Ubiquitous and Mobile Applications. Workshop on Ubiquitous Mobile Information and Collaboration Systems, UMICS'05, Porto, Portugal, 2005.
30. Berners-Lee, T. Axioms of Web Architecture: Metadata. Retrieved from http://www.w3.org/DesignIssues/Metadata.html
31. Berners-Lee, T. The Enquire System Short Description. Retrieved from http://infomesh.net/2001/enquire/manual/
32. Berners-Lee, T. Web Architecture from 50,000 Feet. Retrieved from http://www.w3.org/DesignIssues/Architecture.html
33. Berners-Lee, T. The World Wide Web: Past, Present and Future. *IEEE Computer*, 29 (10), 69-77, 1996.
34. Berners-Lee, T., Fielding, R. and Masinter, L. RFC: 3986 - Uniform Resource Identifier (URI): Generic Syntax. Retrieved from http://tools.ietf.org/html/rfc3986
35. Berry, M.W., Dumais, S.T. and O'Brien, G.W. Using Linear Algebra for Intelligent Information Retrieval. *SIAM Review*, 37 (4), 573-595, 1995.
36. Bershad, B.N., Anderson, T.E., Lazowska, E.D. and Levy, H.M. User-Level Interprocess Communication for Shared Memory Multiprocessors. *ACM Transactions on Computer Systems (TOCS)*, 9 (2), 175-198, 1991.
37. Blackstock, M., Lea, R. and Krasic, C. Toward Wide Area Interaction with Ubiquitous Computing Environments. *Smart Sensing and Context*, pp. 113-127, Springer Berlin/Heidelberg, 2006.
38. Bondi, A.B. Characteristics of Scalability and Their Impact on Performance. 2nd International Workshop on Software and Performance, Ottawa, Ontario, Canada, ACM Press, New York, NY, USA, 2000.
39. Boriah, S., Chandola, V. and Kumar, V. Similarity Measures for Categorical Data: A Comparative Evaluation. Society for Industrial and Applied Mathematics, SIAM Data Mining Conference, Atlanta, GA, USA, 2008.
40. Bouzy, B. and Cazenave, T. Using the Object Oriented Paradigm to Model Context in Computer Go. The First International and Interdisciplinary Conference on Modeling and Using Context, Context'97, Rio de Janeiro, Brasil, 1997.
41. Breese, J.S., Heckerman, D. and Kadie, C. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. The Fourteenth Conference on Uncertainty in Artificial Intelligence, UAI-98, San Francisco, USA, 43-52, 1998.
42. Brown, P.J., Bovey, J.D. and Chen, X. Context-Aware Applications: From the Laboratory to the Marketplace. *IEEE Personal Communications*, 4 (5), 58-64, 1997.

43. Brumitt, B., Meyers, B., Krumm, J., Kern, A. and Shafer, S.A. Easyliving: Technologies for Intelligent Environments. In *Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing*, pp. 12-29, Springer-Verlag, 2000.
44. Buschmann, F., Henney, K. and Schmidt, D.C. *Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing.* Wiley & Sons, 2007.
45. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns.* Wiley, 1996.
46. Bustos, B. and Skopal, T. Dynamic Similarity Search in Multi-Metric Spaces. 8th ACM international Workshop on Multimedia Information Retrieval, Santa Barbara, California, USA, ACM Press, New York, NY, USA, 2006.
47. Byrne, D., Lavelle, B., Doherty, A., Jones, G.J. and Smeaton, A.F. Using Bluetooth and GPS Metadata to Measure Event Similarity in Sensecam Images. 5th International Conference on Intelligent Multimedia and Ambient Intelligence, IMAI'07, Salt Lake City, USA, 2007.
48. C. Traina Jr., Traina, A.J.M., Seeger, B. and Faloutsos, C. Slim-Trees: High Performance Metric Trees Minimizing Overlap between Nodes. In *Proceedings of the 7th International Conference on Extending Database Technology: Advances in Database Technology*, pp. 51-65, Springer-Verlag, 2000.
49. Carlson, D.V. *Aladin: An Extensible Ubiquitous Computing Infrastructure*. Master's thesis, University of Luebeck, 2005.
50. Carlson, D.V., Schrader, A. and Busch, D. Modular Framework Support for Context-Aware Mobile Cinema. *Personal and Ubiquitous Computing*, 12 (4), 299-306, 2008.
51. Carpineto, C., de-Mori, R., Romano, G. and Bigi, B. An Information-Theoretic Approach to Automatic Query Expansion. *ACM Transactions on Information Systems, TOIS*, 19 (1), 1-27, 2001.
52. Carpineto, C. and Romano, G. Effective Reformulation of Boolean Queries with Concept Lattices. 3rd International Conference on Flexible Query Answering Systems, Springer-Verlag, 1998.
53. Carpineto, C. and Romano, G. Order-Theoretical Ranking. *Journal of the American Society for Information Science*, 51 (7), 587-601, 2000.
54. Carson, S.D. A System for Adaptive Disk Rearrangement. *Software—Practice & Experience*, 20 (3), 225-242, 1990.
55. Çelik, T. The Microformat Geo Standard. Retrieved from http://microformats.org/wiki/geo
56. Cerf, V.G. and Kahn, R.E. A Protocol for Packet Network Interconnections. *IEEE Transactions on Communications*, 22 (5), 637–648, 1974.
57. Chakrabarti, K. and Mehrotra, S. The Hybrid Tree: An Index Structure for High Dimensional Feature Spaces. In *Proceedings of the 15th International Conference on Data Engineering*, p. 440, IEEE Computer Society, 1999.
58. Chakrabarti, K. and Mehrotra, S. Local Dimensionality Reduction: A New Approach to Indexing High Dimensional Spaces. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pp. 89-100, Morgan Kaufmann Publishers Inc., 2000.
59. Chakraborty, D., Perich, F., Avancha, S. and Joshi, A. Dreggie: Semantic Service Discovery for M-Commerce Applications. Workshop on Reliable and Secure Applications in Mobile Environments at the 20th Symposium on Reliable Distributed Systems, New Orleans, USA, 2001.
60. Chaudhuri, S. and Gravano, L. Optimizing Queries over Multimedia Repositories. *ACM SIGMOD Record*, 25 (2), 91-102, 1996.
61. Chávez, E., Navarro, G., Baeza-Yates, R. and Marroquín, J.L. Searching in Metric Spaces. *ACM Computing Surveys, CSUR*, 33 (3), 273-321, 2001.
62. Chen, G. and Kotz, D. A Survey of Context-Aware Mobile Computing Research. Dartmouth College, 2000.
63. Chen, G., Li, M. and Kotz, D. Design and Implementation of a Large-Scale Context Fusion Network. First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, MOBIQUITOUS 2004, Boston, Massachusetts, USA, 2004.

64. Chen, H., Finin, T. and Joshi, A. Semantic Web in the Context Broker Architecture. In *Proceedings of the 2nd IEEE International Conference on Pervasive Computing and Communications, PerCom'04*, p. 277, IEEE Computer Society, 2004.
65. Chen, H., Perich, F., Finin, T. and Joshi, A. Soupa: Standard Ontology for Ubiquitous and Pervasive Applications In *Proceedings of the 1st International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pp. 258-267, IEEE Computer Society, 2004.
66. Chen, M.Y., Sohn, T., Chmelev, D., Haehnel, D., Hightower, J., Hughes, J., Lamarca, A., Potter, F., Smith, I. and Varshavsky, A. Practical Metropolitan-Scale Positioning for Gsm Phones. 8th International Conference of Ubiquitous Computing, UbiComp 2006, California, USA, 2006.
67. Christensen, E., Curbera, F., Meredith, G. and Weerawarana, S. Web Services Description Language (WSDL) 1.1. Retrieved from http://www.w3.org/TR/wsdl
68. Ciaccia, P., Patella, M. and Zezula, P. M-Tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pp. 426-435, Morgan Kaufmann Publishers Inc., 1997.
69. Clarkson, K.L. A Randomized Algorithm for Closest-Point Queries. *SIAM Journal on Computing*, 17 (4), 830-847, 1988.
70. Cohen, N.H., Purakayastha, A., Wong, L. and Yeh, D.L. Iqueue: A Pervasive Data Composition Framework. In *Proceedings of the 3rd International Conference on Mobile Data Management*, p. 146, IEEE Computer Society, 2002.
71. Cooper, J.W. and Byrd, R.J. Lexical Navigation: Visually Prompted Query Expansion and Refinement. In *Proceedings of the 2nd ACM International Conference on Digital Libraries*, pp. 237-246, ACM Press, New York, NY, USA, 1997.
72. Costa, A.T., Endler, M. and Cerqueira, R. Evaluation of Three Approaches for Corba Firewall/Nat Traversal. International Symposium on Distributed Objects and Applications, DOA'05, Agia Napa, Cyprus, 2005.
73. Costa, C.A.d., Yamin, A.C. and Geyer, C.F.R. Toward a General Software Infrastructure for Ubiquitous Computing. *IEEE Pervasive Computing*, 7 (1), 64-73, 2008.
74. Coulouris, G., Dollimore, J. and Kindberg, T. *Distributed Systems: Concepts and Design.* Addison-Wesley Longman Publishing Company Inc., 1988.
75. Cox, S. Dcmi Point Encoding Scheme: A Point Location in Space, and Methods for Encoding This in a Text String. Retrieved from http://dublincore.org/documents/dcmi-point/
76. Crockford, D. RFC: 4627 - the Application/Json Media Type for Javascript Object Notation. Retrieved from http://tools.ietf.org/html/rfc4627
77. Croft, W.B. and Harper, D.J. Using Probabilistic Models of Document Retrieval without Relevance Information. *Readings in Information Retrieval*, pp. 339-344, Morgan Kaufmann Publishers Inc., 1997.
78. Davies, N., Cheverst, K., Mitchell, K. and Efrat, A. Using and Determining Location in a Context-Sensitive Tour Guide. *Computer*, 34 (8), 35-41, 2001.
79. Davies, N. and Gellersen, H.-W. Beyond Prototypes: Challenges in Deploying Ubiquitous Systems. *Pervasive Computing*, 1 (1), 26 - 35, 2002.
80. Dawson, F. and Howes, T. RFC: 2426 - Vcard Mime Directory Profile. Retrieved from http://tools.ietf.org/html/rfc2426
81. Dawson, F. and Stenerson, D. Internet Calendaring and Scheduling Core Object Specification (RFC: 2445). Retrieved from http://tools.ietf.org/html/rfc2445
82. DCMI. Dublin Core Metadata Initiative. Retrieved from http://dublincore.org/
83. DCMI. Using Dublin Core. Retrieved from http://dublincore.org/documents/usageguide/
84. Dean, J. and Ghemawat, S. Mapreduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51 (1), 107-113, 2008.
85. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K. and Harshman, R. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, 41 (6), 391-407, 1990.
86. Deshpande, M. and Karypis, G. Item-Based Top-N Recommendation Algorithms. *ACM Transactions on Information Systems, TOIS*, 22 (1), 143-177, 2004.

87. Devlin, K. Situation Theory and Situation Semantics. *Handbook of the History of Logic (Logic and the Modalities in the Twentieth Century)*, 7, 601-664, 2006.
88. Dey, A.K. Context-Aware Computing: The Cyberdesk Project. The AAAI 98 Spring Symposium on Intelligent Environments, Menlo Park, CA, USA, AAAI Press, 1998.
89. Dey, A.K. and Abowd, G.D. Towards a Better Understanding of Context and Context-Awareness. The Workshop on The What, Who, Where, When, and How of Context-Awareness, as part of the 2000 Conference on Human Factors in Computing Systems, The Hague, The Netherlands, 2000.
90. Ding, J., Gravano, L. and Shivakumar, N. Computing Geographical Scopes of Web Resources. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pp. 545-556, Morgan Kaufmann Publishers Inc., 2000.
91. Ding, Z., Eren, M., Jia, L., Giles, C.L. and Hongyuan, Z. Probabilistic Models for Discovering E-Communities. In *Proceedings of the 15th international conference on World Wide Web*, pp. 173-182, ACM Press, New York, NY, USA, 2006.
92. Domshlak, C. and Roitman, H. Rank Aggregation for Automatic Schema Matching. *IEEE Transactions on Knowledge and Data Engineering*, 19 (4), 538-553, 2007.
93. Doulkeridis, C., Valavanis, E. and Vazirgiannis, M. Towards a Context-Aware Service Directory. In *Proceedings of the 4th VLDB Workshop on Technologies on E-Services, TES'03*, pp. 54-65, North-Holland, 2003.
94. Dubinko, M., Kumar, R., Magnani, J., Novak, J., Raghavan, P. and Tomkins, A. Visualizing Tags over Time. In *Proceedings of the 15th International Conference on World Wide Web*, pp. 193-202, ACM Press, New York, NY, USA, 2006.
95. Edwards, W.K. and Grinter, R.E. At Home with Ubiquitous Computing: Seven Challenges. In *Proceedings of the 3rd international conference on Ubiquitous Computing*, pp. 256-272, Springer-Verlag, 2001.
96. Edwards, W.K., Newman, M.W., Sedivy, J., Smith, T. and Izadi, S. Challenge: Recombinant Computing and the Speakeasy Approach. In *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking*, pp. 279-286, ACM Press, New York, NY, USA, 2002.
97. Endres, C., Butz, A. and MacWilliams, A. A Survey of Software Infrastructures and Frameworks for Ubiquitous Computing. *Mobile Information Systems*, 1 (1), 41-80, 2005.
98. EPCglobal Inc. Epcglobal Tag Data Standards Version 1.4. 2008.
99. Evgeniy, G., Susan, D. and Eric, H. Newsjunkie: Providing Personalized Newsfeeds Via Analysis of Information Novelty. In *Proceedings of the 13th International Conference on World Wide Web*, pp. 482-490, ACM Press, New York, NY, USA, 2004.
100. Fagin, R. Combining Fuzzy Information from Multiple Systems. *Journal of Computer and System Sciences*, 58 (1), 83-99, 1999.
101. Fagin, R. Combining Fuzzy Information: An Overview. *ACM SIGMOD Record*, 31 (2), 109-118, 2002.
102. Fagin, R., Lotem, A. and Naor, M. Optimal Aggregation Algorithms for Middleware. In *Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pp. 102-113, ACM Press, New York, NY, USA, 2001.
103. Fickas, S., Kortuem, G. and Segall, Z. Software Organization for Dynamic and Adaptable Wearable Systems. In *Proceedings of the 1st IEEE International Symposium on Wearable Computers*, p. 56, IEEE Computer Society, 1997.
104. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T. *Hypertext Transfer Protocol -- Http/1.1*. The Internet Engineering Task Force, 1999.
105. Fielding, R.T. *Architectural Styles and the Design of Network-Based Software Architectures*. PhD Thesis, University of California, Irvine, 2000.
106. Fielding, R.T. and Taylor, R.N. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology*, 2 (2), 115–150, 2002.
107. Filho, R.F.S., Traina, A., Jr., C.T. and Faloutsos, C. Similarity Search without Tears: The Omni-Family of All-Purpose Access Methods. In *Proceedings of the 17th International Conference on Data Engineering*, p. 623, IEEE Computer Society, 2001.

108. Flinn, J. and Satyanarayanan, M. Energy-Aware Adaptation for Mobile Applications. *ACM SIGOPS Operating Systems Review*, 33 (5), 48-63, 1999.
109. Forman, G.H. and Zahorjan, J. The Challenges of Mobile Computing. *Computer*, 27 (4), 38-47, 1994.
110. Fowler, M. and Scott, K. *Uml Distilled: A Brief Guide to the Standard Object Modeling Language (2nd Edition)*. Addison-Wesley Longman Publishing Company Inc., 2000.
111. Fox, A., Johanson, B., Hanrahan, P. and Winograd, T. Integrating Information Appliances into an Interactive Workspace. *IEEE Computer Graphics and Applications*, 20 (3), 54-65, 2000.
112. Frei, A. and Alonso, G. A Dynamic Lightweight Platform for Ad-Hoc Infrastructures. In *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, pp. 373-382, IEEE Computer Society, 2005.
113. Furnas, G.W., Landauer, T.K., Gomez, L.M. and Dumais, S.T. The Vocabulary Problem in Human-System Communication. *Communications of the ACM*, 30 (11), 964-971, 1987.
114. Gamma, E., Helm, R., Johnson, R. and Vlissides, J.M. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.
115. Garlan, D., Allen, R. and Ockerbloom, J. Architectural Mismatch or Why It's Hard to Build Systems out of Existing Parts. In *Proceedings of the 17th International Conference on Software Engineering*, pp. 179-185, ACM Press, New York, NY, USA, 1995.
116. Garlan, D., Siewiorek, D., Smailagic, A. and Steenkiste, P. Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, 1 (2), 22-31, 2002.
117. Geihs, K. Middleware Challenges Ahead. *Computer*, 34 (6), 24-31, 2001.
118. Ghemawat, S., Gobioff, H. and Leung, S.-T. The Google File System. *ACM SIGOPS Operating Systems Review*, 37 (5), 29-43, 2003.
119. Ghidini, C. and Giunchiglia, F. Local Models Semantics, or Contextual Reasoning = Locality + Compatibility. *Artificial Intelligence*, 127 (2), 221-259, 2001.
120. Ghosh, S. *Distributed Systems: An Algorithmic Approach*. CRC Press, 2006.
121. Glassey, R., Stevenson, G., Richmond, M., Nixon, P., Terzis, S., Wang, F. and Ferguson, I. Towards a Middleware for Generalised Context Management. The First International Workshop on Middleware for Pervasive and Ad Hoc Computing at Middleware 2003, Rio de Janeiro, Brazil, 2003.
122. Gokhale, A. and Schmidt, D.C. Evaluating the Performance of Demultiplexing Strategies for Real-Time Corba. IEEE Global Telecommunications Conference, GLOBECOM '97, Phoenix, Arizona, USA, IEEE Computer Society, 1997.
123. Golder, S. and Huberman, B.A. Usage Patterns of Collaborative Tagging Systems. *Journal of Information Science*, 32 (2), 198-208, 2006.
124. Goldstein, J. and Ramakrishnan, R. Contrast Plots and P-Sphere Trees: Space Vs. Time in Nearest Neighbour Searches. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pp. 429-440, Morgan Kaufmann Publishers Inc., 2000.
125. Goldstone, R.L. and Son, J.Y. Respects for Similarity. *Psychological Review*, 100 (2), 254-278, 1993.
126. Grace, P., Blair, G.S. and Samuel, S. A Reflective Framework for Discovery and Interaction in Heterogeneous Mobile Environments. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9 (1), 2-14, 2005.
127. Gray, P.D. and Salber, D. Modelling and Using Sensed Context Information in the Design of Interactive Applications. In *Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction*, pp. 317-336, Springer-Verlag, 2001.
128. Griffin, E. *Foundations of Popfly: Rapid Mashup Development*. Apress Inc., 2008.
129. Griswold, W.G., Boyer, R., Brown, S.W. and Truong, T.M. A Component Architecture for an Extensible, Highly Integrated Context-Aware Computing Infrastructure. In *Proceedings of the 25th International Conference on Software Engineering*, pp. 363-372, IEEE Computer Society, 2003.

130. Güntzer, U., Balke, W.-T. and Kießling, W. Optimizing Multi-Feature Queries for Image Databases. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pp. 419-428, Morgan Kaufmann Publishers Inc., 2000.
131. Güntzer, U., Balke, W.-T. and Kießling, W. Towards Efficient Multi-Feature Queries in Heterogeneous Environments. In *Proceedings of the International Conference on Information Technology: Coding and Computing*, p. 622, IEEE Computer Society, 2001.
132. Gunther, B.K. Multithreading with Distributed Functional Units. *IEEE Transactions on Computers*, 46 (4), 399-411, 1997.
133. Guo, S. and Yang, O.W.W. Energy-Aware Multicasting in Wireless Ad Hoc Networks: A Survey and Discussion. *Computer Communications*, 30 (9), 2129-2148, 2007.
134. Guttman, A. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pp. 47-57, ACM Press, New York, NY, USA, 1984.
135. Guttman, E., Perkins, C., Veizades, J. and Day, M. RFC: 2608 - Service Location Protocol, Version 2. Retrieved from http://tools.ietf.org/html/rfc2608
136. Guttman, E., Perkins, C., Veizades, J. and Day, M. Service Location Protocol, Version 2 (RFC: 2608). Retrieved from http://www.ietf.org/rfc/rfc2608.txt
137. Hadley, M.J. Web Application Description Language (WADL). Retrieved from https://wadl.dev.java.net/wadl20061109.pdf
138. Handschuh, S. and Staab, S. Authoring and Annotation of Web Pages in Cream. In *Proceedings of the 11th International Conference on World Wide Web*, pp. 462-473, ACM Press, New York, NY, USA, 2002.
139. Hansen, F.A., Bouvin, N.O., Christensen, B.G., Grønbæk, K., Pedersen, T.B. and Gagach, J. Integrating the Web and the World: Contextual Trails on the Move. In *Proceedings of the Fifteenth ACM Conference on Hypertext and Hypermedia*, pp. 98-107, ACM Press, New York, NY, USA, 2004.
140. Harter, A., Hopper, A., Steggles, P., Ward, A. and Webster, P. The Anatomy of a Context-Aware Application. In *ACM Mobile Computing and Networking*, pp. 187-197, Kluwer Academic Publishers, Hingham, MA, USA, 1999.
141. Heidemann, J., Pradkin, Y., Govindan, R., Papadopoulos, C., Bartlett, G. and Bannister, J. Census and Survey of the Visible Internet. In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement*, pp. 169-182, ACM Press, New York, NY, USA, 2008.
142. Held, A., Buchholz, S. and Schill, A. Modeling of Context Information for Pervasive Computing Applications. 6th World Multiconference on Systemics, Cybernetics and Informatics, SCI '02, Orlando, Florida, USA, 2002.
143. Hellerstein, J.M., Naughton, J.F. and Pfeffer, A. Generalized Search Trees for Database Systems. In *Proceedings of the 21th International Conference on Very Large Data Bases*, pp. 562-573, Morgan Kaufmann Publishers Inc., 1995.
144. Henning, M. The Rise and Fall of Corba. *Communications of the ACM*, 51 (8), 52-57, 2008.
145. Henricksen, K. *A Framework for Context-Aware Pervasive Computing Applications.* PhD Thesis, School of Information Technology and Electrical Engineering, The University of Queensland, 2004.
146. Henricksen, K. and Indulska, J. Modelling and Using Imperfect Context Information. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, p. 33, IEEE Computer Society, 2004.
147. Henricksen, K., Indulska, J., McFadden, T. and Balasubramaniam, S. Middleware for Distributed Context-Aware Systems. International Symposium on Distributed Objects and Applications, DOA Agia Napa, Cyprus, 2005.
148. Heumer, G., Carlson, D., Kaligiri, S.H., Maheshwari, S., Ul, H.-W., Jung, B. and Schrader, A. Paranoia Syndrome – a Location Based Pervasive Multiplayer Game Using PDAs, RFID, and Tangible Objects. Third International Workshop on Pervasive Gaming Applications, PerGames 2006, Dublin, Ireland, 2006.
149. Hightower, J. and Borriello, G. A Survey and Taxonomy of Location Systems for Ubiquitous Computing. *Computer*, 34 (8), 57-66, 2001.

150. Hightower, J., Consolvo, S., LaMarca, A., Smith, I. and Hughes, J. Learning and Recognizing the Places We Go. 7th International Conference, UbiComp 2005, Tokyo, Japan, Springer Berlin/Heidelberg, 2005.
151. Hightower, J., LaMarca, A. and Smith, I.E. Practical Lessons from Placelab. *IEEE Pervasive Computing*, 5 (3), 32-39, 2006.
152. Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G., Altmann, J. and Retschitzegger, W. Context-Awareness on Mobile Devices - the Hydrogen Approach. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences, HICSS'03*, p. 292.291, IEEE Computer Society, 2003.
153. Hunter, J., Khan, I. and Gerber, A. Harvana: Harvesting Community Tags to Enrich Collection Metadata. In *Proceedings of the 8th ACM/IEEE-CS Joint Conference on Digital Libraries*, pp. 147-156, ACM Press, New York, NY, USA, 2008.
154. IBM Corporation. Mqseries Version 5.1 Administration and Programming Examples, Ibm Redbooks. 1999.
155. IBM Corporation. Websphere Service Registry and Repository. Retrieved from http://www-01.ibm.com/software/integration/wsrr/
156. IEEE Computer Society. IEEE Recommended Practice for Software Requirements Specifications. Retrieved from http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?isnumber=15571&arnumber=720574&count=1&index=0
157. Indulska, J. and Sutton, P. Location Management in Pervasive Systems. In *Proceedings of the Australasian Information Security Workshop Conference on ACSW Frontiers 2003 - Volume 21*, pp. 143-151, Australian Computer Society Inc., 2003.
158. Indyk, P. Dimensionality Reduction Techniques for Proximity Problems. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 371-378, Society for Industrial and Applied Mathematics, 2000.
159. Indyk, P. *High-Dimensional Computational Geometry*. PhD Thesis, Stanford University, 2001.
160. Indyk, P. Nearest Neighbors in High-Dimensional Spaces. *Handbook of Discrete and Computational Geometry (2nd Edition)*, Goodman, J.E. and O'Rourke, J. (Eds.), Chapman & Hall, 2004.
161. Indyk, P. and Motwani, R. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pp. 604-613, ACM Press, New York, NY, USA, 1998.
162. International Organization for Standardization (ISO). ISO/IEC Standard on UPNP Device Architecture Makes Networking Simple and Easy. Retrieved from http://www.iso.org/iso/pressrelease.htm?refid=Ref1185
163. International Organization for Standardization (ISO). Mpeg Video Technologies -- Part 1: Accuracy Requirements for Implementation of Integer-Output 8x8 Inverse Discrete Cosine Transform (ISO/IEC 23002-1:2006). Retrieved from http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=42030
164. Jagadish, H.V., Ooi, B.C., Tan, K.-L., Yu, C. and Zhang, R. Idistance: An Adaptive B+-Tree Based Indexing Method for Nearest Neighbor Search. *ACM Transactions on Database Systems, TODS*, 30 (2), 364-397, 2005.
165. Järvelin, K. and Kekäläinen, J. Ir Evaluation Methods for Retrieving Highly Relevant Documents. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 41-48, ACM Press, New York, NY, USA, 2000.
166. Java Community Process (JCP). General JCP Questions (FAQ). Retrieved from http://www.jcp.org/en/introduction/faq
167. Java Community Process (JCP). JCP 2: Process Document. Retrieved from http://jcp.org/en/procedures/jcp2
168. Jernigan, C., Bayley, C., Lin, J. and Wright, C. The Locale Project Homepage. Retrieved from http://code.google.com/android/adc_gallery/app.html?id=25

169. Jeronimo, M. and Weast, J. *UPNP Design by Example: A Software Developer's Guide to Universal Plug and Play*. Intel Press, 2003.
170. Jhingran, A. Enterprise Information Mashups: Integrating Information, Simply. In *Proceedings of the 32nd International Conference on Very Large Databases*, pp. 3-4, VLDB Endowment, 2006.
171. Jiang, X. and Landay, J.A. Modeling Privacy Control in Context-Aware Systems. *IEEE Pervasive Computing*, 1 (3), 59-63, 2002.
172. Jolliffe, I.T. *Principal Component Analysis*. Springer-Verlag, 2002.
173. Judd, G. and Steenkiste, P. Providing Contextual Information to Pervasive Computing Applications. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, p. 133, IEEE Computer Society, 2003.
174. Keidl, M. and Kemper, A. A Framework for Context-Aware Adaptable Web Services. In *Proceedings of the 9th International Conference on Extending Database Technology*, pp. 635-636, Springer, 2004.
175. Keidl, M. and Kemper, A. Towards Context-Aware Adaptable Web Services. In *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters*, pp. 55-65, ACM Press, New York, NY, USA, 2004.
176. Kendall, S.C., Waldo, J., Wollrath, A. and Wyant, G. A Note on Distributed Computing. Sun Microsystems Inc., 1994.
177. Kim, C. and Kim, J. A Recommendation Algorithm Using Multi-Level Association Rules. In *Proceedings of the 2003 IEEE/WIC International Conference on Web Intelligence*, p. 524, IEEE Computer Society, 2003.
178. Kimball, A., Michels-Slettvet, S. and Bisciglia, C. Cluster Computing for Web-Scale Data Processing. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, pp. 116-120, ACM Press, New York, NY, USA, 2008.
179. Kindberg, T., Barton, J., Morgan, J., Becker, G., Caswell, D., Debaty, P., Gopal, G., Frid, M., Krishnan, V., Morris, H., Schettino, J., Serra, B. and Spasojevic, M. People, Places, Things: Web Presence for the Real World. *Mobile Networks and Applications*, 7 (5), 365-376, 2002.
180. Kindberg, T. and Fox, A. System Software for Ubiquitous Computing. *IEEE Pervasive Computing*, 1 (1), 70 - 81, 2002.
181. King, J. and Rothenberg, M. Place Lab: User Gathered Data and Open Source Data Repositories. Retrieved from http://people.ischool.berkeley.edu/~jenking/placelab.pdf
182. Kistler, J.J. and Satyanarayanan, M. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems, TOCS*, 10 (1), 3-25, 1992.
183. Kleinberg, R. Geographic Routing Using Hyperbolic Space. In *26th IEEE International Conference on Computer Communications, INFOCOM 2007*, pp. 1902-1909, IEEE Computer Society, 2007.
184. Korpipää, P. and Mäntyjärvi, J. Ontology for Mobile Device Sensor-Based Context Awareness. 4th International and Interdisciplinary Conference, CONTEXT 2003, Stanford, CA, USA, 2003.
185. Korpipaa, P., Mantyjarvi, J., Kela, J., Keranen, H. and Malm, E.-J. Managing Context Information in Mobile Devices. *IEEE Pervasive Computing*, 2 (3), 42-51, 2003.
186. Krafzig, D., Banke, K. and Slama, D. *Enterprise Soa : Service-Oriented Architecture Best Practices*. Prentice Hall, 2004.
187. Krumhansl, C.L. Concerning the Applicability of Geometric Models to Similarity Data: The Interrelationship between Similarity and Spatial Density. *Psychological Review*, 85, 450-463, 1978.
188. Krumm, J. and Hinckley, K. The Nearme Wireless Proximity Server. In *The Sixth International Conference on Ubiquitous Computing, UbiComp 2004*, pp. 283-300, Springer Berlin/Heidelberg, 2004.
189. Kunder, M.d. *Geschatte Grootte Van Het Geïndexeerde World Wide Web*. Master's Thesis, Tilburg University, 2007.
190. Kushilevitz, E., Ostrovsky, R. and Rabani, Y. Efficient Search for Approximate Nearest Neighbor in High Dimensional Spaces. *SIAM Journal on Computing*, 30 (2), 457-474, 2000.

191. Lassabe, F., Canalda, P., Chatonnay, P. and Charlet, D. Refining Wifi Indoor Positioning Renders Pertinent Deploying Location-Based Multimedia Guide. In *Proceedings of the 20th International Conference on Advanced Information Networking and Applications - Volume 2, AINA'06*, pp. 126-132, IEEE Computer Society, 2006.
192. Lathem, J., Gomadam, K. and Sheth, A.P. Sa-REST and (S)Mashups: Adding Semantics to RESTful Services. In *Proceedings of the International Conference on Semantic Computing*, pp. 469-476, IEEE Computer Society, 2007.
193. Lebeck, A.R., Fan, X., Zeng, H. and Ellis, C. Power Aware Page Allocation. *ACM SIGOPS Operating Systems Review*, 34 (5), 105-116, 2000.
194. Lee, C. and Helal, S. A Multi-Tier Ubiquitous Service Discovery Protocol for Mobile Clients. 2003 International Symposium on Performance Evaluation of Computer and Telecommunication Systems, SPECTS 2003, Montréal, Canada, 2003.
195. Lee, D.L. and Chen, Q. A Model-Based Wifi Localization Method. In *Proceedings of the 2nd International Conference on Scalable Information Systems*, pp. 1-7, Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, 2007.
196. Lei, H., Sow, D.M., J. S. Davis II, Banavar, G. and Ebling, M.R. The Design and Applications of a Context Service. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6 (4), 45-55, 2002.
197. Lemire, D. and Maclachlan, A. Slope One Predictors for Online Rating-Based Collaborative Filtering. SIAM Data Mining, SDM'05, Newport Beach, California, 2005.
198. Lemire, D. and Mcgrath, S. Implementing a Rating-Based Item-to-Item. Recommender System in Php/Sql, Technical Report D-01. Université Du Québec, 2005.
199. Li, J. and Wang, J.Z. Real-Time Computerized Annotation of Pictures. In *Proceedings of the 14th Annual ACM International Conference on Multimedia*, pp. 911-920, ACM Press, New York, NY, USA, 2006.
200. Linden, G., Smith, B. and York, J. Amazon.Com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing*, 7 (1), 76-80, 2003.
201. Ling, W.K., Erber-urch, K., Balke, W.-T., Birke, T. and Wagner, M. The Heron Project: Multimedia Database Support for History and Human Sciences. In *Proceedings of the GI Annual Conference, INFORMATIK'98*, pp. 309–318, Springer-Verlag, 1998.
202. Lymberopoulos, D., Lindsey, Q. and Savvides, A. An Empirical Characterization of Radio Signal Strength Variability in 3-D IEEE 802.15.4 Networks Using Monopole Antennas. In *Proceedings of the European Workshop on Wireless Sensor Networks, EWSN 2006*, pp. 326-341, Springer-Verlag Berlin/Heidelberg, 2006.
203. Mackert, L.F. and Lohman, G.M. Index Scans Using a Finite Lru Buffer: A Validated I/O Model. *ACM Transactions on Database Systems, TODS*, 14 (3), 401-424, 1989.
204. Maes, P. Concepts and Experiments in Computational Reflection. *ACM SIGPLAN Notices*, 22 (12), 147-155, 1987.
205. Manning, C.D., Raghavan, P. and Schütze, H. *Introduction to Information Retrieval.* Cambridge University Press, 2008.
206. Maximilien, E.M., Ranabahu, A. and Gomadam, K. An Online Platform for Web Apis and Service Mashups. *IEEE Internet Computing*, 12 (5), 32-43, 2008.
207. McDonald, D.W. Ubiquitous Recommendation Systems. *Computer*, 36 (10), 111, 2003.
208. McKinley, P.K., Sadjadi, S.M., Kasten, E.P. and Cheng, B.H.C. Composing Adaptive Software. *Computer*, 37 (7), 56-64, 2004.
209. McKinley, P.K., Sadjadi, S.M., Kasten, E.P. and Cheng, B.H.C. A Taxonomy of Compositional Adaptation (Tech. Report Msu-Cse-04-17). Department of Computer Science and Engineering, Michigan State University, 2004.
210. Meiser, S. Point Location in Arrangements of Hyperplanes. *Information and Computation*, 106 (2), 286-303, 1993.
211. Middleton, S.E., Shadbolt, N.R. and Roure, D.C.D. Ontological User Profiling in Recommender Systems. *ACM Transactions on Information Systems, TOIS*, 22 (1), 54-88, 2004.

212. Milner, M. The Netstumbler File Format Specification. Retrieved from http://www.stumbler.net/ns1files.html
213. Mitra, M., Singhal, A. and Buckley, C. Improving Automatic Query Expansion. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 206-214, ACM Press, New York, NY, USA, 1998.
214. Mrissa, M., Ghedira, C., Benslimane, D. and Maamar, Z. Context and Semantic Composition of Web Services 17th International Conference, DEXA 2006, Krakow, Poland, 2006.
215. Mummert, L.B., Ebling, M.R. and Satyanarayanan, M. Exploiting Weak Connectivity for Mobile File Access. *ACM SIGOPS Operating Systems Review*, 29 (5), 143-155, 1995.
216. Nakamura, E.F., Loureiro, A.A.F. and Frery, A.C. Information Fusion for Wireless Sensor Networks: Methods, Models, and Classifications. *ACM Computing Surveys, CSUR*, 39 (3), 9, 2007.
217. Nam, B. and Sussman, A. A Comparative Study of Spatial Indexing Techniques for Multidimensional Scientific Datasets. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, p. 171, IEEE Computer Society, 2004.
218. Nam, B. and Sussman, A. Improving Access to Multi-Dimensional Self-Describing Scientific Datasets. In *Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, p. 172, IEEE Computer Society, 2003.
219. National Institute of Standards and Technology (NIST). Secure Hash Standard, Publication 180-2. 2002.
220. National Marine Electronics Association. The Nmea 0183 Standard. Retrieved from http://www.nmea.org/pub/0183/
221. Nepal, S. and Ramakrishna, M.V. Query Processing Issues in Image (Multimedia) Databases. In *Proceedings of the 15th International Conference on Data Engineering*, p. 22, IEEE Computer Society, 1999.
222. Netcraft Ltd. November 2008 Web Server Survey. Retrieved from http://news.netcraft.com/archives/2008/11/19/november_2008_web_server_survey.html
223. Network Working Group. RFC: 1958 - Architectural Principles of the Internet. Retrieved from ftp://ftp.isi.edu/in-notes/rfc1958.txt
224. Network Working Group. RFC: 5023 - the Atom Publishing Protocol. Retrieved from http://tools.ietf.org/html/rfc5023
225. Newman, M.W., Sedivy, J.Z., Neuwirth, C.M., Edwards, W.K., Hong, J.I., Izadi, S., Marcelo, K. and Smith, T.F. Designing for Serendipity: Supporting End-User Configuration of Ubiquitous Computing Environments. In *Proceedings of the 4th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, pp. 147-156, ACM Press, New York, NY, USA, 2002.
226. Nichols, D.M. Implicit Rating and Filtering. In *Proceedings of the Fifth DELOS Workshop on Filtering and Collaborative Filtering*, pp. 31-36, The European Research Consortium for Informatics and Mathematics, 1997.
227. Nielsen, J. One Billion Internet Users. Retrieved from http://www.useit.com/alertbox/internet_growth.html
228. Niemelä, E. and Latvakoski, J. Survey of Requirements and Solutions for Ubiquitous Software. In *Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia* pp. 71-78, ACM Press, New York, NY, USA, 2004.
229. Nievergelt, J., Hinterberger, H. and Sevcik, K.C. The Grid File: An Adaptable, Symmetric Multikey File Structure. *ACM Transactions on Database Systems, TODS*, 9 (1), 38-71, 1984.
230. Noy, N.F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R.W. and Musen, M.A. Creating Semantic Web Contents with Protégé. *IEEE Intelligent Systems*, 16 (2), 60-71, 2001.
231. OASIS. UDDI Version 3.0.2. Retrieved from http://www.uddi.org/pubs/uddi_v3.htm
232. OASIS/ebXML Registry Technical Committee. Oasis EBXML Registry Specification V3.01. Retrieved from http://docs.oasis-open.org/regrep/v3.0/regrep-3.0-os.zip
233. Object Management Group. Corba Component Model, V4.0. OMG Document Formal/2006-04-01, 2006.

234. Object Management Group. Specification for Deployment and Configuration of Component-Based Distributed Applications. OMG Document ptc/03-07-08, 2003.
235. Object Management Group Inc. Common Object Request Broker Architecture: Core Specification Version 3.0.3. 2004.
236. Open Geospatial Consortium. OpenGIS Abstract Specifications. Retrieved from http://www.opengeospatial.org/standards/as
237. Open Geospatial Consortium. OpenGIS Geography Markup Language (GML) Encoding Standard. Retrieved from http://www.opengeospatial.org/standards/gml
238. OpenUDDI. The OpenUDDI UDDI V3 Compliant Server and Client Library. Retrieved from http://openuddi.sourceforge.net/
239. Oracle Corporation. Bea Messageq Product Overview. Retrieved from http://kr.bea.com/products/more/messageq/overview.shtml
240. Organization for the Advancement of Structured Information Standards. Oasis Web Services Security (Wss) Tc. Retrieved from http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss
241. Organization for the Advancement of Structured Information Standards. Oasis Web Services Transaction (Ws-Tx) Tc. Retrieved from http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-tx
242. Orr, R.J. and Abowd, G.D. The Smart Floor: A Mechanism for Natural User Identification and Tracking. In *Proceesings of CHI '00 Extended Abstracts on Human Factors in Computing Systems*, pp. 275-276, ACM Press, New York, NY, USA, 2000.
243. Ort, E., Brydon, S. and Basler, M. Mashup Styles, Part 2: Client-Side Mashups. Retrieved from http://java.sun.com/developer/technicalArticles/J2EE/mashup_2/
244. OWL-S Coalition. Owl-S 1.1 Release. Retrieved from http://www.daml.org/services/owl-s/1.1/
245. Öztürk, P. and Aamodt, A. A Context Model for Knowledge-Intensive Case-Based Reasoning. *International Journal of Human-Computer Studies*, 48 (3), 331-355, 1998.
246. Page, L., Brin, S., Motwani, R. and Winograd, T. The Pagerank Citation Ranking: Bringing Order to the Web. Retrieved from http://dbpubs.stanford.edu:8090/pub/1999-66
247. Palo Alto Research Center. Obje Interoperability Framework (Whitepaper). 2003.
248. Panwar, S.S., Mao, S., Ryoo, J.-d. and Li, Y. *Tcp/Ip Essentials: A Lab-Based Approach.* Cambridge University Press, 2004.
249. Parnas, D.L. On the Criteria to Be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15 (12), 1053-1058, 1972.
250. Pascoe, J. Adding Generic Contextual Capabilities to Wearable Computers. In *Proceedings of the 2nd IEEE International Symposium on Wearable Computers*, p. 92, IEEE Computer Society, 1998.
251. Pazzani, M.J. A Framework for Collaborative, Content-Based and Demographic Filtering. *Artificial Intelligence Review*, 13 (5-6), 393-408, 1999.
252. Perkins, C.E., Alpert, S.R. and Woolf, B. *Mobile Ip; Design Principles and Practices.* Addison-Wesley Longman Publishing Company Inc., 1997.
253. Pinkerton, B. *Webcrawler: Finding What People Want*. PhD Thesis, University of Washington, 2000.
254. Pokraev, S., Koolwaaij, J. and Wibbels, M. Extending UDDI with Context-Aware Features Based on Semantic Service Descriptions. International Conference on Web Services, ICWS'03, Las Vegas, USA, 2003.
255. Ponnekanti, S.R., Johanson, B., Kiciman, E. and Fox, A. Portability, Extensibility and Robustness in Iros. In *Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications*, p. 11, IEEE Computer Society, 2003.
256. Porter, M.F. An Algorithm for Suffix Stripping. *Readings in Information Retrieval*, pp. 313-316, Morgan Kaufmann Publishers Inc., 1997.
257. Prescod, P. Roots of the REST/SOAP Debate. Extreme Markup Languages 2002, Montréal, Québec, 2002.

258. Priyantha, N.B., Chakraborty, A. and Balakrishnan, H. The Cricket Location-Support System. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking, Mobicon2000*, pp. 32-43, ACM Press, New York, NY, USA, 2000.
259. Raento, M., Oulasvirta, A., Petit, R. and Toivonen, H. Contextphone: A Prototyping Platform for Context-Aware Mobile Applications. *IEEE Pervasive Computing*, 4 (2), 51-59, 2005.
260. Ranganathan, A., Al-Muhtadi, J. and Campbell, R.H. Reasoning About Uncertain Contexts in Pervasive Computing Environments. *IEEE Pervasive Computing*, 3 (2), 62-70, 2004.
261. Raverdy, P.-G., Chibout, R., Chapelle, A.d.L. and Issarny, V. A Multi-Protocol Approach to Service Discovery and Access in Pervasive Environments. The 3rd Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services, San Jose, CA, USA, 2006.
262. Ray, R., Kulchenko, P. and Guelich, S. *Programming Web Services with Perl.* O'Reilly & Associates Inc., 2002.
263. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P. and Riedl, J. Grouplens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, pp. 175-186, ACM Press, New York, NY, USA, 1994.
264. Resnick, P. and Varian, H.R. Recommender Systems. *Communications of the ACM*, 40 (3), 56-58, 1997.
265. Riaz, M., Kiani, S.L., Lee, S., Han, S.-M. and Lee, Y.-K. Service Delivery in Context Aware Environments: Lookup and Access Control Issues. In *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 455-458, IEEE Computer Society, 2005.
266. Richardson, L. and Ruby, S. *RESTful Web Services.* O'Reilly Media Inc., 2007.
267. Richardson, T., Bennett, F., Mapp, G. and Hopper, A. Teleporting in an X Window System Environment. *IEEE Personal Communications*, 1 (3), 6-15, 1994.
268. Riva, O. Contory: A Middleware for the Provisioning of Context Information on Smart Phones. In *Proceedings of the 3rd IEEE International Conference on Pervasive Services, ICPS'06*, pp. 47-56, IEEE Computer Society, 2006.
269. Riva, O. and Toivonen, S. A Hybrid Model of Context-Aware Service Provisioning Implemented on Smart Phones. In *Proceedings of the ACS/IEEE International Conference on Pervasive Services*, pp. 47-56, 2006.
270. Robinson, R., Henricksen, K. and Indulska, J. Xcml: A Runtime Representation for the Context Modelling Language. In *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, pp. 20-26, IEEE Computer Society, 2007.
271. Rocchio, J. Relevance Feedback in Information Retrieval. *The Smart Retrieval System*, Salton, G. (Ed.), pp. 313-323, Prentice-Hall, Englewood Cliffs, NJ, USA, 1971.
272. Rom, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R.H. and Nahrstedt, K. Gaia: A Middleware Platform for Active Spaces. *Mobile Computing and Communications Review, SIGMOBILE*, 6 (4), 65-67, 2002.
273. Rosenberry, W., Kenney, D. and Fisher, G. *Understanding Dce.* O'Reilly & Associates Inc., 1992.
274. Roth, M.T., Arya, M., Haas, L., Carey, M., Cody, W., Fagin, R., Schwarz, P., Thomas, J. and Wimmers, E. The Garlic Project. *ACM SIGMOD Record*, 25 (2), 557, 1996.
275. Salber, D. and Abowd, G.D. The Design and Use of a Generic Context Server. In *Proceedings of the Perceptual User Interfaces Workshop, PUI '98*, pp. 63-66, ACM Press, New York, NY, USA, 1998.
276. Salber, D., Dey, A.K. and Abowd, G.D. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, pp. 434-441, ACM Press, New York, NY, USA, 1999.
277. Salton, G. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley Longman Publishing Company Inc., 1989.

278. Salton, G. and Buckley, C. Improving Retrieval Performance by Relevance Feedback. *Readings in Information Retrieval*, pp. 355-364, Morgan Kaufmann Publishers Inc., 1997.
279. Saltzer, J.H., Reed, D.P. and Clark, D.D. End-to-End Arguments in System Design. *Transactions on Computer Systems*, 2 (3), 277 - 288, 1984.
280. Samulowitz, M., Michahelles, F. and Linnhoff-Popien, C. Capeus: An Architecture for Context-Aware Selection and Execution of Services. In *Proceedings of the IFIP TC6 / WG6.1 Third International Working Conference on New Developments in Distributed Applications and Interoperable Systems*, pp. 23-40, Kluwer, B.V., Deventer, The Netherlands, 2001.
281. Santini, S. and Jain, R. Similarity Measures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21 (9), 871-883, 1999.
282. Santoro, N. *Design and Analysis of Distributed Algorithms.* Wiley-Interscience, 2006.
283. SAP News Desk. Microsoft, Ibm, Sap to Discontinue UDDI Web Services Registry Effort. Retrieved from http://soa.sys-con.com/node/164624
284. Sarmenta, L.F.G. *Volunteer Computing*. PhD Thesis, Massachusetts Institute of Technology (MIT), 2001.
285. Sarwar, B.M. *Sparsity, Scalability, and Distribution in Recommender Systems*. PhD Thesis, University of Minnesota, 2001.
286. Sarwar, B.M., Konstan, J.A., Borchers, A., Herlocker, J., Miller, B. and Riedl, J. Using Filtering Agents to Improve Prediction Quality in the Grouplens Research Collaborative Filtering System. In *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*, pp. 345-354, ACM Press, New York, NY, USA, 1998.
287. Satyanarayanan, M. Fundamental Challenges in Mobile Computing. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, pp. 1-7, ACM Press, New York, NY, USA, 1996.
288. Satyanarayanan, M. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, 8 (4), 10-17, 2001.
289. SAWSDL Working Group. Semantic Annotations for WSDL (SAWSDL). Retrieved from http://www.w3.org/2002/ws/sawsdl/
290. Schilit, B.N., Adams, N. and Want, R. Context-Aware Computing Applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pp. 85-90, IEEE Computer Society, 1994.
291. Schilit, B.N., LaMarca, A., Borriello, G., Griswold, W.G., McDonald, D., Lazowska, E., Balachandran, A., Hong, J. and Iverson, V. Challenge: Ubiquitous Location-Aware Computing and The "Place Lab" Initiative. In *Proceedings of the 1st ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots*, pp. 29-35, ACM Press, New York, NY, USA, 2003.
292. Schilit, B.N. and Theimer, M.M. Disseminating Active Map Information to Mobile Hosts. *IEEE Network*, 8 (5), 22-32, 1994.
293. Schilit, W.N. *A System Architecture for Context-Aware Mobile Computing*. PhD Thesis, Columbia University, 1995.
294. Schmidt, A., Aidoo, K.A., Takaluoma, A., Tuomela, U., Van-Laerhoven, K. and Van-de-Velde, W. Advanced Interaction in Context. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, pp. 89-101, Springer-Verlag, 1999.
295. Sellis, T.K., Roussopoulos, N. and Faloutsos, C. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In *Proceedings of the 13th International Conference on Very Large Data Bases*, pp. 507-518, Morgan Kaufmann Publishers Inc., 1987.
296. Shakhnarovich, G., Darrell, T. and Indyk, P. Nearest-Neighbors Methods in Learning and Vision. Theory and Practice. *Pattern Analysis & Applications*, 11 (2), 221-222, 2008.
297. Shamos, M.I. and Hoey, D. Closest-Point Problems. In *Proceedings of the 16th IEEE Symposium on the Foundations of Computer Science*, pp. 151-162, IEEE Computer Society, 1975.
298. Shardanand, U. and Maes, P. Social Information Filtering: Algorithms for Automating "Word of Mouth". In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 210-217, ACM Press/Addison-Wesley Publishing Co., 1995.

299. Shepard, R.N. The Analysis of Proximities: Multidimensional Scaling with Unknown Distance Function Part I. *Psychometrika*, 27, 125-140, 1962.
300. Shepard, R.N. The Analysis of Proximities: Multidimensional Scaling with Unknown Distance Function Part Ii. *Psychometrika*, 27, 219–246, 1962.
301. Shepard, S. *RFID: Radio Frequency Identification.* McGraw-Hill Professional, 2005.
302. Sigurbjörnsson, B. and Van-Zwol, R. Flickr Tag Recommendation Based on Collective Knowledge. In *Proceeding of the 17th International Conference on World Wide Web*, pp. 327-336, ACM Press, New York, NY, USA, 2008.
303. Simcock, T., Hillenbrand, S.P. and Thomas, B.H. Developing a Location Based Tourist Guide Application. In *Proceedings of the Australasian Information Security Workshop Conference on ACSW Frontiers 2003 - Volume 21*, pp. 177-183, Australian Computer Society Inc., 2003.
304. Sinclair, J. and Cardew-Hall, M. The Folksonomy Tag Cloud: When Is It Useful? *Journal of Information Science*, 34 (1), 15-29, 2008.
305. Skopal, T., Hoksza, D. and Pokorný, J. Construction of Tree-Based Indexes for Level-Contiguous Buffering Support. In *Proceedings of the 12th International Conference on Database Systems for Advanced Applications, DASFAA 2007*, pp. 361-373, Springer Berlin/Heidelberg, 2007.
306. Smallberg, D. RFC: 832 - Who Talks Tcp? Retrieved from http://www.isi.edu/in-notes/rfc832.txt
307. Sollenborn, M. and Funk, P. Category-Based Filtering and User Stereotype Cases to Reduce the Latency Problem in Recommender Systems. In *Proceedings of the 6th European Conference on Advances in Case-Based Reasoning*, pp. 285-290, Springer-Verlag, 2002.
308. Song, D., Liu, W., He, Y. and He, K. Ontology Application in Software Component Registry to Achieve Semantic Interoperability. In *Proceedings of the International Conference on Information Technology: Coding and Computing, ITCC'05 - Volume 02*, pp. 181-186, IEEE Computer Society, 2005.
309. Soo, V.-W., Lee, C.-Y., Li, C.-C., Chen, S.L. and Chen, C.-c. Automated Semantic Annotation and Retrieval Based on Sharable Ontology and Case-Based Learning Techniques. In *Proceedings of the 3rd ACM/IEEE-CS Joint Conference on Digital Libraries*, pp. 61-72, IEEE Computer Society, 2003.
310. Sørensen, C.-F., Wu, M., Sivaharan, T., Blair, G.S., Okanda, P., Friday, A. and Duran-Limon, H. A Context-Aware Middleware for Applications in Mobile Ad Hoc Environments. In *Proceedings of the 2nd Workshop on Middleware for Pervasive and Ad-hoc Computing*, pp. 107-110, ACM Press, New York, NY, USA, 2004.
311. Spertus, E., Sahami, M. and Buyukkokten, O. Evaluating Similarity Measures: A Large-Scale Study in the Orkut Social Network. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pp. 678-684, ACM Press, New York, NY, USA, 2005.
312. Sputnik Inc. Rf Propagation Basics (White Paper). Retrieved Sputnik Inc, from: http://www.sputnik.com/docs/rf_propagation_basics.pdf
313. Stavrakas, Y. and Gergatsoulis, M. Multidimensional Semistructured Data: Representing Context-Dependent Information on the Web. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering*, pp. 183-199, Springer-Verlag, 2002.
314. Steinberg, D. and Cheshire, S. *Zero Configuration Networking: The Definitive Guide.* O'Reilly Media Inc., 2005.
315. Strang, T., Linnho-Popien, C. and Frank, K. Cool: A Context Ontology Language to Enable Contextual Interoperability. 4th International Conference on Distributed Applications and Interoperable Systems, DAIS2003, Paris, France, 2003.
316. Strang, T. and Linnhoff-Popien, C. A Context Modeling Survey. Workshop on Advanced Context Modelling, Reasoning and Management at the 6th International Conference on Ubiquitous Computing, Nottingham, England, 2004.

317. Stuntebeck, E.P., Patel, S.N., Robertson, T., Reynolds, M.S. and Abowd, G.D. Wideband Powerline Positioning for Indoor Localization. In *Proceedings of the 10th International Conference on Ubiquitous Computing*, pp. 94-103, ACM Press, New York, NY, USA, 2008.
318. Su, J., Scott, J., Hui, P., Crowcroft, J., Lara, E.D., Diot, C., Goel, A., Lim, M.H. and Upton, E. Haggle: Seamless Networking for Mobile Applications. The 9th International Conference on Ubiquitous Computing, UbiComp 2007, Innsbruck, Austria, 2007.
319. Sun Microsystems. Remote Method Invocation Home. Retrieved from http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp
320. Sun Microsystems Inc. RFC: 1057 - Remote Procedure Call Protocol Specification, Version 2. Retrieved from http://tools.ietf.org/html/rfc1057
321. Sun Microsystems Inc. J2me RMI Optional Package (RMI Op - JSR 66). Retrieved from http://java.sun.com/products/rmiop/
322. Swan, R. and Allan, J. Timemine: Visualizing Automatically Constructed Timelines. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, p. 393, ACM Press, New York, NY, USA, 2000.
323. Tanenbaum, A. *Computer Networks.* Prentice Hall Professional Technical Reference, 2002.
324. Technical Committee ISO/TC 154. ISO 8601 - Data Elements and Interchange Formats — Information Interchange — Representation of Dates and Times (ISO 8601:2004(E)). International Organization for Standardization (ISO), 2004.
325. Tel, G. *Introduction to Distributed Algorithms.* Cambridge University Press, 1994.
326. Terry, D.B., Theimer, M.M., Karin, P., Demers, A.J., Spreitzer, M.J. and Hauser, C.H. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pp. 172-182, ACM Press, New York, NY, USA, 1995.
327. The Internet Corporation for Assigned Names and Numbers (IANA). Mime Media Type Specifications (RFC 2045, RFC 2046, RFC 4288, RFC 4289, RFC 4855). Retrieved from http://www.iana.org/assignments/media-types/
328. The Internet Engineering Task Force (IETF). The Atom Syndication Format (Rfc: 4287). Retrieved from http://tools.ietf.org/html/rfc4287
329. Thomson, G., Nixon, P. and Terzis, S. Towards Adhoc Situation Determination. The 1st International Workshop on Advanced Context Modelling, Reasoning and Management at UbiComp2004, Nottingham, England, 2004.
330. Tilkov, S. A Brief Introduction to REST. Retrieved from http://www.infoq.com/articles/rest-introduction
331. Tversky, A. Features of Similarity. *Psychological Review*, 84, 327-352, 1977.
332. ur-Rehman, W., de-Lara, E. and Saroiu, S. Cilos: A Cdma Indoor Localization System. In *Proceedings of the 10th International Conference on Ubiquitous Computing*, pp. 104-113, ACM Press, New York, NY, USA, 2008.
333. Van-Laerhoven, K. and Cakmakci, O. What Shall We Teach Our Pants? In *Proceedings of the 4th IEEE International Symposium on Wearable Computers*, p. 77, IEEE Computer Society, 2000.
334. Vinoski, S. Demystifying RESTful Data Coupling. *IEEE Internet Computing*, 12 (2), 87-90, 2008.
335. Vinoski, S. REST Eye for the Soa Guy. *IEEE Internet Computing*, 11 (1), 82-84, 2007.
336. Vinoski, S. Serendipitous Reuse. *IEEE Internet Computing*, 12 (1), 84-87, 2008.
337. Vojnović, M. Tagbooster: A System for Ranking and Suggesting Tags. Retrieved from http://research.microsoft.com/~milanv/tagbooster.htm
338. Voorhees, E.M. Query Expansion Using Lexical-Semantic Relations. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 61-69, Springer-Verlag, 1994.
339. Vozalis, E. and Margaritis, K.G. Analysis of Recommender Systems' Algorithms. In *The 6th Hellenic-European Conference on Computer Mathematics and its Applications*, 2003.

340. Vucetic, S. and Obradovic, Z. A Regression-Based Approach for Scaling-up Personalized Recommender Systems in E-Commerce. In *ACM WEBKDD '00*, ACM Press, New York, NY, USA, 2000.
341. W3C Technical Architecture Group. Architecture of the World Wide Web, Volume One. Retrieved from http://www.w3.org/TR/webarch/
342. Waldo, J. The JINI Architecture for Network-Centric Computing. *Communications of the ACM*, 42 (7), 76-82, 1999.
343. Walker, R. Style Decoder. New York Times Magazine. The New York Times Company, 2008.
344. Wang, X.H., Zhang, D.Q., Gu, T. and Pung, H.K. Ontology Based Context Modeling and Reasoning Using Owl. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, p. 18, IEEE Computer Society, 2004.
345. Want, R., Hopper, A., Falc, V. and Gibbons, J. The Active Badge Location System. *ACM Transactions on Information Systems, TOIS*, 10 (1), 91-102, 1992.
346. Want, R. and Pering, T. System Challenges for Ubiquitous & Pervasive Computing. In *Proceedings of the 27th International Conference on Software Engineering*, pp. 9-14, ACM Press, New York, NY, USA, 2005.
347. Want, R., Schilit, B.N., Adams, N.I., Gold, R., Petersen, K., Goldberg, D., Ellis, J.R. and Weiser, M. The Parctab Ubiquitous Computing Experiment. *IEEE Personal Communications*, 2 (6), 28-43, 1995.
348. Ward, A., Jones, A. and Hopper, A. A New Location Technique for the Active Office. *IEEE Personal Communications*, 4 (5), 42-47, 1997.
349. Ward, A.M.R. *Sensor-Driven Computing*. University of Cambridge, 1998.
350. Waterman, M.S. *Introduction to Computational Biology.* Chapman and Hall, London, 1995.
351. Weber, R., Schek, H.-J. and Blott, S. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, pp. 194-205, Morgan Kaufmann Publishers Inc., 1998.
352. Wei, E.J.Y. and Chan, A.T.S. Towards Context-Awareness in Ubiquitous Computing. In *Embedded and Ubiquitous Computing, EUC 2007*, pp. 706-717, Springer Berlin/Heidelberg, 2007.
353. Weiser, M. The Computer for the Twenty-First Century. Scientific American. Scientific American Inc., 94 - 104, 1991.
354. Weiser, M. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, 36 (7), 75-84, 1993.
355. Weiser, M., Welch, B., Demers, A. and Shenker, S. Scheduling for Reduced Cpu Energy. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, p. 2, USENIX Association, 1994.
356. White, D. and Jain, R. Technical Report Vcl-96-101: Algorithms and Strategies for Similarity Retrieval. Visual Computing Laboratory, University of California, La Jolla, California, USA, 1996.
357. Wikipedia. Slope One. Retrieved from http://en.wikipedia.org/wiki/Slope_One
358. Willinger, W. and Doyle, J. Robustness and the Internet: Design and Evolution. *Robust Design: A Repertoire of Biological, Ecological, and Engineering Case Studies*, Jen, E. (Ed.), p. 231, Oxford University Press, 2002.
359. Wishart, R., Henricksen, K. and Indulska, J. Context Obfuscation for Privacy Via Ontological Descriptions. In *Proceedings of the 1st International Workshop of Location and Context-Awareness, LoCA 2005*, pp. 276-288, Springer Berlin/Heidelberg, 2005.
360. Wold, E., Blum, T., Keislar, D. and Wheaton, J. Content-Based Classification, Search, and Retrieval of Audio. *IEEE MultiMedia*, 3 (3), 27-36, 1996.
361. Wong, J. Marmite: Towards End-User Programming for the Web. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*, pp. 270-271, IEEE Computer Society, 2007.

362. Wong, J. and Hong, J. What Do We "Mashup" When We Make Mashups? In *Proceedings of the 4th International Workshop on End-user Software Engineering*, pp. 35-39, ACM Press, New York, NY, USA, 2008.
363. Wong, S.K.M., Ziarko, W., Raghavan, V.V. and Wong, P.C.N. On Modeling of Information Retrieval Concepts in Vector Spaces. *ACM Transactions on Database Systems, TODS*, 12 (2), 299-321, 1987.
364. World Wide Web Consortium (W3C). Composite Capabilities/Preference Profiles: Structure and Vocabularies 2.0 (Cc/Pp 2.0). Retrieved from http://www.w3.org/Mobile/CCPP/
365. World Wide Web Consortium (W3C). Html 4.01 Specification. Retrieved from http://www.w3.org/TR/1999/REC-html401-19991224/
366. World Wide Web Consortium (W3C). Platform for Internet Content Selection (Pics). Retrieved from http://www.w3.org/PICS/
367. World Wide Web Consortium (W3C). Rdf/Xml Syntax Specification (Revised). Retrieved from http://www.w3.org/TR/rdf-syntax-grammar/
368. World Wide Web Consortium (W3C). Semantic Web Initiative. Retrieved from http://www.w3.org/2001/sw/
369. World Wide Web Consortium (W3C). Simple Object Access Protocol (SOAP) 1.1. Retrieved from http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
370. World Wide Web Consortium (W3C). Web Services Addressing (Ws-Addressing). Retrieved from http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/
371. World Wide Web Consortium (W3C). Web Services Architecture, W3c Working Group Note 11. Retrieved from http://www.w3.org/TR/ws-arch/
372. WSMO Working Group. Web Service Modeling Ontology (Wsmo Final Draft 21 October 2006). Retrieved from http://www.wsmo.org/TR/d2/v1.3/
373. Wu, X., Zhang, L. and Yu, Y. Exploring Social Annotations for the Semantic Web. In *Proceedings of the 15th International Conference on World Wide Web*, pp. 417-426, ACM Press, New York, NY, USA, 2006.
374. Xu, J. and Croft, W.B. Improving the Effectiveness of Information Retrieval with Local Context Analysis. *ACM Transactions on Information Systems, TOIS*, 18 (1), 79-112, 2000.
375. Yarin, P. and Ishii, H. Touchcounters: Designing Interactive Electronic Labels for Physical Containers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: the CHI is the Limit*, pp. 362-369, ACM Press, New York, NY, USA, 1999.
376. Zezula, P., Amato, G., Dohnal, V. and Batko, M. *Similarity Search: The Metric Space Approach (Advances in Database Systems).* Springer, 2005.
377. Zhou, D., Bian, J., Zheng, S., Zha, H. and Giles, C.L. Exploring Social Annotations for Information Retrieval. In *Proceeding of the 17th International Conference on World Wide Web*, pp. 715-724, ACM Press, New York, NY, USA, 2008.
378. Zhu, F., Mutka, M. and Ni, L. Splendor: A Secure, Private, and Location-Aware Service Discovery Protocol Supporting Mobile Services. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, p. 235, IEEE Computer Society, 2003.
379. Ziegler, C.-N. *Towards Decentralized Recommender Systems*. PhD Thesis, Albert-Ludwigs-Universität Freiburg, 2005.
380. Zobel, J. and Moffat, A. Inverted Files for Text Search Engines. *ACM Computing Surveys, CSUR*, 38 (2), 6, 2006.

Short Curriculum Vitae of Darren Vaughn Carlson

Personal Information

Grosser Bauhof 5
23552 Luebeck
Germany

Phone: +49-(0)451-889-6614
E-mail: carlson@itm.uni-luebeck.de
Born: 24/2/1972 in Coon Rapids, Minnesota, USA

Education

Master of Science in Digital Media *6/2005*
University of Luebeck, Germany (ISNM)

Bachelor of Science in Electronic Engineering Technology *6/1998*
Minnesota State University, Mankato, Minnesota, USA

High School Degree *6/1990*
Hutchinson High School, Hutchinson, Minnesota, USA

Professional Experience

Founder and Lead Software Engineer *10/2003 - Present*
StreamLabs LLC, Minnesota, USA.

Research and Teaching Assistant *10/2003 - 7/2008*
International School of New Media gGmbH, Luebeck, Germany.

Research Staff Member *3/2000 - 9/2003*
NEC Computer and Communications Research Laboratories, Heidelberg, Germany.

Embedded Systems Engineering Internship *3/1998 - 5/1999*
Elektrobit OY, Oulu, Finland.

Publications

Carlson, D. *Adaptive Ubiquitous Computing Environments.* In: Hasebrook, J., Muhr, G. and Schrader, A. (eds.): Applying Digital Media to Culture, IOS Press, Amsterdam, 2008 (in press).

Carlson, D., Busch, D. and Schrader, A. *Modular Framework Support for Context-Aware Mobile Cinema.* Personal and Ubiquitous Computing, 12 (4), 2008, 299-306.

Detken, K., Martinez, C., Carlson, D., Guljajeva, V., Oja, M.K. and Schrader, A. *ECHOES - A Crazy Multiplayer Pervasive Game.* Workshop for Mobile Gaming (Mobile Gaming '08) at (GI) Informatik'2008, Munich, Germany, September 9, 2008.

Heumer, G., Carlson, D., Jung B. and Schrader, A. *Paranoia Syndrome - A Pervasive Multiplayer Game.* In: Carsten Magerkurth, Carsten Roecker (eds.): Pervasive Gaming Applications - A Reader for Pervasive Gaming Research vol. 2, Shaker Verlag, 2007.

Carlson, D., Schrader, A. and Busch, D. *Mobile Cinema - Context-aware Interactive Video.* Workshop on Investigating new user experience challenges in iTV: Mobility & Sociability at the International Conference on Human Factors in Computer Systems (CHI'2006), Montreal, Canada, April 22, 2006.

Heumer, G., Carlson, D., Kaligiri, S. H., Maheshwari, S., Hasan-Waqar-Ul, Jung, B. and Schrader, A. *Paranoia Syndrome - A Location Based Pervasive Multiplayer Game using PDAs, RFID, and Tangible Objects.* Third International Workshop on Pervasive Gaming Applications - PerGames 2006.

Carlson, D. and Schrader, A. *Ocean: Community-Based, Real-World Ubicomp.* 8th Annual Conference on Ubiquitous Computing (Ubicomp 2006), Workshop: System Support for Ubiquitous Computing (UbiSys'2006), Orange County, California, USA, Sept. 17-21, 2006.

Schrader, A., Ahad, A., Carlson, D., Egan, B., Kaminski, J. and Moser, J. *Das begreifbare Museum - Ein interaktives Informationssystem (Demonstration).* ACM Mensch & Computer 2005, Linz, Austria, September 4-7, 2005.

Jung, B., Schrader, A. and Carlson, D. *Tangible Interfaces for Pervasive Gaming.* 2nd International Conference of the Digital Games Research Association, Vancouver, British Columbia, Canada, June 16-20, 2005.

Carlson, D. *Aladin: An Extensible Ubiquitous Computing Infrastructure.* Master's Thesis, University of Luebeck, Germany, 2005.

Carlson, D., Fan, C., Kassler, A., Niedermeier, C., Schrader, A. and Schorr, A. *MASA: A scalable QoS Architecture.* 7th IASTED International Conference on Internet and Multimedia Systems and Applications, Honolulu, Hawaii, USA, August 13-15, 2003.

Carlson, D. and Schrader, A. *Seamless Media Adaptation with Simultaneous Media Processing Chains.* Poster session at the the 10th ACM International Conference on Multimedia, Juan-les-Pins, France, December 1-6, 2002.

Carlson, D., Hartenstein, H. and Schrader, A. *QoS Orchestration for Mobile Multimedia.* The 1st Workshop on Applications and Services in the Wireless Networks (ASWN'2001), Evry, France, July 25-27, 2001.

PATENTS

Carlson, D. and Schrader, A. *Verfahren zur Uebertragung von zeitsynchronen Daten.* German Patent DE 102 28 861 B4 (04.05.2005). Patent holder: NEC Europe Ltd., 69115 Heidelberg.

Carlson, D. and Schrader, A. *Mechanism for transmission of time-synchronous data.* US Patent (pending): US 2004/0008626 A1 (15.01.2004). Patent holder: NEC Europe Ltd., 69115 Heidelberg.

Carlson, D. and Schrader, A. *Time Synchronous Data Transmitting Method, Data Processing Apparatus and Network.* Japanese Patent: JP 2004/072737 A (04.03.2004). Patent holder: NEC Europe Ltd., 69115 Heidelberg.