i

From the Institute for Telematics
The University of Luebeck

Director:
Prof. Dr. rer. nat. Stefan Fischer

Aspect-Oriented Adaptation Composition and Dynamic Reconfiguration in Multimedia Frameworks

Dissertation
For Fulfillment of
Requirements
For the doctoral degree
of the University of Luebeck
from the faculty of Technology and Natural Sciences –

Submitted by

Muhammad Asadullah Khan
from Rawalpindi, Pakistan

Luebeck, 2007

Abstract

With enormous expansion of the Internet in size, particularly the wireless part, and an ever increasing myriad of distributed multimedia applications, we are moving towards its more ubiquitous use in future. In particular, the number of embedded and resource constrained computers will significantly increase along with the resulting increase in the number of applications. Due to the distributed nature of such systems, the number of inter- and intra-application interactions will rise. These interactions will be unpredictable, dynamic and distributed. Therefore, the system as a whole will need to adapt to the changes as soon as they occur. Contemporary solutions exist in the form of a middleware layer, which is reconfigurable using reflective or aspect-oriented programming. This indirectly enables adaptive execution of the applications running on top of it. The applications interact with this layer via stubs, skeletons or interfaces.

In this thesis the existing development paradigm has been reviewed for its shortcomings and a new paradigm of composing adaptive behaviors has been developed, in the context of multimedia frameworks. Two main issues which have been tackled are static composition of adaptation and its dynamic reconfiguration. This new paradigm is based on the state machine model and is implemented in software as an event-based system. The new paradigm differs from the contemporary work in the sense that it does not rely on any stubs, skeletons or interfaces. Adaptation mechanism is completely confined in a separate layer, at the time of adaptation composition, whereas, at the application load-time, smart patches of code are generated according to the specified adaptation behaviors and are weaved into the supplied application code, using aspect-oriented programming techniques. The source code of the application is not required; weaving is done on the compiled code instead. Thus the actual application code is transformed into new code, which has now adaptation behaviors composed into it. When the transformed code starts execution, it generates events. The events are trapped and adaptively diverted to trigger the application from one state to another, in response to dynamic changes occurring in the entire system. Consequently, the execution pattern of the given code is impacted by changes external to the application and the application code adapts to those changes. Since multimedia application code has been considered in

particular, multimedia data flow undergoes dynamic and adaptive modifications as a result.

Through quantitative and comparative analysis it has been shown that the model performs significantly better than the existing systems, which use middleware as a separate layer underneath the application. Also, this model can handle various types of application oriented multimedia adaptations, which rely on the application code and cannot be adequately handled by existing systems without human involvement. The architecture is portable and realizable in other languages as well. Although the work was done in the context of multimedia frameworks, the principles used by the conceptual model are applicable to any event based framework. The work presented in this thesis concludes by proposing suggestions for further development with particular reference to its portability to different frameworks and extensibility using dynamically reconfigurable hardware.

Acknowledgements

With a deep feeling of satisfaction that comes upon completion of a task, I thank my supervisor Prof. Dr. rer. nat. Stefan Fischer, whose guidance and encouragement throughout my work at IBR, Braunschweig and later at ITM, Luebeck has been a great source of inspiration. I felt a great pleasure, when I was given a free-hand to experiment and explore research ideas. In addition to his valuable technical suggestions during the process of writing scientific papers and this dissertation, he was happy to help solve many of my problems which were not even related to his profession, but I would have faced them as a foreign student in Germany. In particular, I am thankful for all the 'letters' he wrote to the "Auslaenderbehoerde" and different foreign offices and embassies, to handle visa related issues.

I am grateful to the Braunschweig University of Technology and the University of Luebeck for employing me as a Scientific Assistant and providing necessary financial assistance to enable my all conference participations. Prof. Dr.-Ing. Lars C. Wolf deserves my thanks, for all his efforts to handle my visa extension issues with the Braunschweig Foreign Office, after Prof. Fischer moved to Luebeck.

My colleagues, Dr.-Ing. Joerg Diedrich and Frank Strauss deserve thanks in particular, since without them, my initial days in Germany would have been miserable. They were there to help with even those things which often appeared minor, but were practically significant and hard to handle as a foreign student. For me, they were not only 'Live German-English Dictionaries', they also told some wonderful tricks to survive in Germany. They deserve appreciation for all the help they extended to me, including their all 'unsuccessful' efforts to teach me the German Language! My colleague Prof. Dr. Christian Werner, was particularly helpful in finalizing dissertation submission formalities.

Last but above all, I must remember the long time sacrifice of my family, without whose encouragement and support, I would have neither been able to start this task, nor complete it.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

With the enormous expansion of the Internet in size, particularly the wireless part, and an ever increasing myriad of distributed multimedia applications, we are moving towards its more ubiquitous use in future. The number of computers (in particular those which are embedded and resource constrained) will significantly increase. This will lead to an even greater number of applications running on such devices and a corresponding rise in the number of their interactions. These interactions among applications (and different components of the same application) will be distributed, un-predictable and dynamic. Therefore, stable operation of future systems can only be guaranteed if different components constituting an application and different applications making up the entire system are capable of adapting themselves to changes in a coordinated manner. However, looking at the complexity of large distributed systems of resource constrained devices, it is neither feasible nor possible to take into account so many different adaptation scenarios, because the number of situations which may arise during an applications execution lifetime, and interaction patterns of different applications are unpredictable beforehand.

These inter-application, intra-application and system/network-wide interactions, which in-turn contribute to dynamically changing resource requirements during the application runtime, will render the design of adaptable applications very complex and their interactions un-manageable, especially when certain Quality of Service (QoS) is desired. Furthermore, it is not possible to have as many human experts to cope with this situation because it requires a good understanding of specific adaptation Application Programming Interfaces (APIs) and using those APIs in developing every application requires analysis of the entire system dynamics. The solution lies in development of systems which require no or minimal user involvement to adapt their functionality to varying operating environments.

The work presented in this dissertation is motivated by the need to develop frameworks/environments, which can facilitate intelligent adaptations after making legacy multimedia applications (non-adaptive ones) capable of adapting, without any user involvement in programming adaptation behaviors and with minimum (or without) user involvement in configuring adaptation parameters. Although this work is focused on multimedia applications on wireless networks of resource constrained devices, the methodology presented here is applicable to other event-based development frameworks in different domains.

1.2 An Overview of Existing Approaches

Since the early days of distributed computing software like Remote Procedure Calls(RPC) efforts have been mostly focused on hiding low level networking details, providing distribution transparency and certain other such features by introducing an abstraction layer beneath the applications. Later developments like Distributed Computing Environment(DCE-RPC) provided some additional services along with a number of uniform interfaces for the application programmers while masking the network heterogeneity. Further middleware systems including a number of well known technologies like OMG's CORBA [OMG1995], DCOM (which evolved as DDE to OLE to COM to DCOM) [MM1997], Java RMI [Sun1997] matured and are being widely used today, however, the motivation behind all such progress was to ease the development of enterprise applications for which, characterizing parameters are robustness, persistence, transaction security etc. These technologies in their common form are, therefore, not suitable for applications which are required to adapt dynamically to changes in their operating context.

Technologies like CCM [OMG2001] , EJB [Sun2001] and COM [Don1997] , mainly enhanced the capabilities of existing middleware technologies by using component models, enabling reusable service composition, configuration and installation, but the focus remained on business QoS. Other approaches to produce purpose-built middleware for communication, like Adaptive Communication Environment (ACE) [Sch1994], and TAO [Sch1999], were although custom tailored to communication QoS, at least a part of these middleware still remained as a monolithic layer beneath the applications. This is not suited to resource constrained devices due to the relatively large footprint of the middleware.

Further improvements in literature appear in the form of custom-tailored middleware mostly relying on reflective-programming techniques, and can be broadly classified as Adaptive and Reflective Middleware, which facilitate inspection and alteration of the middleware layer and adjusting it to needs. The principal benefit of such technologies is that low level networking details remain hidden when required, but, can be exposed to the application programmer where needed, through reflection. Thus a general purpose middleware can be made domain specific when desired. Two prominent examples are Open ORB [BCA+2001] and Dynamic TAO [RKC1999]. Although in both these research prototypes, reflection is used to configure the ORB, the process of application development and customization is fairly human dependent and requires two phases:

(i) - The applications should be written to benefit from the API offered by the ORB

(ii) - and the ORB must be configured to the operating environment.

A number of projects like BBN's Quality Objects(QuO) [LBS+1998] and middleware based on CORBA compliant ORBs, like Component-Integrated ACE ORB (CIAO) [Wang2003], ZEN [KKS+2003], AspectIX [HBG+2001] are targeted to provide communication oriented QoS by using Meta Object Protocols (MOPS) [Kic1991] or Aspect oriented Programming (AoP) [KLM+1997]. These technologies enable separation of functional and non-functional concerns in the middleware layer. Due to this separation of concerns, the resulting middleware is customizable and can have small footprint as well, however, there are still some limitations discussed below; in particular, when a general purpose middleware is customized to a specific domain.

1.3 Limitations of Existing Work

Despite having undergone many improvements, the existing middleware technologies are still limited in coping with the challenges posed by specific domains, like integrated networks, wireless multimedia on resource constrained devices etc. These limitations are summarized below:

- *Middleware exists as a separate layer beneath the application. Additional abstraction layer is not suitable in case of multimedia applications, because it introduces un-necessary overhead in copying packets and transfer of other control signals to and from this layer. The additional operations consume a significant amount of system resources like the battery power on small devices. Even in those schemes, which offer the option of bypassing the middleware layer, application dependent properties (e.g., multimedia transcoding) cannot be efficiently handled..*

- *The programmer is required to program with some purpose-built QoS API, which needs substantial effort to analyze (rather predict) the runtime system behavior, precisely estimate resource utilization etc, which is never possible due to time varying nature of the interplay between resource requirements and resource fluctuations. This impact is many-fold in wireless networks (the details are given in later chapters). It is a major problem and will always hinder system's performance predictability, throughout its life-time.*
- *These models mainly target system-side adaptation (based on resource reservation and allocation etc.) and have been used for mission-critical distributed real-time embedded applications (which are non-elastic in nature and rely on pre-hand over-booking of the system resources). In order to guarantee QoS, these schemes reserve resources in multiples of the minimum application requirements. Therefore, unless the resource fluctuations are very high, most of over-booked resources go waste for most of the application lifetime. Since multimedia applications can generally tolerate some resource fluctuation and resource over-booking will lead to waste of resources, adaptation based schemes is likely to perform better.*

Implementation type	Flexibility	Efficiency	Programming Ease	Size Suitability
Implemented solely inside application	Less	High	Low; the user needs to devise the adaptation procedures and program them.	Suitable for small foot print devices (because only the application is present, no middleware)
Implemented purely outside the application	High	Less (because the application has no control and is bound to rely on whatever the system provides.	High; because the user does not need to master the adaptation details for each application, a general API will suffice in all cases.	Not Suitable for small foot print devices, because of the existence of additional middleware layer.
Implementation spread across the application and the layer beneath	Relatively high	Low (because the application and the middleware both have to coordinate adaptation decisions, which reduces execution speed)	High; because the user does not need to master the adaptation details for each application, a general API will suffice in all cases. However the applications must be specifically designed to operate in different modes.	Midway between the above two extremes.

Table 1.1 – Comparison of placement of adaptation in overall system

The limitations of the existing models, as described above were in particular related to the systems, which mainly target system-side adaptation by resource reservation and adaptive (re)-allocation. Adaptive software systems can also be classified with respect to the placement of adaptation mechanism inside the system, as shown in table 1.1.

1.4 Aimed Contributions

Considering the limitations and comparison, above, we conclude that:

high execution efficiency and small footprint can be achieved only if the adaptation mechanism is embedded into the application, while maximum flexibility can only be retained when the implementation of adaptation mechanism resides completely outside the application.

The goal of this research is to devise a conceptual model, develop an architecture and realize the developed architecture to fulfill both the above conflicting requirements in an optimal way. This has been done by integrating aspect oriented methodologies existing in the software engineering community with the adaptation mechanisms proposed by the networking community to develop an adaptation composition and dynamic reconfiguration environment. Main contribution of the this research is that it proposes an improved paradigm to engineer domain specific adaptation systems for event based frameworks. The work presented in this dissertation is distinctive in three respects:

- *The system software layer, responsible for adaptation exists separately at the time of composition (due to which the adaptation parts stay separate from the application, giving the benefits of separation of functional and behavioral concerns), but forms a part of the application at runtime (thus giving the performance benefits of embedding adaptation code into the application).*
- *The implementation is custom-tailored to multimedia application on resource devices, due to which it has a very small foot print and as it is particularly in the context of multimedia frameworks (where the skeleton is pre-defined), it does not need programming effort (except the very minimum 'tweaking' in some cases, for fine tuning).*
- *Since its implementation is Aspect Oriented and aspects are additive in nature (code patches are added only when required, thus stripping off the unnecessary adaptation code when not required). These aspects are added to the application's target code (source code in not required) and adaptation is achieved on the data stream.*

Although the research was carried out in the context of multimedia application development frameworks, which are largely event-based, the impact of the concept of using automated aspect weaving for self-managed adaptations is far-reaching. Since the fundamental approach of this work is based on the concept of state machines, the implementation can be extended to other event based development frameworks and other categories of applications can be targeted.

A number of research works and prototypes target context aware application adaptation. Although these efforts are adaptation based and radically different from the resource reservation based approaches (which are widely employed for QoS guarantees), these mainly concentrate on service adaptations, and a few others, which may not be considered as full-fledged middleware (e.g, SMIL) focus on content adaptations. None of these provides the concept of automated weaving of some adaptation mechanism, but, rely on the user to design each individual application according to the anticipated operating context, which is again not a good solution because of the third limitation detailed above.

1.5 Technical Challenges

To reach an optimal compromise between these two conflicting requirements, there are a number of challenges which need to be addressed at different levels.

Technical challenges exist in:

- *Devising the right conceptual model which can capture adaptation requirements with minimum user involvement.*
- *Developing the most feasible architecture to realize the proposed model, which comprehensively incorporates (or be extendable to incorporate) various aspects of application-oriented and system-oriented adaptations.*
- *Realizing the proposed model and architecture using, developing and extending specific language features and software abstractions.*

In the scope of this work it will be proved with the help of a test-case implementation that it is possible to devise an implicit adaptation mechanism for an event based multimedia framework, which can intelligently and without (or with minimal) user involvement, comprehensively map inter-application, intra-application and network-wide adaptation requirements to the underlying development framework and that these requirements can be automatically weaved into prewritten non-adaptive applications, transforming them into adaptive ones, completely getting rid of any middleware layer (at

the expense of loosing some flexibility), thus making adaptation a sole property of the composition and reconfiguration environment.

1.6 Organization of the Thesis

Having given a background introduction to the work presented here, the rest of this dissertation is organized as follows:

Chapter 2 presents a detailed survey of the related work, giving a classification of existing work and introduces the fundamental concepts useful in understanding the rest of this dissertation.

Chapter 3 introduces the conceptual model of the system developed in this research. The model is based on the basic principles of state machines and event based systems. It sets the stage for integrating application oriented adaptation with software engineering techniques of reflection and aspect-oriented programming, in the realm of multimedia frameworks.

Chapter 4 gives an architectural overview of the proposed model in the form of an Adaptive Composition and Runtime Environment for Multimedia Applications (ACREMA).

Chapter 5 details implementation of the proposed architectural model with examples of code snippets to show main parts of implementation.

Chapter 6 explains the results obtained from practical evaluation of the implemented model using test applications. Results include both functional as well as architectural evaluation and are obtained from application execution on an emulated network on Linux Virtual Machines.

Chapter 7 concludes this research with the lessons learnt, proposes some enhancements and gives suggestions for further improvements and future work.

Chapter 2

Background and Survey of Related Work

2.1 Introduction

The work presented in this report has benefited from a wealth of research existing across a multitude of QoS and adaptation related fields. This includes in particular, research related to algorithms and implementation of the fundamental QoS and adaptation concepts, existing work in the domain of mobile and pervasive computing, especially the language features and abstractions for development of adaptive software systems and the related work in development of adaptive software platforms and their architectural models. Keeping in view the wide span of these disciplines and the theme of the work presented in this dissertation, the chapter has been broadly divided into three main sections as follows:

QoS and Adaptation – Fundamental Concepts **(details in section 2.2)**, which provides a background for establishing the requirements and categorizing them. The work presented in this section includes relevant fundamental concepts, overview of the existing algorithms and standards and some example implementations. This section mainly summarizes the work where main emphasis was on design of a QoS provisioning or adaptation algorithm. In these cases the work does not focus on exploring the effectiveness of a particular language feature or a particular architectural model of the software.

Language Features and Software Tools for Adaptation **(details in section 2.3)** This section details certain language features and various language enhancements which provide different techniques to incorporate adaptive behaviors in software. Therefore, the work summarized in this section is focused on exploring the effectiveness of a particular language feature in the development of adaptive software systems (and not on development of adaptation algorithms). It is meant to introduce those enabling technologies which are used by the work presented in the next section. These techniques

mainly include reflection, reification, Meta Object Protocols (MOPs) and Aspect-oriented Programming/Aspect Oriented Software Development (AoP/AOSD).

Software Systems and Models to Realize QoS and Adaptation **(details in section 2.4)** This section describes the employment of the language features and tools discussed in section 2.3, in developing adaptive systems. Examples include adaptive and reflective middleware, aspect-oriented middleware, meta architectures, runtime environments and the related work in these domains. The above mentioned language features and software tools. This is mainly a comparative study of existing adaptive (reflective and/or Aspect-oriented) middleware and runtime environments, including in particular the study of architectural models for resource constrained devices and ubiquitous environments. The work presented here is mainly from the software engineering community and is focused on exploring the effectiveness of a software engineering paradigm or language features to develop adaptive systems.

2.2 Quality of Service and Adaptation - Fundamentals

The International Telecommunication Union (ITU) standard X.902, Information technology – Open distributed processing – Reference Model, refers to QoS as "A set of quality requirements on the collective behavior of one or more objects". Different researchers have given various definitions. We agree on the following definition of Quality of Service (QoS), given by IEEE "The set of those quantitative and qualitative characteristics of a distributed multimedia system, which are necessary in order to achieve the required functionality of an application" [SN2004].

Practically, providing QoS is achieved by striking a balance between resource supply and demand, by allocating (and dynamically reallocating) the resources in high demand to various applications or their components, according to preset contracts. Implementation of such contracts may involve resource costs, service charges, user preferences etc. Two well known standards, for providing QoS in the Internet are *Integrated Services (Intserv)* and *Differentiated Services (Diffserv)*. However, the applicability of these standards is limited to wired, fixed networks. Intserv is aimed at providing QoS guarantees to individual application sessions, whereas, Diffserv is to handle different classes of internet traffic in different ways. Inherent scalability problem due to per-flow state maintenance of the Intserv and the resolution of pushing the load to edge routers in the case of Diffserv, make the suitability of these standards questionable

for wireless networks, in particular. In case of wireless communication, the problem of data transmission and reception itself becomes a problem and due to the same reason, issues like efficient and reliable routing, mobility management, adaptive resource management etc. are hot areas of research in their own right. Such issues are being mostly handled at the network and MAC layers in the networking community.

In general, QoS provisioning as viewed by the available literature can be seen as a combined effect of QoS specification, mapping, routing, resource management and adaptation. *QoS Specification* refers to the definition of the required QoS level in terms interpretable form the viewpoint of a particular system entity. At user level it may only be specification of the human perception [ZKS+2003] (e.g; of a video or audio, in terms of excellent, good, poor etc). At application level, the specification parameters will be application specific (e.g; frame rate, frame size). At system level, these may be buffer size, tolerable delays etc. Similarly, there are parameters related to communication channel signal to noise levels etc.

Therefore QoS provisioning mechanisms are generally implemented using a layered architecture, and quality requirements from each higher level are translated to (mapped onto) the adjacent lower level of the architecture. For example the specification given by the user, or that given by a specific application has to be translated to lower levels of the protocol stack, considering available resources (like specification of the requirements of good, poor or excellent has to be somehow translated into parameters like frame-rate, frame-size etc at application level, or to inter-arrival jitter, delay, throughput of the network channel, device display capabilities, buffer size and available battery life keeping in view the available resources.

Two principle approaches that exist at present to provide QoS in IP networks and differ in their suitability for application domains are: System-Oriented QoS and Application-Oriented Qos (Adaptation). These are discussed in the next two sub-sections.

2.2.1 System (Network) Oriented QoS

A lot of work has been focused in this area in the networks community. This scheme is based on pre-hand resource reservation and relies on some reservation protocol like RSVP [ZDE+1993], dynamic RSVP (dRSVP) [KK2000]. The applications submit their requirements in advance and this information is used in the subsequent phases. The calls are admitted (or denied) on the basis of the information provided before the setup.

The advantage is that it can lead to a stable QoS through out the session. However, this assumption only holds if the applications can precisely estimate their resource requirements throughout the system, which is seldom possible. Due to this reason, the resources will need to be reserved in higher amounts than those demanded by the applications, in order to guarantee the availability of that amount throughout the application lifetime. The downside of all such approaches is that if each application reserves resources in excess amounts, and the situation where these surplus resources are required does not occur or remains for a very short duration, then for a significant period of time those resources which were reserved in surplus will go waste. This will result in guaranteed QoS provisioning at a very high cost. Due to these reasons such an approach is suitable only in mission-critical applications (like DRE, UAV), examples implementations include TAO, CIAO, ZEN.

2.2.2 Application Oriented QoS (Adaptation)

Application Oriented approach to provide QoS is more flexible as compared the one discussed above. The applications do not need to submit an estimate of resource consumption in advance, but benefit from the resources when they are available and adapt themselves to utilize lesser resources when the resources fall below the critical levels. This approach cannot provide any hard QoS guarantees and is suitable only where soft QoS is required (e,g; in certain multimedia applications, if optimum video quality is available it would be better but the user would generally bear with slightly lower quality). This approach is effectively a compromise between guaranteed QoS and *best effort* implementations. This kind of approach to provide QoS is discussed in more detail in the subsequent sections.

More importantly and above all, both the approaches discussed can co-exist in a system. We can have examples of the systems mainly relying on Network Oriented QoS but supporting Adaptation, where adaptation is used only when the resources in the system fall below critical levels.

2.2.3 Combined System QoS and Application Adaptation Approaches

There are some approaches like [FRS2000], where both the aspects have been considered. Such schemes take into account the demands of the application, initially reserving the system resources in surplus and provide QoS guarantees within certain

operational bands. When the application requirements exceed those initially specified (or in case when the demands cannot be met by the underlying system), the application is required to adapt to whatever can be offered by the system. In such cases the mechanisms to realize certain QoS are built into both the application and the system.

Since a discussion of QoS algorithms and mechanisms at lower levels of the protocol stack are not directly within the scope of the work presented in later chapters of this dissertation, only a brief overview has been presented.

2.2.4 Adaptation in Pervasive and Mobile Computing

Since pervasive computing [Wei1993] is about the technologies that disappear, its realization needs addressing the challenges of very large scalability, seamless integration of different technologies, acquisition and management of context and invisibility of the technology enabling all these. Pervasive computing can be viewed as a close companion of mobile computing or as suggested in [SM2003], as a superset of mobile computing. In case of pervasive environments, adaptation takes on several dimensions, since not only the network bandwidth is usually limited, device capabilities (relating to its processing power, display size, available battery power) are also constrained, adaptation can play a pivotal role to realize the vision of pervasive computing, this section therefore, summarizes some existing work related to adaptation in pervasive and mobile environments. While considering adaptation (in particular multimedia adaptation), in the context of pervasive and mobile environments, two most significant facets of the subject are concerned with (i) – characteristics of the environment and (ii) – characteristics of the devices operating in such environments. These two dimensions of pervasive and mobile adaptation are briefly described below.

2.2.4.1 Adaptation with respect to Characteristics of Pervasive Environment

With regard to pervasive computing, adaptation of content and services is critical in accordance with the changes in the environment and device limitations. A significant amount of research aimed at this type of adaptation is found in the literature, encompassing a wide range of techniques from media transcoding proxies to use of different markup languages to describe the content itself. These are the approaches generally using compression, searching, indexing and filtering to manage and scale multimedia data. Early work in this respect, like [KDP+2002] has mostly addressed the

issues like compression and content adaptation and applied these concepts to video adaptation for both streaming media and point to point communication, while keeping main focus on compression of the content. Work like InfoPyramid [SML1999], manages different variations of multimedia data objects with different fidelities (summarized, compressed and scaled variations) and modalities (video, image, text and audio) and generates and selects among the alternatives in order to adapt the delivery to different client devices. [YLC+2002], introduces a method based on clipping of the web content using Web Clipping Markup Language (WCML), using which a clip is automatically extracted from a source page based on a clip specification provided by a content provider and transformed into a target page according to a set of conversion rules. The clips stored in an intermediate meta-language are later transformed into multiple presentation pages in different target markup languages. In a similar work [LL2002], content adaptation using SMIL has been combined with a proxy-based architecture. On contrary to pure content-based adaptation using markup languages, which necessitate generating multiple versions of the same application, combining proxy-based architecture with markup languages enables this generation on the fly, without requiring the content provider to write multiple versions of an application for multiple types of clients.

2.2.4.2 Adaptation with respect to Characteristics of Pervasive Devices

A distinguishing property of any pervasive environment is associated with the devices used in such an environment, which are generally limited in resources, like battery power and display size, in particular. In this regard, researchers have targeted both hardware oriented procedures which includes designing low-voltage devices to *Real Time Dynamic Voltage Scaling* (RT-DVS) [PS2001] strategies for energy conservation on small foot-print devices by integrating adaptive battery consumption strategies and real-time task constraints into the operating system scheduler. Enhancements of such work include [PHS], putting forward the concept of *Energy Aware QoS* (EQoS) that can manage real-time tasks and adapt their execution to maximize the benefits of their computation for a limited energy budget.

Some prominent examples of research specifically related to energy aware multimedia adaptation include [KW2001], proposing software implementation strategies for power-conscious systems to algorithms and middleware frameworks for energy-aware processor reservation [YN2006] to coordinate adaptation of multimedia algorithms and energy

resources [YNG2001]. [MV2003; TSY+2004] present an adaptive middleware solution for power-aware video streaming to mobile hand-held devices by adaptive switching of the device network interface card to sleep mode and adapting the video burst size, while [PS2004] details an approach to strike a balance between the end-system QoS requirements and available battery power on resource constrained devices by dynamically selecting the appropriate transcoders and adaptively scaling the transcoding parameters for streaming video. In a similar approach [CKP2003], MPEG4 Fine Grain Scaling (FGS) has been used by adaptively sending enhancement layers in addition to the base layer to resource-constrained client to achieve energy-aware multimedia transmission. In small devices like PDAs and mobile phones, a significant amount of battery power is sipped by the backlight of the display unit. Especially with regard to video, there are approaches like [PLM+2004; PML+2003] that suggest middleware solutions which target adaptation of the display units backlight to save energy, while preserving video QoS at a reasonable level.

In addition to the research related to optimizing adaptive behaviors with respect to any single characteristic of pervasive computing (as clear from the above examples), existing work that resulted in development of complete pervasive and mobile computing platforms is of great interest and relevance. Work related to pervasive adaptation can be divided into two broad categories [BN2004a]; *context models* that provide a database-style management and interaction and *context ontologies* that focus on a thorough representation of the context knowledge with some reliance on artificial intelligence methods for its manipulation. Since ontology based work is not directly relevant to the research presented in this dissertation, this section focuses mainly on the work that targeted pervasive adaptation in a broader sense and led to the development of complete frameworks or middleware, aimed at achieving system-wide adaptation. Some examples are outlined below:

MobiPADS

Mobile Platform for Actively Deployable Service (MobiPADS)[CC2003] is a reflective middleware specifically designed to facilitate context-aware processing by providing an execution platform to enable active service deployment and reconfiguration of the service composition in response to varying contexts. Unlike most mobile middleware, MobiPADS supports dynamic adaptation at both the middleware and application layers to

provide flexible configuration of resources to optimize the operations of mobile applications.

Gaia

Gaia [RC2000; RHC+2002; RHC+2002a] is a meta-operating system supporting the development of applications for active spaces. An active space is defined as programmable ubiquitous computing environments in which users interact with several devices and services simultaneously. These active spaces must support development and execution of user-centric applications, in which *sessions* associate user data and applications with the users themselves. Users can define different sessions, thus forming a *user virtual space* and can activate and suspend sessions as desired. This allows users to move about with in the active space and still stay connected to their applications.

The core of the Gaia operating system consists of a component management core and it can use CORBA, RMI, SOAP or any other communication middleware. Five basic services offered by Gaia are; the context service, presence service, event management, space repository and the context file-system. Gaia uses LuaOrb, which is based on scripting language Lua, to program active spaces and configure the entities they contain.

Although Gaia is a step forward in the direction of removing the classical interfaces (keyboard, mouse, monitor etc) and takes a step in the direction of pervasive computing, the involvement of the user to configure individual sessions is significant. Also, in a ubiquitous environment a user may not be assumed to be expert in writing language scripts to describe different possible configurations.

Aura

Aura [Garlan2001; SG2002] is an architectural framework to tackle user mobility in ubiquitous computing environments. In Aura, user tasks are represented explicitly, using place holders to capture the user intent and search for suitable configurations in changing environments. Aura's architecture consists of a *task manager, environment manager, context observer and service suppliers.* The task manager is aimed at minimizing user distraction in face of changes. With user movements, all the information relating to a particular task is migrated to the new environment along with negotiating support for that task. The QoS information of the components monitoring a particular task is monitored

and when the requirements cannot be met by the new environment, the environment manager is signaled to find an alternative configuration (if possible). When a task change is explicitly triggered by the user, the task manager takes care of the related house-keeping activity like status saving etc. When the context changes, (the user moves from one environment to another, or someone else walks in the office) context constraints like privacy requirements are readjusted. The service suppliers typically occur in the form of normal applications wrapped such that the wrappers map abstract service descriptions into application specific settings. Service discovery mechanisms of Aura are built on top of existing technologies like Jini.

The main difference between Gaia and Aura is that Gaia emphasizes space programmability by allowing its users to configure the applications to benefit from the resources in their current space [RHC+2002a].

BASE

BASE [BS2003; BSG+2003] is a middleware for pervasive computing based on the concept of micro-broker for resource constrained devices, aimed at providing easy to use abstractions to access remote services and device specific capabilities. The design of BASE is inspired by the research in micro kernels, therefore, instead of providing a whole lot of functionality in the middleware infrastructure, it relies on a flexible plug-in based architecture. Plug-ins can be used to communicate with different local and remote services and to query the device capabilities. The device capabilities are registered at the local registry service and transport protocols at the invocation broker itself. In contrast with the approaches like Gaia discussed above, BASE does not rely on the presence of a specific surrounding where the services could be discovered and used. BASE facilitates different communication models and simple service lookup in the locality of a device is provided by the device registry, which keeps a list of the devices reachable and transport plug-ins to access another device.

PCOM

PCOM is a component system for pervasive computing, which offers application programmers a high level programming abstraction to capture the dependencies among components using contracts [BHS+2004]. In PCOM, adaptation is two fold: at lower level, it uses BASE [BSG+2003], which can swap communication protocols even at

runtime and at the level of application, dependencies among components can be resolved by the adaptation policies embedded in PCOM, without any human involvement, which allows application component selection when more than one suitable components are available. Component dependencies are modeled using contracts. A contract has two parts: the first specifies the corresponding components' requirements (like libraries, memory) while the second part specifies the functionality provided by the component and its dependencies on other components. An application is thus modeled as a tree of components and their dependencies, where the root component identifies the application. PCOM provides three signaling mechanisms to indicate the availability of used components, that of new components and to indicate any change in the quality parameters of a component. These signals can be used by the application programmers to hook the required actions corresponding to different events or use system provided adaptation procedures. The components are atomic with respect to distribution.

AMUN

Autonomic Middleware for Ubiquitous eNvironments (AMUN) [TBP+2005] is a pervasive computing middleware similar to Gaia, which is meant to support a smart office environment, in which personalization issues (like automated telephone call forwarding) related to office assignment to different persons can be managed in an adaptive manner. The underlying structure of the middleware is based on peer-to-peer network, supporting transparent messaging and monitoring. AMUN is a based around the concepts of self-configuration, self-optimization and self-healing. Architecturally, it consists of four main entities; the transport interface, the service interface and the service proxy, the event dispatcher and the autonomic manager. The transport interface provides abstracts the underlying communication platform. The event dispatcher supports the services to send messages and receive messages from other services by registering themselves as listeners.

The main feature of AMUN which makes it suitable for ubiquitous computing is the use of message introspection. Message parameters are given as name, value pairs and only the names and types of the elements are defined using WSDL, therefore, using introspection on messages, it is possible to match parameters to a service description. Also, as the number of message parameters can vary, it is easier to extend the system, for example incorporating a better service after development. For a service that has moved to another node, service proxies are used to forward incoming messages to the new location.

System resources, message latencies and events are monitored by the system to trigger adaptation decisions, e.g., using message latencies and local resources to decide whether it is beneficial to run a service locally or on a remote node.

In addition to the projects described in this section above, there is a significant amount of other related work, which can be classified under the general category of pervasive adaptation. Examples of such research include: Reconfigurable Context Sensitive Middleware (RCSM) [YKW+2002], that models context-sensitive application software as *context-objects.* It uses a context sensitive reconfigurable ORG (R-ORB) which hides the underlying ad-hoc networking details and is responsible for service and device discovery on behalf of the context objects. The implementation has been targeted to support both kinds of communications (direct, like RPC) and indirect (by sharing a common space between interacting applications).

2.3 Language Features and Software Tools for Adaptation

While algorithms and standards for QoS and adaptation provide fundamental concepts and abstractions for various strategies, programming languages and software tools provide the necessary constructs to realize adaptive implementations. Typical language features include code instrumentation, inspection and alteration, dictated by adaptation policies to develop adaptive software. This section presents an overview of these enabling technologies from the programming language and software engineering perspective.

2.3.1 Reflection and Reification

In the context of programming languages, reflection has been defined in [BGW93] as, "*Reflection* is the ability of a program to manipulate as data, something representing the state of the program during its own execution. There are two aspects of such manipulation: *introspection* and *intercessions.* Introspection is the ability for a program to observe and therefore reason about its own state. Intercession is the ability for a program to modify its own execution state or alter its own interpretation or meaning. Both aspects require a mechanism for encoding execution state as data; providing such an encoding is called *reification*."

Different words have been used in the literature to explain this term. For the purpose of discussion in this report, we agree on the following definition, "The process of providing

an external representation of the internals of a system, which allows the internals of the system to be manipulated at runtime, is called *reification*".

Reflection can be structural or behavioral. Structural reflection implies the ability of the language to provide a complete reification of both the program currently executing as well as a complete reification of its abstract data types [ALD+2003]. Therefore, structural reflection provides the ability to alter statically fixed internal data/functional structures and architecture used in a program. A structural reflective system would provide a complete reification of its internal methods and state, allowing them to be inspected and changed. For example, the definition of a class, a method or a function etc can be altered on demand.

Behavioral reflection on the other hand implies the ability of the language to provide a complete reification of its own semantics as well as a complete reification of the data it uses to execute the current program. Therefore behavioral reflection provides the ability to intercept an operation such as a method invocation and alter the behavior of that operation. This allows the program or another program to change the way it functions or behaves.

In brief, structural reflection changes the internal structure of a program, while behavioral reflection alters the actions of a program.

Efficient implementation of structural reflection is much simpler than that of behavioral reflection. Structural reflection features have existed in some form in languages like Lisp and Prolog since long, but, behavioral reflection is a hot topic of research, in particular being investigated for its applications in designing reflective middleware. Various example works making use of behavioral reflection will be discussed in later sections of this chapter.

2.3.1.1 Reflection Support in Contemporary Languages

In programming languages, like Smalltalk, Lisp, Prolog reflection and reflective programming concepts existed since long [DM1995], most modern languages like Ruby, Python, C# etc. also support reflective programming. Limited reflective support was added to java with JDK 1.1 reflection API. Java reflection can be used by (1) – the applications that need to discover and use all of the `public` members of a target object based on its run-time class and (2) – by the applications that need to discover and use the members declared by a given class. These applications need run-time access to the implementation of a class at the level provided by a `class` file. Examples in this category

are development tools, such as debuggers, interpreters, inspectors, and class browsers, and run-time services, such as Java Object Serialization. These applications use instances of the classes `Field`, `Method`, and `Constructor` obtained through different methods of the class `Class`. This allows inspection of the program related metadata like class names, methods, access specifiers, fields etc, within the JDK security restrictions on the fields, methods, and constructors in other objects. It is also possible to load and instantiate classes at runtime and invoke methods on the objects of those classes. An even powerful feature introduced with JDK 1.3 is the possibility of using *proxies*, which support the *interface* of another object (the *target*), such that the *proxy* can substitute for the *target* for all practical purposes through implementation and delegation. The *proxy* therefore acts as an intermediary or a substitute and forwards some or all calls to the target, importantly, it can add method pre-processing or post-processing to the calls when working as an intermediary. This procedure for adding pre and/or post processing to a method call resembles the concept of *before* and *after advice* (explained in section 2.3.4) in Aspect oriented Programming (AoP), however it has certain limitations and is not as powerful as that in AoP. Java 1.5 offers significantly enhanced support for adaptive programming in the form of enhanced reflection features, code instrumentation class and metadata support in the form of annotations.

Among the tools for java byte-code instrumentation, ASM[ASM], BCEL[BCEL], Javassist and Reflex[] provide means of changing class definitions in java. Javassist is more systematic, since it offer an Aspect composition tool Gluonj[], based on top of it, which provides all the basic constructs of an Aspect Oriented Language. Due to its support for code instrumentation and availability of AOP constructs, Gluonj and Javassist were used to form the Aspect Engine in this work.

2.3.2 Meta Object Protocols and Meta Architectures

Gregor Kickzales, in 1991, pioneered the idea of Meta Object Protocols (MOPs) through his groundbreaking work [Kic1991]. The key idea of MOPs is the division of a software system into two levels; the base level and the meta level. The base level describes the actual design of the software to carry out its particular functions, whereas the meta-level keeps information about the base-level and therefore describes different policies which determine the behavior of the base-level. Thus meta-level keeps information about the actual program and this information is known as meta-data. In case of object oriented software, this meta-data is organized into objects, called meta-objects. Thus meta-level can be used to inspect and alter the behavior of the base level. The meta-

interfaces which expose the functions of the base-level are called meta-object protocols, since they specify the means of altering the behavior of base level components, objects or modules. A meta object protocol relies on the fundamental principles of reflection, and should be sufficiently general to permit unanticipated changes to the platform, but be restricted to prevent the integrity of the system [DM1995].

2.3.3 Aspect Oriented Programming

One of the key features of Object Oriented Design is to encapsulate data and functions specific to a goal, in a specific object. Thus an object is a self-contained unit without any information about other objects and others don't have any information about that object, except what the object makes public. In practical software design, there are a number of concerns which cannot be adequately confined into a specific object. A *concern* is a specific requirement or consideration that must be addressed in order to satisfy the overall system goal. Implementation of such concerns affects multiple objects (rather classes) in a system and they are known as *cross-cutting concerns.* For example if a software had been designed without keeping security issues in mind and later on some security policy needs to be implemented, then this will require changes in multiple modules composing the system e.g; it may involve modification of a number of classes. Thus security is a cross-cutting concern, which cuts across several modules of the entire software. Other examples include logging, scheduling etc.

Aspect oriented Programming [Kiczales1996] is a paradigm which facilitates seamless incorporation of cross-cutting concerns into existing software through a mechanism known as *weaving*. Using this software development paradigm, a prewritten program can be 'intelligently patched' at various well defined places, resulting in new code. An aspect oriented language has four main elements which are necessary to carry out the code weaving process. These elements are detailed below.

2.3.3.1 Elements of an Aspect Oriented Language

Join Point

A join point is a well defined location within the actual program code (i.e; the code written by the user to do a specific task), where a concern will crosscut the application. It can be considered as a well defined point within the program, where some additional

code can be meaningfully patched. Examples include, method calls, constructor calls, exception handlers etc. Basically, join points are the places where cross cutting actions are woven in.

Point Cut

A point cut is a program construct which selects the join points (and/or a set of join points). It also collects the context information at those points. For example, a point cut can select a particular method call as a join point and can also capture that method's context like its arguments and the target object. Basically, point cuts provide a kind of link to joint points using the aspect weaver's language constructs. Point cuts are the constructs used to specify weaving rules and join points are the conditions which occur within the primary program flow upon which those rules are satisfied.

Advice

An advice is the additional code, which existed outside the primary program and is to be executed at a join point, specified using a point cut. The advice code is executed, before, after or around join points. If we consider a particular method call as a join point, then *before advice*, refers to the additional code which will be executed before that method invocation, *after advice* is defined as the code that will be executed after that method invocation and *around advice* defines the code which will completely substitute the pre-existing method.

Aspect

An aspect is like a class in an object oriented language. It is a modular unit of code, which contains the point cuts along with corresponding advices.

Different aspect oriented languages use different terms to refer to the above mentioned elements or they extend these basic elements with some language specific features.

2.3.3.2 Aspect Weavers and Related Work

An aspect weaver or an aspect engine is software, which enables compilation of aspects. The process of weaving can be static; meaning that the actual aspects are inserted statically or it may be dynamic; meaning that only insertion hooks are inserted statically and the actual code patching takes place at runtime. There are also cases where weaving is done at load time. Dynamic approaches are generally more flexible, because they allow partial application code modification while the application is running. There are a large number of aspect weavers available for different languages each with its own strengths and weaknesses and a number of those which are language-independent. These weavers differ in the ways they express the crosscutting concerns and how they translate those concerns to form the final system.

2.4 Software Systems and Models for Adaptation

Having presented the language features for software adaptation, this section details the architectural models which have been developed using those language features. In the context of software systems, adaptation refers to extensibility, enhancement of complete or partial software system to tailor it no new needs. Adaptation in software exists in some well known forms, hot-fixing, hot-swapping, software patching etc. Scientific literature in the related area unfolds a whole lot of efforts vested in devising better architectural models for software adaptation. It includes component [] based approaches, using reflection, and recently aspect oriented programming. This section gives an overview of some of the famous projects and a summary of the related work where these techniques of software adaptation have been applied, especially to the middleware layer.

2.4.1 Adaptation's Place in System Hierarchy - The Middleware Level

Traditionally, middleware systems have been developed keeping in mind generic problems like heterogeneity, distribution etc. For example, CORBA [COR], J2EE[J2E] were mainly devised to ease the development of enterprise applications. The middleware level has so far been the most attractive place in the system hierarchy to design adaptive systems [DLS+2004].

2.4.1.1 Reflective and Adaptive Middleware and Related Work

Adaptive middleware is software whose functional behavior can be modified dynamically to optimize for a change in environmental conditions or requirements [LSZ+2001]. Thus

adaptive middleware can be customized to the needs of a specific domain, e.g; embedded control systems with real-time requirements, resource constrained mobile devices etc. Adaptive and reflective techniques have emerged as a new paradigm for the development of next generation dynamic middleware [5] and generally reflection is the primitive technique to achieve adaptation. These techniques enable the system to self-alter to meet the changing environment or user needs. Adaptation can take place autonomously or semi autonomously, on the basis of the systems deployment environment, or according the user defined policies [BCC+1999]. Primary requirements of a runtime adaptive system are: measurement, reporting, control, feedback and stability. Being adaptable is only a feature which a middleware system may have, and this feature is realized using reflective programming. Therefore, in literature, adaptive and reflective middleware are treated together (sometime synonymously).

RAFDA

The Reflective Architecture Framework for Distributed Applications (RAFDA) [PWK+2003] is a reflective framework enabling the transformation of an non-distributed application into a flexibly distributed equivalent one. RAFDA allows an application to adapt to its environment by dynamically adapting its distribution boundaries. It can transform a local object into a remote object and vice-versa, allowing local and remote objects to be interchangeable. RAFDA achieves flexible distribution boundaries by substituting an object with a proxy to a remote instance. The transformation process takes place at the bytecode level. Points of substitutability are identified and an interface is extracted for each substitutable class, then every reference to the substituted class is transformed to use the extracted interface. The proxy implementation provides different transport options including Simple Object Access Protocol(SOAP), Remote Method Invocation(RMI) and Internet Inter-ORB Protocol(IIOP). Policies determine substitutable classes and the transport mechanism to be used.

mChaRM

The multi-Channel Reification Model (mChaRM) [CA2000], is a reflective implementation which reifies and reflects directly on communications. This model does not operate on base-objects but on the communication among base-objects, resulting in a communication-oriented model of reflection. It abstracts and encapsulates inter-object

communications and enables the meta-programmers to enrich and/or replace the predefined communication semantics. mChaRM handles a method call as a message sent through a logical channel between a set of senders and receivers. The model supports reification of such logical channels into logical objects called multi-channels. A multi-channel can enrich the messages (method calls) with new functionality, thus allowing a finer reification-reflection granularity than that use din other approaches. mChaRM is specifically targeted for developing complex communication mechanisms and has been used to extend standard Java RMI to support multicast RMI.

GARF and CodA

GARF [GGM1993] and CodA [McAffer1995] are considered to be milestone in reflective research. GARF is a tool that supports the design and development of reliable distributed applications by wrapping distribution primitives of a system to create a uniform abstract interface which allows the basic behavior of the system to be enhanced.

CodA is a project aimed at fine grain decomposition of the meta-level architecture. Its main goal was to allow decomposition based on logical behavior, thus it was mainly meant to deal with the problem of *monolithic meta-architectures.* CodA eliminated this by using multiple meta-objects, each one describing a single small behavioral aspect, instead of one large object describing many aspects of an objects behavior. For example if *distribution* is a behavioral concern, then in CodA, it can be decomposed into smaller aspects like, message sending, receiving, queuing etc.

Open ORB

OpenORB [BCA+2001] is a good example of a full-fledged reflective middleware. It relies on component paradigm and provides reflective features by defining three fundamental concepts: components, interfaces and bindings. Reflective facilities in OpenORB support inspection and dynamic adaptation of multiple aspects of components and bindings [BCB+2002]. These facilities are organized into four meta-models: the interface, architecture, interception and resources meta-model. An excellent and detailed architectural as well as performance analysis of OpenORB has been presented in [LPP+2005].

Interface meta-model provides access to the external view of components and bindings, enabling enumeration of provided interfaces and discovery of new interface definitions. The interception meta-model enables dynamic attachment of interceptors to interfaces which allows insertion of pre- and post- processing functionality. The architecture meta-model provides access to the internal structure of components and bindings, represented as an object graph; it provides operations to retrieve, insert, remove and replace components and explicit bindings, as well as operations to manipulate the local bindings connecting them. The resources meta-model provides access to underlying resources and resource management. Specifically, the meta-model captures diverse types of resources at different levels of abstraction (e.g. buffers, user-level threads, and kernel-level threads), and provides control over the distribution of resources among *tasks*, defined as units of resource allocation. Tasks are invocation sequences that can span multiple components distributed over different address spaces.

DynamicTAO and UIC(LegORB)

DynamicTAO [KRL+2000] is a reflective ORB built as an extension of TAO [SLM1998]. TAO is a modular and configurable middleware platform based on design patterns. TAO uses the strategy design pattern [GHJ+1995] to encapsulate different aspects of the middleware implementation and provide reconfigurability. However, TAO itself is not reflective in nature and is aimed at static hard real-time systems (avionics in particular) and does not provide sufficient support for on-the-fly configuration of strategies, once the ORB has been statically configured. DynamicTAO on the other hand is a reflective ORB and allows inspection and reconfiguration of its internal engine. This is achieved by exporting an interface for transferring components across the distributed system, loading and running modules into the ORB runtime and inspecting and modifying the ORB configuration state.

The Universally Interoperable Core (UIC) , (previously known as LegORB [RMK+2000]) is a reflective ORB targeting environments with limited resources, such as handheld devices. LegORB adopts a microkernel-like approach, where the core contains only the low-level essential components. The application programmer implements customized policies, or selects them from a collection of policies available with the ORB package (marshaling, demarshaling, specific GIOP implementations, etc.). UIC defines a skeleton of abstract components that encapsulates standard functional aspects of ORBs (e.g. marshalling strategies and concurrency policies), and it can be specialized to form

different personalities (e.g. CORBA client-side personality or Java RMI personality). Specialization in UIC involves developing concrete components that conform to the abstract components and inserting them into the skeleton structure [LPP+2005].

2K

2K [KRC+2000] is an operating system that incorporates most of the dynamic reconfiguration functionality of middleware. On top of the 2K microkernel, it uses dynamicTAO and LegORB (renamed as UIC), as reflective ORBs. In 2K, resource management responsibilities of the operating system supplemented with algorithms for QoS provisioning, including admission control, negotiation, reservation and regeneration. The application programmers can, therefore, access the system's dynamic state and can implement application-specific adaptations, while the system guarantees that the QoS is preserved. The services provided include standard CORBA services (e.g. naming, trading, and security service) as well as services for automatic configuration, resource management, and code distribution.

Multe-Orb

Multe-ORB [EKP+2000; KKP2001] is a reflective multimedia object request broker suited to Low Latency, High Throughput Environments. It handles application QoS requirements, by supporting QoS specification on CORBA binding and protocol level. Reflection is achieved by reifying the binding compositions and allowing any component to be inspected, thus manipulating the object graph by inserting, removing or replacing individual components using meta object protocols. This concept of open bindings is similar to the one introduced in [BCD+1997]. Application specific QoS parameters are instantiated as a set of protocol modules that collectively achieve the required QoS goals. Multe-Orb aims at integrating an end-to-end QoS solution with a standardized ORB in the least intrusive way. However, provisioning of QoS is supported only for CORBA request-reply invocations.

K-Components

K-Components [DC2001; JC2001; JV2004] is an implementation of a dynamically adaptable architectural meta-model. It is designed to build dynamically adaptable software architectures whose configuration is stored as a typed connected graph, where

the vertices are interfaces that are labeled with the component instances that implement them. Interfaces are connected by directed edges labeled with connector properties, which represent the reconfigurable properties of the connector such as the ability to change its communication protocol, etc. The entry point in the program is represented as the root of the vertex.

The graph is automatically generated from the component definitions and the actual implementation code. It is stored and managed by a meta-level component called the configuration manager. Dynamic reconfiguration is achieved using reflective code called *adaptation contracts.* These adaptation contracts specify conditional transformations based on architectural constraints. They are implemented as metalevel objects that can be loaded and unloaded at run-time using the configuration manager. The integrity of the system is maintained by the configuration rules specified on the edges of the graph and by a reconfiguration protocol that ensures that vertices involved in the reconfiguration are in a safe state. Further adaptation contracts also take care of the management of incoming and outgoing dependencies of the system. The adaptation code is kept separate from the computational code by using a special Adaptation Contract Description Language (ACDL) to specify this code. The K-Components system implements components with an architecture metamodel, and adaptations contracts to support reconfiguration.

Quartz

Quartz [SC2000] defines an architecture that provides support for quality of service (QoS) specification and enforcement in heterogeneous distributed computing systems. Applications requiring QoS enforcement use the mechanisms provided by Quartz to specify their requirements. In order to enforce the required QoS, Quartz employs the resource reservation protocols available in the target network and operating system. Quartz defines two main levels of abstraction for QoS specification, the application level and the system level. A Translation Unit that is part of the middleware architecture is responsible for translating the application defined QoS specification to a set of parameters corresponding to the available protocol of the hosting platform.

The central component in Quartz is the QoS agent, that is composed of the Translation Unit and multiple System Agents associated with the reservation protocols responsible for administering the use of the available resources. Quartz provides no support for extending binding types. As it is not a generic middleware architecture, but rather a QoS

architecture. Quartz provides extensive infrastructure for dealing with resource adaptation and reservation. Adaptation rules can be specified at the system and the application levels. At the system level, the System Agent is used to adapt to environmental condition including network or host resources. The System Agent is also responsible for monitoring of the resources that are occupied by the reservation protocol it corresponds to. At the application layer, Quartz employs a declarative attribute-value based syntax for resource reservation. Application adaptation is provided in the form of call-backs from the middleware. Applications are notified only in case the middleware is unable to provide the resources originally requested.

2.4.1.2 Aspect Oriented Middleware and AoP Frameworks

A number of systems, including our implementation presented in this report have often combined reflection and AoP together. While application of reflective techniques eases the reconfiguration of middleware, application of aspect oriented programming to develop middleware aids in modularizing the cross cutting concerns in the middleware layer. System wide concerns like persistence, transactional communication, security, QoS, and synchronization cannot be easily modularized and the code for handling them is often spread across (rather entangled in) different modules. Therefore, many research and commercial middleware has chosen the middleware layer as the right place to use Aspect Oriented Programming for middleware adaptation.

Since the very fundamental requirements were put forward in the discipline of software engineering, *cohesion* and *coupling* were defined. Well designed software must show a high degree of cohesion while minimizing the inter-module coupling. In the domain of middleware, a key challenge is the achievement of accurate modularization at the level of objects, components, agents etc. However, this is not a straight forward task and involves many cycles of re-factoring during the development phases. A large number of research projects have applied AoP successfully, to modularize systemic concerns in the middleware layer, producing various aspect oriented middleware. Among them, a broader classification can be done with respect to the application of AoP in their development;

- those systems which pre-existed and have been extended (or made domain specific), for example those applying AoP to existing CORBA, CORBA compliant or Corba Component Model (CCM) compliant by using AoP.

- those which have been engineered from the ground up to benefit from AoP. This mainly includes the systems where CBSE has been combined with AoP while designing the system or AoP has been used to custom-tailor the system to pervasive computing needs.

Both these types of systems have been reviewed in the following subsections:

2.4.1.2.1 Aspect Oriented QoS Extension of CORBA and CORBA Compliant Systems

Early works with respect to middleware aspectization have been focused on extending CORBA with aspects hence providing some QoS support. This type of work has mostly been focused on customizing existing middleware to support real-time and real-time embedded systems. Early examples include: The Quality Objects (QuO) framework [ZBS1997], which addresses the issue of QoS using CORBA by extending the CORBA-IDL with a Quality Description Language (QDL). QDL is actually a set of Quality Description Languages [LBS+1998], which are used to specify possible QoS states, the system resources and mechanisms for measuring and controlling QoS, and behavior for adapting to changing levels of available QoS at runtime. QuO supports QoS at COBRA Object level, by allowing the user to specify an application's expected usage patterns and QoS requirements for a connection to an object. The QoS usage specification is at the object level (like methods per second) and not at the communication level (like bits per second). Thus an object may have several connections to the same object, each with different system properties. [DLS+2004] shows development of adaptive distributed applications using QuO framework.

In [HCG2001], *AspectJ* has been used to incrementally add adaptation features to existing ACE-ORB(TAO) [Schmidt1998] middleware, by using their *AspectIDL* on the java real time event channel. AspectIX [HBG+1998] is a CORBA compliant ORB and supports QoS management on per-service basis, by using fragmented objects called Qoslets, which travel from the server to the client and cooperate to achieve end-to-end QoS. Management of Adaptive QoS enabled Services (MAQS) [BG1997a; GB2001] is a framework developed using MICO (an open source CORBA compliant ORB) [RP1997] and QoS IDL (QIDL). QIDL [BG1998] is an extension of the OMG IDL with QoS definitions. A distinguishing benefit of this approach is that unlike the BBN's QuO, QIDL does not introduce a new language for writing aspects and does not use a separate aspect weaver, instead, it implicitly extends the existing IDL with the notion of QoS.

2.4.1.2.2 Component Based Aspect Oriented Frameworks

The related work discussed in this section contains those examples where AoP has been applied to component based systems. Various projects mentioned below mainly differ from each other in their join point model (invasive or non-invasive) along with certain features specific to each of them. Invasive Dynamic AOP breaks the component architecture by weaving code within the base component implementation, i.e. behind the interface contracts, whereas non-invasive approaches utilize the component interfaces as point-cuts, and hence these aspects are implemented as interceptors on the interfaces. The former approach tends to rely on code re-writing techniques, such as bye-code rewriting as supported by tools such as Javassist. In contrast, non-invasive approaches tend to rely on behavioral reflection mechanisms such as interception to dynamically introduce or remove aspects [Blair].

JBoss AOP

JBoss AOP is a Java based aspect oriented framework. Aspects and other constructs are written using java and which are then bound with the applications using java 5.0 annotations or XML. Point cut and advice bindings are resolved at runtime. Jboss AOP is normally used together with the JBoss Application server. Its microkernel layer delivers light weight component model and for the same reason it has found place in various embedded systems. The join point model of JBoss is Invasive.

Lasagne

Lasagne [TVJ+2001] is an Aspect-Oriented Middleware for Context-Sensitive and Dynamic Customization of Distributed Services. In Lasagne, aspects are woven non-invasively at system runtime and the selection of the aspects to be composed is context sensitive. These are the two features which make it more dynamic in comparison with other examples in the same class. The aspect oriented approach of Lasagne is based upon extensions, where an extension encapsulates a slice of behavior that updates multiple components at the same time. For example an authentication extension may crosscut a number of components involved in a client-server request. A service will have some contextual properties attached to it. Interceptors attached to the components of the middleware inspect the values of those properties and decide which extensions to execute. In Lasagne aspects are woven/unwoven dynamically and this feature is similar

to the JAC implementation (discussed next). The dynamicity is achieved by policy selection on the client. However, in case of Lasagne the distribution mechanism is provided by the regular ORB, hence distribution as an aspect would be relatively difficult to implement.

JAC

Java Aspect Components (JAC) [PSD+2004] is an open source middleware project at Object Web [JAC], and is aimed at developing Aspect Oriented Middleware. It has been written in Java and provides dynamic AoP features. It relies on the use of containers, which are similar to J2EE. The two core mechanisms that JAC relies upon are in order to extend the application semantics, are dynamic wrappers and meta-model annotations. Through the configuration interfaces provided by them, new aspects can be integrated with running applications. The framework also facilitates, the use of distributed point-cuts, thus enabling cross-cutting structures which are not in the same host. There is a library of pre-defined aspects in JAC, which can be used by the application programmers or alternatively, new aspects can be defined. Among the predefined aspects are; session, persistence, transaction, deployment, GUI, authentication, caching, integrity and consistency aspects.

The distribution related, predefined aspects are a special feature of this framework. For code instrumentation, JAC is dependent on BCEL, which wraps the classes at load time and then aspect components are instantiated later on. Aspects can be woven and unwoven during the application runtime. The core distribution mechanism in JAC is based upon two kinds of application components; a deployment aspect which is used to create a distributed application and a set of distributed aspects which implement distributed protocols .

The key concept introduced by JAC is the notion of aspect component. An aspect component is the software entity that captures a crosscutting concern. Due to this feature, JAC is very powerful in applying system-wide concerns, since the all parts of a distributed application automatically get updated. To achieve this, JAC comes with a container mechanism. The containers host both business objects and aspect component instance. They are remotely accessible using either CORBA or Java RMI.

Prisma

Prisma [APC+2004; ASJ+2003; PRJ+2003] is another attempt to combine the benefits of component based software engineering (CBSE) with aspect oriented software development (AOSD). A unique feature of PRISMA is that it does not have the notion of the base program to which aspects may be woven, instead, functionality is considered as another aspect. Thus in this case aspects are woven together. PRISMA can be used as a framework to evolve architecture of complex information systems. It relies on requirements-driven evolution, which is supported by means of a meta-level and the reflexive properties of PRISMA Architecture Description Language (ADL), which have been implemented as a middleware. PRISMA specifies different characteristics (like distribution, safety, context-awareness, coordination etc) of an architectural element (e.g, a component or connector) using aspects, and its architecture is evolved at meta-level using the specified properties. In PRISMA the distribution aspect has to be added to the set of aspects types of a conceptual model in order to enable the specification of software architectures of distributed systems. The distribution aspect specifies the features and strategies that manage the dynamic location of instances of architectural elements in a software architecture. The distribution aspect deals with all the properties related to distribution and changes in location. Each architectural element with a distribution aspect must have a location.

PRISMA is an architectural model that can be used to describe the architecture of software applications based on components and aspects. Applications designed with PRISMA have to be implemented using different kinds of object-oriented and/or aspect-oriented languages.

PROSE and MIDAS

MIDAS (MIDleware Adaptive Services) [FPA2003; PAG2003] is a system based on PROSE (PROgrammable Service Extension) that allows applications to self-organize into spontaneous information systems, but without relying on a fixed infrastructure. MIDAS is a middleware layer for adaptive services, and Prose is the language providing the facilities required by MIDAS. MIDAS is based on the *spontaneous container* concept. A spontaneous container is a container that adapts computer appliances to the environment where they are being executed. It works dynamically for entire service communities, which are built dynamically, using dynamic service discovery. Although in this

dissertation, it has been classified from the point of view of its architecture and placed under this sub-section, with respect to its features it can also be classified under middleware for pervasive systems. MIDAS does not have special features furthermore than exposed in EJB/Jini, which is the middleware layer over which MIDAS is constructed.. PROSE/MIDAS is designed for Java, and to be used over EJB or Jini. They offer a dynamic weaving mechanism based on a JITcompiler.

JAsCo

JAsCo [SVJ2003] is basically an aspect oriented language, combining the component based design concepts with aspect oriented programming. It introduces the notions of an *aspect-bean* and a *connector*. An aspect-bean is like a normal java bean, however it contains extra code to realize a particular behavior of a component. It also specifies a deployment hook. The connectors are used to deploy one or more hooks in a specific context (the so called traps). These concepts of hooks, traps and aspect beans are functionally very similar to the concepts of joinpoint, point and an aspect, introduced earlier, except that JAsCo uses them with components. The traps should be known before the execution is started, which makes the approach a bit limited, however, if it is used together with JAsCo HotSwap-2 (which requires JVM1.5+), runtime insertion and removal of traps is supported, which overcomes its limitations and makes the approach more dynamic.

Jadabs

Jadabs [AG2005] is a light weight middleware for pervasive computing, based on service oriented architecture (SOA). It describes applications by annotating components and services with metadata. By using this metadata, new dependencies can be evaluated at runtime to adapt the application with extensions provided or required by the environment. Jadabs can use different underlying network technologies and enables communication in centralized as well as decentralized environments by using peer-to-peer communication paradigm. The unique feature of Jadabs is that it offers both *segmented containers*, suitable for powerful devices like laptops and *monolithic containers*, which are somewhat limited in their support, but suitable for small devices like mobile phones. Jadabs makes use of PROSE for segmented containers and Nanning [NAN] for monolithic containers for aspect weaving.

2.5 Summary of the Related Work and Limitations

Adaptive mobile applications are generally built using two approaches; either the adaptation is performed by the system which underpins the application or the application itself monitors and adapts to change [ECD+2001]. These two approaches differ from each other in the levels of efficiency and flexibility. Programming adaptive behaviors into applications is more efficient, but is practically very cumbersome, it is difficult for the application designer to predict the runtime behavior of an application and in particular, to estimate all possible scenarios which can occur at runtime on a system-wide basis [KF2005]. Handling the adaptation responsibility to the underlying system (usually middleware) is less efficient but more flexible, since the application code does not need to explicitly handle adaptive behaviors. Most of the recent research has benefited from the latter approach by using different techniques to develop QoS middleware, among which are included Meta Object Protocols (MOPS) [KR1991] for separation of concerns and behaviors, reflection and Aspect oriented Programming. A wealth of research is available in literature, mainly in the area of adaptive, reflective and in particular aspect oriented middleware, where specific languages for describing QoS aspects have been developed, and existing middleware has been custom tailored to provide QoS. A comprehensive comparison of different aspect weavers and aspect oriented middleware is provided in [LPP+2005]. While different aspect oriented middleware systems have their own pros and cons, a commonality found in them is the assumption that, the applications running on top of such systems are using the API's provided by the middleware. The systems based on this assumption give the benefit of interoperability, and will generally be capable of adapting a wide range of applications, if aspects are directly woven into the application code, this will be more efficient and will be feasible if adaptation is required for a specific class of applications and can be used to transform non-adaptive applications into adaptive ones. An example of this is found in TRAP/J project [SMC+2004]. While TRAP/J can be used as a tool to make applications adaptive, this requires selecting classes to be adapted and defining adaptive behaviors for them.

Adaptation, in all its forms (software, resource or service), affects multiple elements of a system and has been identified as a cross-cutting concern, particularly in the context of pervasive computing [RK2004]. In case of wireless networks, the communication problems are even more complicated due to the fact that not only the application demands remain fluctuating, the available resources like the network bandwidth, in particular, are also neither constant nor precisely known at any instant.

Reflective implementations of middleware depending upon the component model suffer from an inherent property of the component model, that is, since reflective facilities are indispensable from the component model (because all components maintain meta-information about themselves, which is discovered through reflection to facilitate plugability), the associated overhead scales up with the number and size of components in a system. Therefore, on resource constrained devices in particular, it is not feasible to keep a general purpose middleware layer, and customize it when required. In some cases customization of a large middleware to specific environment can prove complicated. This suggests the need for domain-specific middleware.

Thus, despite being beneficial in comparison with the classical middleware technologies, effectiveness of reflective middleware is only proven for the middleware developers, because, they can develop a general purpose reflective middleware once and then customize it using reflection, to market it faster. Thus applying reflection to engineer middleware itself can be a good example of software reusability, but, its use to application programmers is limited.

The common factor of all the existing approaches is the inherent idea of providing the user with an API, using which the user can design applications on top of it, which can then benefit from the underlying middleware features. The work presented in this report is based on the existing concept of separation of concerns and behaviors, with the main difference that we are implementing it in more autonomous fashion. Our approach can be seen as an example of the same principles, however, since it is in the context of a framework (Java Media Framework : JMF), where the structure of applications is pre-defined, it gives us the benefit of specifying adaptive concerns at a higher level of abstraction and system-wide manner in the form of profiles and makes the adaptations more self-managed, at the cost of limiting the applicability to a specific framework.

Chapter 3

Aspect-Oriented Model for Adaptive Code Generation

3.1 Introduction

A wealth of research has gone into developing and deploying various strategies for providing Quality of Service (QoS). On one hand are the approaches providing guaranteed QoS for non-elastic applications and on the other hand are those attempting to adapt the resource usage to whatever is available, applicable in case of elastic applications. In the networks community, adaptation refers to altering the transmission of data by different means like compression, transcoding or adaptive routing, resulting in *adaptive behavior of data flow*. In the software engineering community, adaptation research is concentrated in adapting the software itself to enhance its functionality at runtime (hot-fixing; updating application components without taking the application offline), facilitating content or service adaptation, resulting from *adaptive code modifications*. Despite seemingly being two different research directions, it is possible to combine these two approaches, in such a way that for any multimedia application, *adaptive behavior of the multimedia data stream* can by achieved through *adaptive modification of that application's code*. Therefore, software abstractions and techniques used to map adaptation requirements for any distributed computing software can be enhanced to be applied in the context of multimedia frameworks, which are used to develop multimedia applications.

This chapter presents a conceptual model based on state machines and state sets, and maps adaptation requirements given at a higher level of abstraction onto the application

target code using Aspect Oriented Programming techniques. The model developed in this research relies on four constituents:

- A System State Machine
- An Application State Set
- A Profile State Set
- A Resultant State Machine

These are the main components of the model and are described in detail in the following sections.

3.2 System State Machine

The purpose of the System State Machine is to keep track of past resource availability, present state and future resource availability prediction. When the application starts, the state of the resource availability represents the default (or current) state of the systems. At any given time, the System State Machine consists of the default state/reference state (representing current resource levels) and different other states, each of which can be characterized by a unique combination of resource levels. An example of System State Machine is shown in the figure below:

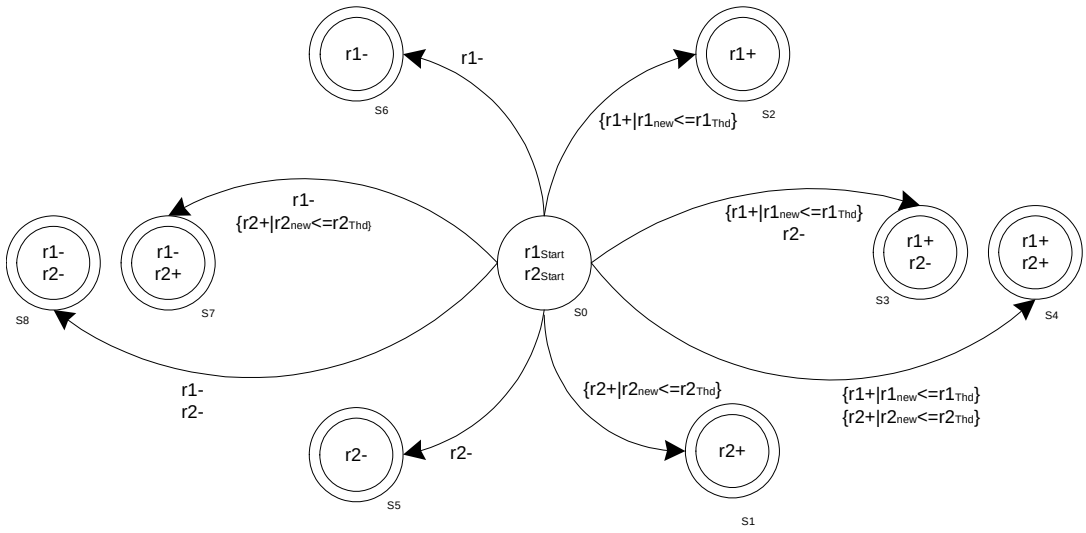


Fig. 3.1 – A System State Machine with arbitrarily chosen state names, showing all possible states

Fig 3.1 represents a System State Machine, with two resources. Possible combinations of these variations have been shown in the arbitrarily named states, S1 to S8, while S0 is the default (current) or start state. Taking this current state as the reference, all other states have been marked with resource name, followed by a '+' or a '-' sign, indicating an increase or a decrease in the resource. Thus, 'r1+' inside a state bubble means that resource r1 will increase as compared to the current state. Similar meanings can be attached to other symbols. The conditions on the edges, indicate restrictions on switching to any terminal state, e.g., $\{r1+|r1_{new}<=r1_{Thd}\}$ on the edge from S0 to S2 indicates that the System State Machine can switch to state S2 only if there is an increase in the level of resource r1, such that the new level (or the increased level) of r1 is less than or equal to the threshold level of r1. A threshold level of any resource is that level beyond which the system is likely to show some unpredictable behavior at any time (e.g., if 95% of the peak CPU availability is set as the CPU threshold, then it simply means, that more than 95% CPU load is not desirable and since it may make the system unstable). Similarly, in case of available bandwidth, it may be defined as 3% packet loss, (meaning that a packet loss higher than that is not tolerable etc). A state that is labeled with an increase (or a decrease) in the resource level, only tells that it is reached when that specific resource increases (or decreases); it does not define new states on the basis of the magnitude of increase or decrease. Due to this reason, at any point in time, any terminal state of the machine is reached in single transition.

Once the System State Machine switches from one state to another, the new state to which the machine transitioned, now becomes the current (or reference) state. The resource increase and/or decrease possibilities are then evaluated again and a new System State Machine is constructed. With the new machine, some of the existing states may no longer be available, some new states may become available, which did not exist in the previous state machine, or the new state machine may consist of exactly the same states as the previous machine (this case is easily understandable if we keep in mind that a state only shows *the type of resource variations* which led to that state *not the magnitude of resource variations*). Due to this reason, the state machine does not show any transition back to the start state. An example of state transition, where the System State Machine switches to a new state is shown in the figure 3.2a,b,c.

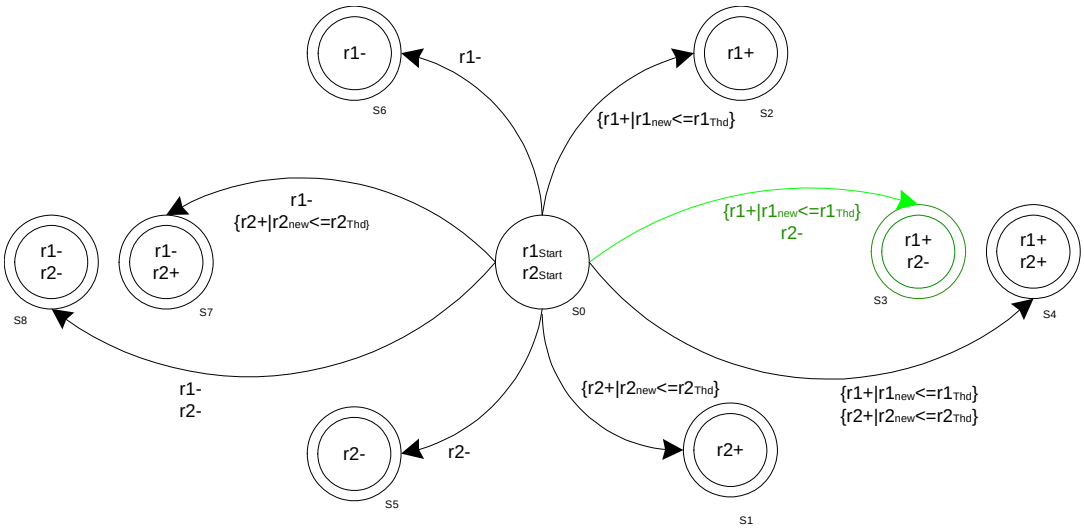


Fig 3.2a – First transition of System State Machine

Assuming that the System State Machine takes the transition from state S0 to S3, as shown in the diagram above, then, as soon as the transition is complete, a new state machine is constructed. This transition will occur only when utilization of resource r1 increases and that of r2 decreases at the same time.

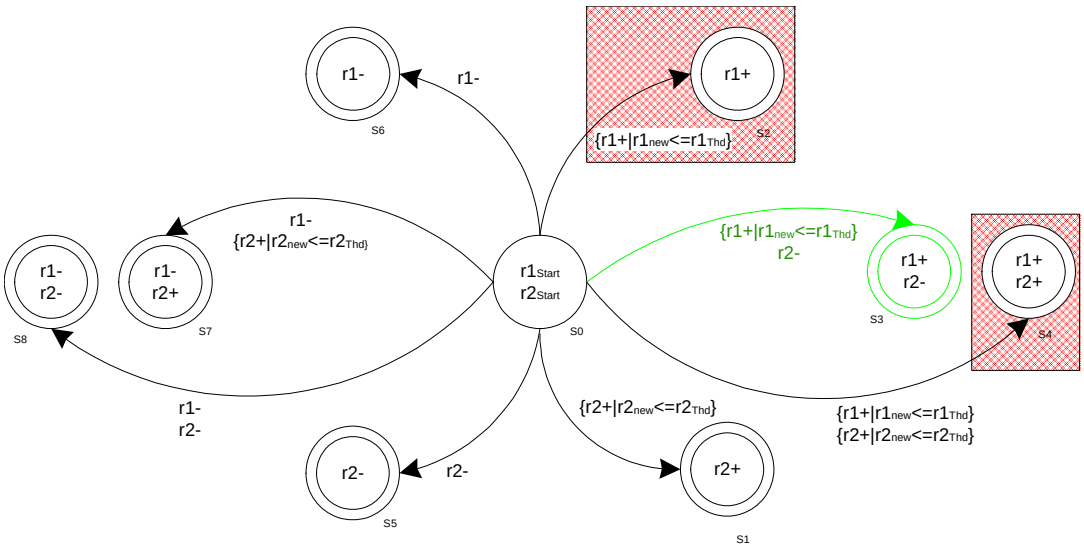


Fig 3.2b – System state machine taking a transition pushing resource limits to threshold level.

If the increase in r1 is enough to push it to the threshold level, then the new System State Machine will be as shown in figure 3.2b. Now all other states (i.e., S2 and S4) which could have been reached due to increase in the consumption of r1, will not be available, therefore they have been shown masked (shown in pink rectangles) in figure 3.2b. Therefore, after this transition, the new System State Machine will be as shown in figure 3.2c.

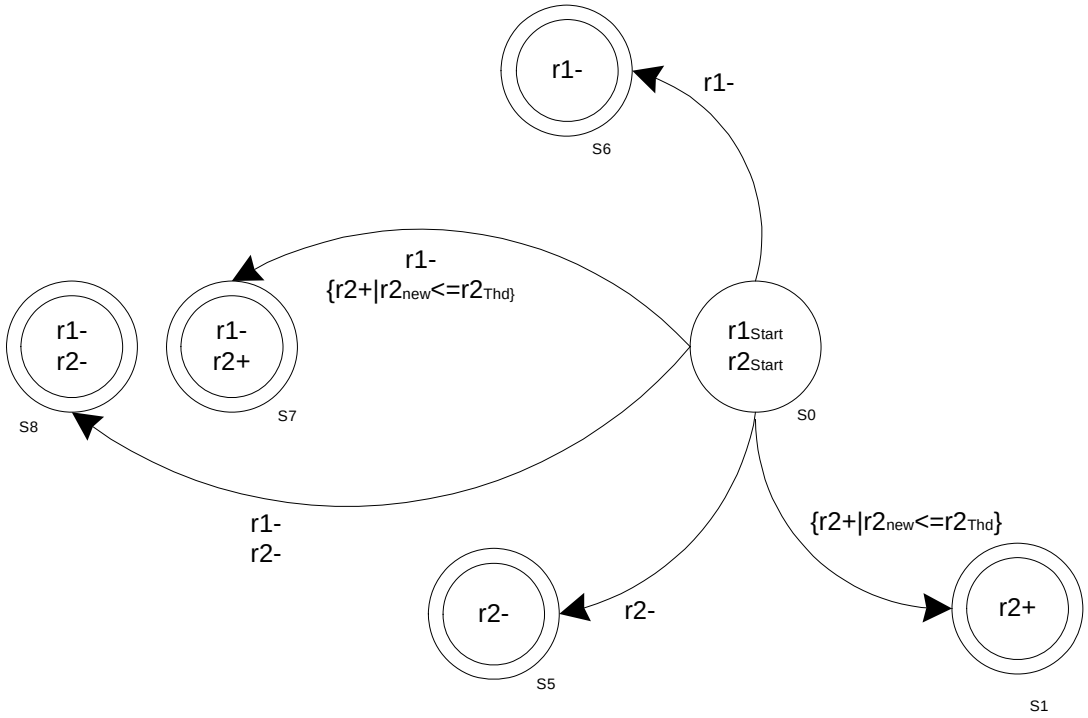


Fig 3.2c – Resulting System State Machine upon completion of transition from S0 to S3

It is notable that the System State Machine resulting upon the completion of transition has the current (or default/reference state) S0, which was S3 in the actual machine before the transition (fig. 3.2a).

In this example, a System State Machine has been shown with combinations of only two resources, in practice however, the resources are more than that and the state machine is correspondingly complex. In practical implementation of this model, switching of states can occur as a result of resource fluctuation or when the application switches from one mode to another. For example, when the application switches from one state to another to realize a video format change, it may overburden some resource,

due to which in the new application execution state, some system resource states may not be available, meaning that the number of possible transitions from this new state are somewhat limited or it may happen that new resource combinations (states) become available as a result of positive resource fluctuation or as a result of the application state switching. Thus a System State Machine keeps changing throughout the execution time.

3.3 Application State Set

When a multimedia application is running, it has some characteristics with respect to the multimedia stream. For example it can have a specific bit rate, frame rate, play-out buffer size etc. Different elements like codecs can have further tunable parameters, e.g., video size may also be variable etc. At any particular point in time, therefore, a multimedia application can also be given a characteristic representation with realizable combinations of such parameters.

A collection of all valid combinations of such characterizing parameters can therefore, define an application state set. As a simple example, let us assume, *videosize* and *frame-rate* be the only two characterizing parameters for an H.263 application, denoted by *s* and *f* respectively. Also assume that the application supports QCIF and SQCIF sizes and frame rates of 5 frames per second and 10 frames per second. Different valid combinations of the two parameters (size and f-rate) will be.

$S_{a1} = f(5), s(SQCIF)$ $\qquad$ $S_{a2} = f(10), s(SQCIF)$

$S_{a3} = f(5), s(QCIF)$ $\qquad$ $S_{a4} = f(10), s(QCIF)$

where S_{ai}, $1 <= i <= 4$, denotes the *i*th element (state) in application state set.

Or the set of application states can be written as:

$S_{app} = \{S_{a1}, S_{a2}, S_{a3}, S_{a4}\}$

Thus, a set of application states as shown above identifies different modes of execution, to which the given application can switch. Each of these states has associated resource requirements. In case of applications which show some degree of adaptability by default, the application state set consists of more than one element, while in the applications which are non-adaptable by default, the application state set will consist of only one element. In such cases, where the application is either not adaptable by default and/or extension of its default adaptability is desired, changes to application code are

necessary, such that the application state set has mode elements (states) to which the application can switch. (Details of application code manipulation are given in chapters 4 and 5).

3.4 Profile State Set

Since an adaptive application has to fulfill user demands of adaptive behavior, its execution is dependent on different user preferences. Therefore, similar to an application state set a profile state set is defined. Each element of the profile state set defines a state which consists of a combination of properties that the user wants the application to satisfy. These states defined by the user have to be specified in some preferential order[1]. For example, a user may like an H.263 video to be in one of the states given below.

$S_{u1} = f(5), s(CIF)$ $S_{u2} = f(10), s(CIF)$ $S_{u3} = f(15), s(CIF)$

$S_{u4} = f(5), s(QCIF)$ $S_{u5} = f(10), s(QCIF)$ $S_{u6} = f(15), s(QCIF)$

where *f* and *s* denote frame-rate and video size respectively and

S_{ui}, $1 <= i <= 6$, denotes the *i*th element (state) in user's profile state set.

Then the user's profile state set can be written as: $S_{prof} = \{S_{u1}, S_{u2}, S_{u3}, S_{u4}, S_{u5}, S_{u6}\}$

The profile state set for any user must specify all the states, which are demanded from the application during its entire life time. It should also be noted that the parameters, defining different states exist in groups, and each group varies from the other most of the

[1] Since, sets do not offer sufficient means to arrange elements in a 'preferred order', more details on this specification of preferences will be presented in chapter 5, using XML examples.

time, not only in the values of these parameter but also in the number of parameters. For example, if in addition to H.263, MJPEG adaptations are also needed, then those options must also be added to the profile state set. In this case, size parameter for H.263 and MJPEG will be different. For all the states contained in the profile state set, the preferences defined by the user along with the resource costs, a state machine can be constructed (as explained in the next section with implementation details in chapter 4 and 5).

3.5 Realizable State Machine

As described above, the application state set and the profile state set, both are related to the states of execution of the application. Therefore, the application state set contains all those states, in which the given application code can execute while the user's profile state set lists all the states in which the user wishes the application to be executable. This possible difference in the states of execution of the given application code and those demanded by the user, is fulfilled by modifying the application code. *A state machine whose states are obtained by intersection of the application state set and the user's profile state set and edges derived from the user preferences, is called the Realizable State Machine.* It indicates the modifications required in the default application code, necessary to achieve the desired adaptive behaviors. Therefore, depending upon the type of elements (definition of states) in the two given sets, there are four possible results of set intersection, and the type of result determines what will be used to derive the edges of the Realizable State Machine. The edges are derived either from the user supplied adaptation preferences or from the adaptation priority tables of the system or using both. The four possible results of set intersection are discussed below.

(i) - $$S_{app} \cap S_{prof} = S_{app} = S_{prof}$$

This is the simplest case. It happens when the application is inherently adaptive and the user demands exactly what is provided by the given application code. In this case, there exists a possibility that the application had only one state (i.e., a non-adaptive application) and the user also does not demand any adaptation. (There is a very little chance of this happening in a practical situation). In such cases, the Realizable State Machine is derived from S_{prof} , by taking all elements as distinct states and the adaptation

priorities configured[2] by the user to form the edges of the machine. This means that even in the cases, where adaptation provided by the application without any code modifications is the same as the user desires, state transitions of the Realizable State Machine are directed by the adaptation configurations given by the user.

(ii) - $S_{app} \bigcap S_{prof} = S_{app}$ and $S_{app} \subset S_{prof}$

Mostly, this is the type of relationship that occurs in practice between what the application offers and what the user demands. It is the case when the application offers some (or only one state of execution at minimum; which is the case of no adaptation). And user demands additional adaptations. In this case too, the Realizable State Machine is derived from S_{prof}, by taking all elements as distinct states and the adaptation priorities configured by the user to form the edges of the machine. The Application State Set and the Profile State Set in section 3.2.2 and 3.2.3 come under this category.

(iii) - $S_{app} \bigcap S_{prof} = S_{prof}$ and $S_{prof} \subset S_{app}$

This is a very rare case, and can occur only when the given application offers a good degree of adaptation and the users demands are less. To construct a Realizable State Machine in this case, for those states which are common to both S_{prof} and S_{app}, the edges are derived from the adaptation configuration supplied by the user and for those states

[2] Configuration of adaptation preferences by the user are discussed in chapter 4 and 5.

which are not common to both the sets, default transition conditions of the application stay untouched.

(iv) - $S_{app} \bigcap S_{prof} = \phi$

This is the situation where both S_{app} and S_{prof} are disjoint sets (they have nothing in common) which means that the adaptation provided by the application (if any) is not needed by the user, but the user has made completely different demands. In this case the Realizable State Machine is constructed entirely from S_{prof} and the edges of the machine are also derived from the adaptation preferences supplied by the user. In this case adaptive code has to be intelligently patched into the supplied application code. The resultant code after the application has undergone this code transformation represents the states of the Realizable State Machine in software and the transitions are implemented as state switching conditions. Since, the adaptations (if any) provided by the initially supplied application code are not needed at all, execution of the supplied code may be masked at different places, using *around advice.* (Detailed explanation is given in chapter 5).

In all the cases of obtaining the realizable machine states through application and user's profile state set intersection, the edges of the machine are derived from the user adaptation preferences, starting from the highest to the lowest. The reverse link in each

case is added automatically between the two states involved in the forward transition. Assuming the user's preferences[3] given below,

$f(15), s(CIF)$; $pref\,1$
$f(10), s(CIF)$; $pref\,2$
$f(5), s(CIF)$; $pref\,3$
$f(15), s(QCIF)$; $pref\,4$
$f(10), s(QCIf)$; $pref\,5$
$f(5), s(QCIF)$; $pref\,6$

Then if the given application is non-adaptive by default (the most frequent case), then the user's state set of section 3.2.3, will be used alone for derivation of the Realizable State Machine as shown in figure 3.3.

[3] In fig 3.3, the edges are marked with the changing conditions. They could have been marked with the preference numbers; like *pref1*, *pref2*, etc. similar to XML implementation files in chapter 5.

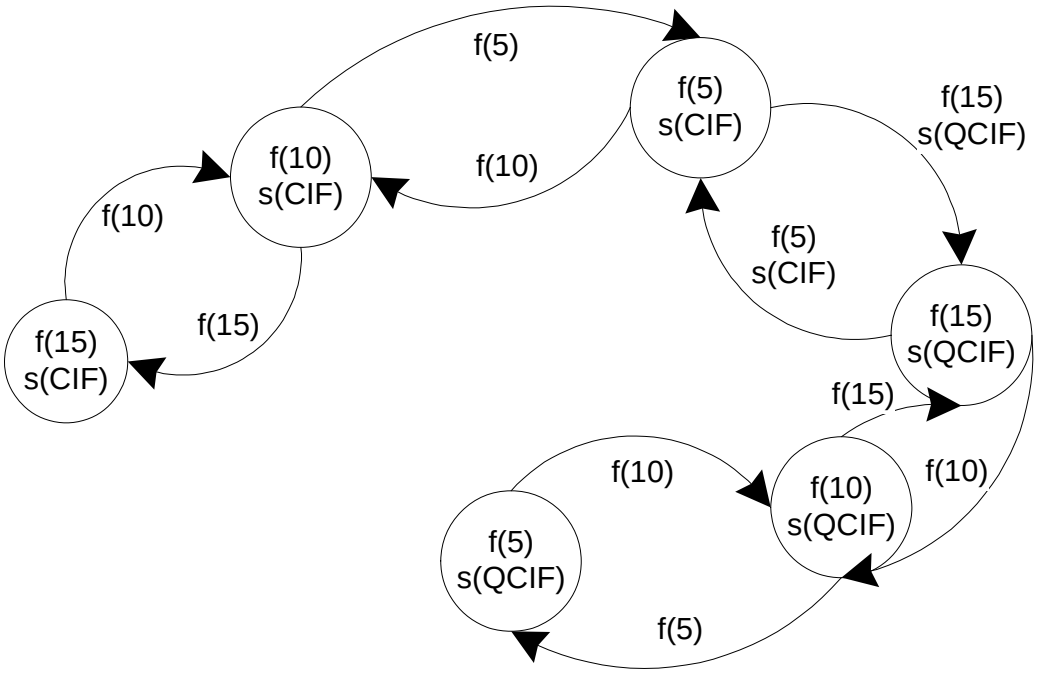


Fig. 3.3 – A sample Realizable State Machine

In practice, user's adaptation preferences can be of greater complexity and so is the corresponding Realizable State Machine.

In cases where the given application shows adaptive characteristics (e.g., the application state set given in section 3.2.2), the intersection of S_{app} and S_{prof} will contain some elements representing the states to which the default application code can switch, therefore, to switch across those states, the transitions for the Realizable State Machine can be derived from the internal adaptation tables. An internal adaptation table lists the effects of changing adaptation parameters upon system resources. An internal adaptation table is shown below.

In order to determine the transitions for the elements coming from the application state set, the priority shown in table 3.1 is used; the lower the number, the higher the priority and vice-versa.

Adaptation	Effect on Network	Effect on Server CPU	Effect on Client CPU	Overhead	RAM (Buffer) Requirement	Battery Life	Impact	Priority
Compression↑	Data Sent↓	Load ↑	Load ↓	High	-	-	System wide	5
Format↔	-	-	-	Medium	-	-	System wide	4
Size↓	Data Sent↓	Load↓	Load ↓	Medium	↓	↑	System wide	3
Pre/ Post Swap ↓	-	-	Load ↓	Medium	-	-	Client side	2
Dimension ↓	-	-	Load ↓	None	↓	↑	Client side	1

Table 3.1 : Adaptation types and their corresponding effects on system resources[4]

3.6 Towards a Practical Model

Previous sections have only described the model itself, without any reference to it the use of software engineering techniques to realize it. This section gives a principle overview of the use of adaptive software engineering techniques and language features which help realize the proposed adaptation model.

Multimedia frameworks (e.g; Java Multimedia Framework, Microsoft Direct Show, largely open source GStreamer etc.) are based on the flow-graph[5] model. Components of

[4] ↑shows increase, ↓shows decrease, ↔ only means change form one to another, - shows not known or negligible change.

[5] While different multimedia frameworks have been using the terms; *filter graph*, *flow graph*, *pipes* etc, a general term *media processing chain* or simply a *processing chain* will be used in this dissertation.

a multimedia framework are characterized by their media processing states, therefore, execution of multimedia applications developed using multimedia frameworks is realized by switching the states of constituent application components. Thus the way an entire application executes is determined by the way its components transition from one state to another, which directly depends upon which events are fired. Use of different media processing elements can be controlled by altering the application code and once desired elements are used, tunable parameters of such elements can be tweaked to gain adaptive performance.

Therefore, a realizable model of adaptive execution (i-e; adaptive processing of multimedia data in the processing chain) is based on the fundamental principle of mapping the states of the Realizable State Machine onto the states of underlying framework components and adaptively controlling the flow of state-transition events. In principle, this task can be decomposed into Static Composition and Dynamic Reconfiguration phases.

Since the Realizable State Machine is the resultant of the application state machine and the user preferences/profiles, its states inherently represent possible adaptive behaviors contained in them. During the Static Composition phase, by using Aspect oriented Programming (AoP), code required to carry-out the state transitions dictated by the Realizable State Machine are woven into those framework components which are used by the application. Thus the given application code is aspectized with several *before*, *after* and *around* advice to facilitate interception, diversion and masking of the events used by the given application code. Additional adaptive code is also hooked with these event interceptions, thus at the application runtime, pre-processing, post-processing or even

completely new code (at certain places) is executed in response to different state transitions. Thus, in accordance with the user preferences/ profiles, the media processing chain generated may be partially or completely different from the original applications' media processing chain. In practical situations, instead of one chain, several chains exist, each with its own characteristic media processing capabilities and associated resource demands. At application runtime, when a request to adapt to certain situation has to be fulfilled, the chain suited best to that kind of processing is installed (this gives coarse-grain adaptation) and then it the tunable parameters of the media processing elements in that chain are tweaked (this gives fine-grain adaptation). In summary, the default media processing chain of the multimedia framework gets altered[6]. This is shown in figure 3.4 a, b, c and d.

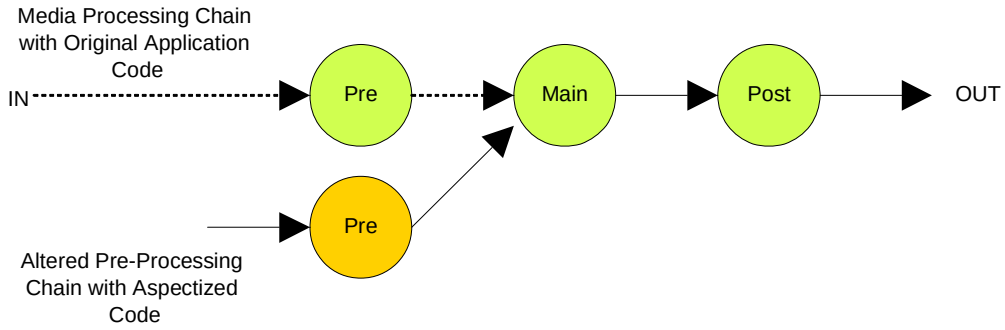


Fig. 3.4a – Altering pre-processing chain of the given application code

[6] It is notable, that the change in the pre, post or main processing chain elements may either result in complete replacement of that element with a new one, or only masking of some methods and/or addition of some others.

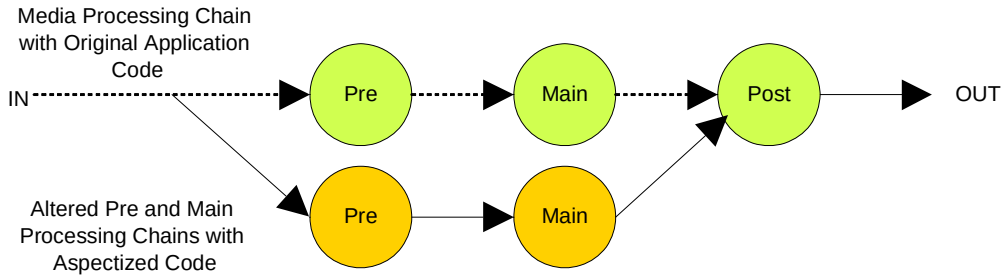


Fig. 3.4b – Altering pre and main processing chains of the given application code

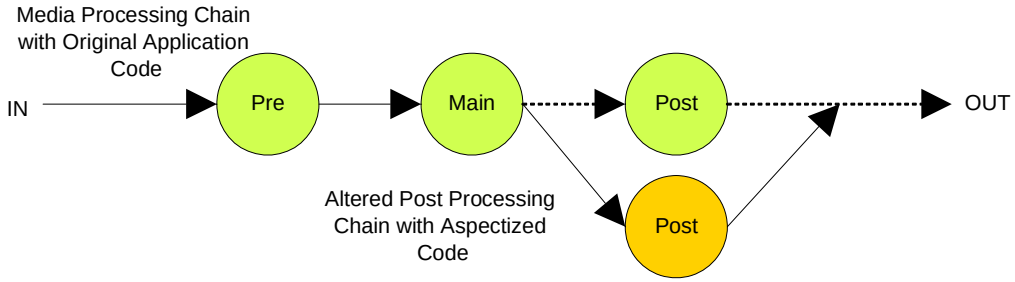


Fig. 3.4c - Altering post processing chain of the given application code

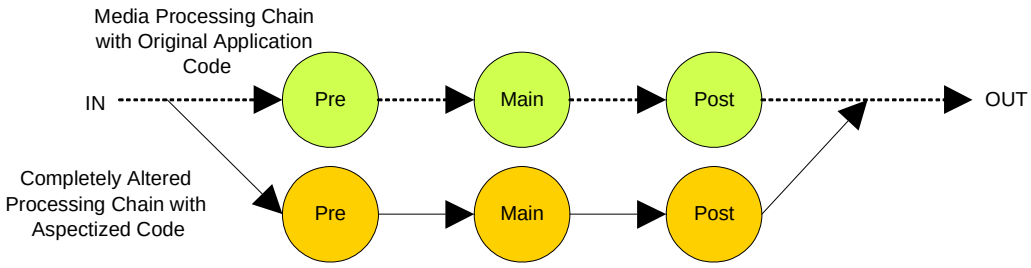


Fig. 3.4d – Altering entire processing chain of the given application

In addition to figure 3.4a,b,c and d, at times more than one media processing chains can exist in parallel to each other and switching from one to another is done at runtime. In all the cases, during the Dynamic Reconfiguration phase, already aspectized application's media processing chain, which is now hooked with an adaptation policy, is directed to fine-tune the parameters associated with each media processing element (like a codec, a renderer, a packetizer etc). It is notable that in addition to the cases shown in figure 3.4

above, there are several other possibilities of code aspectization. In many cases, instead of completely swapping a processing chain component with another one, only modification of some methods can give the desired adaptive behavior, so that only that modification is made to the code without swapping the existing chain with a new one. Subsequent chapters, discuss the theoretical model developed in this chapter from a practical standpoint, with particular reference to Java Media Framework (JMF).

For practical implementation of any adaptation, a number of factors relating its type are to be considered. An adaptation parameter may be device specific (e.g; LCD backlight switching on/off, changing the play-out buffer size on the client etc.) it may be user specific (e.g; each user defining a different profile for the same content type), or instead of determining the local device behavior, it may impact system-wide adaptation decision (e.g; incase of format change, server and client both have to adapt in a coordinated manner; details in sec. 4.4).

Chapter 4

Adaptation Composition and Runtime Environment for Multimedia Applications (ACREMA)

4.1 Introduction

The adaptation model presented in the last chapter can be easily realized in software if the applications are adaptive by default; i.e; if they have been programmed to have modal adaptation characteristics within themselves, however, for non-adaptive applications (those which only have minimal functionality) but need to show adaptive behavior, some architectural abstractions that can transform non-adaptive code to execute in an adaptive manner must be developed. This chapter provides architectural and implementation details of how the adaptation model discussed in the previous chapter can be extended to non-adaptive (legacy) multimedia applications. It also discusses some test applications and how the results can be applied in general. The first section describes media processing elements of the Java Media Framework (JMF), since sample application code from JMF was used to realize the concept for the purpose of testing.

4.2 JMF Media Processing Elements

The JMF API mainly consists of several interfaces and a few classes, which abstract audio and video processing devices used in daily life. Some features of these media processing elements are tunable, while the framework itself is extendable as a whole to incorporate new functionality. The table given below, lists important media processing elements of JMF along with the possible adaptive behaviors that can be attached to each one of them.

Element Name	Possible Adaptation Features
Processor	A Processor is the basic element which can be used in a number of ways, starting from capturing media from input devices, reading from URL or files etc. It can be used to add adaptive behaviors like size of the display picture.
RTPManager	RTPManagers are responsible for RTP session management. Thus any kind of adaptation related to controlling the network response can be done by using adaptive method code to an RTPManager.
Codecs (Several	Of all the elements, codecs are the most versatile for the purpose of tenability. Many different types

Types)	of coarse and fine adaptations are possible on audio and video, including format, frame-rate, bit-rate, encoding quality, and different other codec specific parameters.
Buffer	A playout buffer can be managed adaptively, and may help reduce jitter.
Renderer	A renderer is an abstraction of an output device. Thus several client side adaptations can be realized by adaptively controlling the renderer.
Device Ports	These elements abstract physical device ports.
Effect	It is one of the special plug-ins and by writing custom Effect plug-in, various media processing features of the JMF API can be enhanced, e.g; any kind of manipulation of individual frames or transcoding the input stream etc.
Packetizers	Packetizers are responsible for converting the data generated by the application for transmission over the network (e.g; encoding for RTP transmission). Depacketizers do the reverse.

Table 4.1 –Java Media Framework's media processing elements

The above table shows media processing elements controlling the behavior of a multimedia application. Each application typically uses several of these elements and each element has several versions, each one with a different set of properties (e.g; the same codec by one vendor may behave differently the one from another vendor etc). Different characteristics of each of such elements are controlled by a number of methods, spread across many interfaces/classes. Such methods need to incorporate code changes in order to give adaptive behavior to the entire application. To make the given non-adaptive application, adaptive, code changes have to be made to several methods spread across the entire application code, *adaptation* is therefore considered as a cross-cutting concern in this context. That is why AoP has been used to handle it.

4.3 Architectural Overview of the System Model

The model presented in the previous chapter only describes how user's adaptation requirements and the properties of the given multimedia application code are merged together, giving a theoretical manifestation of the adaptable code. Practically, this process starts with exploring the given application code using *reflection* and exploring the media processing elements present in the application along with methods which can manipulate those media processing elements. The given application code itself resides on top of the ACREMA layer as shown in figure 4.1.

Thus, initially the application in question only consists of its basic functionality, and all the adaptation behaviors are contained inside ACREMA. Transforming a non-adaptive application into an adaptable one is completed in two phases. During the first phase when the application is loaded on top of ACREMA, ACREMA generates code patches according to the Realizable State Machine and weaves them into the given application code at different points. It is notable that these operations are carried out on byte code of

the application (source code is not at all required). This weaving process exports some 'hooks' inside the given application code at well-defined adaptation points.

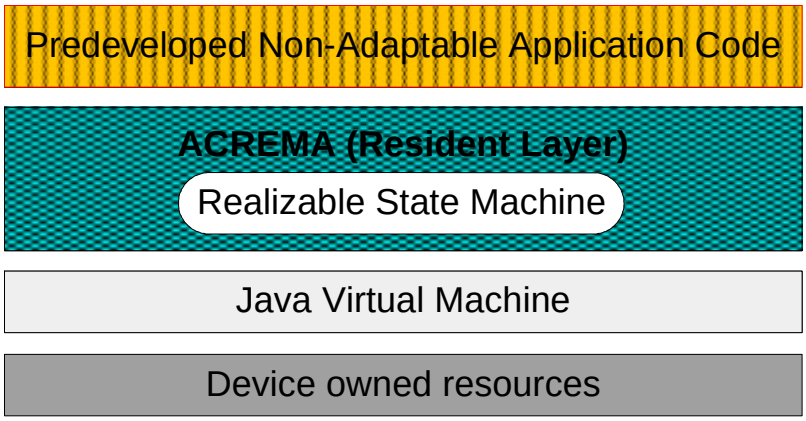


Fig 4.1: Application loaded on top of ACREMA resident layer

These 'hooks' which are now part of the modified application-code on one hand and can be tweaked according to the adaptation policy and on the other hand, can be used to link to adaptive code patches at the application load time or to adaptively divert the flow of control during application runtime. In terms of AOP, this whole process is finding the appropriate join points, composing pointcuts and weaving-in advice code. Figure 4.2 illustrates the concept.

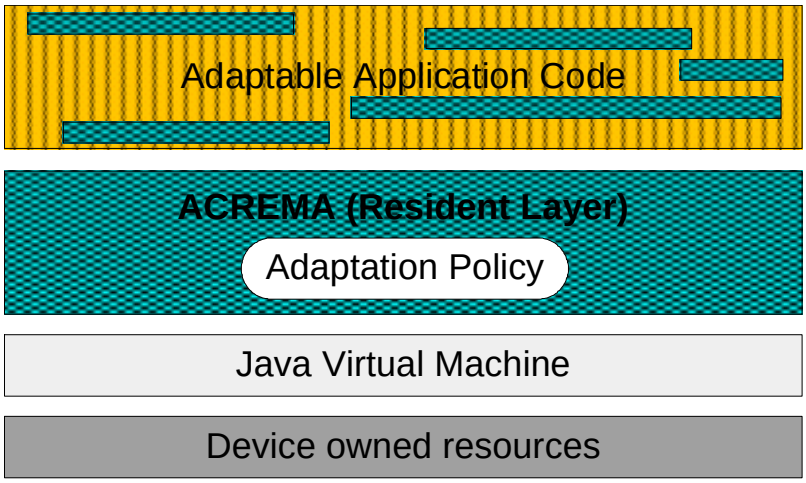


Fig 4.2: Aspectized application code produced after generated adaptation behaviors have been weaved-in.

The given non-adaptive application code now contains adaptation hooks (green patches inside yellow, shown in figure 4.2).

At an abstract level, the entire system consists of two main phases: the *static composition phase*, which is responsible for identifying the appropriate joinpoints in the given application code along with the adaptation advice to be woven at those joinpoints and the *dynamic reconfiguration phase*, which uses the weaved advice code along with different adaptation policies and available resource levels, to realize runtime adaptation. The entities constituting the whole system are shown in the figures 4.3 and 4.4.

4.3.1 Static Composition

The Static Composition, is meant to produce the runtime adaptable code from the given non-adaptable application code. A conceptual overview of the process is shown in figure 4.4. The lower portion of figure 4.4, where the Mapping Interface is shown producing the Realizable State Machine has already been explained in the last chapter, while the upper portion which is transforming the supplied application code according to the Realizable State Machine, using an *aspect engine* is the subject of section 4.5.

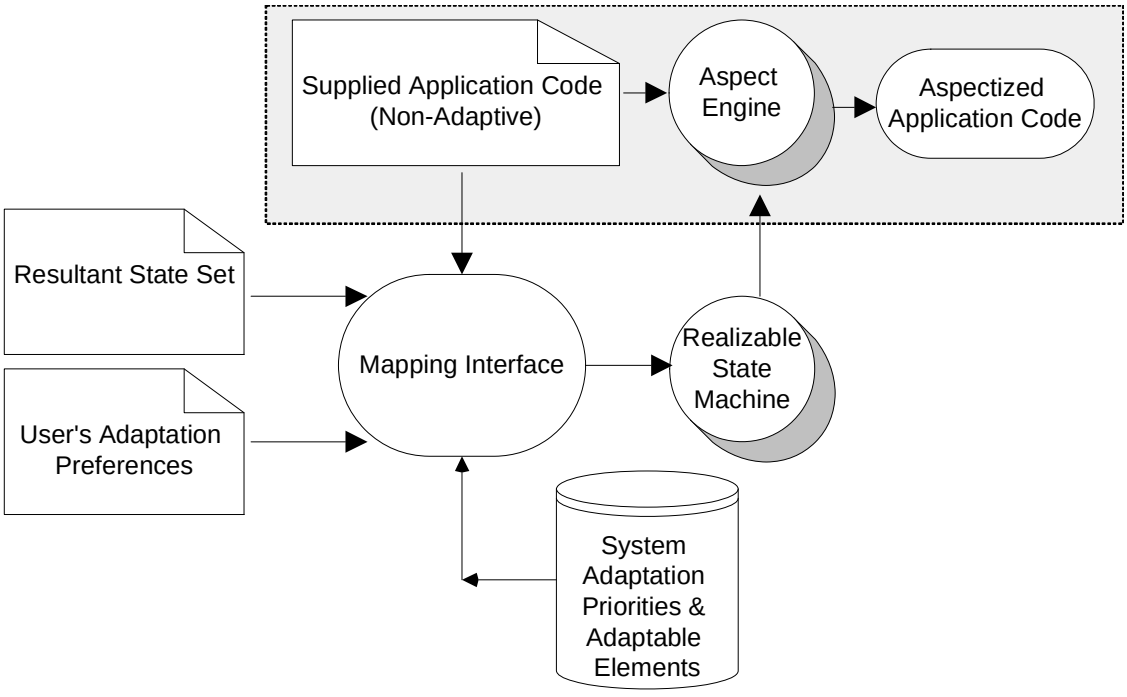


Fig. 4.3 – Static Composition Phase

The Mapping Interface takes a list of adaptable elements of all the system elements available in the JMF API, along with the provided non-adaptive byte code for the application. The application code is analyzed (details in section 4.5) for all possible adaptive behaviors which can be added to the classes and methods used by the given application. The pointcuts are identified at this stage. In addition to specifying the types of adaptive behaviors desired, mostly other information related to the characteristics of the device (e.g., battery and cpu power, cpu and lcd adaptation features, network interfaces etc.) can also be specified. If there are any conflicting (non-realizable) adaptive behaviors in the specification, they are filtered out. The Realizable State Machine produced by the mapping interface is used to weave the adaptive code into the supplied application code, thus transforming it into adaptable code (shown as 'aspectized code' in figure 4.4). Since the aspect engine is a load-time aspect-weaver, the functions of JMF API, used by the user application are aspectized at the application load time. It is notable that due to load-time nature of the weaving process, when the application is loaded, the adaptation hooks are inserted and when the application completes execution and is unloaded, the application code is again the same as before loading. The changes made to the code are not permanent. The aspectized application code is made to divert its flow of control at specified points to the code patches, which give the overall execution an adaptive behavior.

4.3.2 Dynamic Reconfiguration

Dynamic Reconfiguration of the application is done to achieve adaptation at runtime. It is realized by executing the aspectized code produced during the Static Composition phase along with the current input from the System State Machine at any time. Vital resource levels like CPU usage, remaining battery, available network bandwidth and RAM usage are obtained by the resource monitor, along with the pre-programmed resource threshold limits. The aspectized application code, which now has various states of operation is switched from one state to another by the adaptation engine, in accordance with the possible transitions of the System State Machine. The System State Machine, is indirectly used to predict future adaptation possibilities by the Adaptation Engine.

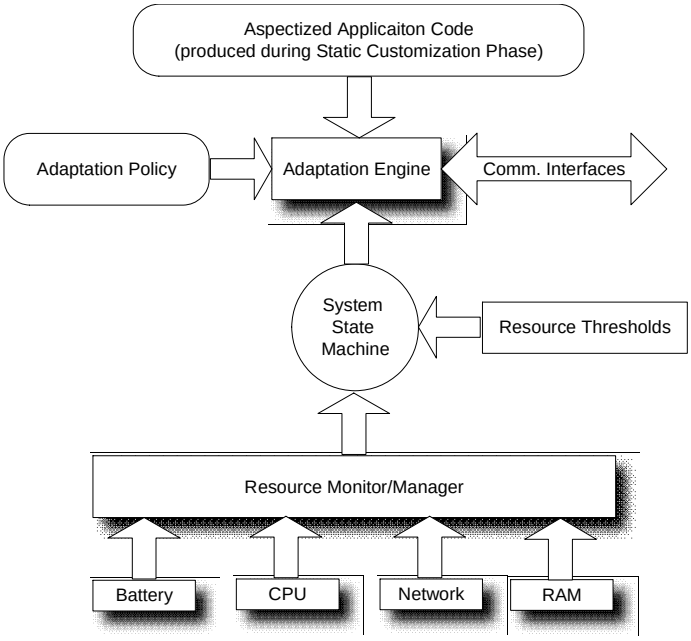


Fig 4.4: Dynamic Reconfiguration Phase

The time varying input to the Adaptation Engine, fed by the System State Machine, only predicts adaptation possibilities and anticipates the effects on the system resources. However, the application may not be capable of taking adaptation suggested by the System State Machine (because resource variations are independent of both the type of application and the user desires). Therefore, when its time to adapt, the Adaptation Engine has to choose from only those choices provided by the System State Machine, which are realizable by the aspectized code according to the constraints provided by the adaptation policy.

It may be noted that the user's adaptation preferences which together with the profile state set, partly (or in some cases fully) define the Realizable State Machine during the static composition phase are different from the adaptation policy used at runtime (figure 4.5). The difference is that the adaptation preferences in the static composition describe all the desired 'possibilities' of adaptation and enable the application for *coarse-grain adaptation,* while the adaptation policy used at runtime is for *fine-grain adaptation*, hence the name *dynamic reconfiguration phase*. A coarse grain adaptation is characterized by the way it impacts system resources. In case of coarse-grain adaptation, whenever the application switches from one mode to another, the change in system

resource utilization is significant, while for fine-grain adaptations, this change is very minor. Examples of course-grain adaptation include, compression scheme change, format adaptations etc, while tuning codec parameters, quality factor etc are examples of fine-grain adaptation.

In actual implementation, adaptation requirements are translated onto different plug-ins of the JMF. Therefore the process of mapping different profiles and preferences to make applications adaptive is effectively a two step process. In the first step, aspects are generated from profiles and are woven across different plug-ins constituting a JMF application. In the second step, adaptation preferences dynamically reconfigure the plug-ins. For example, normal codecs are made rate-adaptive by inserting mechanism for frame dropping and adaptive bit-rate control, special effects like converting a colored image to grey scale are adaptive enabled/disabled at runtime, while insertion of hooks takes place statically. This reconfiguration of media processing elements facilitates fine-grain adaptation and management of complete RTP sessions.

4.4 System-wide Adaptation Coordination

Since the application software can be spread across a complete network, adaptation is viewed as a system-wide coordinated activity. System-wide communication of initial negotiation and runtime statistics from different resource state machines existing on communicating peers, along with adaptation decisions from the server and synchronization signals are sent on the communication interface shown in figure 4.4. In all cases a machine with more resources and the one that is acting as a server, stays the adaptation master (controller of adaptation decisions) and the client remains adaptation slave. Therefore, the process of system-wide adaptation starts with the client application registering itself with the server side, mentioning the node's IP address and communication port and the server, registering the data with the client. In case of multiple clients, separate media processing chains exist on the server, because, the adaptation capabilities of all the clients are generally not the same.

During application runtime when there is a need to adapt, depending upon the type of adaptation required (whether due to scarcity of a local resource or a network resource) the client tries to adapt locally by altering the local application parameters, if possible (e.g., by making the picture small or swapping a local application component like a heavy-weight video renderer with a lightweight one). If the adaptation needs involvement of the

server, the client sends an adaptation request along with the adaptation type. The client determines the type of adaptation from the parameters of the target state of its own *Realizable State Machine*. If the adaptation cannot be handled locally on the client, then the server coordinates the system-wide adaptation. For example, if currently the client is receiving an H.263 video with 15 frames per second (the present state of it Realizable State Machine) and the fluctuation in the system resources switch its *System State Machine* to a new state, where it's *Realizable State Machine* must transition to a target state defined by H.263 video of 10 frames per second, then the adaptation request sent to the server will be:

```
<?xml version="1.0" encoding="utf-8"?>
<adaptreq type = "adaptdown">
      <format>h263</format>
      <framerate>10</framerate>
  </adaptreq>
```

After parsing the adaptation request, the server will signal the client if the request cannot be fulfilled, otherwise, when the Realizable State Machines of both the client and the server have reached a common state to switch, the client will be signaled to switch to the target state and the server will fulfill the request. When the client's request is refused, the client can send another request later. Apart from the adaptation request, the client periodically sends feedback reports.

If the server needs to adapt, the server will start the adaptation negotiation in a similar manner with the client. However, in this case when the server's and the client's Realizable State Machines both have a common state to which they can switch, the server has to trigger the adaptation. All adaptation co-ordination messages are sent via *comm. Interfaces* shown in fig. 4.4.

4.5 State Machines and Code Transformation

Figure 4.3 and figure 4.4 above, only described the phases of *static composition* and *dynamic adaptation* at a conceptual level, without giving details of implementation in software. Both these only serve as a 'bridge' to explain how the theoretical model presented in chapter 3 is related to the software implementation. In both these figures, the grey highlighted portions are mainly related to the software implementation of the adaptation using AoP and are explained in figure 4.5.

Entire ACREMA layer shown in fig 4.5 is an elaboration of the abstract view given in figure 4.3 and 4.4. It has been shown enclosed inside a dotted boundary, which is further

subdivided into two parts by a vertical dotted line, in order to separate the static composition phase from the dynamic reconfiguration phase. Both the static composition and the dynamic reconfiguration phases proceed in four passes, shown by the two dotted arrows in the figure 4.5. During the static composition phase, each pass starts by scanning the given byte code for adaptable properties of the components that the application loads by default (e.g; a default codec may support multiple frame rate or bit rates etc). This is done by the ACREMA Event Listener, which captures the events fired when the media processing chain takes transitions. Further details of these passes are shown in figure 4.6. The four different blocks (Composers) shown in figure 4.5, inside the static composition phase, correspond to four different classes of adaptation described in section 4.6. These four blocks enable composition of adaptive behaviors in the given application's byte code during the four composition passes shown in figure 4.6.

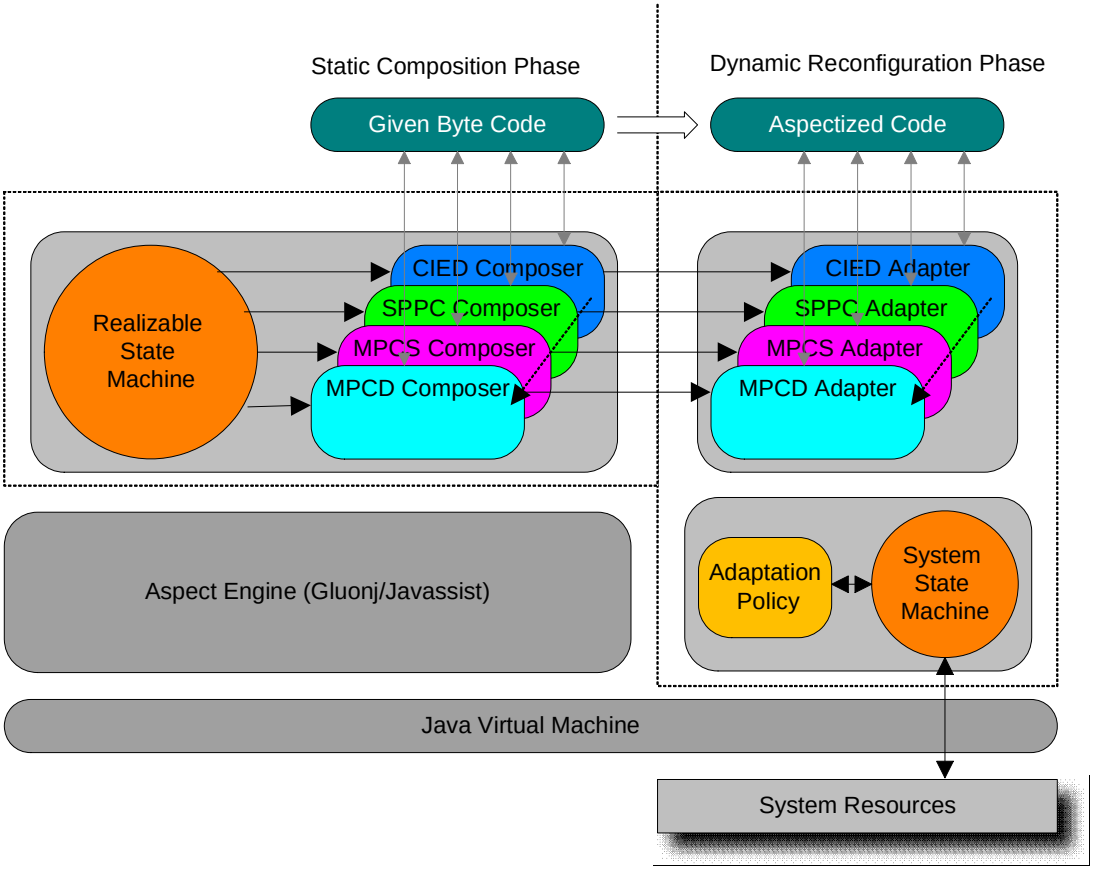


Fig 4.5 – An overview of the application code processing through the static composition and dynamic reconfiguration phases[7].

---

[7] In fig. 4.5, other parts of the system like Realizable State Machine generation, network communication etc have been deliberately omitted to avoid cluttering.

Upon completion of the static composition phase, the aspectization process is complete and the aspectized code of the given application which is now capable of showing adaptive behaviors is used by the dynamic reconfiguration phase. Four adaptors shown in figure 4.5 correspond to four different classes of adaptation discussed in section in 4.6. According to the adaptation policy and in response to the state transitions taken by the System State Machine at runtime, these four adaptors invoke the adaptive code patches corresponding to the state transitions of the Realizable State Machine.

```
Pass -1:
   Scan the given code, locate media processing elements;
   Deduce the application type (whether, local or network);
   Use before advice, intercept and prioritize event queue;
   Generate Realizable State Machine;
   Enable all CIED adaptations;

Pass -2:
   Read Realizable State Machine;
   Locate Pre- and Post Processing Adaptation Requirements;
   Mask conflicting methods, using around advice;
   Weave in new Aspects and Generate new Realizable State Machine;

Pass -3:
   Read Realizable State Machine;
   Locate Static Main Processing Adaptation Requirements;
   Mask conflicting methods, using around advice;
   Weave in new Aspects and Generate new Realizable State Machine;

Pass -4:
   Read Realizable State Machine;
   Locate Dynamic Main Processing Adaptation Requirements;
   Mask conflicting methods, using around advice;
   Weave in new Aspects and Generate new Realizable State Machine;
   Enable Parameter Tuning for Runtime Reconfiguration;
   Report Static Composition Completion;
```

Fig. 4.6 – Static Composition Passes

4.6 Adaptation Classification

There can be several ways to classify the adaptations carried out by ACREMA, e.g; categorizing with respect to the type of application (e.g; running locally on the device playing stored media, executing on the network as a simple client and server, a multiparty

conference where several sessions are to be managed separately etc). Classification can also be done on the basis of the type of implementation (i.e; how are the adaptive behaviors injected in into the application). Another possible categorization is with reference to the type of adaptive code injection and replacement which is more related to ACREMAs' architecture and type of the supplied byte-code of the application. In the subsequent sections, the classification will be done from an architectural standpoint with respect to the type of Aspect oriented features used.

4.6.1 Code Interception Event Diversion Adaptation (CIED)

This type of adaptation is characterized by its existence anywhere in the application code (i-e; it can effect any of the *Pre, Post* or *Main* processing chain(s)) and are handled by intercepting running code and diverting events. For any non-adaptive application, this is the simplest type of adaptation and basically requires identification of tunable media processing elements (if any) constituting the entire chain, along with their properties which may be tuned to get the desired behavior. Examples of the applications requiring this type of adaptation include any application that is to play or record a simple audio/video from/on the local device and the user preferences demand very simple adaptation. Typical adaptation requirements in such cases are related to demands on bit-rate, frame-rate, display size, different codec parameters etc.

Since this type of adaptation is characterized by its property of Dynamic Stream Reconfiguration, it requires minimum code manipulation and the Static Composition phase is the simplest for such cases. This phase mainly involves identification of the media processing element(s) which are responsible for giving the desired output. Identifiers pointing to all such elements are listed, event ACREMA event listeners are registered to divert to the flow of control to the Adaptation Engine and corresponding Aspects are added. The UML sequence diagram below shows the process.

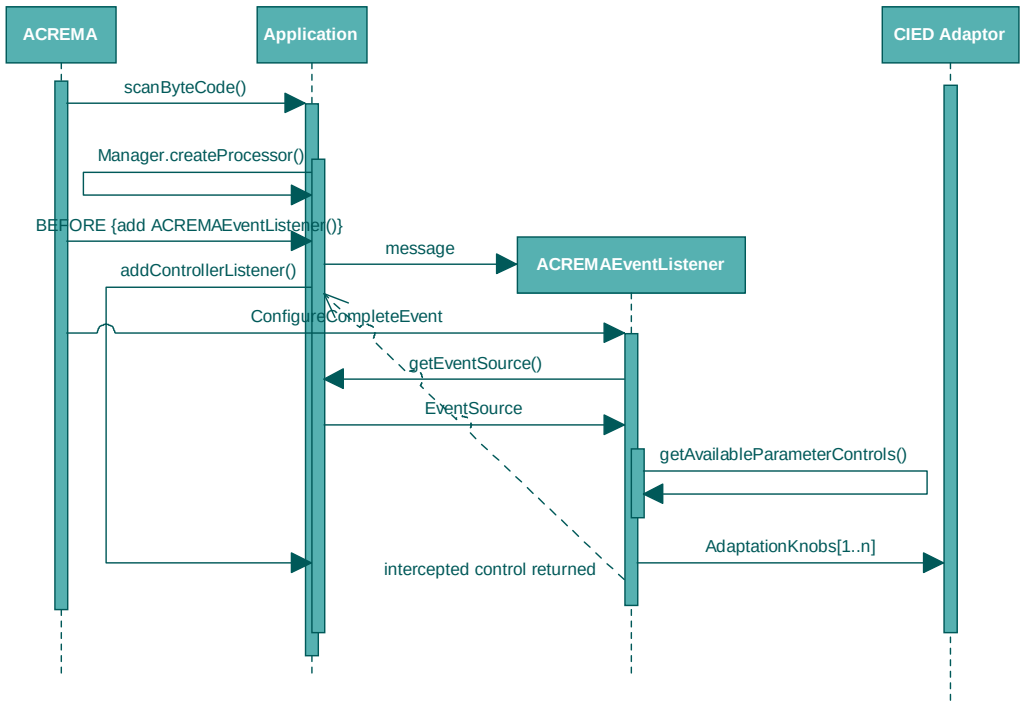


Fig 4.7: Sequence of operations to weave-in CIED Adaptations

As soon as an adaptive element (like a Media Processor) is created, the pointcut matches the condition and by adding *before advice*, the execution of the application is intercepted and the flow of control is diverted to add ACREMAEventListener, which is responsible for extracting all tunable parameters as dictated by the Profile S.M. The flow of control is returned to the given application after all Runtime Adaptation Knobs have been exported to the CIED Adaptor, which is responsible for parameter-tuning at runtime, under the constraints of the System S.M. Use of *before advice* here ensures that all events are intercepted by ACREMA, before they reach the actual application.

Since this type of adaptation, does not involve code injection into the application (instead it works only by intercepting the given code and diverting events), it relies entirely on the default media-processing chain of the application and the extent of CIED adaptations that can be extracted from the given application code depends upon the number and type of media-processing elements loaded by the application. (Implementation example in sec.5.2.1)

4.6.2 Static Pre or Post-Processing Chain Adaptation(SPPC)

SPPC Adaptations are characterized by their existence in pre- and/or post-processing media chains, due to which the additional code required to impart adaptive behaviors is also in the pre- and/or post processing chains. This type of adaptation involves relatively more work to be done during the Static Composition phase. Examples can include adaptation choices made by the user which cannot be fulfilled by tuning existing media processing elements of the given application. Thus, the requirements are met by adding additional code fragments which can display the desired adaptive behavior. Practical examples of this type may include applications involving custom processing of a media track (if it is desired that the size of the playback display be variable, it may need addition of a display components, having properties of screen size scaling, or it may be desired to have a light-weight video-renderer than one the application already has, (in order to suit the deployment on a resource constrained device). Another example may be addition of media pre-processing or post-processing elements in the codec chain (e.g; a user opts for some adaptation which, although, can be provided by the default codec used by the application, but the output produced cannot be displayed by the default video-renderer or the codec in its present form takes only specific input video size, in the pre-processing chain, then the pre-processing chain may need code modifications to handle additional sizes. Similarly, some post processing (e.g; RGB to YUV conversion or vice-versa) may be needed.

This kind of adaptation is relatively complicated than the previous one and is mainly carried out on different pre and/or post-processing elements of the chain. The UML sequence diagram below shows the operations involved during the Static Composition of the given application from this type of adaptations.

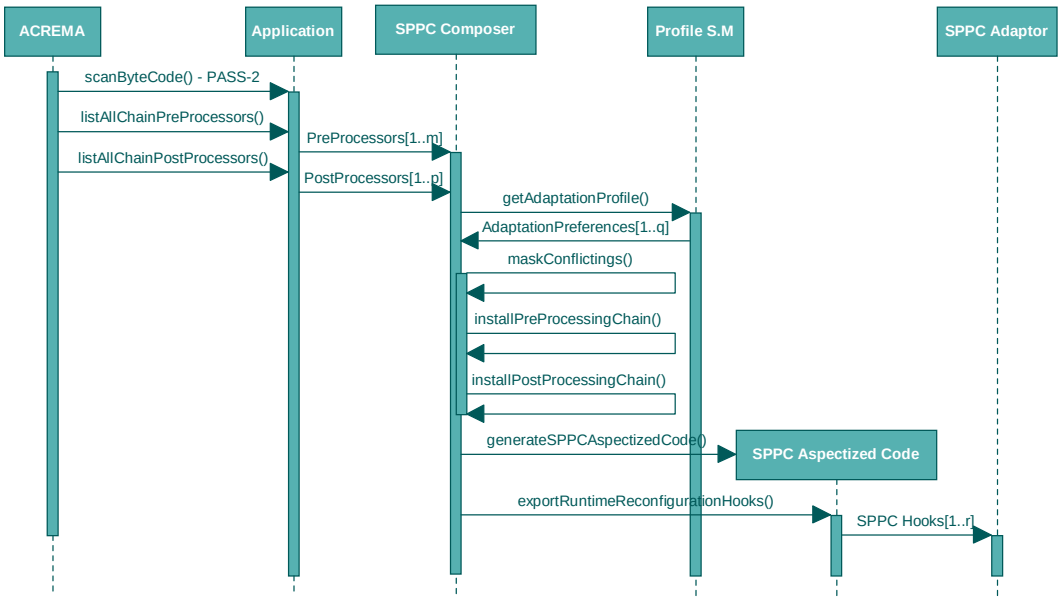


Fig 4.8: Sequence of operations to weave-in SPPC Adaptations

All pre and post processing elements are obtained, their media processing properties explored (through *reflection*) and in view of the constraints set by the Profile S.M, their default properties are altered. This may need only partial code injection (using *before* or *after advice*, to add additional code, e.g; in case of video size change) or complete replacement of a specific element (using *around advice*, e.g; in case of swapping default video-renderer with a leight-weight renderer). As a final step, all adaptation hooks are obtained by exploring the configurable properties of the modified pre and/or post-processing chain and exported for runtime adaptation.

All related pointcuts are identified, the joinpoints formed and Aspect Advice is woven to carry out desired adaptive processing. (Complete implementation example in sec.5.2.2)

4.6.3 Main Processing Chain Static Adaptations (MPCS)

This type of adaptation exhibits a higher degree of complexity and is found in the applications which are written with their basic functionality in mind, but need to adapt to a different format. It is particularly related to installation of custom codecs. These adaptations are characterized by their requirement to modify the main processing chain. A typical example of this type of adaptation involves transcoding from one format to another, where the transcoding format is selected statically and is not altered

dynamically. That is, it has the transcoding property as a coarse-grain adaptation requirement during static composition phase and may or may not have fine-grain adaptation requirements during dynamic reconfiguration phase. A practical example may not even involve media display or play-out, but only, reading one format and storing in another format or reading in one format, transcoding on the fly and transmitting in another format. UML sequence diagram given below shows the operations involved.

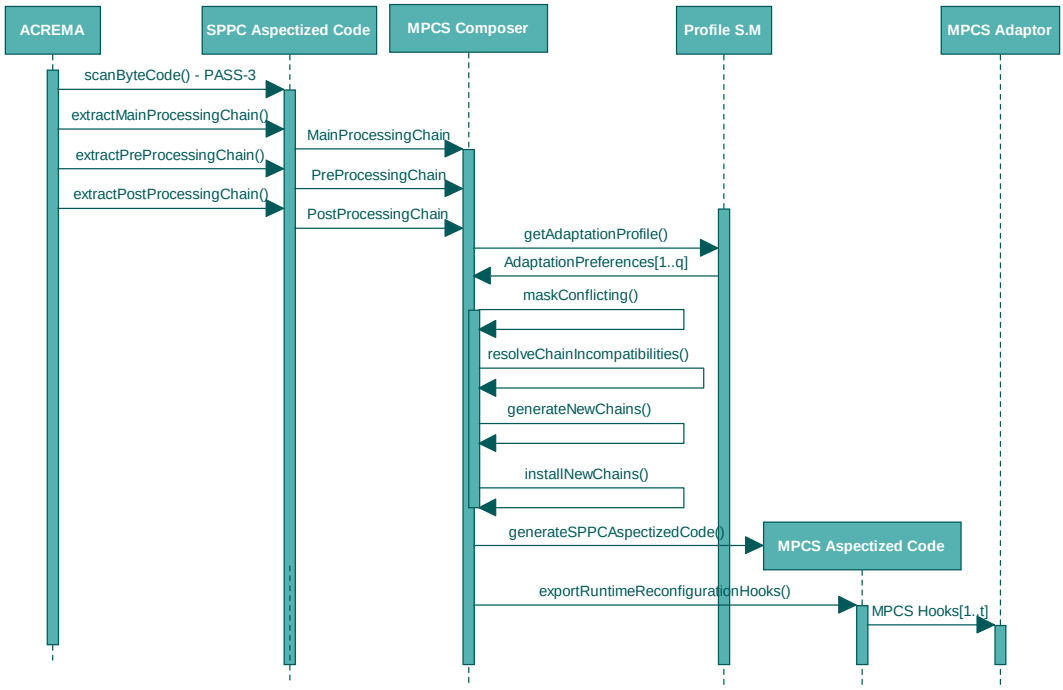


Fig 4.9: Sequence of operations to weave-in MPCS Adaptations

In addition to swapping the existing main-processing chain with the new one, an additional (and iterative) step in this case is to resolve the incompatibilities arising with the installation of the new codec. This conflict resolution may require changes to pre and/or post-processing chains (e.g; if a codec accepts YUV input instead of RGB, then color scale needs conversion, which is basically a pre-processing operation). Like the previous cases, runtime adaptation hooks from the resulting aspectized code are passed to the adaptor for runtime tuning. (Implementation example in sec.5.2.3).

4.6.4 Main Processing Chain Dynamic Adaptations (MPCD)

This type of adaptation has the highest degree of complexity, both during the static configuration phase as well as during the Dynamic Reconfiguration phase. It is different from all other adaptation types due to its resource conflicting nature and very high overhead. This type of adaptations are resource-conflicting becaue, when the system adapts with respect to one resource, another resource gets overloaded mainly due to heavy house-keeping overhead. These adaptations are basically on the main processing chain, but, occasionally effect pre- and post-processing chains as well. Thus complexity of the runtime phase is the main factor which differentiates it from PMCS case discussed above. Example of this kind of adaptation is found where a codec swap is needed at runtime. In this case mostly an existing processing chain has to be completely replaced with another, and this process of chain swaps may continue on to multiple sets of pre, post and main processing chains. Additional code to facilitate synchronized switching of the media stream from one chain to another is generated. All runtime adaptation hooks from all complete chains are then exported in sets (each set corresponding to a complete chain), therefore, switching from one chain to another can exhibit entirely new adaptive behavior of each element of the chain under consideration. This type of adaptations have maximum overhead due to the fact that each time a new chain is installed while the application is running, the one already in place has to be garbage collected. (Complete implementation example in sec.5.2.4).

Table 4.2 gives a summary of basic adaptation types, categorized on the basis of implementation type.

Adaptation	Identifying Property	Implementation	Overhead	Nature
CIED	Exists anywhere in the program code	Implemented by intercepting running code and diverting events	Least	Non-Conflicting
SPPC	Only in pre and/or post-processing chains	Implemented by injecting additional code in pre and/or post processing chain(s)	More than CIED	Non-Conflicting
MPCS	Basically in the main processing chain, but can effect pre and/or post processing chains	Implemented by one-time static modification of the main (and possibly pre and/or post) processing chain(s) using code injection	Comparable with SPPC	Non-Conflicting
MPCD	Basically in the main processing chain, but mostly effects pre and/or post processing chains	Implemented by one or more static code modifications of the main (and possibly pre and/or post) processing chain(s) using code injection along with addition of runtime switching and stream synchronization code	Highest	Resource Conflicting

Table 4.2 – Summary of adaptation classification

4.6.5 Multiple Code Manipulation Adaptations (MCMA)

Each of the adaptation types discussed above corresponded to the four passes of application code processing in fig.4.5 and fig.4.6. Since the formation of Realizable State Machine is also dependent on User's Profile State Set in addition to the Application State Set, and the User's Profile State Set is defined by the user along with the adaptation preferences (also) given by the user, real life adaptation requirements are much complex than the 4 basic categories of adaptation discussed above. The complexity mainly arises from the fact that to fulfill user's adaptation demands, the processing of the given application code has to undergo multiple passes (shown in fig. 4.6). Therefore MCMA represents the real life adaptation scenarios where multiple locations in the application code are aspectized. This is the most complex adaptation situation and can virtually handle all types of requirements, therefore, is the most practical situation also. However, the corresponding complexities of both the Static Composition and Dynamic Reconfiguration phases are proportionally high. Another important feature of these adaptations lies in the fact that they have to do an additional pass to scan the application for such methods which can interfere with the ACREMA adaptation implementations and mask all such methods, so that the user may not be able to deliberately or accidentally tamper with the system. These adaptations are characterized by existence of multiple adaptation instances of the types described in sec.4.5.1 to sec.4.5.4. This requirement of making multiple changes throughout the code-base, where these changes satisfy well defined conditions, justifies the use of Aspect oriented Programming to handle these practical cases of adaptation. (Complete implementation and evaluation example is in Chapter 6.)

4.6.6 Adaptations requiring ACREMA Extensions

The adaptations which are not strictly confined to application code modifications, but rely on several other factors (e.g; different cross-layer adaptations, which can be carried-out effectively without making any changes to the application code), fall under this category. Although, such cases were not directly within the scope of the work presented here and have not been implemented fully in ACREMA however, a limited extension related to adaptations with respect to device characteristics have been incorporated. This can incorporate system wide adaptations (e.g, any specific adaptive resource management strategy of any network transmission methodology like composition filters etc, layered transmission with online format conversion into multiple content qualities, without requiring to write multiple content types).

Chapter 5

ACREMA Implementation

This chapter explains specification of adaptation preferences/profiles along with code snippets to show some sample code injection examples. To implement the concepts discussed in the last chapters, Java was the language of choice, since it provided all desired language properties (reflection, meta-data handling, AoP), and a wide variety of aspect engines, each with a host of features. On top of that Java Media Framework (JMF) was available to design the test applications and prove the suitability for an existing platform, therefore application development was carried out in JMF.

5.1 Specification of Adaptation Preferences and Profiles

Adaptation preferences are accepted from the user using a GUI and are internally kept in XML files. A sample adaptation preferences file is shown in fig. 5.1. Different switching possibilities given by the user, along with the default application code determine the possible code instrumentation (adaptive code injection into the given applicaiotn code), thus defining the states of the application and user state sets, while preferences define the transitions of the Profile State Machine.

Although, resource usage in multimedia compression largely depends upon the type of image/video being encoded/decoded, from a number of experiments, the resource overheads were computed, and based on resource consumption in different types of operations, adaptations tables were devised.

Adaptation	Type	Implementation Overhead	Reliability	Comments
Format change↑	Conflicting	complicated, high	can be faulty	consecutive invocations not recommended
Size change↑	Non-conflicting	simple, low	can be faulty only first time	consecutive invocations possible
Frame rate change↑	Non-conflicting	very simple	Never faulty	consecutive invocations possible
Image Quality change↑	Non-conflicting	Simplest	Never faulty	Step-wise invocations possible

Table 5.1 – Different adaptation types and relative overheads

In addition to the coarse-grain adaptation types described in table 5.1, other fine-grain adaptations also exist. For example, a list of tunable codec parameters is in case of H.263, the parameters, which in-turn is represented as enabling different modes of the codec. The default order for codec parameters are no-options, PB-enable, AC-enable, AP-PB-enable and AP-PB-AC-enable.

```
<?xml version="1.0" encoding="utf-8"?>
<preferences>
      :
      :
      :
      <compression standard = "h263">
            <codec pref = "1">
                  <adaptation type = "format">
                        <size pref = "1">16CIF</size>
                        <size pref = "2">4CIF</size>
                        <size pref = "3">CIF</size>
                        <size pref = "4">QCIF</size>
                        <size pref = "5">SQCIF</size>
                  </adaptation>
                  <adaptation type = "framerate">
                        <fps pref = "1">highrate</fps>
                        <fps pref = "2">midrate</fps>
                        <fps pref = "3">lowrate</fps>
                        :
                        :
                        :
                  </adaptation>
            </codec>
            :
            :
            :
      </compression>
      :
      :
      :
</preferences>
```

Fig 5.1: An excerpt of user's adaptation preferences

Considering the above mentioned sample profile and adaptation preferences, the Profile State Set with a default state is derived.

For generation of the Profile State Set, device profile, network profile etc are, taken into account in addition to user preferences for desired adaptive behavior. A device

profile describes device characteristics like, capabilities to switch on/off the LCD backlight, cpu speed-stepping modes, low-power-stand-by mode availability etc. The network profile describes the connection possibilities and preferences e.g; it can contain preferences with respect to pricing (per minute usage in GPRS, dependent on the amount of data transfer in case of GSM, free in case of WLAN etc) which helps the system decide how expensive an adaptation would be[8]. A sample profile is shown in fig 5.2.

5.2 Derivation of the Resultant State Machine

The first step in aspectizing a pre-written application code is the identification of elements which can be used to weave adaptive behaviors. This is done once for whole framework API (in this case JMF API). The classes, their identifiers and methods, which represent and control the functionality of media processing elements are identified and corresponding information stored in a repository. Depending upon the given application code, the ACREMA Resident Layer thus appropriately identifies all the joinpoints in the given code (details in the next four sub-sections), such points are usually spread across the whole code-base, spanning multiple classes. These joinpoints are used in the Pointcut(s) and Advice for adaptive execution is woven.

---

[8] In the simulation examples for evaluating ACREMA's architecture and application test cases, switching between WLAN, GSM or GPRS part was not implemented (due to restrictions of the evaluation environment). Further more this is not within direct scope of the work presented here; this type of adaptation comes under ACREMA's extensions.

```
<?xml version="1.0" encoding="utf-8"?>
<profile>
      <clientside>
            <renderer pref = "1">lightweight</renderer>
            <renderer pref = "2">awt</renderer>
            <renderer pref = "3">native</renderer>
            <renderer pref = "4">java</renderer>
            <dimension pref = "1">large</dimension>
            <dimension pref = "2">medium</dimension>
            <dimension pref = "3">small</dimension>
            :
            :
            :
      </clientside>
      <deviceproperties>
            <cpu type = "ARM">
                  <freq>200Mhz</freq>
                  <scaling = "3">speedstep</scaling>
            </cpu>
            <ram>64M</ram>
            <lcd level = "2">backlight</lcd>
            :
            :
            :
      </deviceproperties>
      <resourcecosts>
            <connection cost = "high">gprs</connection>
            <connection cost = "low">gsm</connection>
            <conection cost = "none">wlan</conection>
            :
            :
            :
      </resourcecosts>
</profile>
```

Fig. 5.2 – An excerpt of sample profile

5.2.1 CIED Code Transformation and Parameter Tuning

This type of adaptation mainly involves identification of the software elements, which can be operated in an adaptive manner. Although, some kind of load-time composition is needed in all cases, in this case, the patched code only exports adaptive features of already existing components inside the application. Therefore, the adaptation is mainly confined to parameter tuning of existing components. To illustrate the example, we take a prewritten application available from Sun Microsystems as a sample java media framework application.

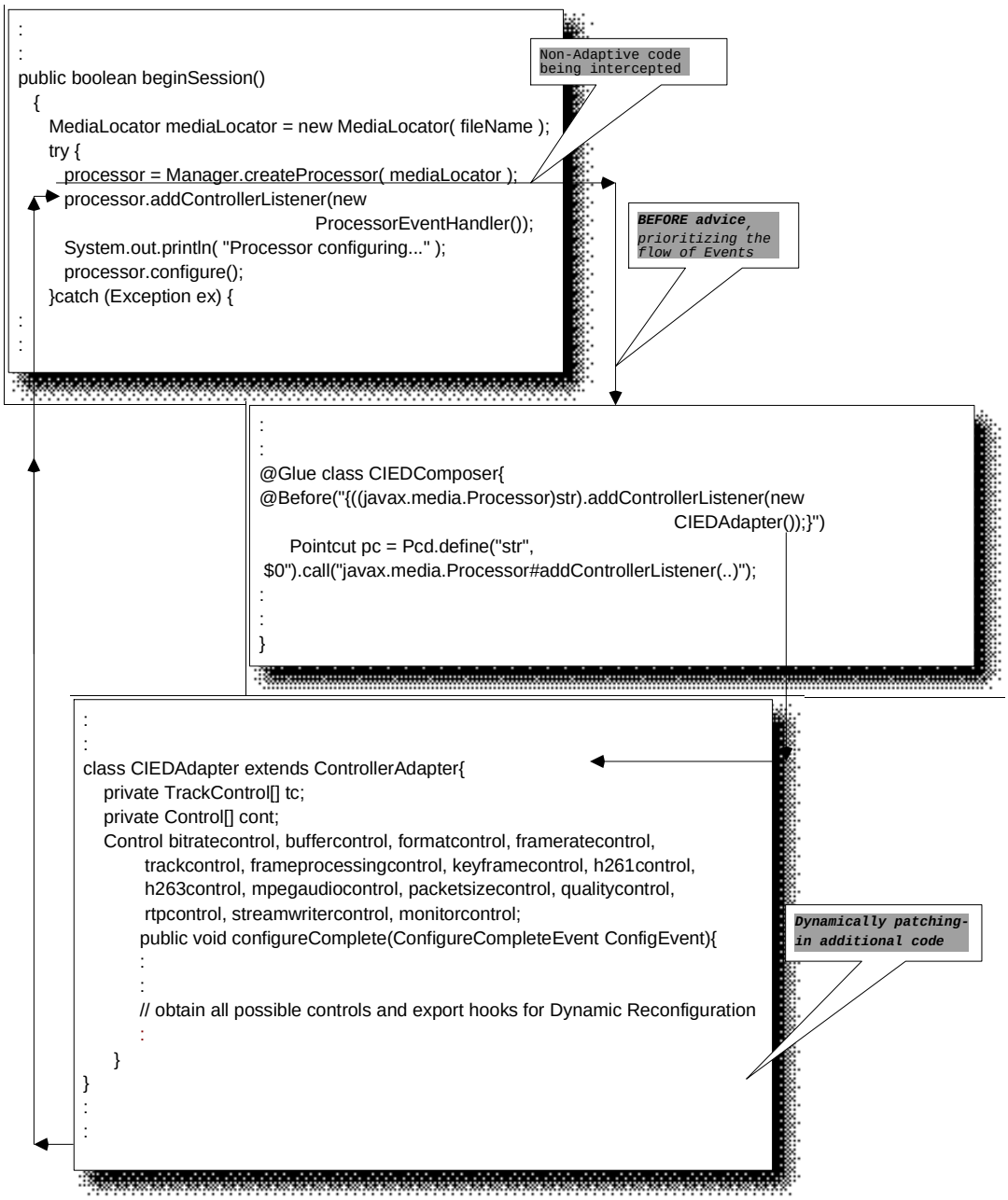


Fig 5.3: Given application byte-code being intercepted by the Pointcuts residing outside the application, advice being woven and runtime adaptation hooks being exported as a result

Since the adaptation can only be carried out on the application written using JMF, the framework API was searched thoroughly for the media elements which can be vital in adapting the multimedia data stream. Considering the following application excerpt in fig 5.3, which declares an element of type `javax.media.Processor`. It is one such element which processes multimedia data and can be used to perform adaptive media processing. Since this type of adaptation can is realized by event-channel interception, to divert the control flow and monitor the operations and states of the Processor, the ACREMA Resident Layer adds event listeners on it using sample AoP snippet shown in fig. 5.3.

At runtime, as soon as the method, `p.configure()` has completed execution, before it has exited from its body, the pointcut defined in the figure matches the condition and weaves-in the corresponding advice, thus effectively enabling the control flow diversion upon event reception, to the class `ACREMAEventListener`. This class in-turn captures the source of the event (`javax.media.Processor`) and adds event listeners on it to keep track of its states. In the example code it gets a reference to the `trackcontrols` and thus can access and adaptively alter the behavior of individual media track to be played by the application. It is worth noting that the use of *@before* registers the ACREMAEventListener by intercepting the application code in such a way that in the event queue, the desired events are captured by ACREMA, *before* they can be passed to the application, so that they can be diverted (and manipulated) adaptively.

Upon completion of this whole chain of events (defined as Static Composition), the given non-adaptive application, is now capable of showing the sample adaptive behavior. Like this simple example of a behavior addition, when the entire application code is aspectized with any such adaptation advice, a number of adaptations can be realized. The AoP code shown in fig-5.3 captures the reference to the argument passed to the function `configure()` of class `javax.media.Processor`. Its clear from fig-5.3 that its the variable p of type `javax.media.Processor`. The sample Aspect Code defines and alias to the parameter passed and the alias is then used by the advice of this aspect code to register an event listener with that variable. Therefore, when the byte-code corresponding to the program snippet shown in fig-5.3 is loaded on top of the ACREMA Resident Layer, first of all, the code is analyzed and by taking a reference to the media processing element, ACREMA Event Listener is attached to it, which keeps monitoring that element for adaptive processing of the media stream passing through it. Once event channel interception is completed successfully, the different Controls get hooked to ACREMA for dynamic reconfiguration according to adaptation preferences, and changing system states.

5.2.2 SPPC Code Transformation and Parameter Tuning

Like the example shown above, in this case too, the events are intercepted to divert the flow of control, however the difference here is that additional code is patched. In contrast to the case described above, where different `Controls` (e.g., bit-rate, frame-rate etc) were obtained by intercepting the events and only fine grain adaptation was possible through dynamic reconfiguration of the existing code, now additional code patched-in provides the hooks for fine-grain adaptation These adaptations also rely on dynamic parameter tuning, but the tunable parameters are of those components, which did not form the part of the given application code, but were, injected by ACREMA at load time and then at runtime, there parameters are adaptively tuned. Or the cases of load-time component swapping, where a pre-existing component of the application code is replaced with another for adaptive processing, however, this component must be a part of media pre-processing or media post-processing chain.

An example can be the situation, where the application (an RTP Client) was developed for a powerful computer, is now required to execute on a resource-constrained device like a PDA. Due to use a resource limited nature of the target device an adaptation possibility is to use a light-weight video renderer. Thus instead of using the default renderer, a new renderer is to be used with this pre-written non-adaptive application, for which the user only needs to have the byte-code available. Then the device characteristics of the new device will need to be specified (in the device profile). Keeping this adaptation requirement in view, the sample code shown below will undergo an adaptive transformation through ACREMA, the details of this process are shown in fig. 5.4.

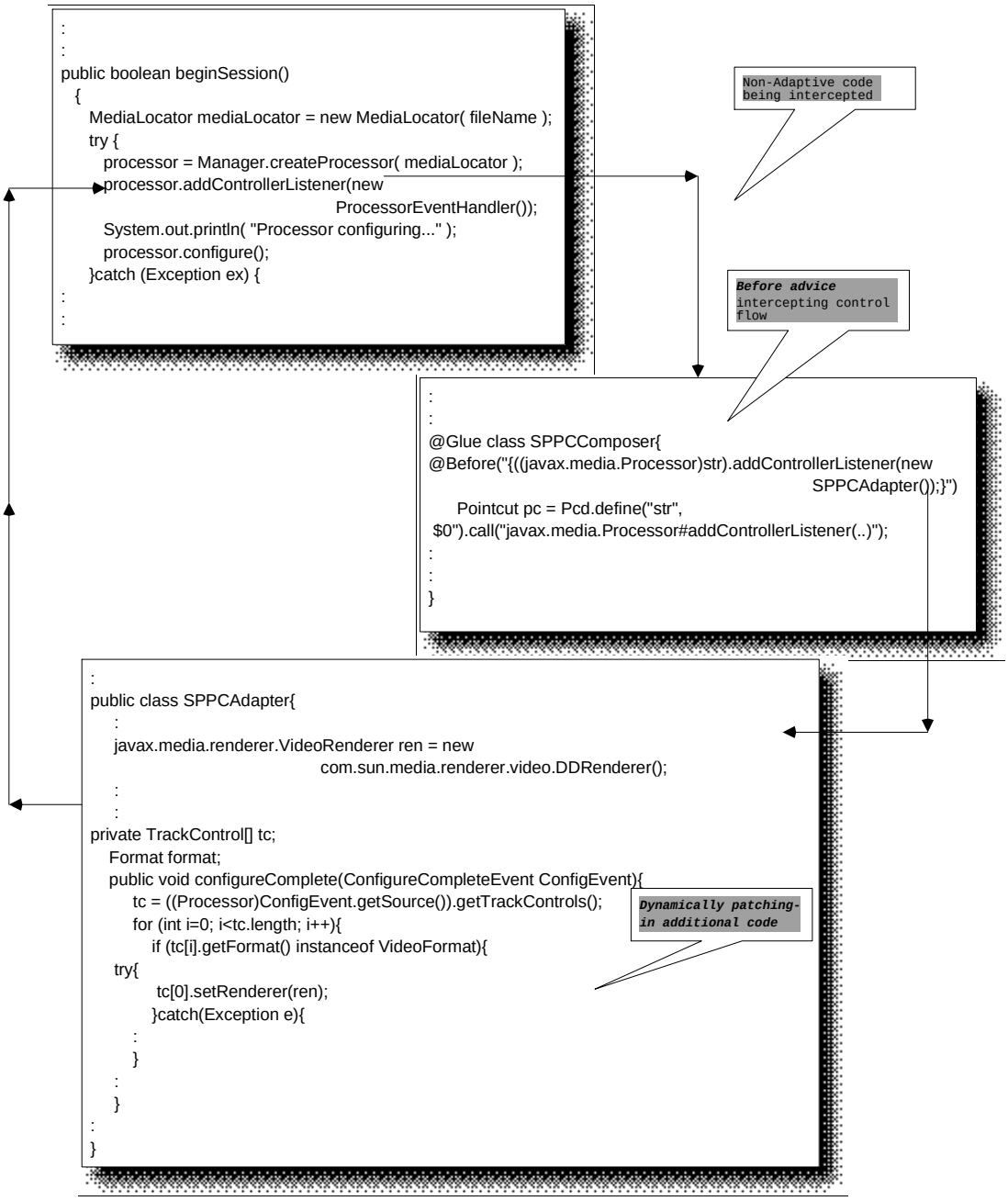


Fig 5.4 (a) showing code interception and patching in case of SPPC adaptations

Depending upon the application code a similar analysis is done on other media processing elements, including codecs, input and output devices, communication

channels etc, in order to monitor their processing behaviors and then adaptive behavior is given to them by altering their default way of data processing.

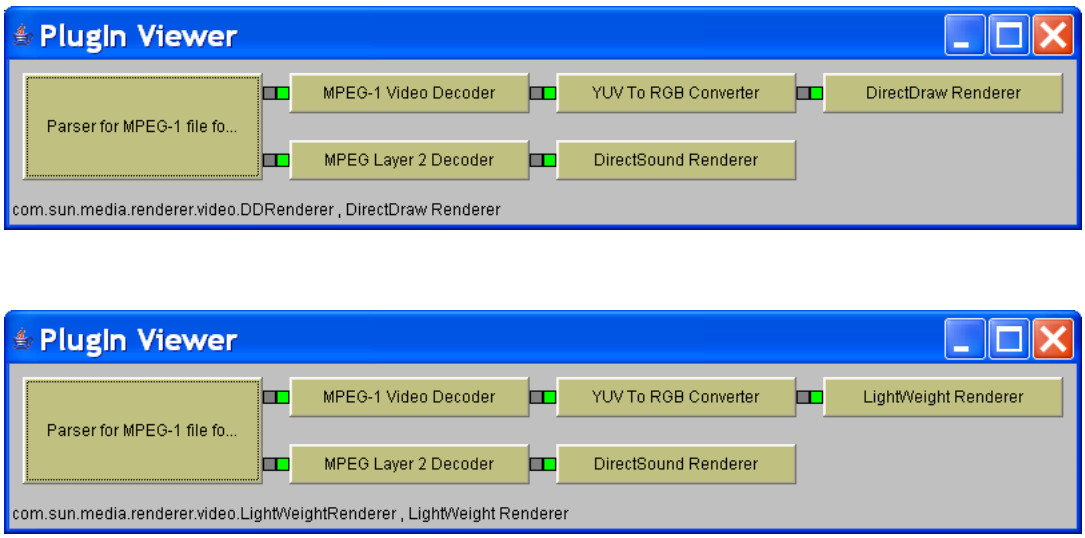


Fig 5.4 (b): showing the resulting component swap (DirectDraw Renderer swapped with LightWeight Renderer), as the result of above advice weaving.

Having added the advice, corresponding to variations in the resource demands, when the element which has been made adaptive switches across different modes of operation, lead to generation of the Realizable State Machine. In the example code shown above, since a method call was trapped and it ultimately completed upon addition of a custom codec to the already provided application code. This codec provides the following tunable parameters and corresponding to different modes of the codec, there are resource variations as the table above.

5.2.3 MPCS Code Transformation and Parameter Tuning

Example of this includes format change adaptations where the codec is completely swapped on the fly. This type of adaptation is vitally different from the previous one by its nature of affecting the *main processing chain* of multimedia data processing, whereas the adaptation mentioned above affects the *pre-processing* or *post-processing chains.* Due to this reason the implementation overhead of MPCS is higher than that of the previous cases, however, this overhead is mostly confined to application load time. A practical example shown below is of an application where the application was initially designed to use a specific compression and the user wants a different one.

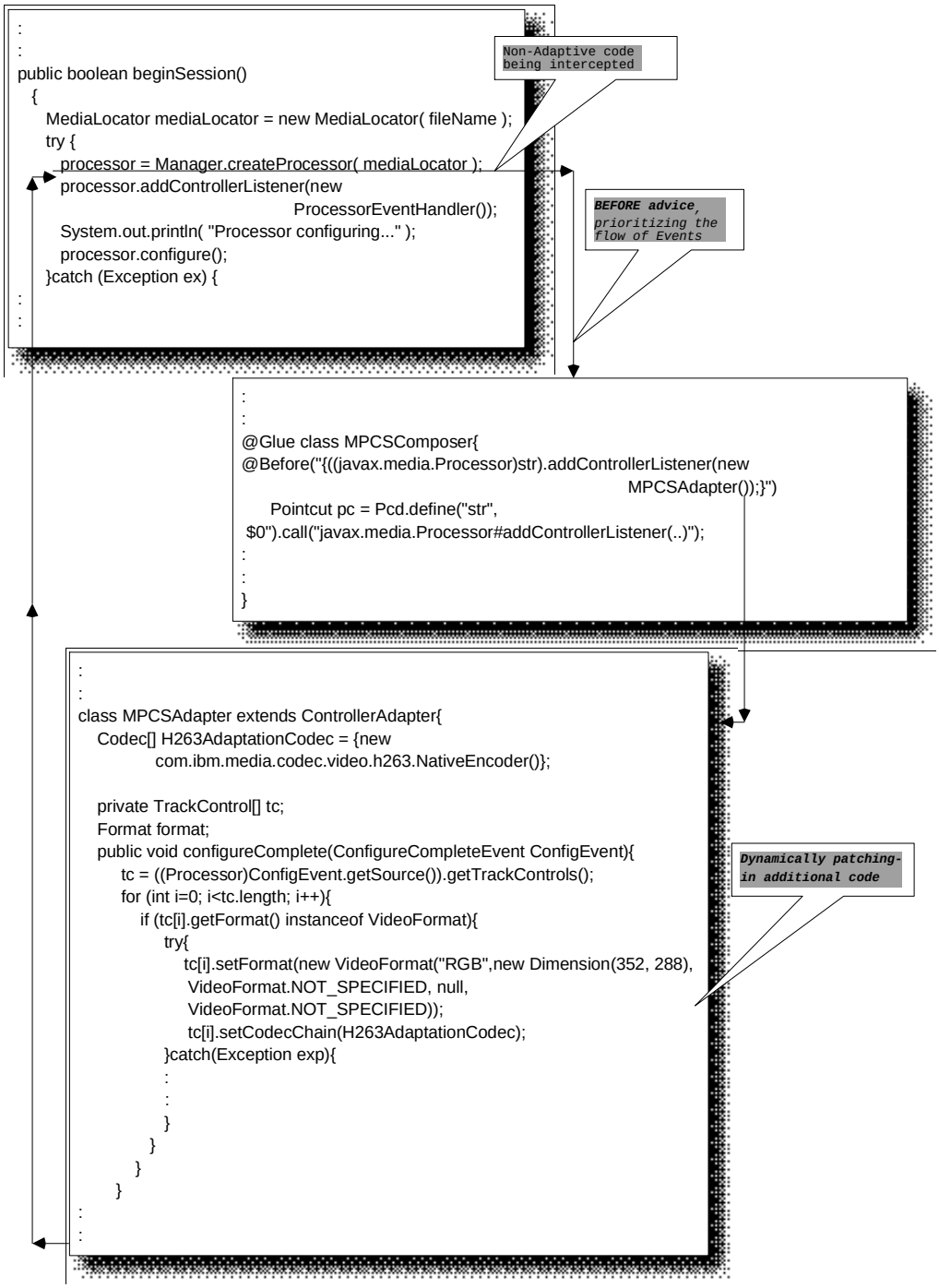


Fig 5.5: A codec chain being swapped with a new H.263 codec during the static composition phase of an MPCS adaptation.

5.2.4 MPCD Adaptation Implementation

In contrast with the adaptations in the previous cases, this is the final phase of applcaiton processing and scans the provided non-adaptive byte-code, for necessary code swaps, in view of the profile state machine, when multiple media processing elements are needed in the main processing chain of the application, such that during application runtime, such changes need to be incorporated on the fly. A typical example of this type of adaptation is implemented as a dynamic codec swap requirement.

In MPCD Adaptations several complete media processing chains are installed during Static Composition phase and during the Dynamic Reconfiguration phase, those chains are activated. This, however involves additional effort to synchronize media data, stopping one chain, to switch to another and so on. Due to this, in practical applications, MPCD adaptations get lowest priority and are used rarely, since they often lead to resource usage conflicts.

Implementation details of different adaptations as given in this chapter, differ from each other mainly during Static Customization phase. During Dynamic Reconfiguration, all adaptations are invoked in response to custom events, fired due to transitions of the System State Machine.

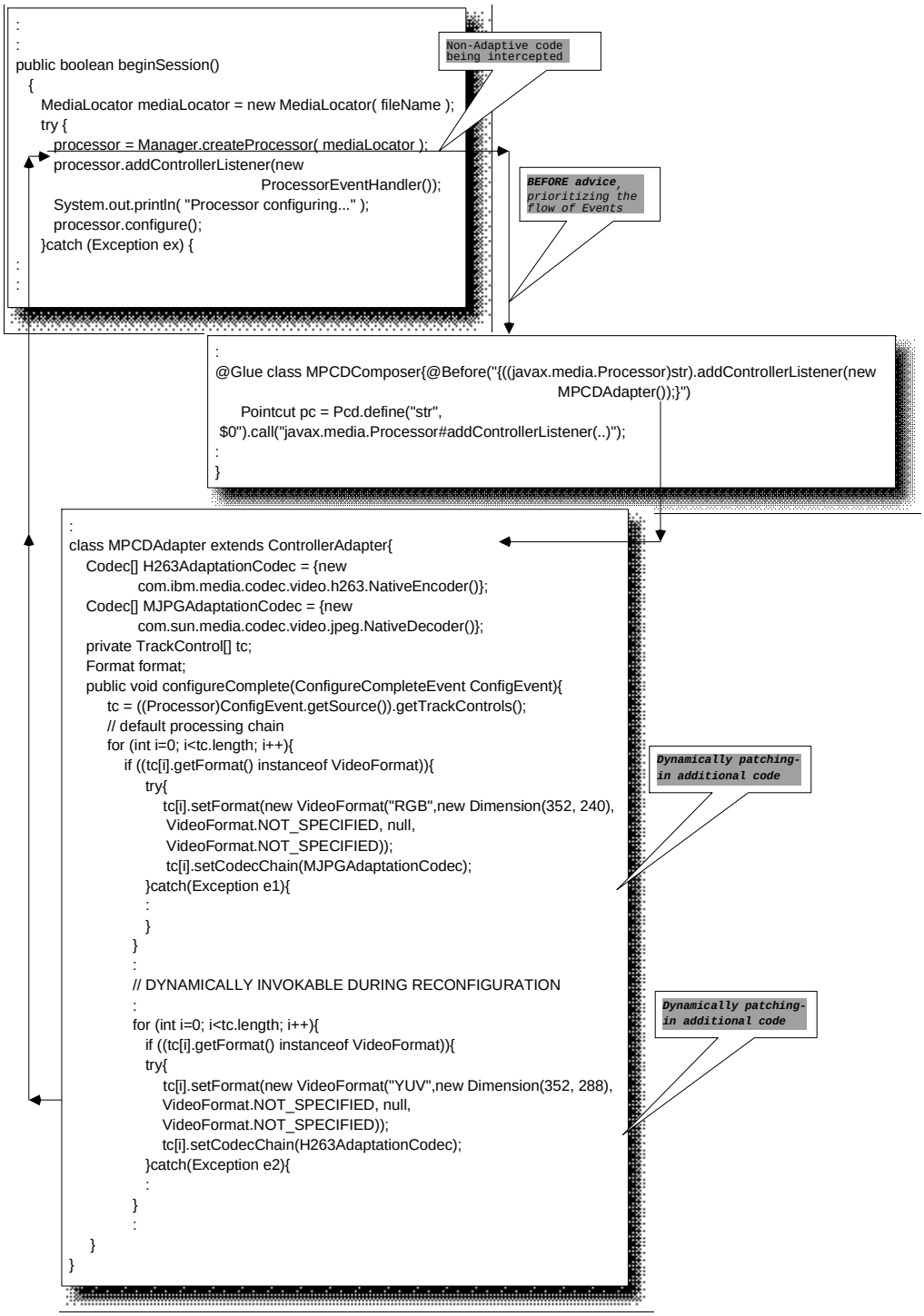
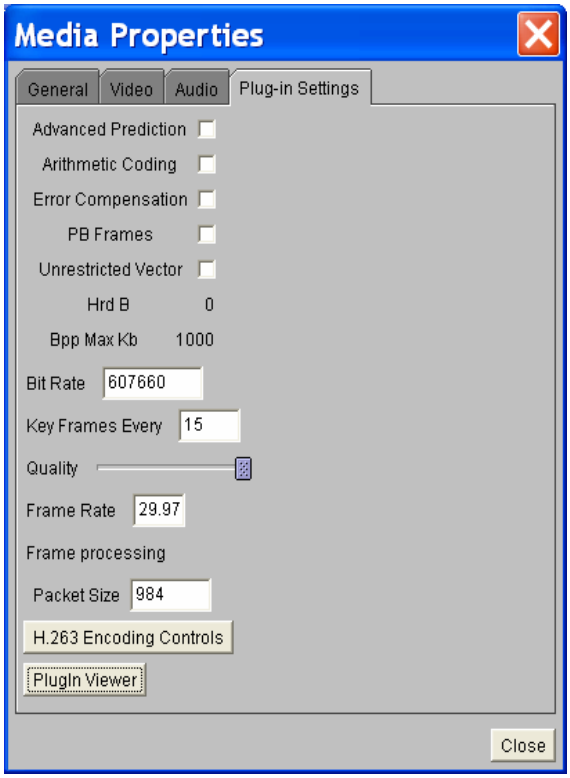


Fig 5.6 (a): Multiple processing chain installation of a DMFC adaptation. I/O sync. code left out for simplicity.



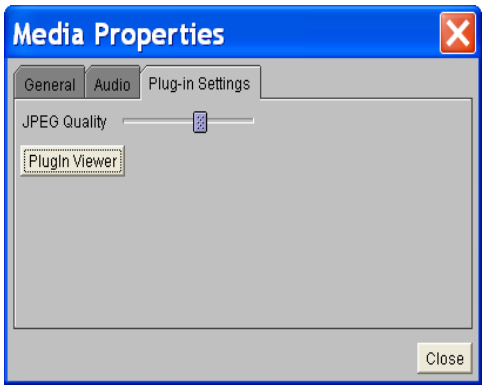


Fig 5.6(b): resulting change in dynamic adaptation hooks, (initially JPEG quality control was available), now in addition to H.263 quality control a number of other fine tuning parameters are exported as adaptation hooks.

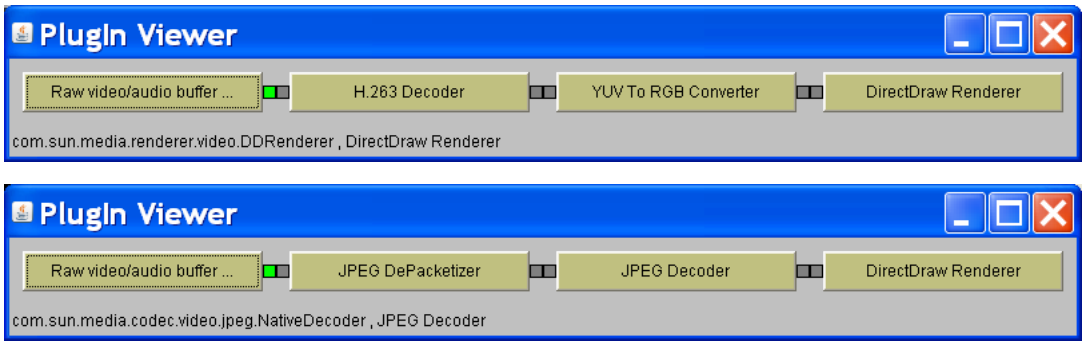


Fig 5.6(c): multiple elements of the media processing chain have been swapped, as a result of main chain incompatibility resolution process (described earlier in sec.4.5.4)

Chapter 6

ACREMA Evaluation

6.1 Evaluation Test Bench

This chapter provides details of the evaluation setup and quantitative measurements made in order to measure the efficiency of ACREMA's architecture and adaptive code injection, compilation and execution overheads. The evaluation of ACREMA was conducted in an emulated test-bench, which simulates situations that can occur in real life. The experimental setup to emulate virtual machines was realized as shown in the diagram below:

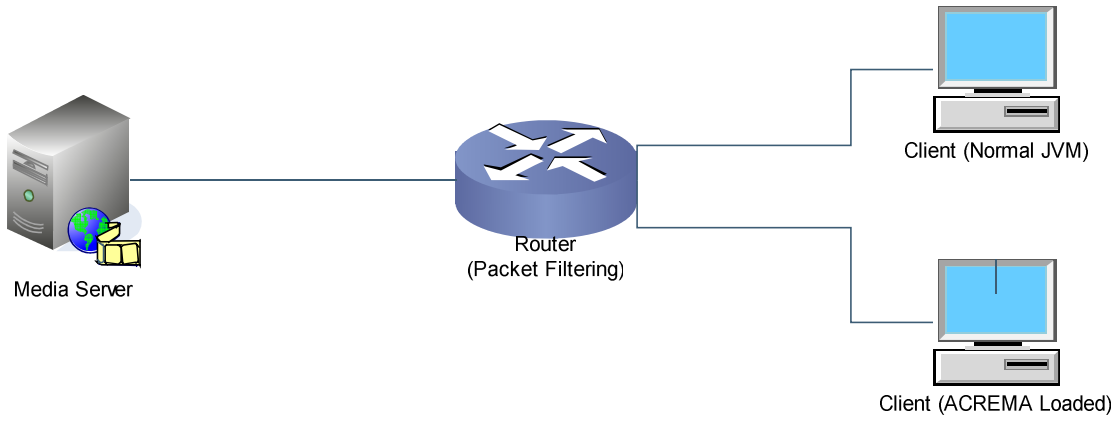


Fig 6.1 – Evaluation test bench

Using virtual machines, a network of clients and server was emulated on an Intel Dual Core, 2.0 GHz, machine such that one client is running application on ACREMA while the other without it. Different scenarios of network bandwidth variation were simulated by selective packet forwarding/dropping through the emulated router while CPU load variations were simulated by randomly, starting and stopping graphic animation applications, so that the emulated virtual machines running ACREMA were subject to varying CPU load.

Different compression schemes make use of different varying properties of the media data, therefore, to test the response of the applications running in ACREMA, a sample uncompressed movie file was streamed under ideal network and CPU conditions and sample profiles of transmission in H263 and MJPEG were generated. Having established the following relationship under ideal conditions, adaptation policy was devised.

– MJPEG bandwidth requirement >> H263 bandwidth requirement

– MJPEG Client CPU load > H263 Client CPU load

– MJPEG Server CPU load << H263 Server CPU load

Since main purpose of the work presented in this dissertation was to test the effectiveness of the principles of AoP, when applied to achieve multimedia adaptation, on by aspectizing pre-written applications in a pre-designed framework (and not to devise a comprehensive adaptation algorithm) adaptation policy was made dependent upon user's preferences, along with the default adaptation priorities coded in the internal tables.

Different aspects of overall evaluation can be categorized under three broad classes of :

- Architectural Evaluation
- Application Test Cases, and
- Qualitative Evaluation

6.2 Architectural Evaluation of ACREMA

In order to measure different architectural overheads, a high precision timer, based or the actual hardware of the machine was used by accessing the microprocessor registers. On an Intel Dual Core machine with 2GHz clock frequency, the maximum precision achieved was 25.4222 micro-seconds.

Since adaptations in ACREMA take place in two different phases: Static Composition Phase and Dynamic Reconfiguration Phase, ACREMA's own architectural evaluation was done in terms of load time and runtime latencies. Load time latencies occur due to that fact that the application code in Aspectized at load time, while runtime latencies are the adaptation invocation latencies. Fig 6.2 shows Static Composition Phase latencies averaged over 10 runs of test cases.

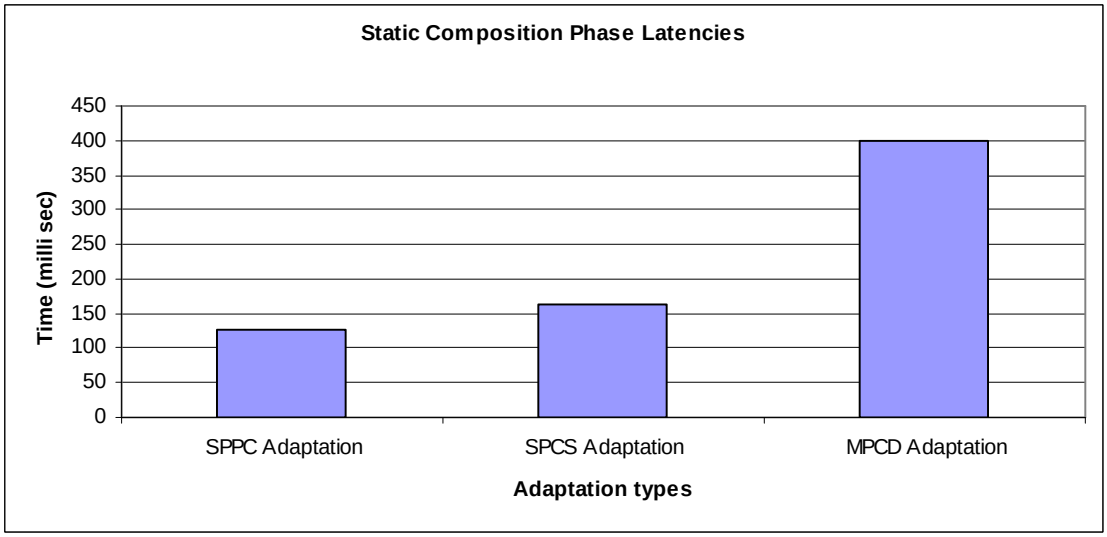


Fig. 6.2 –Adaptation Composition Latency Graph (load time)

Since CIED Adaptations rely on event interception only, the latencies in this case are only runtime latencies. For the other three types (SPPC, MPCS and MPCD Adaptations), latencies are mainly loadtime. For these 3 different cases of adaptation ranged from 125.792 milli-sec in case of SPPC Adaptations (minimum) to 398.8715 milli-sec in case of Dynamic Flow Diversion, where adaptive code for dynamic codec swap was to be injected codec swap was involved at runtime. This figure may seem a significant overhead, however it is worth noting, that all this overhead is load-time overhead.

Table 6.1 and fig. 6.3 show relative adaptation invocation latencies for different invocations. Each bar is an average of 10 readings. The normal case depicts the case of hand-coded adaptation (in blue), while the instrumentation done by ACREMA is plotted in pink. These are the per-invocation latencies, however, in an application there may be several adaptation invocations, in which case the latencies will add up.

Adaptation	Normal	ACREMA
Bit Rate Adaptations	1525	2414
Buffer Control Adaptations	2131	2454
Format Control Adaptations	2041	2898
Frame Processing Control	2052	2884
Frame Rate Control	2048	3000
H.263 Control	2043	2083
Key Frame Control	2019	2840
Monitor Control	2061	3062
Mpeg Audio Control	2039	2844
Packet Size Control	2067	2900
Quality Control	2123	2980
RTP Control	2041	2947
Media Track Controls	2074	2836
Stream Writer Control	2021	2969

Table 6.1 – Comparison of Adaptation invocation Latencies

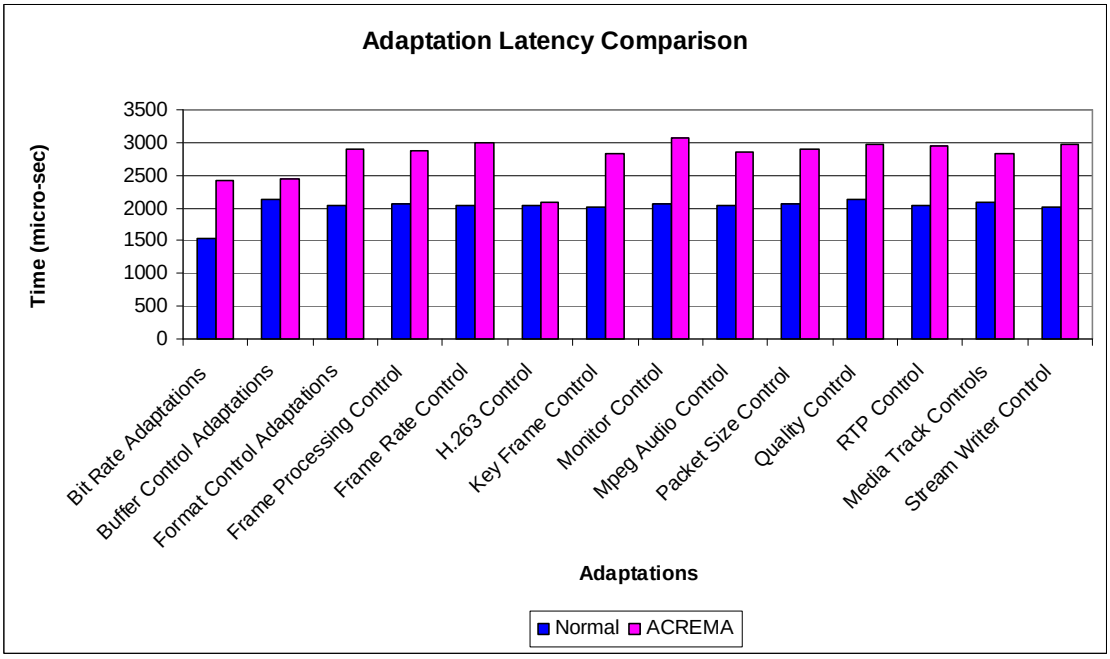


Fig 6.3 – Adaptation Invocation Latency Graph (runtime)

From the graphs of adaptation composition latencies (fig 6.2) and dynamic adaptation invocation latencies (fig. 6.3), it can be inferred that irrespective of the simplicity or

complexity of a real life adaptation situation, adaptation composition latencies are much higher than the adaptation invocation latencies. Thus ACREMA successfully shifts most of the overall temporal overhead of any adaptation to load time, while minimum (in most of the cases negligible) latencies appear during application execution time. This is in accordance with the general understanding of adaptations found in contemporary scientific literature, according to which, it's better to start an application with some delay, in order to avoid runtime disturbance, than to start an application quickly and disturb it afterwards.

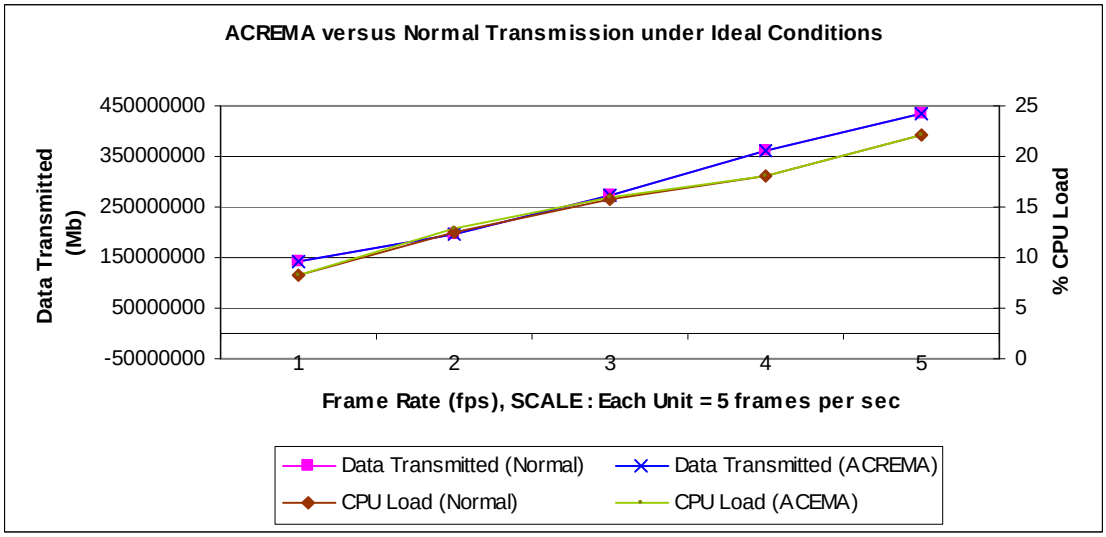


Fig 6.4 – Comparison of Network Bandwidth Requirements

ACEMA was evaluated for runtime overhead on the CPU and the network in comparison with the normal case (i-e; application running without ACREMA), under ideal conditions. The results are shown in fig. 6.4. It was observed that there is no significant load on the CPU or the Network. A minor difference can be seen from the graph, in case of CPU, however this is with in the range of experimental error. Thus under ideal network conditions, ACREMA does not overload system resources more than they will be loaded without ACREMA.

6.3 Application Test Case Evaluation

As described in (sec.4.5.1 to sec.4.5.4) and (sec.5.2.1 to sec.5.2.4), the applications can be categorized with respect to the type of code (event) interceptions (diversions) they utilize. In order to further ensure fair evaluation of ACREMA architecture and implementation, the application type used in all cases of simple and complex adaptations discussed below, was kept the same, so that any discrepancies in quantitative analysis can be avoided by keeping the measurement-reference constant for all different types of simple and complex adaptations. For the four main categories of adaptations, application performance was measured as detailed below.

The performance evaluation is based around taking pre-written non-adaptive application using JMF API and evaluating them for the efficiency of adaptation advice injection. The applications tested in this regard fall under two main categories: those which were written with minimum functionality (e.g, a simplest application consisting of an RTP server and a corresponding client), and those which the user has already written to incorporate some feedback or monitoring (a case of pre-implemented limited adaptive behavior).

In all the sample evaluations of the system presented in the subsequent sections of this chapter, the worst case scenario was put to test. The worst case scenario is defined by the following two characteristics:

- The given application's byte-code should be completely void of any kind of default adaptation behavior (so that any kind of pre-existing behavior e.g; programmed parameter tuning etc. neither adds to nor subtracts from the adaptive behavior(s) dictated to the application by ACREMA).
- The given application code and the adaptation preferences described by ACREMA, should exist in two completely independent units (so that the genuine efficiency of Aspect oriented Programming paradigm, applied in the context of this thesis can be evaluated in true spirit of the paradigm itself).

Sections 6.3.1 to 6.3.4 show single invocation of a specific type of adaptation. In these cases, except the one specific adaptation under test, all others were masked from triggering, while a case of multiple adaptations is shown in section 6.3.5.

6.3.1 Code Interception Event Diversion (CIED) Adaptations

These adaptations are the simplest to implement and have the minimum overhead, however are limited in their adaptation capabilities. It incorporates all those cases, where no additional code needs to be injected to the given application code and the adaptation policy works by hooking itself with the media processing events of the application through the *Aspects*, used by ACREMA. The figure below shows Quality Factor Adaptation of an MJPEG video stream from 8 down to 2 and then up to 10 (maximum). Quality Factor mainly effects Server CPU, this is a non-conflicting adaptation. The adaptation was triggered by starting graphic application at time point 65 which ended on 104. The adaptation latency in this case is negligible (approximately 3 milliseconds directly obtainable from fig. 6.3).

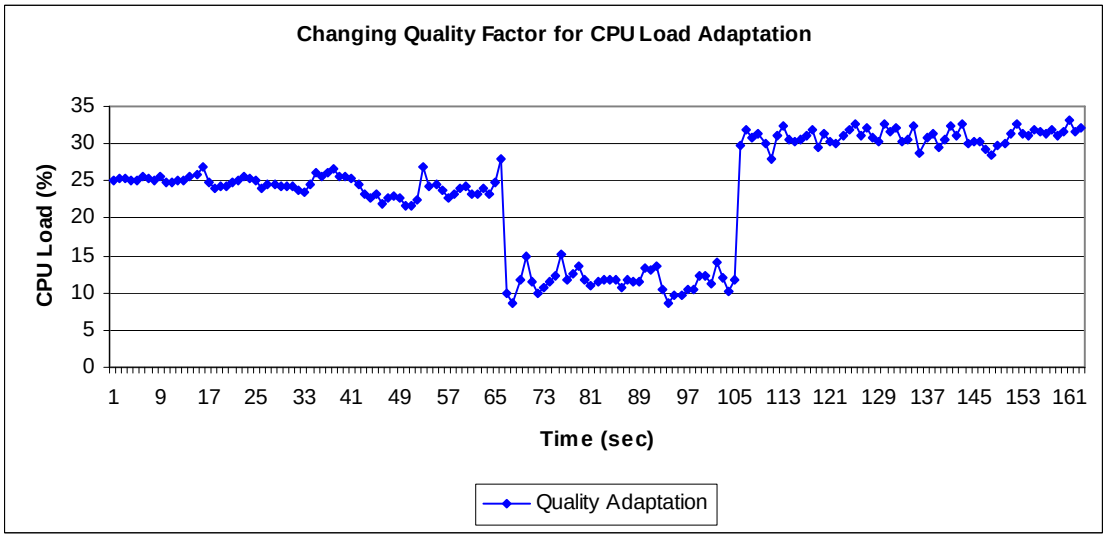


Fig 6.5 – Server CPU Load adaptation by varying MJPEG Quality Factor, with negligible adaptation latencies.

6.3.2 Static Alteration of Pre or Post Processing Chain (SPPC Adaptation)

An example of SPPC adaptation is encountered when adaptive alterations to the post processing media chain is required by swapping a heavy weight component (*DirectDrawRenderer*) with its light weight equivalent (*LightWeightRenderer*). In the example adaptation scenario shown in fig. 6.8. Since this is an adaptation that affects, the client side only, network transmission has not been shown in the figure. In the figure below, at sample time point 70, the default video renderer is swapped with its light weight equivalent and the application on the resource constrained client device, demands lesser CPU share. This is an example of a non-conflicting post-processing chain local

adaptation, which can be carried out in isolation and synchronization of the client and server is not required.

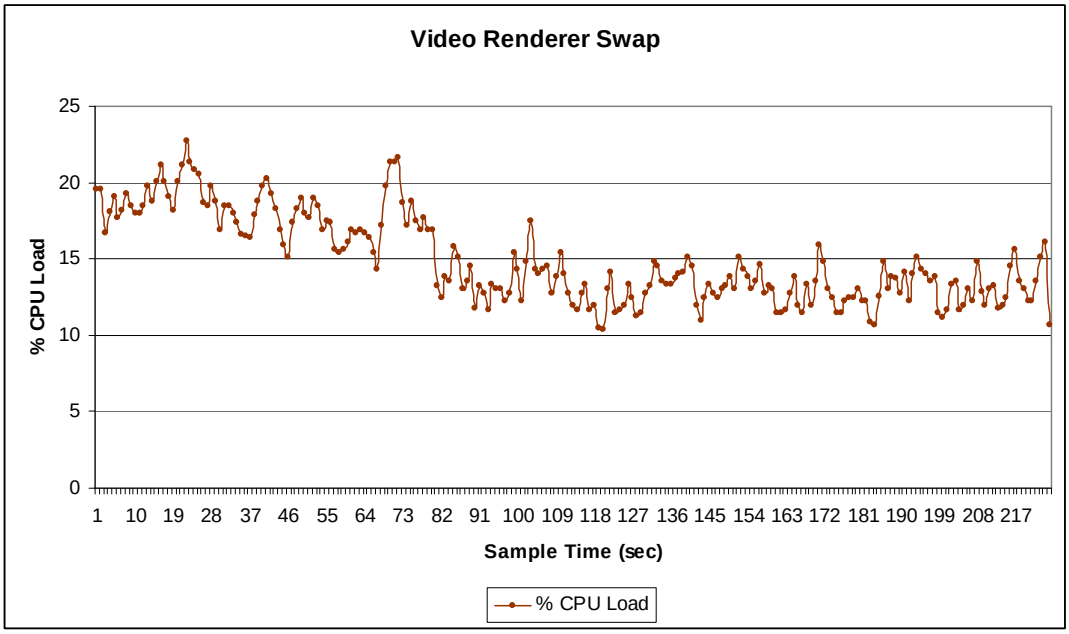


Fig 6.6 – Static Pre/Post Processing Adaptation –Example of Client-only adaptation

6.3.3 Static Alterations of Main Processing Chain

The implementation was tested to adapt a pre-written (non-adaptive) video conferencing application transmitting H.263 coded CIF size frames in WLAN, to adapt to bandwidth variations. The results are shown in the fig 6.9. The application was configured to adapt down (decrease frame rate) and adapt up (increase frame rate) according to the two threshold values (arbitrarily set). As an example of adaptation policy in this case, reaching the upper threshold value of percentage packet loss triggers the 'adapt-down' behavior, while staying in the 'adapt-down state' for five consecutive time-points triggers the 'adapt-up' behavior. In the figure, '*adapt-down*' is invoked at time-point 13, upon crossing the threshold of 5% loss, and again at time-point 25, while '*adapt-up*' is invoked at after 5 stable time-points at 30. (The experimental results shown in this case were obtained by simulating the scenario on a 1.1 Ghz, single processor PC).

Both the *adapt-down* and *adapt-up* operations take place instantaneously, with a very small invocation latency.

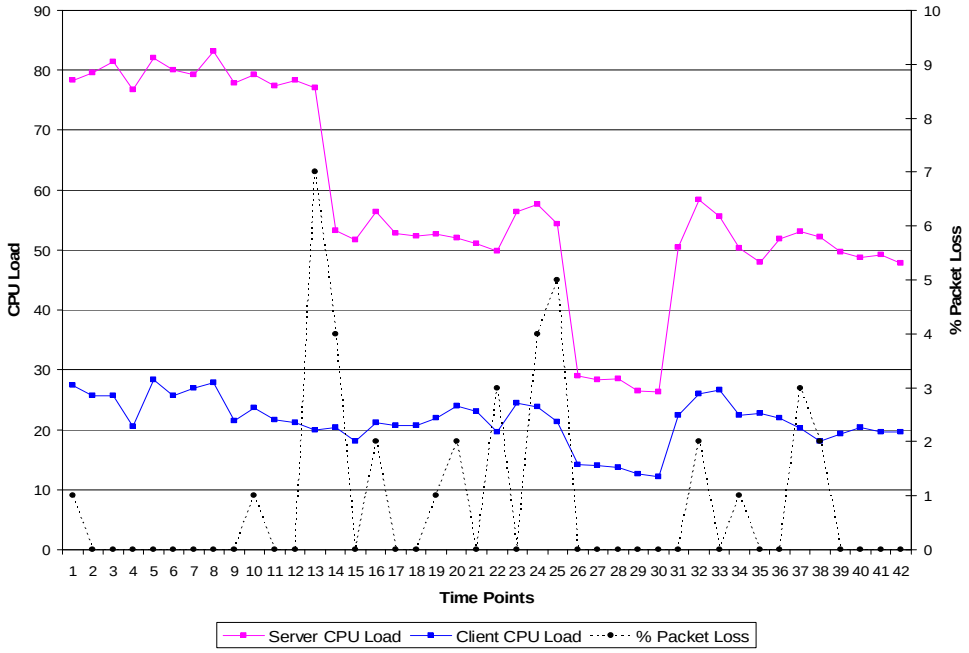


Fig 6.7 – Main Processing Chain Static Adaptation – Changing frame-rate in H.263 video

6.3.4 Dynamic Data Flow Diversion

This type of adaptation is the most resource intensive and incurs the heaviest load-time overhead. The performance of ACREMA for such adaptations was measured by starting

a client-server session, where initial transmission was in MJPG format in a stable system state. At time=35, packet loss starts increasing and the system remains stable no longer. As the packet loss continues for the next 5 points, a format adaptation is triggered, which results in stopping the current flow, synchronizing and starting a new media data flow with H.263 codec. However, as soon as ACREMA starts adaptation operation, and both client and server synchronize to initiate the codec swap and flow diversion operation, CPUs on both sides experience a heavy load. It is notable that dynamic behavior of this adaptation is resource-conflicting (i-e; adapting the application to relieve one resource, result in temporary overload of another)[9]. Fig 6.8. a significantly long adaptation latency due to the same reason. The data transmission even drops down to zero, which occurs due to switching from one media processing chain (chain swapping discussed in chapter 3) to another.

[9] This also explains presence of a small wait time between successive adaptations, in the adaptation policy, because, if the adaptation policy is void of a small wait-time between two consecutive adaptations, the system may go into a thrashing state, where, the overall resource consumption by the adaptation engine would increase the overall resource consumption by the application under test.

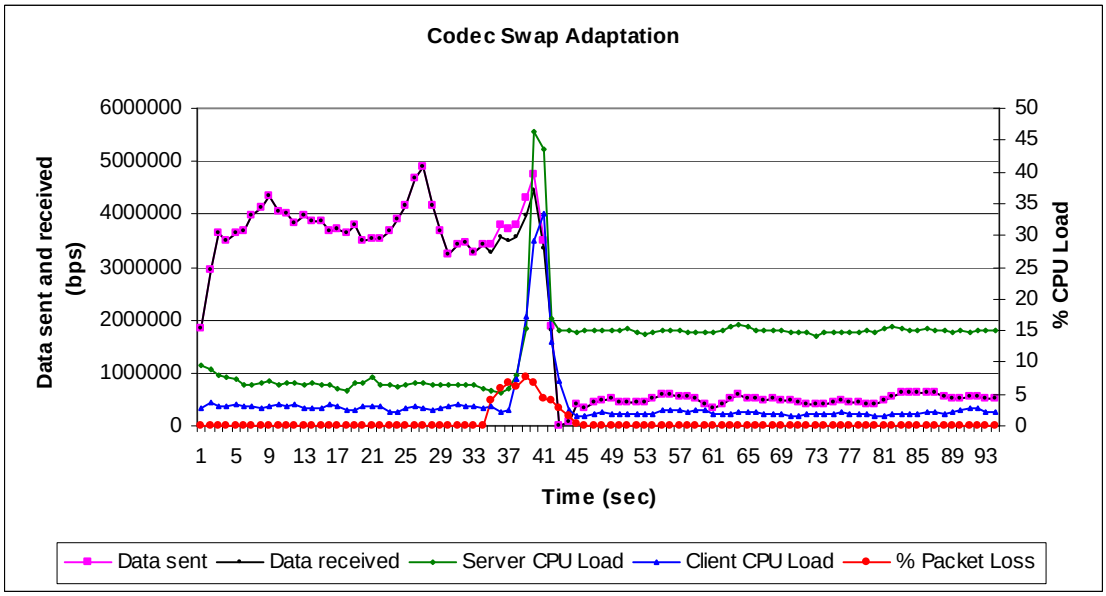


Fig 6.8 – Codec Swap Adaptation – Conflicting Adaptation Example

6.3.5 Multiple Adaptation Application Test Case

Fig 6.9 shows results of mixed and multiple frame-rate and size adaptation invocations on an MJPG video in response to simulated packet loss due to custom firewall switching on and off on the router, shown in fig 6.1. The allocated bandwidth (shown red) in fig 6.9 was periodically increased and decreased in several steps to observe the adaptation response. It may be noticed that this variation in allocated bandwidth stays constant for some time before changing. It is due to the fact that adaptation is not based on instantaneous values, but uses a moving average instead. Moreover, whenever there is a need to adapt, the adaptation process is not very abrupt. Also, at times, when the packet loss reduces to zero, if the adaptation engine has already adapted-down, then it may trigger adapt-up operation, however, at the same time, the allocated bandwidth may further reduce. An example of such a deceptive decision may be observed at point 25, 100 and 101 and 105 where the already adapted application is trying to adapt-up, while the allocated bandwidth is still not sufficient.

It may also be observed that even in presence of ACREMA, although the percentage packet loss decreases considerably, it is not zero, because, whenever an adaptation is

invoked, it is triggered with some delay (a 'lazy' response). The adaptation is invoked on the basis of packet loss reported by a three-term moving-averager, therefore the response in not instantaneous.

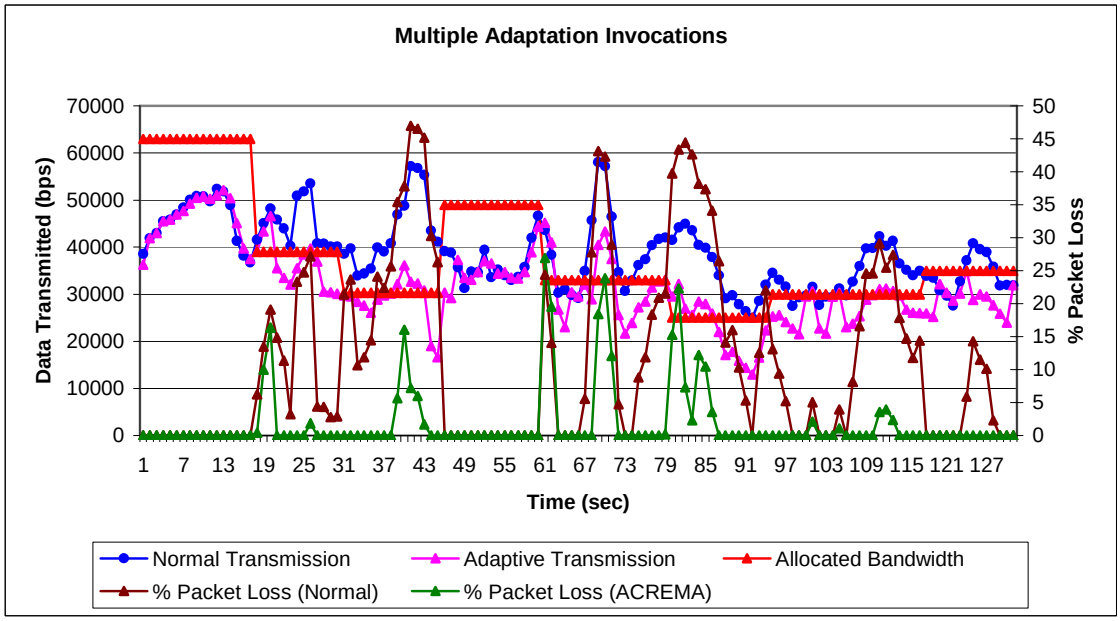


Fig 6.9 – Multiple adaptation invocations in a complex real-life situation

At the same time it can also be observed that ACREMA can reduce packet loss by a significant amount (the actual quantity will depend upon a number of factors and varies from case to case).

6.4 Qualitative Evaluation

Qualitative evaluation can be done in the cases, where quantitative evaluation is not possible. In case of ACREMA, the following factors may be considered for qualitative evaluation.

6.4.1 Scalability

Scalability can be addressed from two view points: scalability of the software system as well as scalability of performance of the system, with respect to multimedia application type and size. With respect to the size of application (software), ACREMA,

can adaptively aspectize an application of any size, however, if complexity of adaptation requirements to be fulfilled is increased by the user, then the latencies will increase correspondingly. The exact increase in latencies will be dependent upon the size and type of the pre-written non-adaptive software application.

With respect the complexity of the multimedia, ACREMA should scale very well, because communication overhead of ACREMA is negligible. There may be some inherent limitation on communication scalability due to RTCP, which can be attributed to the protocol (not ACREMA). Runtime computational overhead depends mainly upon the number of times adaptation is invoked. Static computational overhead will be considerable, however, it can be tolerated in most of the cases, since it occurs only at load time.

6.4.2 Generality

Although, the implementation presented here has been specific to Java Media Framework, its' architecture relies on fundamental principles of *reflection* and *Aspect oriented Programming.* This makes ACREMA a system capable of being developed and extended to various other languages and media frameworks.

6.4.3 Co-Existence

ACREMA in its present state is limited to application oriented adaptations, which don't rely on any specific network service and are not bound to any particular protocol. Although it has been tested with RTP/RTCP, since JMF facilitates use of custom transport, ACREMA is not limited to a specific protocol too. It does not interfere with any other adaptation system existing on the same device which may be meant to handle adaptive features of other applications. Therefore, it can co-exist on a system with any other such system, as long as ACREMA is independently responsible for adapting multimedia applications.

6.4.4 Limitations

The limitations described in this section relate to display of unpredictable behavior. It can mainly happen when the user deliberately incorporates some code which can trigger any actions, and the events, method signatures or the system elements, involved with realization of those behaviors are the same which are used by ACREMA to inject automated adaptations. The reason for this anomalous response is the fact that, actions carried out depend upon the order in which the events are caught by the user application

and the ACREMA. This limitation, however, occurs only when the user violates the fundamental principle of keeping the behavioral aspects separate from the application core functionality and also by making some programming error (i-e, the limitation is only seen in the cases, where the user deliberately programs an application to defeat the system!).

Chapter 7

Outlook and Future Directions

7.1 Contributions

While in existing research and most of contemporary work, middleware has been extensively used as a means to achieve objectives like distribution transparency by increasing the level of abstraction, the abstraction achieved so far is still limited to the extent that requires an application developer to develop additional code which serves as an interface between the application and the middleware. During application loadtime and execution the middleware stays as a separate layer underneath the application. This adds to the processing overhead on one hand and the abstraction provided is still limited since some 'interfacing code' needs to be developed by the application developer on the other hand. The Aspect oriented approach presented in this dissertation increases the level of abstraction by providing the application developer means of configuration (instead of requiring to develop 'interfacing code'), reduces the software footprint since the middleware does not exist as a separate layer underneath the application while increasing the execution efficiency at the same time, since there are no extra copying and un-copying operations involved in contrast with middleware. However, all this is achieved at the cost of being a domain-specific environment (i-e., works with multimedia applications only), which may not be seen as a major limitation in this case, since the aim was to solve a domain specific problem. The contribution of this work lies in the fact that two conflicting properties of *efficiency* and *abstraction* which have been difficult to handle in contemporary middleware implementations has been successfully (but domain specifically) tackled in this dissertation.

Table 7.1 summarizes the contribution of this work along with a comparison with existing solutions.

Existing Solutions	ACREMA	Comparison
Middleware layer exists as a separate layer beneath the application. Additional layer is not suitable in case of multimedia applications because it will introduce un-necessary overhead in copying packets and transfer of other control signals to and from this layer. Even those approaches which handle stream based flows by bypassing this layer, do so partially and they are unable to properly handle application-side adaptations.	The System Software Layer responsible for adaptation exists separately at the time of Static Composition (giving the benefits of separation of functional and behavioral concerns), but weaves itself into application at runtime (thus giving the performance benefits of embedding adaptation code into the application.	ACEMA ↑↑[10]
The programmer is required to program QoS or Adaptation behavior using some purpose-built API, which may at times involve some kind of knowledge of the runtime circumstances of the system. (precise knowledge of this type is generally not available due to complex runtime system resource and requirements fluctuations)	The programmer does not need to program using any purpose-built API and knowledge of complex adaptations is also not required. Only users preferences, together with the application type and device profiles will generate the adaptation requirements and program them.	ACREMA ↑
Mainly target system-side adaptations, involving pre-hand resource reservation, which can lead to waste of reserved resources in case of multimedia applications, which can tolerate some QoS degradation	Mainly targets application-side adaptations, involving no pre-hand resource reservation. In case of multimedia applications which are mostly elastic in nature, this will generally do well, however in some cases, this may temporary disrupt the communication.	Generally, ACREMA ↑ In some cases, ACREMA ↓

Table 7.1 – Summary of Contributions

7.2 Future Extensions

Having presented and evaluating ACREMA in the last few chapters, future work can take mainly two directions:

---

[10] ACEMA ↑↑ denotes ACREMA is much better. ACREMA ↑ denotes ACEMA is better.

7.2.1 Software Related Extensions

- Since *Aspects* are mainly used for software re-factoring and handling concerns spread across several modules, ACREMA was developed to address unmanaged application-oriented adaptation. Since Aspects can be used to make adaptive modifications to existing non-adaptive code and also because adaptation, by its very nature is a cross-cutting concern. In addition to that, ACREMA can be extended to incorporate any kind of cross-layer or resource-oriented adaptations. Therefore, system-oriented QoS work may be combined to complement the solution provided and a comprehensive approach for cross-layer adaptation (including the application-side adaptations) can be realized, to address the QoS and Adaptation problem more comprehensively.

- With respect to support for multimedia development frameworks, the current implementation can be extended with DSJ[Direct Show Java], to support a more feature rich multimedia API. Since the prototype implementation was based around JMF, Mobile Multimedia API (MMAPI), can be easily supported on the client side. The architecture of ACREMA relies on fundamental principles of *reflection* and *Aspect oriented Programming* and not only reflective features are now supported by most of contemporary languages, a wide range of aspect weavers also exists for a number of languages. Therefore, the same system can be implemented in other languages (e.g; on .Net platform, C# or others). This will enable support for multimedia frameworks already developed in other languages (e.g; largely open source G-Streamer).

- The adaptation process handled in current implementation was based on a number of *if-then-else* decisions. However, the adaptation process by its very nature can be better handled using fuzzy / neural algorithms. Therefore, future work may also be in the direction of integrating such adaptation algorithms.

7.2.2 Dynamically Reconfigurable Hardware Related Extensions

During the course of this project the following two observations were made regarding computational overhead:

- It was observed a number of times, that modifying a pre-written software can involve significant amount of overhead, which in case of multimedia can be intolerable at times. For example in case of codec swap in particular, both the Client and Server CPUs are overloaded, due to the requirement to load a number of java classes, since most of the codecs use Java Native Interface (JNI), this process is slowed down and synchronizing media streams further slows down the over all task.
- Multimedia processing (like encoding and transcoding in particular) involve a number of mathematical operations, due to which performance of a software codec is not as good as a hardware codec.

Considering implementation of such operations in hardware, in particular dynamically reconfigurable FPGA's seems promising. In this regard, researchers have already done some fruitful work using dynamically partially reconfigurable hardware, where different hardware configurations are used by dynamically changing data paths to switch across several algorithm implementations. For example, [AFK+2006] and [PAK+2007] provide such a solution for network processors, while [CHP+2006] presents a reconfigurable multimedia audio player on FPGA, which can download a configuration bit-stream from a remote database, in case of a codec being absent and reconfigure the FPGA with the new codec.

Another motivating factor that suggests the implementation of ACREMA's architecture on dynamically reconfigurable hardware is due to the fact that *Aspects* offer a very good way of code instrumentation, and the applicability of the *Aspect Oriented Programming* to modular reconfigurable computing has been formally studied and proved in [VB2007].

As a natural extension of the work presented in this dissertation, implementation of ACREMA's architecture in dynamically reconfigurable hardware is therefore proposed. The proposed implementation will mainly consist of a general purpose CPU and an FPGA which can be dynamically, partially reconfigured (e.g., Xilinx Virtex II Pro) on board. Communication between the JVM running on the CPU and codecs implemented in the FPGA can be established by using JNI. Using *Aspect Oriented Programming*, all responses to handle transcoding and codec swap adaptations (which have a very high overhead) can be intercepted and served by the hardware codecs implemented inside the FPGA. As swapping a codec in software involves loading a number of new classes,

unloading the already loaded ones, running garbage collector, synchronizing media streams etc, the CPU is overloaded in the process and the user notices a temporary transmission interruption. In case of implementation on dynamically reconfigurable FPGA, AOP may be used to intercept the class loading operation. Swapping a codec in hardware will then be a matter of loading the reconfiguration bit-stream to one area of the FPGA while the other is still in use (which is actually a very fast operation). The data flow paths can then be attached to the newly installed codec.

Since different codecs share several common internal functional units, on a runtime reconfigurable hardware, exploring a finer degree of runtime reconfigurability, such that codec are partially swapped would also be of interest. It is believed that in case of successful implementation of the proposed system, ACREMA's performance can be enhanced many folds and a device as small as a mobile phone may be powerful enough to adaptively transcode multimedia!

List of Abbreviations

Acronym	Definition
ACE	Adaptive Communication Environment
ACM	Association for Computing Machinery
ACREMA	Adaptation Composition and Reconfiguration Environment for Multimedia Applications
AI	Artificial Intelligence
AOP	Aspect-Oriented Programming
AOSD	Aspect Oriented Software Development
API	Application Programming Interfaces
ASM	Assembler (A Byte Code Instrumentation Tool)
BCEL	Byte Code Engineering Library
CBSE	Component Based Software Engineering
CCM	CORBA Component Model
CIAO	Component Integrated ACE Orb
CIED	Code Interception Event Diversion
CIF	Common Intermediate Format
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
DCE	Distributed Computing Environment
DCOM	Distributed Component Object Model
DDE	Dynamic Data Exchange
DRE	Distributed Realtime Embedded
DVS	Dynamic Voltage Scaling
EJB	Enterprise Java Beans
FGS	Fine Grain Scaling
GIOP	General Inter-Orb Protocol
GPRS	General Packet Ratio Service
GSM	Global System for Mobile communications
GUI	Graphical User Interface
IDL	Interface Description Language
IEEE	Institute of Electrical and Electronics Engineers
IIOP	Internet Inter-Orb Protocol
IP	Internet Protocol
ITU	International Telecommunication Union
J2EE	Java 2 Enterprise Edition

Acronym	Definition
JDK	Java Development Kit
JMF	Java Media Frameworks
JPEG	Joint Photographic Experts Group
JVM	Java Virtual Machine
LCD	Liquid Crystal Display
LVM	Linux Virtual Machine
MAC	Medium Access Control
MJPEG	Motion JPEG
MJPG	Motion JPEG
MMAPI	Mobile Media Application Programming Interface
MOP	Meta Object Protocol
MPCD	Main Processing Chain Dynamic
MPCS	Main Processing Chain Static
MPEG	Motion Picture Experts Group
OMG	Object Management Group
ORB	Object Request Broker
QCIF	Quarter Common Intermediate Format
QDL	Quality Description Language
RGB	Red Green Blue
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RSVP	ReSerVation Protocol
RTP	Real Time Protocol
RTCP	Real Time Control Protocol
SMIL	Synchronized Multimedia Integration Language
SNR	Signal to Noise Ratio
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPPC	Simple Pre/Post Processing Chain
SQCIF	Sub-Quarter Common Interchange Format
TCP	Transmission Control Protocol
UAV	Unmanned Arial Vehicle
UML	Unified modeling Language
WCML	Web Clipping Markup Language
WLAN	Wireless Local Area Network
WSDL	Web Service Definition Language
XML	eXtensible Markup Language

Bibliography

[AFK+2006] Albrecht, C., Foag, J., Koch, R., Maehle, E. **DynaCORE – A Dynamically Reconfigurable Coprocessor Architecture for Network Processors**. In *Euromicro Conference on Parallel, Distributed and Network-Centric Processing*. 2006.

[AG2005] Andreas Frei and Gustavo Alonso. **A Dynamic Lightweight Platform for Ad-Hoc Infrastructures**. In *PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*. 2005.

[ALD+2003] Atallah S.B. and Layaida O. and De Palma N. and Hagimont D., *Dynamic Configuration of Multimedia Applications*, **N PROC. OF THE 6TH IFIP/IEEE INTERNATIONAL CONFERENCE ON MANAGEMENT OF MULTIMEDIA NETWORKS AND SERVICES (MMNS'03). BELFAST SEPT** (2003) : .

[APC+2004] Ali N. and Perez J., *Implementation of the PRISMA Model in the. Net Platform*, **PROC. OF DYNAMICA M{\'A** (2004) : .

[ASJ+2003] Ali NH and Silva J. and Jaen J. and Ramos I. and Carsi JA and Perez J., *Mobility and Replicability Patterns in Aspect-Oriented Component-Based Software Architectures*, **PROCEEDINGS OF** (2003) 15: p. 820--826.

[ASM] . http://asm.objectweb.org/.

[BCA+2001] Blair G.S. and Coulson G. and Andersen A. and Blair L. and Clarke M. and Costa F. and Duran-Limon H. and Fitzpatrick T. and Johnston L. and Moreira R. and others, *The Design and Implementation of Open ORB 2*, **IEEE DISTRIBUTED SYSTEMS ONLINE** (2001) 2: p. 1--40.

[BCB+2002] Blair Gordon S. and Coulson Geoff and Blair Lynne and Duran-Limon Hector and Grace Paul and Moreira Rui and Parlavantzas Nikos. **Reflection self-awareness and self-healing in OpenORB**. In *WOSS '02: Proceedings of the first workshop on Self-healing systems*. 2002.

[BCC+1999] Gordon S. Blair and Fabio M. Costa and Geoff Coulson and Hector A. Duran and Nikos Parlavantzas and Fabien Delpiano and Bruno Dumant and Horn and Jean-Bernard Stefani. **The Design of a Resource-Aware Reflective Middleware Architecture**. In *Reflection '99: Proceedings of the Second International Conference on Meta-Level Architectures and Reflection*. 1999.

[BCD+1997] Blair G.S. and Coulson G. and Davies N. and Robin P. and Fitzpatrick T., *Adaptive Middleware for Mobile Multimedia Applications*, **PROCEEDINGS OF THE 8TH INTERNATIONAL WORKSHOP ON NETWORK AND OPERATING SYSTEM SUPPORT FOR DIGITAL AUDIO AND VIDEO (NOSSDAV)** (1997) : p. 259--273.

[BCEL] . http://bcel.sourceforge.net/.

[BG1997a] Becker C. and Geihs K., *MAQS-Management for Adaptive QoS-enabled Services*, **PROCEEDINGS OF IEEE WORKSHOP ON MIDDLEWARE FOR DISTRIBUTED REAL-TIME SYSTEMS AND SERVICES** (1997) : .

[BG1998] Becker C. and Geihs K., *Quality of Service—Aspects of Distributed Programs*, **INTERNATIONAL WORKSHOP ON ASPECT-ORIENTED PROGRAMMING AT ICSE'98 KYOTO/JAPAN 1998** (1998) : .

[BHS+2004] Becker C. and Handte M. and Schiele G. and Rothermel K., *PCOM-a component system for pervasive computing*, **PERVASIVE COMPUTING AND COMMUNICATIONS 2004. PERCOM 2004. PROCEEDINGS OF THE SECOND IEEE ANNUAL CONFERENCE ON** (2004) : p. 67--76.

[BN2004a] Becker C. and Nicklas D., *Where do spatial context-models end and where do ontologies start? A proposal of a combined approach*, **PROCEEDINGS OF THE FIRST INTERNATIONAL WORKSHOP ON ADVANCED CONTEXT MODELLING REASONING AND MANAGEMENT IN CONJUNCTION WITH UBICOMP 2004** (2004) : .

[BS2003] Becker C. and Schiele G., *Middleware and application adaptation requirements and their support in pervasive computing*, **DISTRIBUTED COMPUTING SYSTEMS WORKSHOPS 2003. PROCEEDINGS. 23RD INTERNATIONAL CONFERENCE ON** (2003) : p. 98--103.

[BSG+2003] Becker C. and Schiele G. and Gubbels H. and Rothermel K., *BASE-a micro-broker-based middleware for pervasive computing*, **PERVASIVE COMPUTING AND COMMUNICATIONS 2003.(PERCOM 2003). PROCEEDINGS OF THE FIRST IEEE INTERNATIONAL CONFERENCE ON** (2003) : p. 443--451.

[CA2000] Cazzola W. and Ancona M., *mChaRM: a Reflective Middleware for Communication-Based Reflection*, **DISI UNIVERSITA DEFLI STUDI DI MILANO TECHNICAL REPORT: DISI-TR-00-09** (2000) : .

[CC2003] Chan ATS and Chuang SN, *MobiPADS: A Reflective Middleware for Context-Aware Mobile Computing*, **IEEE TRANSACTIONS ON SOFTWARE ENGINEERING** (2003) 29: p. 1072 -- 1085.

[CHP+2006] Castillo, J., Huerta, P., Pedraza, C., Martinez, J. I. **A Self-Reconfigurable Multimedia Player on FPGA**. , 2006.

[CKP2003] Kihwan Choi and Kwanho Kim and Massoud Pedram. **Energy-aware MPEG-4 FGS streaming**. In *DAC '03: Proceedings of the 40th conference on Design automation*. 2003.

[DC2001] Jim Dowling and Vinny Cahill. **The K-Component Architecture Meta-model for Self-Adaptive Software**. In *REFLECTION '01: Proceedings of the Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*. 2001.

[DLS+2004] Duzan G. and Loyall J. and Schantz R. and Shapiro R. and Zinky J., *Building adaptive distributed applications with middleware and aspects*, **PROCEEDINGS OF THE 3RD INTERNATIONAL CONFERENCE ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT** (2004) : p. 66--73.

[DM1995] Demers F.N. and Malenfant J., *Reflection in logic functional and object-oriented programming: a short comparative study*, **WORKSHOP ON REFLECTION AND METALEVEL ARCHITECTURES AND THEIR APPLICATIONS IN AI. IJCAI** (1995) 95: pp. 29-38.

[Don1997] Don Box, *Essential COM*, (1997) : .

[ECD+2001] Efstratiou C. and Cheverst K. and Davies N. and Friday A., *Architectural requirements for the effective support of adaptive Mobile applications*, **PROCEEDINGS OF THE SECOND INTERNATIONAL CONFERENCE ON MOBILE DATA MANAGEMENT** (2001) : .

[EKP+2000] Eliassen F. and Kristensen T. and Plagemann T. and Rafaelsen H.O., *MULTE-ORB: Adaptive QoS aware binding*, **WORKSHOP ON REFLECTIVE MIDDLEWARE (RM 2000). NEW YORK USA** (2000) : .

[FPA2003] Frei A. and Popovici A. and Alonso G., *Event based systems as adaptive middleware platforms*, **WORKSHOP OF THE 17TH EUROPEEAN CONFERENCE FOR OBJECT-ORIENTED PROGRAMMING JULY** (2003) : .

[FRS2000] Foster I. and Roy A. and Sander V., *A quality of service architecture that combines resourcereservation and application adaptation*, **QUALITY OF SERVICE**

2000. IWQOS. 2000 EIGHTH INTERNATIONAL WORKSHOP ON (2000) : p. 181--188.

[GB2001] Geihs K. and Becker C., *A framework for re-use and maintenance of Quality of Servicemechanisms in distributed object systems*, **SOFTWARE MAINTENANCE 2001. PROCEEDINGS. IEEE INTERNATIONAL CONFERENCE ON** (2001) : p. 470--478.

[GGM1993] Garbinato B. and Guerraoui R. and Mazouni K., *Distributed Programming in GARF*, **PROCEEDINGS OF THE WORKSHOP ON OBJECT-BASED DISTRIBUTED PROGRAMMING** (1993) : p. 225--239.

[GHJ+1995] Gamma Erich and Helm Richard and Johnson Ralph and Vlissides John. Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley Professional Computing Series). . Addison-Wesley Professional, 1995.

[Garlan2001] Garlan D., *Aura: Distraction-Free Ubiquitous Computing.*, **ENGINEERING FOR HUMAN-COMPUTER INTERACTION, 8TH IFIP INTERNATIONAL CONFERENCE, EHCI 2001, TORONTO, CANADA, REVISED PAPERS** (2001) : pp. 1-2.

[HBG+1998] Franz J. Hauck and Ulrich Becker and Martin Geier and Erich Meier and Uwe Rastofer and Martin Steckermeier. **The AspectIX ORB Architecture**. In *Object-Oriented Technology ECOOP'98 Workshop Reader*. 1998.

[HBG+2001] Hauck, F. J., Becker, U., Geier, M., Meier, U., Rastofer, U., Steckermeier, M.. **AspectIX: a Quality-Aware, Object-Based Middleware Architecture.**. In *3rd IFIP International Conference on Distributed Applications and Interoperable Systems*. 2001.

[HCG2001] Hunleth F and Cytron R and Gill C, *Building Customizable Middleware using Aspect Oriented Programming*, **WORKSHOP ON ADVANCED SEPARATION OF CONCERNS (OOPSLA?01)** (2001) : .

[JAC] . http://jac.objectweb.org/.

[JC2001] Jim Dowling and Vinny Cahill, *Dynamic Software Evolution and the K-Component Model*, **WORKSHOP ON SOFTWARE EVOLUTION, OOPSLA 2001** (2001) : .

[JV2004] Jim Dowling and Vinny Cahill, *Self-Managed Decentralised Systems using K-Components and Collaborative Reinforcement Learning*, **PROCEEDINGS OF THE WORKSHOP ON SELF-MANAGED SYSTEMS (WOSS'04)** (2004) : .

[KDP+2002] Karunanidhi A. and Doermann D. and Parekh N. and Rautio V., *Video analysis applications for pervasive environments*, **PROC. 1 STINTERNATIONAL CONFERENCE ON MOBILE AND UBIQUITOUS MULTIMEDIA OULU FINLAND** (2002) : p. 48--55.

[KF2005] Khan, M. A., Fischer, S. **A Reflective Runtime Environment for Dynamic Adaptation of Streaming Media on Resource Constrained Devices**. , 2005.

[KK2000] Kuo, G.S. and Ko, P.C.. **Dynamic RSVP forMobile IPv6 in Wireless Networks**. In *51st IEEE Vehicular Techonology Conference*. 2000.

[KKP2001] Kristensen T K.T.A.P.T., *Implementing configurable signalling in the MULTE-ORB*, **IN 4TH IEEE CONFERENCE ON OPEN ARCHITECTURES AND NETWORK PROGRAMMING (IEEE OPENARCH'01)** (2001) : .

[KKS+2003] Krishna, A.S. Klefstad, R. Schmidt, D.C. Corsaro, A.. **Towards Predictable Real-time Java Object Request Brokers**. In *The 9th IEEE Real-Time and Embedded Technology and Applications Symposium, 2003.*. 2003.

[KLM+1997] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V.,

Loingtier, J.M., Irwin, J.,. **Aspect-Oriented Programming**. In *European Conference on Object-Oriented Programming (ECOOP)*. 1997.

[KR1991] Kiczales, G and Des Rivieres, J. The art of metaobject protocol. . MIT Press Cambridge MA USA, 1991.

[KRC+2000] Fabio Kon and Roy H. Campbell and M. Dennis Mickunas and Klara Nahrstedt and Francisco J. Ballesteros. **2K: A Distributed Operating System for Dynamic Heterogeneous Environments**. In *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing (HPDC'9)*. 2000.

[KRL+2000] Kon F. and Rom{\'a, *Monitoring security and dynamic configuration with the dynamicTAO reflective ORB*, **MULTIMEDIA MIDDLEWARE WORKSHOP** (2000) : p. 121--143.

[KW2001] Kshirasagar Naik and David S. L. Wei, *Software implementation strategies for power-conscious systems*, **MOBILE NETWORKS AND APPLICATIONS** (2001) 6: p. 291--305.

[Kic1991] Kiczales G.. The art of metaobject protocol. . MIT Press Cambridge MA USA, 1991.

[Kiczales1996] Kiczales G., *Aspect-oriented programming*, **ACM COMPUTING SURVEYS (CSUR)** (1996) 28: .

[LBS+1998] Loyall J.P. and Bakken D.E. and Schantz R.E. and Zinky J.A. and Karr D.A. and Vanegas R. and Anderson K.R., *QoS Aspect Languages and Their Runtime Integration*, **LECTURE NOTES IN COMPUTER SCIENCE** (1998) 1511: .

[LL2002] T. Lemlouma and N. Layada, *SMIL Content Adaptation for Embedded Devices*, **SMIL CONFERENCE EUROPE 2002** (2002) : .

[LPP+2005] Loughran N, Parlavantzas N, Pinto M, Fernández L.F, Sánchez P, Webster M and Colyer A, *AOSD-Europe-ULANC-10*, (2005) : .

[LSZ+2001] Loyall, Joseph P. Schantz, Richard E. Zinky, John A. Pratim Pal, Partha Shapiro, Richard Rodrigues, Craig Atighetchi, Michael Karr, David A. Gossett, Jeanna Gill, Christopher D.. **Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications**. In *ICDCS '01: Proceedings of the The 21st International Conference on Distributed Computing Systems*. 2001.

[MM1997] Mowbray, T. J and Malveau, R.C.. CORBA Design Pattern. . Wiley, New York, 1997.

[MV2003] MOHAPATRA S. and VENKATASUBRAMANIAN N., *Proactive Energy-Aware Streaming to Mobile Hand-Held Devices*, **PROCEEDINGS OF THE IEEE 5TH MOBILE AND WIRELESS COMMUNICATION NETWORKS (MWCN)** (2003) : .

[McAffer1995] McAffer J., *Meta-level programming with CodA*, **PROCEEDINGS OF ECOOP** (1995) 95: .

[NAN] , *http://nanning.codehaus.org/overview.html*, () : .

[OMG1995] OMG, *Common Object Request Broker: Architecture and Specification, Revision 2.0*, (1995) : .

[OMG2001] Object Management Group, *CORBA 3.0 New Components Chapters,*

OMG TC Document ptc/2001-11-03 edition, (2001) : .

[PAG2003] Popovici A. and Alonso G. and Gross T., *Just-in-time aspects: efficient dynamic weaving for Java*, **PROCEEDINGS OF THE 2ND INTERNATIONAL CONFERENCE ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT** (2003) : p. 100--109.

[PAK+2007] Pionteck, T., Albrecht, C., Koch, R., Maehle, E., Huebner, M., Becker, J. **Communication Architectures for Dynamically Recon¯gurable FPGA Designs**. , 2007.

[PHS] Pillai P. and Huang H. and Shin K.G., *Energy-Aware Quality of Service Adaptation*, () : .

[PLM+2004] Pasricha S. and Luthra M. and Mohapatra S. and Dutt N. and Venkatasubramanian N., *Dynamic Backlight Adaptation for Low-Power Handheld Devices*, **IEEE DESIGN AND TEST OF COMPUTERS** (2004) : .

[PML+2003] Pasricha S. and Mohapatra S. and Luthra M. and Dutt N. and Venkatasubramanian N., *Reducing backlight power consumption for streaming video applications on mobile handheld devices*, **PROC. FIRST WORKSHOP EMBEDDED SYSTEMS FOR REAL-TIME MULTIMEDIA** (2003) : .

[PRJ+2003] P'erez J. and Ramos I. and Ja'en J. and Letelier P. and Navarro E., *PRISMA: Towards Quality Aspect Oriented and Dynamic Software Architectures*, **3RD IEEE INTERNATIONAL CONFERENCE ON QUALITY SOFTWARE (QSIC 2003) DALLAS TEXAS USA NOVEMBER** (2003) : p. 6--7.

[PS2001] Padmanabhan Pillai and Kang G. Shin. **Real-time dynamic voltage scaling for low-power embedded operating systems**. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*. 2001.

[PS2004] C. Poellabauer and K. Schwan, *Energy-Aware Media Transcoding in Wireless Systems*, **PROCEEDINGS OF THE SECOND IEEE INTL. CONFERENCE ON PERVASIVE COMPUTING AND COMMUNICATIONS (PERCOM 2004)** (2004) : .

[PSD+2004] Pawlak R. and Seinturier L. and Duchien L. and Florin G. and Legond-Aubry F. and Martelli L., *JAC: an aspect-based distributed dynamic framework*, **SOFTWARE PRACTICE AND EXPERIENCE** (2004) 34: p. 1119--1148.

[PWK+2003] Portillo A. R, Walker S., Kirby G., and Dearly A., *A Reflective Approach to Providing Flexibility in Application Distribution*, **PROC. 2ND INTERNATIONAL WORKSHOP ON REFLECTIVE AND ADAPTIVE MIDDLEWARE**

ACM/IFIP/USENIX INTERNATIONAL MIDDLEWARE CONFERENCE (MIDDLEWARE 2003) RIO DE JANEIRO BRAZIL (2003) : .

[RC2000] Roman M. and Campbell R.H., *Gaia: enabling active spaces*, **PROCEEDINGS OF THE 9TH WORKSHOP ON ACM SIGOPS EUROPEAN WORKSHOP: BEYOND THE PC: NEW CHALLENGES FOR THE OPERATING SYSTEM** (2000) : p. 229--234.

[RHC+2002] Roman M. and Hess C.K. and Cerqueira R. and Ranganathan A. and Campbell R.H. and Nahrstedt K., *Gaia: A Middleware Infrastructure to Enable Active Spaces*, **IEEE PERVASIVE COMPUTING** (2002) 1: p. 74--83.

[RHC+2002a] Roman M and Hess C and Cerqueira R and Ranganathan A and Campbell RH and Nahrstedt K, *A middleware infrastructure for active spaces*, **PERVASIVE COMPUTING IEEE** (2002) 1: .

[RK2004] Rashid A. and Kortuem G., *Adaptation as an aspect in pervasive computing*, **WORKSHOP ON BUILDING SOFTWARE FOR PERVASIVE COMPUTING AT THE 19TH ACM SIGPLAN CONF ON OBJECT-ORIENTED PROGRAMMING SYSTEMS LANGUAGES AND APPLICATION (OOPSLA 2004) VANCOUVER CANADA** (2004) : .

[RKC1999] Roman, M., Kon, F.,and Campbell, R.H., *Design and Implementation of Runtime Reflection in Communication Middleware: The DynamicTAO Case*, **19TH IEEE CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, WORKSHOP ON E-COMMERCE AND WEB-BASED APPLICATIONS** (1999) : pp. 122-127.

[RMK+2000] Roman M, Mickunas M.D, Kon F. and Roy Campbell, *LegORB and Ubiquitous CORBA*, **IN IFIP/ACM MIDDLEWARE'2000 WORKSHOP ON REFLECTIVE MIDDLEWARE.** (2000) : .

[RP1997] Romer K. and Puder A., *MICO: CORBA 2.0 implementation*, **USER AND PROGRAMMER GUIDE COMPUTER SCIENCE DEPARTMENT UNIVERSITY OF FRANKFURT GERMANY** (1997) : .

[SC2000] Siqueira F. and Cahill V., *Quartz: A QoS Architecture for Open Systems*, **PROCEEDINGS OF IEEE INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS (ICDCS 2000)** (2000) : p. 197--204.

[SG2002] Sousa J.P. and Garlan D., *Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments*, **PROCEEDINGS OF THE 3RD WORKING IEEE/IFIP CONFERENCE ON SOFTWARE ARCHITECTURE** (2002) : p. 25--31.

[SLM1998] Schmidt D.C. and Levine D.L. and Mungee S., *Design of the TAO real-time object request broker*, **COMPUTER COMMUNICATIONS** (1998) 21: p. 294--324.

[SM2003] Saha D. and Mukherjee A., *Pervasive computing: a paradigm for the 21st century*, **COMPUTER** (2003) 36: p. 25--31.

[SMC+2004] Sadjadi S.M. and McKinley P.K. and Cheng B.H.C. and Stirewalt R.E.K., *TRAP/J: Transparent generation of adaptable java programs*, **PROCEEDINGS OF THE INTERNATIONAL SYMPOSIUM ON DISTRIBUTED OBJECTS AND APPLICATIONS (DOA'04)** (2004) : .

[SML1999] Smith J.R. and Mohan R. and Li C.S., *Scalable multimedia delivery for pervasive computing*, **PROCEEDINGS OF THE SEVENTH ACM INTERNATIONAL CONFERENCE ON MULTIMEDIA (PART 1)** (1999) : p. 131--140.

[SN2004] Steinmetz, R., Nahrstedt, K.. Multimedia Systems. . Springer, 2004.

[SVJ2003] Suvee D., Vanderperren W., Jonckers V, *JAsCo: an aspect-oriented approach tailored for component based software development*, **PROCEEDINGS OF THE 2ND INTERNATIONAL CONFERENCE ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT** (2003) : p. 21--29.

[Sch1994] Schmidt D.C.. **ACE: an Object-Oriented Framework for Developing Distributed Applications**. In *6th USENIX C++ Technical Conference*. 1994.

[Sch1999] Schmidt, D. C. and Cleeland, C., *Applying Patterns to Develop Extensible ORB Middleware*, **IEEE COMMUNICATIONS** (1999) : .

[Schmidt1998] Schmidt D.C., *An Architectural Overview of the ACE Framework: A Case-study of Successful Cross-platform Systems Software Reuse*, **USENIX LOGIN MAGAZINE TOOLS SPECIAL ISSUE NOV** (1998) : .

[Sun1997] Sun Microsystems:, *Java Remote Method Invocation Specication*, (1997) : .

[Sun2001] Sun Microsystems, *Java 2 Platform Enterprise Edition*, (2001) : .

[TBP+2005] Trumler W. and Bagci F. and Petzold J. and Ungerer T., *AMUN—autonomic middleware for ubiquitous environments applied to the smart doorplate project*, **ADVANCED ENGINEERING INFORMATICS** (2005) 19: p. 243--252.

[TSY+2004] M. Tamai and T. Sun and K. Yasumoto and N. Shibata and M. Ito, *Energy-aware Video Streaming with QoS Control for Portable Computing Devices*, **(NOSSDAV 2004)** (2004) : .

[TVJ+2001] Truyen E., Vanhaute B., Joosen W., Verbaeten P., Jørgensen B.N, *Dynamic and Selective Combination of Extensions in Component-Based Application*, **PROCEEDINGS OF ICSE** (2001) : pp. 233-242.

[VB2007] Vinh, C. P., Bowen, J. P.. **A Formal Approach to Aspect-Oriented Modular Reconfigurable Computing**. , 2007.

[Wang2003] Wang, N., Schmidt, D.C., Gokhale, A., Rodrigues C., Natarajan, B., Loyall J.P., Schantz, R.E., and Gill, C. D. QoS-enabled Middleware. Qusay H. Mahmood. Wiley and Sons, 2003.

[Wei1993] Weiser M., *Some Computer Science Problems in Ubiquitous Computing*, **COMMUNICATIONS OF THE ACM,** (1993) : .

[YKW+2002] Yau, S.S., Karim, F., Wang, Y., Wang, B. and Gupta, S.K.S, *Reconfigurable context-sensitive middleware for pervasive computing*, **PERVASIVE COMPUTING IEEE** (2002) 1: pp. 33 - 40.

[YLC+2002] Yang S. and Lee H. and Chung K. and Kim H., *A Content Provider-Specified Web Clipping Approach for Mobile Content Adaptation*, **4TH INTERNATIONAL SYMPOSIUM ON MOBILE HUMAN-COMPUTER INTERACTION** (2002) : p. 324--328.

[YN2006] Wanghong Yuan and Klara Nahrstedt, *Energy-efficient CPU scheduling for multimedia applications*, **ACM TRANSACTIONS ON COMPUTER SYSTEMS** (2006) 24: p. 292--331.

[YNG2001] Yuan W. and Nahrstedt K. and Gu X., *Coordinating energy-aware adaptation of multimedia applications and hardware resource*, **PROCEEDINGS OF THE 2001 INTERNATIONAL WORKSHOP ON MULTIMEDIA MIDDLEWARE** (2001) : p. 60--63.

[ZBS1997] Zinky J.A. and Bakken D.E. and Schantz R.E., *Architectural support for quality of service for CORBA objects*, **THEORY AND PRACTICE OF OBJECT SYSTEMS** (1997) 3: p. 55--73.

[ZDE+1993] Zhang, L., Deering, S., Estrin, D., Shenker, S., Zappala, D., *RSVP: A New Resource Reservation Protocol*, **IEEE NETWORK MAGAZINE** (1993) 7: pp. 8-18.

[ZKS+2003] Zink M., Kuenzel O., Schmitt, J.and Steinmetz, R., *Subjective Impression of Variations in Layer Encoded Videos*, **INTERNATIONAL WORKSHOP ON QUALITY OF SERVICE** (2003) : p. 137--154.

Appendix A : List of Author's Publications

Muhammad A. Khan and Stefan Fischer, "ACREMA - An Adaptive Composition and Runtime Environment for Multimedia Applications", ***in the 32nd Euromicro Conference on Software Engineering and Advanced Applications (SEEA 2006), Aug.-Sept. 2006, Cavtat, Croatia.***

Muhammad A. Khan and Stefan Fischer, "Towards Unmanaged Multimedia Adaptations using Automated Aspect Weaving", ***short paper in the International Conference on Software Engineering Research and Practice (SERP'2006), June 2006, Las Vegas, USA.***

*Muhammad A. Khan and Stefan Fischer, "*A Customizable, Reconfigurable Deployment Environment for QoS-aware Multimedia Application", ***in the 4th international workshop on Adaptive and Reflective Middleware (ARM-2005), Nov. 2005, Grenoble, France.***

*Muhammad A. Khan and Stefan Fischer, "*A Reflective Runtime Environment for Dynamic Adaptation of Streaming Media on Resource Constrained Devices*"*, ***proceedings of 38th Hawaii International Conference on System Sciences (HICSS-38), Jan 2005, Hawaii, USA.***

*Muhammad A. Khan and Stefan Fischer, "*A Reflective Runtime Environment for Dynamic Adaptation of Streaming Media*"*, ***workshop proceedings, the 5th ACM/IFIP/USENIX International Middleware Conference (Middleware 2004), Oct 2004, Toronto, Canada.***