Coordinated Adaptation for Adaptive Context-aware Applications

Christos Efstratiou

M.Sc. (Lancaster 1998)

Diploma (Patras, Greece 1996)

Computing Department

Lancaster University

United Kingdom

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

OCTOBER 2004

Abstract

Coordinated Adaptation for Adaptive Context-aware Applications

Christos Efstratiou
Computing Department
Lancaster University

Submitted for the degree of Doctor of Philosophy.
October 2004.

The ability to adapt to change is critical to both mobile and context-aware applications. This thesis argues that providing sufficient support for adaptive context-aware applications requires support for *coordinated adaptation*. Specifically, the main argument of this thesis is that coordinated adaptation requires applications to delegate adaptation control to an entity that can receive state information from multiple applications and trigger adaptation in multiple applications. Furthermore, coordination requires support for reconfiguration of the adaptive behaviour and user involvement. Failure to support coordinated adaptation is shown to lead to poor system and application performance and insufficient support for user requirements.

An investigation of the existing state-of-the-art in the areas of adaptive and context-aware systems and an analysis of the limitations of existing systems leads to the establishment of a set of design requirements for the support of coordinated adaptation. Specifically, adaptation control should be decoupled from the mechanisms implementing the adaptive behaviour of the applications, applications should externalise both state information and the adaptive mechanisms they support and the adaptation control mechanism should allow modifications without the need for re-implementation of either the application or the support platform.

This thesis presents the design of a platform derived from the aforementioned requirements. This platform utilises a policy based mechanism for controlling adaptation. Based on the particular requirements of adaptive context-aware applications a new policy language is defined derived from Kowalsky's Event Calculus logic programming formalism. This policy language allows the specification of policy rules where conditions are defined through the expression of temporal relationships between events and entities that represent duration (i.e. *fluents*). A prototype implementation of this design allowed the evaluation of the features offered by this platform. This evaluation reveals that the platform can support coordinated adaptation with acceptable performance cost.

Declaration

This thesis has been written by myself, and the work reported herein is my own. Many of the ideas in this thesis were the product of discussions with my supervisors Prof. Nigel Davies and Dr. Adrian Friday. The work reported in this thesis has not been previously submitted for a degree in this, or any other form.

Christos Efstratiou

Acknowledgements

My most sincere thanks are due to my first supervisor Prof. Nigel Davies. Throughout my work his remarkable ability to see beyond the obvious and his insightful comments helped me get passed some difficult obstacles in my research. I want to thank him for his patient supervision and most importantly the dedicated support he offered me during the writing of this thesis. Moreover, I owe him thanks for the research opportunities he offered me beyond the strict field of my Ph.D. I consider it an honour to have worked with and known him.

I would like to express my deepest thanks to my second supervisor Dr. Adrian Friday. His ability to boost my moral, his technical guidance and supportive supervision were invaluable throughout my Ph.D. His technical knowledge and his ability to suggest feasible solutions to complex problems helped me overcome difficult design and implementation issues. I owe him thanks for his enthusiasm and his constructive criticism over the past few years.

Thank you to my colleagues in the Computing Department at Lancaster, for helping to create a pleasant and friendly work environment. Particular thanks are due to Dr. Keith Cheverst who guided me during the first years of my Ph.D. I want to thank him for his guidance in research document writing and his valuable suggestions on my research. I would also like to thank Prof. Gordon Blair and Dr. Lynn Blair for their interest in my work over the past few years. Moreover, I would like to thank Dr. Keith Mitchell and Dr. Matt Storey for their comments and discussions over research and non-research related issues. Thanks to all the guys in the the Skylab that I worked with: Maomao Wu, Oliver Storz, Fahd Al-Bin-Ali, Prasad Boddupalli.

I would like to thank in particular my colleague and dear friend Dimitris Pezaros. He has been both a person that I could chat with and complain about my work, but most importantly he was the friend that I could share a beer with and enjoy listening to his guitar.

I would like to express my gratitude to all those beyond the department that made my time at Lancaster so enjoyable, especially Dr. Antonis Sapountzis for the nights we've spent chatting and drinking ouzo.

Special thanks are due to Vassia Markidou for her emotional and moral support throughout most of my Ph.D. work. I owe her so much...

Contents

List of Tables

List of Figures

CHAPTER I

Introduction

Contents

1.1 Adaptive and Context-Aware Applications

During the last decade we have witnessed a significant shift of the computing industry's focus towards the mobile user. A range of handheld computers with varying capabilities are now widely available and technologies providing wireless communication are offered in many forms (e.g. GSM [Adams95, Mouly92], IEEE 802.11 [IEEE97], Bluetooth [Bluetooth99a, Bluetooth99b]) as required by different application domains. This increasing interest in mobile computing has highlighted the fact that the characteristics of mobile environments have significant differences compared to those of traditional desk-top computing. In particular, mobile environments are tightly coupled with the notion of *change* [Davies98c]. Indeed, mobility is by definition related to changes in users' environments as they move. Moreover, the characteristics of mobile devices imply changes related to the availability and quality of resources, such as power supply and connectivity. These facts have been one of the main drives for research in the area of mobile computing. From this broad area two dominant research themes have emerged of interest to this thesis: the support for *mobile adaptive applications* and the development of *context-aware systems/ applications*.

Adaptation became an important research issue during the first half of the '90s when efficient support for streaming multimedia applications was one of the leading research targets [Campbell94, Diot95]. Most of the knowledge acquired during this time was transferred into the mobile world where the requirement for adaptation was further intensified by the variation in resource availability such as network connectivity and power supply. Indeed, in mobile environments adaptation has been applied not just for multimedia applications, but rather for every system component.

The "birth" of context-aware applications was stimulated by Mark Weiser when his vision of ubiquitous computing required context-aware functionality to be offered by future mobile systems [Weiser93]. Following the increasing interest of the computing industry in mobile computing, a range of new technologies (handheld computers, environment sensing technologies, etc.) offered new possibilities for applications that can monitor the user's environment and modify their behaviour accordingly.

At a high level, these two trends may appear like two independent parallel paths dealing with different aspects of mobility. The former, driven by the inherent restrictions of mobile technologies (varying network quality, limited battery life, etc.) is focusing on the development of adaptive applications as a solution for offering the best utilisation of resources. The latter, motivated by the availability of new environment monitoring technologies (location tracking, service discovery, etc.) is focusing on enriching the mobile user's experience through the development of applications capable of modifying their behaviour according to the mobile user's context without direct interaction. However, in this thesis we identify that the underlying principles for both categories of systems is the same: *applications that modify their behaviour due to external changes*[1]

Even though present researchers have tended to focus on one of these domains, it is reasonable to conceive that future mobile systems will combine both of these characteristics (adaptation triggered by changing network QoS, power availability, user context, service availability, etc.). Therefore, each individual mobile application will allow adaptation triggered by a variety of different system or contextual attributes. The consideration of a system that supports such adaptive applications is the main target of this thesis.

1.2 Motivation

The importance of adaptation in distributed systems, and in particular mobile distributed systems, has been identified by a number of researchers [Davies94b, Noble95, Katz94]. In [Badrinath00] a conceptual framework for network and client adaptation that fits most currently available mobile adaptive systems is described. In this framework, adaptation is illustrated as a mechanism where application-specific adapters are triggered to perform modifications on a network stream when certain changes are monitored in the system's environment or the availability of resources. This high level description of network adaptation, where the behaviour of the system is modified according to changes in the system's environment bears many similarities with the behaviour of context-aware systems.

Context-aware computing was first defined by Schilit and Theimer [Schilit94b] in 1994 to be software that "adapts according to its location of use, the collection of nearby people and objects, as well as changes to those objects over time". This early definition was based on a quite limited notion of context (location and proximity). Dey offers a

[1]Here we don't make any distinction between context related changes (e.g. location) and resource related changes (e.g. power availability). Indeed for a single application both cases refer to changes that are external to the application.

broader definition of context :

Definition 1: *Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between the user and the application, including the user and the applications themselves.*[Dey01]

This definition of context can practically include any kind of information that can characterise the situation of a participant in an interaction, be that the availability of resources or the quality of a network channel. In this sense, any application defined as adaptive in traditional terms, is actually a context-aware application.

It may be quite clear that adaptive applications are actually a sub-set of context aware applications, however not all context aware applications can fit in the conceptual framework describing adaptive systems. Indeed, context-aware applications can support features such as [Dey01]:

- presentation of information and services to a user.
- automatic execution of a service for a user.
- tagging of context to information for later retrieval.

This list includes possible ways that context can be used for the development of context-aware applications. The second feature identifies the way that applications can respond to changes by modifying their behaviour. This is a functionality that has clear similarities to the operation of an adaptive system: changes make an application modify its behaviour.

In order to explicitly specify the target domain of this thesis and also, avoid conflicts with terminology, it is necessary to define adaptive context-aware applications:

Definition 2: *Adaptive context-aware applications are applications that modify their behaviour (adapt) according to changes in the application's context. The term context is used in accordance with definition 1 being any information that can be used to characterise the situation of an entity.*

According to the above definition, the set of adaptive context-aware applications is a subset of context-aware applications and a superset of traditional adaptive applications (Fig 1.1). Following Dey's definition of context, the application's context can be any

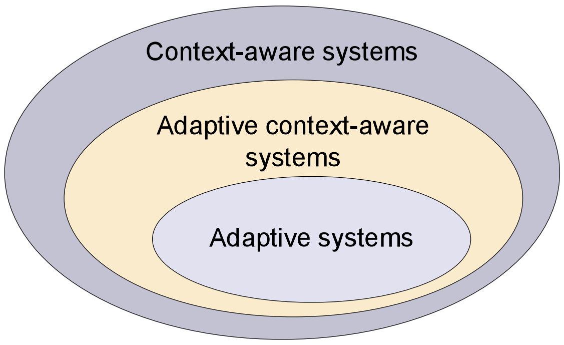


Figure 1.1: Sub-set relationship of context-aware, adaptive context-aware and traditional adaptive systems

information that characterises the application's situation. This can include the availability of resources, the preferences of the user, or the existence of other applications in the system that may interfere with the application.

An adaptive context-aware application should be expected to be able to adapt to a variety of contextual triggers. However, adaptive applications typically consider a particular resource that is the prime cause of adaptation (usually the quality of the network connection), while context-aware applications consider the application's *external* context. Furthermore context-aware applications tend to have no consideration of co-existing applications that they may share resources with.

The approach followed by existing researchers to isolate the domains of adaptation and context-awareness raises questions about the possible implications for a system that should be able to support both adaptive and context-aware applications. In particular, it is not clear how a system should behave when applications are capable of adapting to both contextual and resource related triggers. The interaction of multiple context-aware adaptive applications adapting to a number of different but possibly inter-related environmental triggers may cause instabilities and undesirable behaviour. Furthermore, coordination between these applications will need to consider both the abilities of the applications to adapt, as well as the multiple triggers that may require adaptation. The problem of coordination and interdependent adaptation is further intensified by the fact that applications are developed with no prior knowledge of other applications that may coexist at runtime and the effects their predefined behaviour may have on the system or the user expectations (this issue is discussed in detail in section 3). Considering these observations it is possible to identify a number of potential shortcomings or problems that may occur when adaptive context-aware applications are co-located within the same system, in an ad-hoc approach:

- **Inefficient use of available resources:** Consider a scenario where an application that is power-aware may run on the same system with an application that is network-aware (able to adapt in response to changes on the available bandwidth). In a situation where the first application reduces the use of the network in response to limited power supply, expecting power consumption to be reduced, the second application may monitor the consequent increase in available bandwidth and increase their network usage. As a result of these independent adaptations, the power saving action taken by the first application would be negated by the actions of the second.

- **Conflicts:** The previous scenario is actually a case of conflicting adaptive reactions of two applications. Given that both applications are developed independently without awareness of co-existing applications with different adaptation objectives, it is unreasonable to expect that the applications themselves will be able to resolve such a conflict without help from a coordinator provided by the system or the user.

- **Disregard of user preferences:** Most current adaptive or context-aware applications either ignore the user involvement in the specification of the application's behaviour, or restrict that involvement to the specific subset of context triggers they are aware of. However, user requirements can have implications on the way applications collaborate. In more detail, allowing the users to specify the behaviour of individual applications may not always provide satisfactory over-all system behaviour. Indeed, user requirements may necessitate coordinated adaptation of multiple applications. A possible scenario might include a video-streaming player that will degrade the stream bandwidth when the user is in their car, in order to allow timely delivery of traffic information for a co-running traffic monitoring application. Such a configuration scheme would require a system-wide approach to coordinated adaptation allowing combination of several triggering attributes (location and network bandwidth in this scenario) and control over multiple applications.

- **No extensibility:** In the two scenarios presented above one solution is to allow the system to make certain applications aware of more contextual triggers than the ones they were designed for. These contextual triggers may relate to external contextual information or the state of coexisting application. The first scenario, for example, could be resolved either by requiring the second application to incorporate power awareness in its behaviour or to be aware of the activities of other applications in the system. The support for extensibility however, can not be left to the application designers, most importantly because the specific extensions that may be required depend on the configuration of the system and the user preferences.

In order to understand the reasons that can lead to such undesirable behaviour taking place, it is necessary to investigate the fundamental characteristics of adaptation. In particular, adaptation can be defined as a combination of three conceptual entities:

- A *monitoring* entity to monitor a number of contextual attributes that may trigger the application to adapt. The monitoring entity can either be part of an application or the system itself. The information monitored may be of interest to more than one application.
- An *adaptation policy* that is responsible for deciding if and when the application should adapt based on the information gathered by the monitoring entity. An application is designed with a set of policies that implement the application's default behaviour. These default policies cannot perform special purpose coordinated decisions, mainly because the application developer is not aware of the possible configuration of the target system.
- The adaptive *mechanism* that performs the necessary changes when triggered by the adaptation policy. The adaptive mechanism is tightly coupled with the semantics of the application.

Based on these definitions, this thesis claims that the main reasons for the shortcomings of existing systems supporting adaptive context-aware applications relate to their design approach. Specifically:

- Adaptive systems tend to couple adaptation policies and adaptation mechanisms. In most cases these are both implemented as a single component that is bound to the semantics of the actual application.
- The monitoring entities offering information about attributes that can cause adaptation typically do not allow sharing of that information with other applications

in the system. Specifically, adaptation policies can not obtain information about other monitoring entities related to either the state of other applications or new contextual attributes.

- Adaptation policies are usually hard-coded either within the adaptive applications or the system platform supporting coordination. This fact does not allow reconfiguration of the system and moreover does not allow user involvement in order to specify their requirements in relation to the behaviour of the system.

Following these observations this thesis claims that sufficient support for coordination, conflict resolution, extensibility and user involvement can be achieved through the design of an adaptation support platform that satisfies the following design requirements:

1. Decoupling adaptation policies and adaptation mechanisms. Since adaptation mechanisms are generally related to the semantics of applications it is often necessary for them to be part of an application's implementation. However, adaptation policies that define when and how an application should adapt should be decoupled from the application's implementation.

2. Externalisation of application state. Monitoring entities that may be part of applications or system components monitoring the system's environment (e.g. a location monitoring module) should externalise that information. This would allow the adaptation support platform to retrieve information across multiple applications and/or multiple system components.

3. Externalisation of applications' adaptation mechanisms. With the decoupling of adaptation policies and adaptation mechanisms, an adaptation support platform can handle adaptation policies in a system-wide manner. This functionality requires applications to allow the adaptation support platform to trigger adaptation as defined by the adaptation policies. Thus the applications should expose an interface that allows the invocation of adaptation methods by the platform.

4. Enable the modification of the adaptation policies. As stated earlier, one of the reasons for insufficient support for multiple adaptive context-aware applications is the fact that applications do not have any prior knowledge of the configuration of the end-system and possible interdependencies between co-existing applications.

Therefore, the adaptation support platform should allow the reconfiguration of the system's behaviour in order to achieve coordination and conflict resolution. Moreover, the user should be able to express their requirements by modifying the adaptation policies that govern existing applications.

In order to prove this claim this thesis presents the design of an adaptation support platform that satisfies the aforementioned requirements. This platform uses a policy based mechanism for specifying adaptation policies. This approach allows the modification of the system's behaviour without the need for a re-implementation of either the applications or the platform. In the process of identifying a policy language that satisfies the requirements for adaptive context-aware applications a new language was defined that was derived from the Event Calculus logic programming formalism [Kowalsky86]. The Event Calculus Policy Language allows the definition of adaptation policies that can incorporate state information from multiple applications and system components, invoke adaptation over multiple applications and allow the user to modify existing adaptation policies or add new ones.

This thesis also presents a prototype implementation of this design and a thorough evaluation of the features of this prototype. Specifically, the ability of the adaptation platform to invoke adaptation actions in multiple applications is shown to allow the coordinated adaptation of multiple applications. The incorporation of multiple triggering information is shown to allow the extensibility of existing adaptive applications by allowing the definition of policy rules that incorporate additional adaptation triggers from the default triggers defined by the applications. Finally, the support for modification of the adaptation policies allows the resolution of conflicts and the active involvement of the user in the specification of the system's behaviour.

1.3 Road Map for the Thesis

This thesis is established in the following steps:

- Chapter 2 presents an investigation of existing adaptive and context-aware systems. The chapter provides a brief historical overview on the advances in adaptation and context-aware computing. Following the historical overview, a critical analysis of existing adaptation systems and context-aware systems is presented.

This critical analysis is driven by a set of criteria questioning the support of existing systems in terms of coordination, extensibility, reconfiguration and user involvement. The chapter concludes with a summary of this critical analysis.

- Chapter 3 presents an analysis of the potential problems that may occur when multiple adaptive context-aware applications are combined in an ad-hoc manner. In particular, a set of theoretical scenarios are presented and discussed. This analysis allows the identification of some general conclusions about the behaviour of existing systems and the reasons that specific problems can occur when supporting multiple adaptive context-aware applications. The chapter concludes with a set of design requirements that should be satisfied by a platform that supports coordinated adaptation for multiple adaptive applications.

- Chapter 4 presents the design of a platform supporting coordinated adaptation for multiple adaptive applications. The chapter includes a discussion about how the requirements defined in the previous chapter can be mapped onto a platform that supports coordination, extensibility, reconfiguration and user involvement. More specifically, the design of this platform requires applications to externalise their adaptation interface specifying their adaptive mechanisms and a set of state variables reporting their state. The platform uses those interfaces in order to retrieve state information from the applications and trigger adaptation as and when needed. The adaptation control mechanism is realised through a policy based mechanism. Specifically, the Event Calculus Policy Language is defined as a possible language for defining adaptation policies.

- Chapter 5 presents the implementation of a prototype implementation of the architecture for supporting coordinated adaptation. The chapter identifies the possible configurations for the implementation of this architecture with respect to the level of distribution of the platform's components. The presentation of the prototype supporting adaptation on a single host includes a detailed description of the prototypes components and an analysis of the evaluation engine implemented for the processing of Event Calculus Policy rules.

- Chapter 6 presents an evaluation of the prototype platform. This evaluation includes a qualitative evaluation of the platform's characteristics and a quantitative evaluation of the platform's performance. In more detail, the qualitative evaluation examines the level of support offered by the prototype in terms of coordination, conflict resolution, extensibility and user involvement. The performance evaluation examines the behaviour of the platform against a set of scalability factors, including the number of applications in the system, the number of policy rules, the complexity of the policy rules, etc.

- Chapter 7 summarises the work presented in this thesis. Special attention is drawn to the main contributions of this work and further research issues that arise from this work are discussed.

CHAPTER II

Adaptive and Context-aware Systems

Contents

2.1 Overview

The objective of this chapter is to provide an overview of the state of the art in the field of adaptive systems and to examine current adaptive and context-aware systems against a set of criteria that will be used as the basis for the analysis presented in chapter 3. The first sections of this chapter provide a historical survey of developments in adaptation and context-aware computing. Next a set of criteria are established for a critical analysis of existing adaptive and context-aware systems. The criteria defined are: the level of support of existing systems in terms of coordination, extensibility, reconfiguration and user involvement. A detailed presentation of a range of adaptive and context-aware systems is presented in the following sections. Finally this chapter concludes with a set of general observations derived from the critical analysis of existing systems.

2.2 The Emergence of Adaptive Systems

Throughout the history of computer science, the term *adaptation* has been used in a variety of different contexts. While no uniform definition for adaptation has been identified, adaptation in the field of mobile systems is most commonly linked to resource availability [Noble98] or network quality of service [Davies94a, Katz94]. As described in chapter 1 the target domain of this thesis includes application adaptation triggered by any type of context. Therefore this overview section will describe advances in both adaptive and context-aware systems.

2.2.1 Adaptive Network Protocols

Since the Internet was first established the varying characteristics of the underlying infrastructure and the “best effort” approach adopted by the Internet imposed a requirement for adaptive flow control. The transmission control protocol (TCP) used by the Internet for reliable communication provides a minimum level of adaptive congestion control [Jacobson88]. Specifically, in order to handle network congestion TCP uses a *congestion window* that determines the amount of data allowed for transmission in order to avoid congestion. TCP increases or decreases the congestion window in response to perceived network congestion (i.e. lost segments) in order to achieve better utilisation of the available bandwidth.

Since congestion control was introduced in TCP in 1980 several other enhancements have been proposed to provide more adaptive network behaviour including adaptive queue management in routers and adaptive retransmission time-outs.

With the widespread deployment of wireless communication new requirements were established for offering sufficient support for error prone communication. Early research identified TCP's adaptation strategy as inappropriate for wireless communication [Caceres94]. Wireless networks are characterised by losses due to transmission errors and handoffs. Caceres found that TCP interprets these losses as congestion and invokes congestion control mechanisms and retransmission of the lost segments, degrading the performance of the communication. To address such problems a number of communication protocols have been developed in order to support the explicit requirements of wireless communication [Bakre95, Amir95].

2.2.2 Distributed Multimedia

With the emergence of the Web at the beginning of the '90s, the focus of the research community was targeted on the efficient dissemination of multimedia content. The prime aim of this effort was the efficient support for real-time video and audio communication over the Internet.

One of the characteristics of multimedia is its high dependency on the timely transmission of their data. Video and audio packets that do not reach the destination on time for playback are considered useless packets. Thus protocols such as TCP are unsuitable for multimedia traffic as they involve retransmission of lost packets, adding unnecessary delays.

The Real Time Protocol (RTP) [Schulzrinne96] was introduced as a transport protocol suitable for realtime multimedia traffic. RTP uses UDP as the underlying transport protocol and time-stamping for controlling the sequence and flow of packets. Each RTP packet is time-stamped at the source with the time the packet should be played out at the destination. The flow of multimedia traffic is controlled through the Real Time Control Protocol (RTCP).

The introduction of the RTP/RTCP protocol allowed the development of simple adaptive mechanisms in multimedia applications. Delay adaptation, for example, uses buffering at the destination to allow adaptation to variance in packet delays (jitter). The well known *vat* audio-tool [Jacobson94] maintains an estimate of the average and standard deviation of the transmission delay. Based on these parameters *vat* can compute a correctly sized play out buffer eliminating any distortions caused by various packet delays.

Another research trend that had great impact in the development of adaptive multimedia applications was the introduction of a wide variety of multimedia encoding schemes. Such schemes include MPEG-1/MPEG-2 [RFCb], MPEG-4 [RFCc], Apple's Quicktime [Apple], H261 [RFCa], H320 [ITU], etc. Each of these schemes has different characteristics in terms of required bandwidth, tolerance to packet loss, required CPU power, etc.

These transmission requirements expressed by the encoding protocols introduced the need for a mechanism that can allow applications to express these requirements to the underlying network infrastructure and request guaranties that they can be satisfied. The notion of Quality of Service (QoS) refers to the capability of a network to provide guaranties on the quality of service offered to a network application. Typical QoS support includes an API that applications can use in order to express their resource requirements (e.g. maximum delay, throughput, packet loss, jitter, etc.) and possibly reserve these resources. When these requirements can not be satisfied by the underlying network the application is notified in order to renegotiate their requirements.

The combination of QoS support and different multimedia encoding protocols allowed the development of much more sophisticated adaptive multimedia applications. Typical examples include video tools that can switch between different encoding mechanisms [Davies98a], dynamic fine tuning of encoding parameters of a single encoding protocol [Yeadon96, Walpole97] (allows adaptation when video encoding schemes cannot be changed through the dynamic introduction of filters). In multicast environments like MBONE, tools like vic [McCanne95b] and vat [Jacobson94] use adaptive techniques to allow media streaming over the multicast backbone of the Internet.

2.2.3 Mobile Systems

Early developments in mobile computing were mostly concerned with file system access and the problems caused by disconnected operation. CODA [Kistler91] is a typical example of a system dealing with disconnected file access. Similar problems but in a more specific domain were also targeted by projects dealing with disconnected database access [Demers94].

The popularity of the Web in the Internet gave a push to an increasing effort to allow wireless access to web content. In these efforts the actual aim shifted away from the disconnected operation and moved to issues concerning reliable transmission of data over low bandwidth links. The typical network infrastructure used by most systems was based on a client-proxy-server scheme. A web client was allowed to access the content of a web server through a proxy that was responsible for all necessary transformations/adaptations before the data was transmitted over the wireless link. The techniques used by these systems included re-encoding of images, text compression, data perfecting, etc. The result of this effort was the specification of the WAP [WAP99] standard for web access over GSM networks.

The second half of the '90s show a shift of the research community, abandoning special purpose mobile applications using ad-hoc development techniques and moving towards the development of general purpose middleware for mobile environments. Here the lessons learned by the support for adaptive distributed multimedia were transferred to the mobile environment.

The research in the area of middleware for mobile systems was (and still is) spread over a wide range of different methodologies and research directions. A number of systems utilised the tuple space paradigm to allow communication decoupling across space and time [Davies98c, Johanson02]. Other systems incorporated open/reflective architecture designs in order to allow flexible adaptation [Blair00]. Middleware supporting mobile agents [Johansen97] and code mobility [Joseph97] was introduced in order to allow applications to split their processing and network requirements between mobile and/or fixed nodes in the network.

An additional research topic that gradually became important at the end of the '90s and beginning of '00s, is the support for low power consumption applications and power-aware adaptation [Flinn99, Havinga99].

2.2.4 Context-aware systems

The first generation of context-aware systems was mostly influenced by the visions of Mark Weiser [Weiser93] and the work on the PARC Tab project at Xerox PARC

[Schilit94a]. One interesting characteristic of the first context-aware systems is that, in most cases, the only type of context considered was the physical location of the user and the proximity of other users and services.[Long96, Brown95]

Following the PARC Tab experiment, new research projects extended the notion of context beyond the physical location, including information such as the user preferences, the existence of other users in the surrounding environment, the quality of the wireless network connectivity, etc. [Davies99]. However, even these systems followed an ad-hoc approach to the development of context-aware applications.

The end of the '90s and beginning of this millennium was a time where a lot of research was focused on context representation and the development of middleware platforms that could be used for the creation of context aware applications [Salber99, Dey00]. However, even today the question "what is context" is still open for debate. Today, there are no standards concerning context representation and most research groups follow their own proprietary designs.

2.2.5 Application Aware Adaptation

One of the aims of early research in mobile computing was the development of systems that can provide transparent mobility support. The target of this approach was to push all the functionality related to mobility into the system and allow applications to operate as if they were operating in a fixed environment. The specific requirement for adaptation was considered a feature that should be provided transparently without any involvement of the application. Systems such as Coda [Kistler91], for example, offers application transparent adaptation for file system access.

Research in the second half of the '90s indicated that transparent adaptation has certain limitations. In particular, it is not possible for general purpose adaptation platform to provide sufficient adaptation support for the requirements of all applications. Noble in [Noble95] suggests the use of application-aware adaptation. In particular, middleware platforms supporting adaptation should notify applications about possible changes in the environment and allow them to adapt. This approach allows applications to implement adaptive mechanisms that are more fitting to their requirements. Indeed, application developers have better knowledge about the semantics of the applications. Therefore adaptive applications can implement their own mechanisms for adaptation

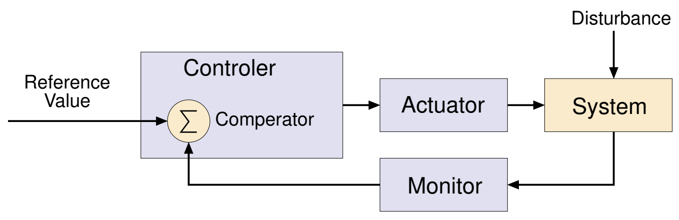


Figure 2.1: Block diagram of a feedback control system

and rely on the middleware platform for general purpose adaptation support. A number of middleware platforms follow this trend [Friday96, Blair00]. A particularly interesting example is the work on Odyssey [Noble95] where the Coda file system offering transparent adaptation is modified in order to allow application-aware adaptation.

The work presented in this thesis follows an application-aware approach in supporting adaptation.

2.3 Overview of Existing Adaptive Systems

As highlighted in the previous section, the physical limitations of mobile environments along with the increasing need for multimedia and context-aware services drove the research community towards the adoption of adaptive approaches in the development of such systems. The following sections offer a deeper investigation of current research in these areas, revealing their possible limitations in supporting multiple adaptive context-aware applications.

2.3.1 Abstracting Adaptation

In order to identify in more detail what are the actual limitation of current approaches, it is first necessary to examine the fundamental mechanisms supporting adaptation.

The theoretical model that has been proposed for describing adaptive systems is based on feedback control theory [Cen97, Meng00, Kokar99]. The feedback control theory was initially used in engineering for developing hardware control systems. Figure 2.1 shows a typical feedback control system. The *controller* helps the system maintain a reference value of a control variable, while reducing the system's sensitivity to disturbance. The controller interacts with the system through *monitors* and *actuators*.

A monitor measures the controlled variable, and is the source of the feedback. The controller's output causes the actuator to adapt the system's behaviour in response to disturbance, or changes in the system's environment.

In the analysis presented here we propose a simplified closed loop system as the basic abstraction of an adaptive system, borrowing from the design of the feedback control systems. This abstraction includes three distinct functional elements (Figure 2.2):

- *Monitor*: The first element performs the monitoring of a specific source of information that is 'interesting' for the adaptive mechanism. This information source could be, for example, the availability of a specific resource such as power or a contextual trigger such as the system's physical location.
- *Controller*: The second element is the controlling mechanism that takes decisions concerning the adaptive reaction of the system. This decision is based on the information received by the monitor. This controller could, for example, state that when the power supply drops below a specific threshold then a reaction is necessary.
- *Actuator*: The third element is the actual adaptation mechanism that performs the specific adaptive action as directed by the controller. For example, an actuator might reduce the network bandwidth consumed by an application. Note that this reaction may in turn have an impact on the initial source of information, i.e. causing a change in the rate that the available power drops. This last link between the actuator and the initial resource being monitored does not necessarily exist in all systems. Most context-aware systems for example do not affect the initial resource that triggered their change of behaviour.

This theoretical model allows the description of abstract concepts such as "extensibility", "reconfigurability", etc. as explicit design characteristics that adaptive systems should satisfy. In the next section these abstract concepts are used as the bases for establishing a set of criteria for the analysis of existing adaptive and context-aware systems. These criteria are defined in relation to the abstract model of adaptation discussed.

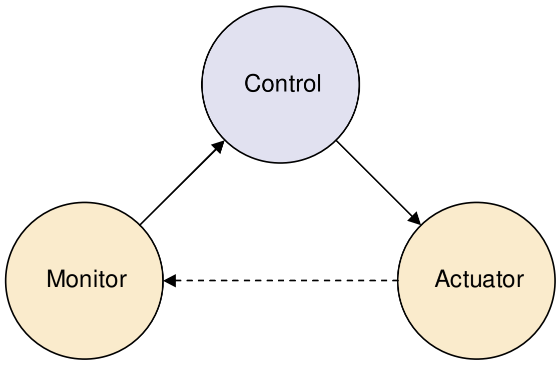


Figure 2.2: Basic adaptation cycle

2.3.2 Assessment Criteria

Before analysing the design characteristics of current adaptive and context-aware systems, it is necessary to establish a set of criteria that will guide this investigation. These criteria will allow us to establish some general principles about how these systems operate focusing on the issues that are of importance for this work. These principles will be the basis for the analysis presented in chapter 3.

In this thesis we are concerned with supporting mobile systems that consist of a collection of independently developed adaptive context-aware applications. As described in Definition 2, adaptive context-aware applications are applications capable of adapting to a variety of contextual triggers such as the availability of resources, the preferences of the user, or the existence of other applications in the system that may interfere with the application. Considering the focus of this thesis we can clearly identify a number of key characteristics that are of importance for this work:

- Each adaptive application may require information about a wide variety of contextual attributes that may be used as triggers for adaptation.
- The target environment is a system where multiple adaptive applications will co-exist and possibly interfere with each other.
- The application developer may not be aware of any possible interference or undesirable side-effects between applications when designing their own application.

Based on these observations we define a set of criteria that will be used in the assessment of the existing mobile adaptive and context-aware systems.

2.3.2.1 Coordination

In an environment with multiple coexisting adaptive context-aware applications it is important to support coordinated adaptation (Chapter 3). Adaptive responses by individual applications may have contradicting effects or cause instabilities [Efstratiou00]. Coordinated adaptation can overcome conflicting situations and increase overall system stability.

Mapping this statement to the theoretical framework of the basic adaptation cycle (Section 2.3.1) by the term *coordination* we specify the ability of a controller to trigger actuators that relate to more than one application. In more detail, a coordinated adaptive reaction involves multiple applications triggered to perform particular actions, in contrast to isolated adaptation where a controller triggers only one application. In practice, this functionality requires adaptation controllers to be able to retrieve information from multiple applications (i.e. have access to multiple monitoring entities) and be able to trigger adaptation on multiple applications (i.e. be able to trigger multiple actuators).

2.3.2.2 Extensibility

As previously described (Chapter 1), the target of this thesis is to provide support for adaptive applications able to adapt to an extensible set of contextual triggers. Current research efforts have already identified certain contextual attributes that can become triggers for adaptation. These include, among others, network QoS [Davies98c], power supply [Flinn99] and physical location [Cheverst00]. However, the possible types of contextual attributes that may trigger adaptive reactions by applications can not be defined as a static set of triggers, since future mobile systems should be able to incorporate new adaptation triggers as and when they become regarded as important.

Mapping this statement to the theoretical framework of the basic adaptation cycle, by the term *extensibility* we specify the ability of a controller to receive input by more than one monitor. Moreover, we consider how possible it is for an existing controller with a given set of monitors to be modified in order to receive input by more, possibly newly created monitors. Most value is given to the ability of system to allow existing applications to extend their behaviour without the need for re-implementation.

2.3.2.3 Reconfigurability

Independently developed adaptive applications are constructed under a set of assumptions that the developer had to make about the target operating environment. However, in environments where multiple adaptive applications coexist in the same system, it is expected that interdependencies between adaptive reactions of individual applications will lead to undesirable behaviour or even conflicts (see Chapter 3 for a detailed discussion).

In situations such as these, the ability of the mobile system to allow reconfiguration of their adaptive behaviour is vital: under certain conditions applications may be required to modify the default adaptive strategies defined by the developer. Therefore the system should provide the means for modifying the system's adaptive behaviour.

Mapping this criterion into the theoretical framework, by the term *reconfigurability* we specify the ability of a controller to allow modification of their behaviour. These modifications may be considered as related to the previously mentioned criteria. In particular, the incorporation of additional monitors and actuators into an existing adaptation cycle may require modifications to the behaviour of the controller.

2.3.2.4 User Involvement

The importance of user involvement in the operation of adaptive systems has often been neglected in current adaptive systems [Efstratiou01]. In a system where adaptive behaviour may require to be modified, as described in the previous criterion, user involvement can allow the user to specify the desirable modification.

According to our theoretical framework, we require the controller to allow the user to inspect its behaviour and potentially modify it according to their requirements. In terms of existing systems we investigate the ability of the system to provide user awareness about its adaptive behaviour and allow the user to modify that behaviour.

The following sections present a survey of existing system supporting adaptation and context-awareness. In particular, this survey includes projects that support adaptation following a wide range of design approaches such as, application aware adaptation middleware, mobile agent based middleware, reflective middleware, etc. The context-aware related section focuses on projects that are related to the reactive response of applications to contextual changes, e.g. location triggered, proximity triggered adaptation.

2.3.3 Independent Adaptive Applications

Independent adaptive applications (also known as Laissez-Fair Adaptation [Noble98]) are applications that adapt independently without the need for any system support. Applications in this category include commercial systems such as the Microsoft Windows Media Player [Microsoft03] or the RealPlayer [Real03] and research tools, such as vic [McCanne95a] and vat [Jacobson94]. In such systems, applications monitor the availability of resources and make their own adaptation decisions in isolation of other applications or the system.

The laissez-faire approach provides a substantial benefit. No system support is required, a feature that is essential for commercial systems where the operating system is a fixed commodity.

However, the laissez-fair approach does not support application concurrency: applications, operate in isolation from the rest of the system, unaware of other applications possibly sharing the same resources. Moreover, the monitoring information received by an individual application may not always reflect the metrics that can be achieved by a system monitor that is aware of all involved parties.

Criteria based analysis:

- **Coordination:** Not possible. Applications act in isolation.
- **Extensibility:** Not possible. Applications have a fixed set of adaptive triggers that they can react to.
- **Reconfiguration:** Depends on the application, but most available systems do not provide any mechanisms for reconfiguration.
- **User involvement:** Depends on the application, but most available systems do not allow the user to modify the application's behaviour.

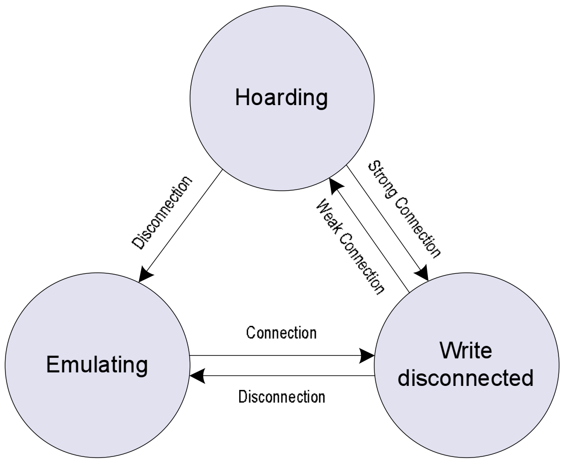


Figure 2.3: The Coda state transition diagram

2.3.4 Middleware-based Systems

2.3.4.1 Coda

The Coda filesystem [Satyanarayanan90] was developed in Carnegie Mellon University as an extension to the work done on the Andrew File System [Satyanarayanan85]. Coda is a highly available replicated file system offering disconnected operation for mobile clients. In Coda file servers maintain a state transition mechanism consisting of three states [Kistler91, Mummert95]: *hoarding*, *emulating* and *write disconnected* (see figure 2.3).

In the hoarding state, a mobile client pre-fetches in the local cache the user's set of working files. The pre-fetching can be initiated periodically or at the user's request. The set of files to be cached is determined by the hoarding database that is constructed using file access traces and can be modified by the user.

When the client looses connection with the file server, Coda moves into the emulation state. In this state the file system allows modification of the cached files as if they were still connected to the file server.

All file access operations performed in the emulation state are logged in order to be replayed when the client re-connects to the file server. Replaying the logged actions allows Coda to update the database with any changes which took place when disconnected. This update process is performed in the write disconnected state. After the logged actions have been replayed, the files in the file server are up-to-date with the files in the client's local cache. However, certain situations may require user intervention. In particular, when the client is connected through a low speed connection, file updates may take relatively long time. When the estimated time for a file update exceeds a certain threshold (called the *patience threshold*) user intervention is required to specify whether updates should be postponed until a high speed connection is established. During the update state, inconsistencies may be discovered. These inconsistencies can be resolved by either application specific resolvers or by direct user intervention. At the end of the write disconnected state, if the client is connected to the server through a high speed link, the file system moves back to the hoarding state.

Criteria based analysis:

- **Coordination:** No mechanisms to support coordination are offered.
- **Extensibility:** Not applicable. Coda is a special purpose system targeting disconnected file access.
- **Reconfiguration:** Coda offers mechanisms to specify how conflicts should be resolved. However, the general behaviour of the system is static and can not be reconfigured.
- **User involvement:** User involvement is required as part of the resolution of conflicts in data updates.

2.3.4.2 Odyssey

The Odyssey system [Noble98] was created as a generalisation of the Coda system in order to support media-specific adaptive communication for mobile clients. As with Coda, Odyssey works on the assumption that a mobile network consists of light weight mobile clients connected over wireless links to fixed servers with high processing capabilities and no power limitations.

The general model of operation is based on monitoring the levels of resources such as network QoS, CPU and battery power and notifying adaptive applications when the levels of these resources do not satisfy the applications' requirements. When such notifications reach an application, it is the application's responsibility to perform the necessary adaptations and renegotiate new resource levels with the platform.

The monitoring and negotiation of resources is performed by the *viceroy*. The viceroy is the common point where all applications express their requirements in terms

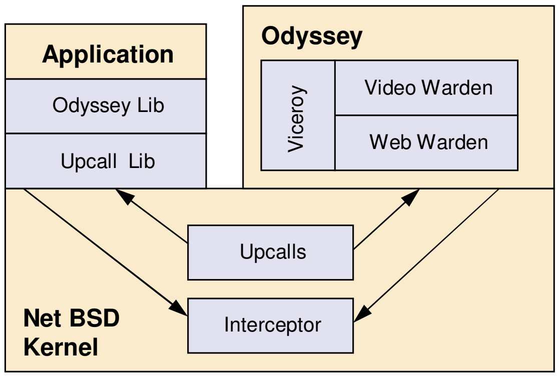


Figure 2.4: The Odyssey system

of resource windows (an application will be satisfied as long as the resource levels are within the bounds of the resource windows). The viceroy is responsible for monitoring the system's available resources and sending notifications to the applications when the resource levels exceed the bounds of the resource windows.

The notifications sent to an application will in most cases require an adaptive reaction from the application and a specification of a new resource window. Odyssey offers a set of media specific agents that can perform modifications on the fidelity of media transmitted over the network, in response to changing resource levels. Each of these agents, called a *warden*, is specialised for a certain media type (video, audio, etc.) and can offer media specific adaptation. For example a video warden can modify the frame rate, encoding, dimensions, etc. of a video stream according to different levels of required resources.

Criteria based analysis:

- **Coordination:** Odyssey considers coordination as a mechanism for sharing resources between applications. In this sense Odyssey allows the coordinated sharing of resources between multiple applications through the viceroy notifications. The available system resources are handled by the Odyssey platform according to the requirements of all applications in the system. Whenever an adaptive action is necessary Odyssey can trigger multiple applications as needed. However, this coordination support does not fully satisfy the criterion specified in section 2.3.2.1 as no explicit coordination of actions is involved.

- **Extensibility:** In Odyssey it is possible to extend the existing system and support a number of resource attributes that can trigger adaptation, including network QoS, power, cost, etc. However, any such extension would require the implementation of the necessary viceroys that would deal with the particular resource and the re-implementation of applications that will use that viceroy. The prototype implementation supports adaptation triggered by changing network QoS only.

- **Reconfiguration:** The Odyssey platform does not offer any mechanisms for modifying the default adaptive behaviour of applications or the system itself.
- **User involvement:** There is no support for user involvement in modifying the system's adaptive behaviour.

2.3.4.3 MOST

The MOST system is a collection of tools supporting collaboration among field workers in the power distribution industry [Friday96]. The underlying support for adaptation in MOST is achieved through an application aware platform that facilitates the creation of explicit binding objects that encapsulate network connections between two or more applications. The explicit binding object allows querying about the connection's QoS attributes thus breaking the transparency between the connection's characteristics and the application. Therefore the applications are able to perform adaptation in response to changing network QoS levels.

At the application level MOST supports user awareness. In particular, the MOST interface offers indications about the quality of the communication links with other parties. These indications however do not give information about any possible adaptive reactions that the application might have taken.

Criteria based analysis:

- **Coordination:** The open binding approach followed by MOST (and by a number of middleware platforms discussed later) allows the sharing of information between applications. In particular each binding object allows monitoring of their interface by multiple applications. This fact means that applications can potentially adapt based on information related to other applications in the system. However, this cannot be considered as coordination as applications do not coordinate the actions they take.
- **Adaptation attributes:** MOST supports adaptation triggered by changing network QoS only.

- **Reconfiguration:** There is no built-in support for modifying the system's adaptive behaviour.

- **User involvement:** MOST supports user awareness of the attributes being monitored but there are no mechanisms that allows the user to modify the adaptive behaviour of the system.

2.3.4.4 Rover

Rover is a toolkit developed at MIT that "*combines re-locatable dynamic objects and queued remote procedure calls to provide unique services for "roving" mobile applications"* [Joseph97]. The Rover toolkit offers applications a distributed object system based on a client/server architecture. It supports mobile communication based on two ideas: *relocatable dynamic objects* (RDOs) and *queued remote procedure calls* (QRPCs)

An RDO is an object (code and data) that can be dynamically loaded to a server from a client or vice versa. A mobile aware application includes RDOs for the data types manipulated by the application and exchanged with the server. Moreover, it defines portions of the application that run on the client and portions that run on the server. By transferring part of the application's functionality to the server, the application can reduce the client-server communication requirements.

The communication between RDOs is performed through queued remote procedure calls. QRPCs is a communication mechanism that allows applications to continue to make non-blocking remote procedure calls even when a host is disconnected. The queued requests and responses are exchanged upon network reconnection. Conflict detection and resolution is offered by the RDOs involved in a transaction.

Criteria based analysis:

- **Coordination:** In the Rover toolkit each client/server couple is isolated from any other coexisting client/server applications. Therefore there is no exchange of information between applications and there is no mechanism to support coordination.

- **Extensibility:** The Rover toolkit supports adaptation triggered by changing network QoS only.

- **Reconfiguration:** There is no system support for reconfiguration.
- **User involvement:** There is no consideration for user involvement. Such functionality is the application's responsibility.

2.3.4.5 TACOMA/TACOMA Lite

The TACOMA project [Johansen97] developed at the University of Tromosø focuses on the idea of code mobility and agent technology. The TACOMA project offers a full set of tools for the development of mobile agents.

TACOMA offers support for *weak mobility* [Fuggetta98] where individual agents are responsible for saving their execution state (and possibly filtering out unnecessary state information) before migrating to a new host. This is in contrast to *strong mobility* [Fuggetta98] where the system forces the agent to move by saving the agent's state and restoring it after migration. Strong mobility is supported by systems such as Telescript [White94], Agent-Tcl [Gray96] and Ara [Peine97] while weak mobility is supported by Aglets [Tai99] and Voyager [Glass99].

In order to make the state saving and restoring process easier for the developer, TACOMA uses an abstraction of folders, briefcases and meeting operations. Agents keep their data in folders that they can either carry with them or store in folder cabinets on hosts. An agent can exchange data with a local or a remote agent using briefcases. The meet operation is the abstraction of a remote procedure call between agents.

TACOMA Lite is an extension of the TACOMA project targeting light-weight hand held devices. The main difference between TACOMA and TACOMA Lite is that the latter provides support for disconnected operation: whenever agents need to migrate to a disconnected host, they are queued and transferred upon reconnection.

Criteria based analysis:

- **Coordination:** TACOMA supports the exchange of information between applications. However, this feature alone is not sufficient enough to allow applications to coordinate their actions.
- **Extensibility:** Not applicable.
- **Reconfiguration:** There is no system support for reconfiguration. The mobile

application is responsible for providing the necessary mechanisms that will allow different configurations.

- **User involvement:** There is no consideration for user involvement. Such functionality is, again, the application's responsibility.

2.3.4.6 Bayou

Bayou [Terry95, Demers94] is a weakly consistent replicated database system that supports read or update operations by mobile users who may be disconnected from other users as individuals or as a group. The emphasis of this system is on supporting automated application specific conflict detection and resolution and on supporting application controlled inconsistency.

In more detail, Bayou provides a replicated database system supporting a variety of non-realtime collaborative applications, such as shared calendars, e-mail and document editing. Mobile clients can read and/or write to any server without waiting for changes to be propagated to all servers. Updates are exchanged between servers periodically in *anti-entropy* sessions in order to achieve consistency between replicas.

However, conflicts in data updates may occur while the application is not accessible. Bayou supports automatic application-specific conflict detection and resolution. Applications provide *dependency checks* and *merge procedures* that are used by the servers in order to detect and automatically resolve conflicts. These procedures are executed in each server allowing, eventually, the replicated database to reach a consistent state.

Criteria based analysis:

- **Coordination:** No mechanism is provided to support coordination.
- **Extensibility:** Bayou is designed to support disconnected and loosely connected operation. No other adaptation triggers are considered.
- **Reconfiguration:** Reconfiguration of the conflict detection mechanism can be achieved by modifying dependency checks or merge procedures, without the need for modification of the rest of the infrastructure.
- **User involvement:** Bayou provides user awareness but it does not offer any means to modify the system's behaviour.

2.3.4.7 Mobiware

The Mobiware [Kounavis01] system developed at Columbia University is a middleware toolkit that controls an open active programmable mobile networks [Tennenhouse97]. The term *open* here means network components (e.g. mobile devices, access point, switches and routers) offer a well defined interface to allow the implementation of new signaling, transport and adaptive QoS management algorithms. In Mobiware these devices are represented as distributed objects based on the Common Object Request Broker Architecture (CORBA).

The Mobiware network comprises an ATM based programmable fixed network with wireless access points. Mobiware defines a set of programmable objects that abstract over certain entities of the network. *Mobile device objects, access point objects* and *switch server objects* abstract mobile devices, access points and network switches respectively. A set of objects that can be located anywhere in the fixed network offer adaptation services. In more detail, the *QoS adaptation proxy (QAP)* allows mobile devices to probe and adapt to changing resource availability over the wireless link. The *mobile agent objects* are responsible for managing hand-offs when triggered by the mobile device.

On top of this infrastructure, Mobiware offers a set of mechanisms for controlled hand-offs and mobile soft-state. The controlled hand-off is a mechanism that permits graceful hand-off of an active data flow from the network to the mobile device with a minimum hand-off dropping probability. During a hand-off initiated by the mobile device, a mobile agent triggers the network switches so that the data flow is delivered to the mobile device through both access points. Eventually the mobile device switches completely to the new access point and the traffic through the old access point is canceled.

The mobile soft-state mechanism provides QoS adaptation support to mobile devices. Mobile soft-state results in the periodic negotiation of bandwidth requirements between the mobile device and QAP. A mobile device sends periodic refresh messages as part of the soft-state probing mechanism. During the refresh phase mobile devices respond to any changes in allocated bandwidth by adapting.

Criteria based analysis:

- **Coordination:** General adaptation in Mobiware is achieved through a soft-state

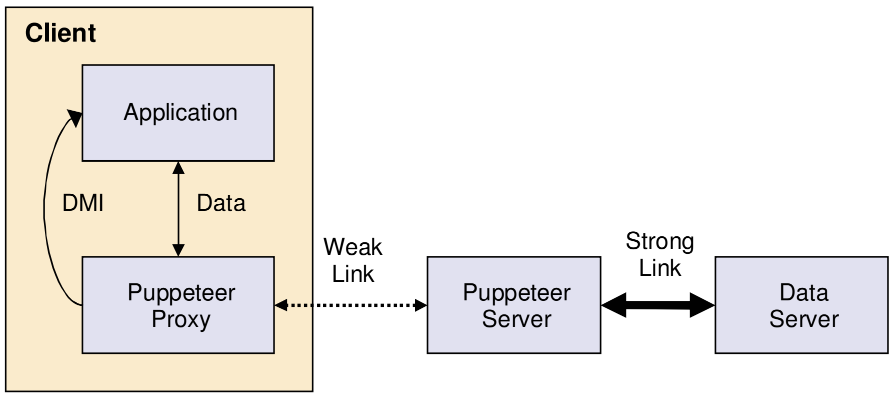


Figure 2.5: The Puppeteer system

mechanism. The goal of this mechanism is to notify the mobile applications and allow them to adapt as needed. Therefore the actual adaptation is performed individually not allowing any coordination among applications.

- **Extensibility:**The Mobiware toolkit supports adaptation triggered by changing network QoS only.
- **Reconfiguration:** The open architecture approach makes possible the reconfiguration of the system. In particular the adaptation decision mechanism can be modified by replacing the mobile agent object with a new one. In such a case, however, re-implementation of certain objects will be required.
- **User involvement:** No user involvement or awareness is supported.

2.3.4.8 Puppeteer

Puppeteer [deLara01] is a project that provides application-specific adaptive behaviour to component based applications. Puppeteer is not a full middleware platform but rather a methodology for offering adaptive behaviour to existing applications without modifying the actual application.

Puppeteer requires that an application exposes a run-time interface that allows the system to view and modify the data it operates on (called the Data Manipulation Interface - DMI). Their prototype implementation is based on Microsoft's COM architecture and uses COM and OLE interfaces to manipulate applications such as PowerPoint and Internet Explorer.

The adaptive mechanism comprises an application-specific Puppeteer client proxy and a corresponding server. The client proxy is responsible for triggering bandwidth adaptation and resource management. It is also in charge of controlling and monitoring the application using the DMI interface. The Puppeteer server is assumed to have a high speed link to the data server. It is responsible for parsing documents, exposing their structure and fetching document components as requested by the client proxy.

When an application controlled by Puppeteer requests a document over the Internet the corresponding Puppeteer client monitors the actual behaviour of the application and fetches the document according to hard-coded policies. For example, in the PowerPoint scenario the client requests the active slide and presents it to the application while it pre-fetches the following slides in the background, thus reducing the delay experienced by the user.

Criteria based analysis:

- **Coordination:** In Puppeteer each client proxy operates in isolation controlling the corresponding application in an uncoordinated manner.
- **Extensibility:** The Puppeteer approach requires the development of client-server couples that are tightly bound to the actual structure of each individual application. Any possible change to incorporate new adaptation attributes would require re-implementation of the client-server couples.
- **Reconfiguration:** For the same reasons presented in the previous criterion, any modifications of the behaviour of the system would require re-implementation of the client-server modules.
- **User involvement:** Due to the requirement for transparency, no user involvement is supported.

2.3.4.9 TAO

TAO [Schmidt98] is a CORBA 2.0 compliant middleware framework that allows clients to invoke operations on distributed objects without concern for object location, programming language, OS platform, communication protocols. One of the main aims of TAO is to provide high-performance, real-time communication, with full support for end-to-end QoS guarantees.

A key motivation for ORB middleware is to support reusable middleware components that handle common tasks, such as interprocess communication, that can be easily integrated in an application. TAO aims to extend this functionality by allowing dynamic reconfiguration of the available ORB components during installation or during run-time. This way an application developer can dynamically configure the underlying middleware platform according to their needs. For example, an application can configure the middleware's characteristics in order to take advantage of the availability of a high-speed ATM network.

In order for this level of flexibility to be possible TAO defines a set of *patterns* which are actually predefined IDL definitions for certain types of components. For example the *wrapper facade* pattern encapsulates I/O communication mechanisms like the socket API, the *reactor* pattern encapsulates an event handling and dispatching mechanism, etc. As a result an application can communicate with a middleware component through the pattern allowing the implementation of the component to be changed or replaced as needed.

There is also a version of TAO for handheld devices called LegORB [Román00]. LegORB takes advantage of the configuration mechanisms provided by TAO in order to create a minimal ORB middleware with only the components required to achieve CORBA compliance.

Criteria based analysis:

- **Coordination:** No support for coordination is provided.
- **Extensibility:**TAO supports adaptation triggered by changing network QoS only.
- **Reconfiguration:** TAO allows run-time re-configuration of the system through the modification of the existing components or their replacement with new ones.
- **User involvement:** There is no mechanisms to allow user involvement in the modification of the system behaviour.

2.3.4.10 Open-ORB

Open-ORB [Blair00] is a reflective middleware platform developed at Lancaster University. The platform follows a component model where components are described by a set of provided interfaces. There is also support for interfaces supporting continuous media interactions. Explicit binding is supported where the result is a binding object with an interface that can be used for QoS monitoring. Moreover, components have a built-in event mechanism that can be used to register for notifications on changes of QoS.

In more detail, every component in Open-ORB has an associated *meta space* that can be used for inspection and adaptation of the underlying infrastructure of the component. For example when dealing with a binding object the meta model expressed by the component could represent an object graph including an MPEG compressor and decompressor and an RTP protocol component. This structure can also be exposed recursively, for example the RTP component can expose the two peer components (connected to the MPEG compressor and decompressor) and a UDP/IP component handling the traffic between the peer components.

The adaptation mechanism supported by Open-ORB consists of a collection of components that can be inserted in a components object graph when needed. More specifically, a monitor component collects statistics on the level of QoS archived by the running system and raises events when QoS violations occur. A controller component is responsible for implementing adaptation policies in response to the events raised by the monitor component. This component is in turn divided into two components the strategy selector and the strategy activator which together realise the adaptation policy.

One of the important characteristics of Open-ORB is the fact that components can be configured or even replaced at runtime. Therefore a particular adaptation policy can be replaced by a new one when needed.

Criteria based analysis:

- **Coordination:** In Open-ORB all network bindings offer an event interface that allows applications to register for changes in the QoS of a particular connection. Therefore it is possible for multiple applications to register for the same events and coordinate their adaptive reactions. However, achieving coordination through these notifications would be entirely the application's responsibility and not part of the functionality offered by the system.
- **Extensibility:** Open-ORB supports adaptation triggered a number of system resources (network QoS, memory, CPU, etc). However, extending this mechanism

to include new, possibly contextual attributes would require re-implementation of the system and/or the application.

- **Reconfiguration:** Open-ORB allows run-time re-configuration of the system through the modification of the existing components or their replacement with new ones.

- **User involvement:** There are no mechanisms to allow user involvement in the modification of the system behaviour.

2.3.4.11 OpenCORBA

OpenCORBA [Leboux99] is a CORBA broker based on a reflective approach. Its architecture enables the reification of its internal characteristics in order to allow applications to modify and adapt them at run-time.

OpenCORBA follows a similar approach to Open-ORB (see above) where each middleware class is associated with a *meta class* that can be used for introspection and adaptation. OpenCORBA follows a more transparent approach to communication by offering the meta class as the means for communication with the actual middleware class not allowing direct access to the middleware class itself. This approach is used for dynamic adaptation of the underlying communication mechanisms. Dynamic adaptation mechanisms supported by OpenCORBA include, different communication protocols (Java RMI, future Corba DII), object migration, object replication, etc. All these adaptive mechanisms can be invoked by the system without affecting the design of the application.

Criteria based analysis:

- **Coordination:** The introspection mechanisms offered by OpenCORBA can be used by multiple applications to identify the conditions of the underlying network. However, coordination relies on the applications themselves.

- **Extensibility:** OpenCORBA supports adaptation triggered by changing network QoS only.

- **Reconfiguration:** OpenCORBA allows reconfiguration of the underlying network mechanisms used by the platform.

- **User involvement:** There are no mechanisms to allow user involvement in the modification of the system behaviour.

2.3.5 Context-aware Systems

Many research project are concerned with the development of context aware systems. The following sections present some representative context-aware systems.

2.3.5.1 Guide

The GUIDE [Cheverst00] system has been developed to provide visitors to the city of Lancaster with information that is tailored to their context. The types of context supported by GUIDE include the physical location of the mobile device, the preferences of the user, the weather conditions, etc.

The GUIDE system consists of a wireless cellular network with small non-overlapping cells, a set of cell servers associated with each cell and interconnected through a fixed network, and a number of mobile devices (such as tablet PCs and PDAs). The location of the device is determined by the specific cell that the mobile device is currently in. The cell servers periodically beacon a location id. This location id is used by the mobile device in order to give information to the user about the location they have just visited. The user interface offered by the GUIDE system is a modified web browser. In particular the web browser tailors the information presented to the user according to their preferences or attraction related attributes, such as if the attraction is closed.

Criteria based analysis:

- **Coordination:** Not applicable. GUIDE is a single application system.
- **Extensibility:** In the GUIDE system the support for specific contextual triggers is hard-coded within the application. Therefore extending the system to support new contextual or adaptation attributes would require re-implementation.
- **Reconfiguration:** The behaviour of the GUIDE system in terms of adaptation or user notification is hard-coded within the system. There is no support for reconfiguration of that behaviour.

- **User involvement:** The user involvement is limited to the specification of certain contextual attributes, such as their interests and preferences.

2.3.5.2 Cyberguide

Cyberguide [Long96] is a location based context-aware indoor mobile tour guide. Visitors at the GVU Centre at Georgia Tech carrying Apple MessagePads retrieve information according to their location and orientation. The location tracking mechanism used by the Cyberguide is based on information gathered from a series of ceiling based infrared sensors. Each sensor sends a vertical infrared beam covering a small cell. As the user moves from one cell to the other the Cyberguide application can identify the location and assume the orientation of the user.

In terms of architecture, Cyberguide follows a modular approach where the system is composed of special purpose components such as the *navigator* (positioning component), the *cartographer* (map component), the *librarian* (information component) and the *messenger* (communication component). Each of these components can be replaced with a new implementation without affecting the rest of the system.

- **Coordination:** There is no system support that controls or coordinates the applications' behaviour.
- **Extensibility:** The modular approach used by the Cyberguide makes it possible to modify the existing functionality. For example there has been a prototype where the location mechanism has been replaced with a GPS based one. However, it is not possible to add more contextual attributes without modifying the rest of the system.
- **Reconfiguration:** There is no support for reconfiguring the systems behaviour without re-implementation.
- **User involvement:** The user cannot modify the system's behaviour.

2.3.5.3 PARC Tab

The PARC Tab [Schilit94a] is a project developed at Xerox Parc as an attempt to realise the vision of ubiquitous computing described by Weiser [Weiser93]. In the PARC Tab project users carry small custom built hand-held devices that use infrared as a communication and location tracking mechanism. The system is designed for indoor operation where each office acts both as a communication cell and a location identifier. One of the characteristics of PARC Tab is the sharing of contextual information among participants. Therefore it is possible to extend the location information gathered with things like proximity of other users or physical objects.

The underlying infrastructure of PARC Tab uses general purpose configurable mechanisms that describe how context should be used. In particular, the *automatic contextual reconfiguration* allows the system to modify the information presented to the user according to the location of the user or the proximity of other users. The *location based commands* allows the execution of programs according to the physical location of the user. Finally the *context triggered commands* use a simple event language where users can defined notification messages when certain contextual criteria are fulfilled.

- **Coordination:** The PARC Tab does not offer any mechanism for coordinating the execution of context triggered commands or programs.
- **Extensibility:** Even though the PARC Tab is based on a very flexible architecture, all the mechanisms provided are coupled with the location information. Therefore the incorporation of new types of adaptation attributes would require re-implementation of the system.
- **Reconfiguration:** The PARC Tab system offers a wide range of configuration mechanisms allowing the users to tailor the behaviour of the system to their own needs.
- **User involvement:** The user can actively specify or modify the behaviour of the system.

2.3.5.4 Context toolkit

The Context toolkit [Salber99] is collection of tools that aim to provide reusable context-sensing components that can by used for the development of context aware applications. The design of the Context toolkit is influenced by the design of graphical user interface toolkits. More specifically the Context toolkit is built around the notion of context widgets: components that encapsulate the context acquisition mechanism and provide a well known interface. Examples of context widgets include the *identity presence* widget that gives information about the presence of a person in a specific location, the *activity* widget that provides information about the level of activity sensed in a room or the *phone use* widget.

The design of the Context toolkit is based on the combination of three types of entities:

- Context Generator: A context generator is the component that acquires raw data from a sensor and provides it to a widget. A context generator could be, for example, a GPS driver, an active badge reader, etc.
- Context Interpreter: A context widget should provide their information in a given format possibly different from the raw data received by a context generator. A context interpreter is the component that interprets the raw data received by a generator to the format that should be exposed by the widget.
- Context Server: A context server acts as an aggregation widget combining several widgets in order to provide higher level contextual information. For example a combination of the identity presence and the activity widget could be used to create a *meeting* widget.

The implementation of the context toolkit is based on the use of XML for describing the attributes offered by a context widget. An application can register with a widget for notifications describing conditions under which a notification should be fired.

- **Coordination:** Not applicable. The prime target of Context toolkit is to provide a flexible mechanism for an application to acquire contextual information. The issue of coordination is the application's responsibility.
- **Extensibility:** The use of a general mechanism for the specification of new context widgets allows the easy incorporation of new adaptation attributes in the Context toolkit. However an existing application would need to be modified before it could take advantage of a new widget.
- **Reconfiguration:** Automatic reconfiguration of widgets can take place transparently by switching between different context generators. At the application level it is the developers responsibility to provide such functionality.

- **User involvement:** The user can not actively modify the system's behaviour.

2.3.5.5 Cooltown

Cooltown is a project developed by HP Labs to support "web presence" for people, places and things [Kindberg01]. The main idea behind Cooltown is that every entity in the real world (person, place or object) is given a globally unique URL that provides information about the particular entity. A roaming user can discover the URL corresponding to an entity and retrieve information related to that entity.

In general the Cooltown project utilises the web paradigm in order to allow easily configurable access to context related information. Specifically, the user can retrieve information about entities that are close to their current location.

In terms of infrastructure, Cooltown assumes that roaming users have a mobile device that is connected to the World Wide Web (possibly through a wireless link). The mobile device can discover or sense the URL locator that corresponds to a particular entity. Cooltown supports three methods of acquiring the URL related to real world entities. Specifically, *discovery* includes a protocol for service discovery where the user's device multicasts a request for all entities in their environment and receives their corresponding URLs. *Direct sensing* includes a mechanism where entities advertise their web presence by sending a wireless signal in the form of a beacon. The mobile device can receive this beacon when it gets close to the related entity. It is then possible to automatically load the related URL to a web browser and see information about the entity. The *indirect sensing* mechanism follows the same approach as the direct sensing but instead of using a mechanism to advertise the URL directly to the mobile device, other means are used as a lookup key to discover the URL related to the entity. For example, through the reading of barcode keys the mobile device can request the URL for the entity that corresponds to the particular barcode.

- **Coordination:** The Cooltown infrastructure does not include any support for coordinating or controlling multiple entities in the user's environment.
- **Extensibility:** The model of automatic discovery and the use of a standard communication protocol (HTTP/HTML) allows the connection of mobile devices to any available entity in their environment.

Project	Coordination	Extensibility	Reconfiguration	User Involvement
Stand alone	No	No	No	No
Coda	No	No	No	No
Odyssey	Possible	Possible	No	No
MOST	Possible	No	No	No
Rover	No	No	No	No
Tacoma	Possible	No	No	No
Bayou	No	No	Yes	No
Mobiware	No	No	Yes	No
Puppeteer	No	No	No	No
TAO	Possible	No	Yes	No
Open-ORB	Possible	No	Yes	No
OpenCORBA	Possible	No	Yes	No
Guide	No	No	No	No
Cyberguide	No	Yes	No	No
PARC Tab	No	No	Yes	Yes
Context Toolkit	No	Yes	Yes	No
Cooltown	No	Yes	No	No

Table 2.1: Current adaptive and context-aware systems: Support for non-transparent adaptation, extended adaptive triggers, reconfiguration and user involvement.

- **Reconfiguration:** Considering that the Cooltown project does not concern the adaptive behaviour of applications the requirement for reconfigurability of the system's behaviour is inapplicable.

- **User involvement:** The user is mainly a spectator that receives information.

2.4 Discussion

The review of existing systems presented brings out some interesting characteristics (Table 2.1). In particular it is possible to identify common design characteristics followed by certain groups of approach. In more detail:

- Stand alone adaptive applications appear to be quite inflexible in terms of coordination, extensibility and reconfiguration. This is quite reasonable considering that a stand alone application can only consider their own environment and serve a specific purpose as expressed during the design of the application.

- Mobile systems tend to target a limited range of contextual attributes that can

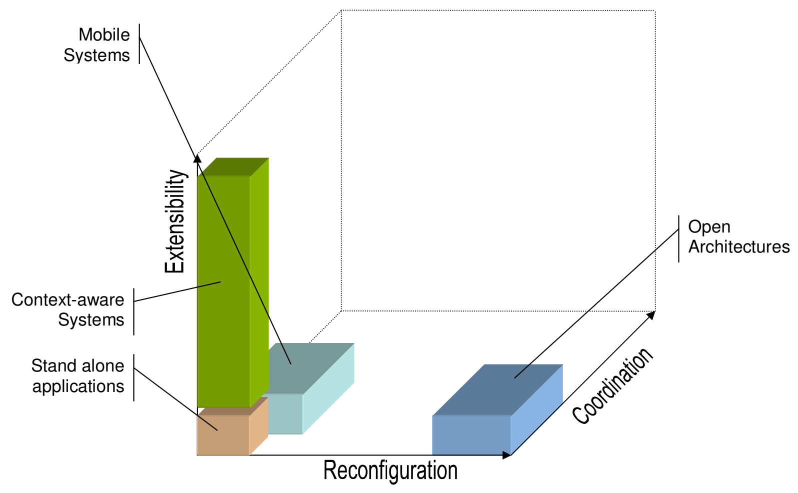


Figure 2.6: Supporting multiple triggers, coordination and reconfiguration.

act as adaptation triggers. In most cases these attributes are related to the QoS offered by the network. In some cases the design of the mobile systems offer the mechanisms to share application state information. However, coordination based on that shared information is generally not supported.

- Open architectures (such as TAO, Open-ORB and OpenCORBA) support flexible reconfiguration of the system's infrastructure. The use of a *reflective* design seen in these systems, allows components within the system to be modified or even replaced during run-time without affecting the operation of the applications active in the system.
- Many of the context-aware systems try to support an extensible mechanism for accessing new contextual attributes that may be of interest to mobile applications. However, in most context-aware systems the actual adaptive behaviour (i.e. how an application responds to context changes) remains part of the individual application. As a result coordination between applications is not possible to achieve.

A very interesting observation is apparent if we try to lay existing research efforts in a three dimensional diagram where *extensibility*, *coordination* and *reconfiguration* are the three axes (Figure 2.6). In this diagram it is clear that each individual research domain is targeting one or in some cases two of these characteristics. However there has been no effort to support all three characteristics in the same system. It should be noted that the issue of user involvement is not represented in this diagram. As discussed in chapter 3 user involvement is considered a cross-cutting feature that extends over all these issues. In chapter 3 we provide an analysis of the implications for systems that do not take into account all these characteristics and highlight the requirements for a system that overcome these problems.

2.5 Summary

This chapter offered a review of existing research in the areas of adaptation and context awareness. In more detail, a brief overview of the emergence of adaptation is given including references to distributed multimedia systems, mobile systems and context-aware systems. The chapter then provides an in-depth review of those adaptive and context-aware systems relevant to this thesis. The review is based on a set of assessment criteria, namely: *coordination*, *extensibility*, *reconfiguration* and *user involvement*. Finally, this review concludes that no existing systems provide full support for all these characteristics. An analysis of the importance of this finding is given in chapter 3.

CHAPTER III

Analysis

Contents

3.1 Overview

As presented in the previous chapter, existing adaptive and context aware systems are targeting specific areas within the domain of adaptive context-aware systems. This chapter presents an analysis of the design principles that govern existing approaches and advocates the need for a new approach. In particular, a set of scenarios is presented that illustrate the limitations of existing systems in supporting multiple adaptive context-aware applications within the same system. Each of these scenarios is followed by an analysis section that identifies the limitations of the design approach followed by existing systems and introduces a possible approach to overcome them. Lastly, a set of design requirements is presented for a system that can successfully provide support for multiple adaptive context-aware applications.

3.2 Challenges in Adaptation

This section illustrates possible limitations in existing systems and gives a short analysis on the reasons behind these limitations. The analysis of each of the issues presented will use references to the theoretical model of the basic adaptation cycle presented in section 2.3.1.

3.2.1 Coordinated Adaptation

3.2.1.1 Scenario

This scenario illustrates how the lack of coordination between adaptive applications can lead to inefficient power management on a mobile system. One existing approach for handling power management, i.e. the ACPI [ACP99] model, is to enable the operating system to switch hardware resources into low power mode when not in use, e.g. spinning down the hard-disk. This approach requires that applications leave hardware resources in an idle state for sufficient periods of time to make the transition between idle and active states worthwhile. Although this approach is suitable when only one application is running on a mobile device, the approach can prove ineffective when multiple applications or system services are sharing hardware resources. In more detail, the lack of coordinated access to hardware resources can result in poor utilisation of the shared resource and therefore sub-optimum power management. For example, consider the case of multiple applications that implement an auto-save feature. In the absence of any coordination between applications each application may choose to checkpoint its state to the disk at an arbitrary time, without considering the state of the disk (i.e. spinning or sleeping). In contrast, if applications are able to coordinate their access to the hard-disk then access to the disk can be clustered, allowing longer periods of inactivity. It follows that the latter approach is more power efficient than the situation in which usage of the hard-disk is completely unregulated.

Further scenarios illustrate the benefits of coordination when taking into account the user experience. For example, a mobile device that is connected through a low bandwidth wireless link would typically experience network congestion: coordinating network applications so that applications less important to the user suspend network activity in favour of the more important ones could provide a much better user experience.

3.2.1.2 Analysis

The term *coordinated adaptation* refers to the ability of an adaptive system to invoke adaptive reactions on multiple applications in a coordinated manner so as to achieve a common goal. In the scenarios presented, coordinated adaptation would be required in order to overcome the power inefficiency and to satisfy the users requirement for a more efficient utilisation of the limited network bandwidth.

As illustrated in chapter 2, existing systems offer limited support for coordination. In particular, most context-aware systems do not consider the support for coordinated adaptation between multiple applications. The design principle behind these systems that restricts their support for coordination is the fact that actuator components and control mechanisms are usually hard-coded within the applications. Therefore its application is only capable of triggering adaptation to itself.

In contrast, some mobile platforms are trying to offer a form of coordination in terms of resource sharing. This limited support for coordination is based on the fact that adaptation support platforms can collect information about the state of existing applications and use that information in order to share system resources according to the applications' needs. However, this approach has its limitations. In particular, the adaptation support platform does not have any control over the adaptation actions taken by the applications. This is again related to the fact that adaptation mechanism and adaptation control is encapsulated within the application.

The design approach where application state information can be accessible by external entities and the fact that adaptation mechanisms and control are encapsulated within the application have influenced the proposition of requirements R1 and R2 (Section 3.3).

3.2.2 Conflicting Adaptation

3.2.2.1 Scenario

In this scenario, we illustrate the potential problems that can occur in a system that utilises separate adaptation mechanisms for different attributes. We consider a hypothetical mobile system that utilises two independent adaptation mechanisms, one for managing power and the other for managing network bandwidth. The two mechanisms can conflict with one another as the following example illustrates. If the system needs to reduce power consumption, the power management mechanism will request those applications that are utilising network bandwidth to postpone their usage of the network device in order to place the network device into sleep mode. As a consequence of applications postponing their use of the network, the available network bandwidth increases. However, the network adaptation mechanism will detect this unused bandwidth and notify applications to utilise the spare bandwidth. In this way, the request to utilise available bandwidth is in direct conflict with the request to postpone network usage.

This example highlights the problem of relying on independent and uncoordinated adaptation mechanisms. The reason behind conflicting cases such as these is the fact that adaptation control entities have no knowledge about the state of other applications in the system and the possible interdependencies of adaptation actions of multiple applications. A system supporting multiple adaptive applications should provide the mechanisms to resolve potential conflicts. In particular, the ability to reconfigure the adaptation support system and/or the adaptation controlling entity of the applications is of great importance. Indeed, a system that allows reconfiguration without re-implementation of the applications would allow the resolution of such conflicts.

3.2.2.2 Analysis

In a system with multiple adaptive context-aware applications conflicts will inevitably occur. The actual reason behind conflicts is the lack of awareness of the application developer of the characteristics of the target system and the possible interdependencies between applications, or the possible side-effects of certain adaptive reactions that may affect co-existing applications in such environments.

In the scenario presented here the network triggered adaptive application is unaware of the importance of power consumption in the adaptation strategy employed by other applications in the system. The solution in this scenario is to modify the control mechanism that triggers adaptation in this application so that it will take into account the available power of the system. Generalising this approach, most conflicting situation can be resolved by modifying the control mechanisms in the adaptation cycles of the involved applications.

Existing mobile and context-aware systems appear to be unable to resolve such conflicting situations. Indeed, such systems keep the control mechanism of adaptation hard-coded within the system and therefore do not allow any modification. In contrast, open architectures allow for such modifications and as a consequence these conflicts could be resolved by modifying or replacing an existing control mechanism with a new one. These facts lead to the specification of requirements R1 and R4 described in section 3.3.

3.2.3 Extensibility

3.2.3.1 Scenario

This scenario considers the extension of an existing application with the inclusion of additional contextual triggers. In this case, we consider a common MP3 player application able to playback local audio files. Assuming that this application is used on a mobile environment supporting location-awareness a possibly desirable extension of the application would be to allow the automatic control of the player's volume based on the user's context. For example, an interesting feature could include the automatic turning down of the player's volume when the user is walking through a shared office space or when they pass in front of office doors.

This feature would require the incorporation of a new contextual attribute to the existing application. Assuming that the functionality for lowering the volume down is already implemented by the application, this scenario advocates a mechanism where this functionality can be triggered when the location of the user changes. In practice it requires the addition of a control mechanism that can retrieve location information from the environment and trigger the particular function on the MP3 player.

3.2.3.2 Analysis

One of the characteristics of adaptive context-aware applications is the fact that they may be triggered to adapt to a variety of different contextual attributes, be that the availability of a specific resource, or the user's context.

The survey of existing systems presented in chapter 2 revealed that existing system support a limited number of possible adaptive triggers. In particular most of the mobile middleware systems consider only the network QoS as a possible trigger for adaptation. Moreover they do not offer any mechanism to extend their support for adaptation triggered by other environmental attributes. This feature is reflected in their design approach by the tight coupling of their monitoring mechanism and their control mechanism. In more detail, mobile middleware systems, such as Odyssey, Coda, Most, Mobiware, etc. have the adaptation control mechanism as a hard-coded element within their middleware infrastructure. A similar approach is presented by all application specific systems, such as GUIDE, Puppeteer, etc. where the monitoring and control mechanisms are hard-coded within the application.

A different approach is used by systems offering middleware support for context-aware systems, such as the Context-toolkit and Parc Tab. In these systems the context monitoring mechanism is decoupled from the actual control mechanism that triggers adaptation. This allows the introduction of new contextual triggers into the system with minimal effort. Indeed, a system that decouples the monitoring entity and the control entity and uses a well defined method for connecting the adaptation controller with monitoring entities would support extensibility. The observation leads to the requirement R3 (Section 3.3) for the externalisation of application state information.

3.2.4 User Involvement

3.2.4.1 Scenario

This scenario considers the case of two applications, an adaptive web browser and an application for viewing a video stream, competing for the same resource, (network bandwidth). In particular, following a drop in available bandwidth the two applications could react using one of the following adaptive strategies:

1. The web browser could stop downloading in order to dedicate its portion of bandwidth to the other application.
2. Both applications could adapt and share the available bandwidth equally.
3. The video viewer could adapt by reducing its bandwidth requirement, e.g. by reducing its frame rate, in order to enable the web browser to utilise a greater share of the available bandwidth, e.g. if an important download is taking place.

The reaction that would be most appropriate depends on the user's requirements and additional contextual information, such as the importance of a particular download. In order for the two applications to adapt differently under different conditions there is a clear need for reconfiguration of the adaptation policies as directed by the user needs.

3.2.4.2 Analysis

The need for reconfiguration has been illustrated by almost all scenarios presented in section 3.2. In all presented cases an existing problem would be solved by modifying the way some particular adaptive decision is being taken. The scenario presented here extends the notion of reconfiguration by introducing the involvement of the user on the way the system should behave.

In the systems presented in chapter 2 it is clear that there is a lack of support for reconfiguration in most system types. This inability to reconfigure is directly related to their design approaches. In more detail, both mobile adaptive systems and context aware systems typically have their decision mechanisms coupled with their monitoring mechanisms or their adaptive mechanisms. Therefore, it is not possible to modify the behaviour of the system without altering their overall architecture.

A different approach is used by the open architectures. More specifically, open architectures follow a modular approach where all components are distinct entities with predefined interfaces bound together. This approach allows the possible replacement or modification of a system module without affecting the rest of the system. The Open-Orb in particular allows such replacements to take place even during runtime.

However, even in open architectures, the user involvement in the specification of the system behaviour is neglected. Any modification or reconfiguration that can be done in these systems requires the re-implementation of the system part that needs to change. Therefore the possible reconfiguration choices are prescribed by the system developer who has to include any alternative configurations within the deployed system. Moreover, any possible reconfiguration actions can only be performed from within the system which again leads to the system developer as the only possible actor that can specify the system's behaviour.

The lack of a mechanism where the adaptation mechanism can be modified by the end user implies a requirement for reconfiguration of the adaptive control mechanism without the need for re-implementation (requirement R4).

3.2.5 Conclusions

Any simple adaptive system follows the basic adaptation cycle presented earlier. However, in a system where several applications or multiple triggering attributes exist adaptive decisions and actions may depend on information that spans several applications and information sources. Therefore, it is important for the underlying adaptation support mechanism to allow adaptive decisions to consult a variety of different adaptation attributes and trigger adaptation on a number of coexisting applications.

As illustrated in the previous sections, existing systems offer limited support for these features. In more detail, most of the existing systems follow a design approach that couples the decision mechanism with either the monitoring mechanism or the adaptive action.

As seen in scenario 3.2.2, an adaptive mechanism that is triggered in order to reduce the level of power consumption may have a side effect on the level of available network bandwidth of the system. These side effects are the main cause of conflicts. In a system with multiple adaptive applications it is reasonable to expect that conflicts will happen. However, as these conflicts are highly dependent on the actual configuration of the end system, it is not realistic to expect that the application developer would be able to provide appropriate conflict resolution mechanisms a priori.

As a solution to these problems the system should be modified in order to overcome these conflicts. In most cases the approach that allows conflict resolution is to coordinate the applications' adaptive behaviour in a suitable way. Coordination can be considered as a desirable feature for the end user not only in terms of conflict resolution but also in significantly improving the user experience, as described in scenario 3.2.4. In an abstract sense, the system should act as a glue that will coordinate applications and system components in order to allow them to collaborate in harmony without conflicts or undesirable behaviour. In order for this functionality to be achieved it is necessary for adaptive applications to follow a design where all three entities described in the adaptation cycle are decoupled and clearly identifiable. Moreover, the system supporting adaptation should be in charge of the controlling entities of the applications' adaptation cycles. This way the adaptation support system would be able manage and allow reconfiguration of the adaptation behaviour of the applications.

Another issue that is apparent from the scenarios is that regardless of whether coordination, conflict resolution or user involvement is concerned, the participation of the user in specifying the system's behaviour is vital. However, in most current adaptive applications the adaptation policies are not distinct elements within the adaptive cycle. Indeed, adaptation policies are typically hard-coded within either the monitoring process or the adaptive mechanism. To allow the necessary level of control over the behaviour of the system the adaptation policies must be decoupled from the adaptation mechanisms themselves. Moreover, these policies should be defined in a language flexible enough to allow the specification of conditions that can include multiple triggering events that may be introduced in the system over time.

3.3 Requirements

The previous section has analysed the limitations of current approaches for supporting adaptive context-aware applications. In particular, these approaches lack the appropriate support for enabling applications to adapt to numerous different attributes in a coordinated and reconfigurable way. A new approach is therefore required which provides support for coordinated, system-wide interaction between adaptive applications and the complete set of attributes that could be used to trigger adaptation.

This section considers a set of requirements that could be used to develop an appropriate architecture for supporting adaptive mobile applications.

3.3.1 R1. Decouple Adaptation Control and Adaptive Actions

One of the issues described earlier is the lack of support for coordination in existing systems. This is caused by the lack of awareness of application developers of the possible configuration of the target system and thus the applications that will co-exist at runtime. Therefore, in order to design a system where multiple applications can coordinate their adaptive behaviour it is not feasible to rely on applications to achieve coordination without external support. As described in section 2.3.2.1 the ability of a system to support coordination is expressed through the ability of adaptation controllers to trigger adaptation to actuators of multiple applications. Combining the two observations it is clear that supporting adaptation would require the controlling entity to be handled by the system so that adaptation triggering can be directed to more than one application at once. In order to achieve such a feature it is necessary to decouple adaptation controllers and the implementation of the adaptation mechanisms. Through this decoupling we can construct a system component responsible for handling the adaptation control mechanisms of all applications in a system, taking into account interdependencies and required coordinated operation.

3.3.2 R2. Export Application State

Supporting both extensibility and coordination requires an adaptation mechanism that can take into account information about the state of multiple applications and/or information collected from context-monitoring entities. For example, the fact that a video player is actively streaming video over the network may be of importance to other network based applications when choosing to adapt. Therefore, applications should externalise information about their state and allow adaptation controllers to take that state information into account. Moreover, system components that monitor contextual information should externalise that information in order to allow adaptation controllers to have access to that contextual information.

3.3.3 R3. Export Adaptive Mechanisms

Following the requirement for decoupling adaptation control mechanisms and adaptation methods it is necessary to define the necessary requirements that allow the adaptation controllers to invoke adaptation methods. As discussed in section 2.3.2.1 adaptation methods are in general bound to the semantics of the actual application. Indeed, the adaptation methods that an application can support depend entirely on how the actual application is implemented. It is the application developer that implements an application in a way that permits certain adaptive behaviour to be performed.

Following this observation, a system where adaptation controllers and adaptation mechanisms are decoupled should include a mechanism where adaptive applications can export their adaptation interface. In more detail, applications should allow adaptation controllers to dynamically inspect the applications' adaptation interface and invoke adaptive mechanisms as and when needed.

3.3.4 R4. Enable Modification of Adaptive Behaviour

Adaptation support systems should provide a mechanism where application adaptation control can be reconfigured without the need for re-implementation of either the application or the adaptation support platform. Moreover, the design of the adaptation controller should allow modifications by the end user thus allowing the user to explicitly specify how the system should behave.

This requirement for modification of the adaptation control mechanism is perhaps the core requirement for tackling the issues presented in section 3.2. Overcoming conflicts in most cases would require modifications of the adaptation control mechanisms of the conflicting applications. Improving the use of system resources may be achieved by coordinating the adaptation on multiple applications. This could be done by having adaptation control mechanisms that can invoke adaptation of multiple applications. User preferences could be expressed by modifying the behaviour of existing control mechanisms to better suit their needs. Extending the behaviour of an existing application, adding awareness of more contextual attributes could be achieved by appropriately modifying their control mechanisms so that they take into account monitoring information offered by other entities in the system. This approach could, for example, turn an adaptive web browser into a location-aware web browser by modifying their adaptation control mechanism so that it will take into account location information provided by a GPS device attached to the system. In this thesis we argue that such behaviour requires a flexible policy based approach. This approach is discussed in detail in the following chapter.

3.4 Summary

This chapter illustrated the possible shortcomings of existing applications when considered in an environment with multiple adaptive context-aware applications. In particular, the issues of coordination, adaptation conflicts, user involvement, etc. were highlighted and analysed. Following the analysis of the reasons behind these shortcomings, a list of design requirements was presented. The following chapter presents the design of platform supporting adaptation based on the aforementioned requirements.

CHAPTER IV

Design

Contents

4.1 Overview

In this chapter the design of a platform supporting coordinated adaptation for adaptive context-aware applications is presented. The chapter begins with a discussion of the requirements presented in chapter 3 and their implications for the design of a supporting platform. Specifically, the discussion walks through the requirements and presents how coordination can be achieved through a platform that supports these requirements. Following this discussion an architecture for supporting coordinated adaptation is presented. This architecture uses a policy-based mechanism for controlling adaptation allowing coordination and reconfiguration. The subsequent sections then discuss in detail each component of the architecture including potential design approaches for realising this architecture. The last section of this chapter is dedicated to the presentation of the Event Calculus Policy Language, a language that was designed to satisfy the requirements of a policy-based adaptation system supporting adaptive context-aware applications. A prototype implementation of the platform design is presented in chapter 5.

4.2 Architectural Discussion

In order to identify desirable features for a platform to support coordinated adaptive context-aware applications it is necessary to analyse the fundamental characteristics of adaptation. As discussed in chapter 2, the operation of an adaptive application is similar to the operation of control systems. More specifically, a simplified closed loop system can be considered as an abstraction of an adaptive system (Figure 4.1). This adaptation system includes three entities: the monitoring entity feeding the system with information that may cause adaptation, the control entity that is responsible for deciding when adaptation is required and the actuator that implements the adaptive mechanism.

In typical adaptive or context-aware applications all three components are part of the application itself. This is principally a consequence of previous research that has shown [Noble97] that fully transparent adaptation platforms cannot provide sufficient support for the requirements of all applications: current approaches in the design of adaptive applications advocate the breaking of the transparency and the shift of adaptation mechanisms away from the system and into the application itself. Indeed, it is normally the

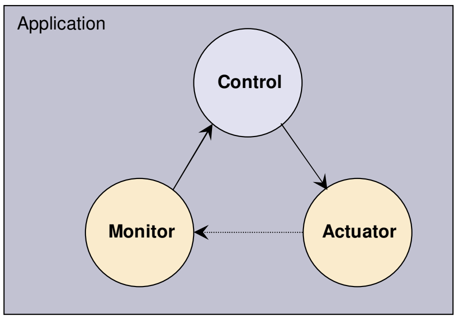


Figure 4.1: Basic adaptation cycle

application developer that has a clear knowledge of the application's semantics and requirements. Therefore, the developer can best implement the adaptation mechanism required for a specific application. In typical examples of adaptive applications the application developer implements two of the three adaptation entities: the actuator that is directly related to the logic of the application and the control entity that is usually hard-coded as a static component that decides when adaptation is required. In these systems the monitoring entity is offered by the system (e.g. a power monitoring tool). In certain cases even the monitoring entity retrieving the specific information that is necessary for the operation of the adaptive application is implemented by the application developer (e.g. in [Davies99, Microsoft03, Real03]). It is also quite common for such systems to blur the boundaries between the three entities, sometimes combining two or even all three of them. In particular, it is quite common for adaptive systems to combine the control entity with the actuator [Kistler92, deLara01], while context-aware systems tend to combine the monitoring entity with the control system [Cheverst00, Schilit94a].

In chapter 3 the analysis section showed clearly why this static architecture cannot support systems with multiple adaptive context-aware applications: lack of coordination can lead to conflicts and low performance while lack of user involvement can lead to insufficient support for the user requirements. Chapter 3 concludes with a list of requirements for a system that can overcome the aforementioned limitations.

The first requirement (R1) for supporting adaptive context-aware applications is to decouple the adaptation control and the actuator. This requirement is based on the fact that the mechanisms implementing adaptation actuators are tightly linked with the semantics of the application and therefore should be part of the application's implementation. In contrast, the adaptation controls are entities that play a more general role: the

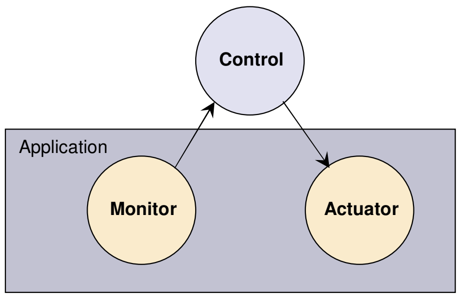


Figure 4.2: Decouple adaptation control and adaptation action

role of an adaptation controller is to receive input in terms of value changes and produce output in terms of invocations of adaptation mechanisms. Therefore, it is possible to follow the same design guidelines for designing all adaptation controllers regardless of the application being controlled. In addition to this fact the decoupling of adaptation control and adaptation actions allows adaptation control entities to be externalised and become part of the system supporting adaptation. This means that the application developer is no longer responsible for implementing the adaptation control mechanism but they can rely on the system support offering the control mechanism (Figure 4.2). Finally, the need for decoupling adaptation control and adaptation actions is a prerequisite for meeting the the requirements for externalising application state and adaptive mechanisms, necessary to achieve coordinated adaptation.

The second requirement (R2) for supporting adaptive context-aware applications is to externalise application state information. This requirement is based on the fact that information collected by an application as part of their monitoring functionality may be of importance for the adaptation controllers involved in the adaptation cycles of other applications in the system. Externalising information reduces the need for replicating similar monitoring functionalities in many applications. Moreover, the state of an application running in a system can be an important factor for the decision of other applications about when and how to adapt. In terms of platform design this requirement suggests a system design where adaptation control entities can receive monitoring information from multiple applications (Figure 4.3). In practice this means that the adaptation controller of a single application is not isolated from the activities of other applications active in the system. This externalisation requires an open design approach where applications can express their state through a specified interface definition language. Moreover, since the actual state variables that are reported by applications are not known in advance, the system supporting adaptation must be able to dynamically parse the interface exported by the application and construct the appropriate components

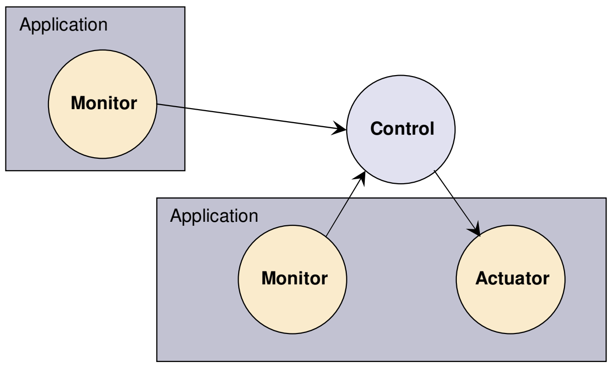


Figure 4.3: Externalise application state

that will collect application state information during runtime.

The third requirement (R3) for supporting adaptive context-aware applications is for applications to externalise their adaptive mechanisms. Satisfying this requirement means that adaptation control entities do not have to be related to a single application. As all adaptive applications in an adaptive system externalise their adaptation mechanisms it is possible for adaptation controllers to invoke adaptation in multiple applications. This functionality allows the implementation of coordinated adaptation: a single adaptation controller responsible for implementing a specific adaptation policy can trigger multiple applications to perform coordinated actions as required (Figure 4.4). In terms of design, this requirement can be supported by expressing an application interface as described in the previous paragraph. More specifically, the application interface exported by applications should include the definitions of methods corresponding to adaptation actuators. These methods could then be called by external entities in order to request that the application performs a specific adaptation action. As discussed above, application interfaces should be dynamic, i.e. the external entity invoking actions on applications should be able to dynamically marshall the data required to perform the

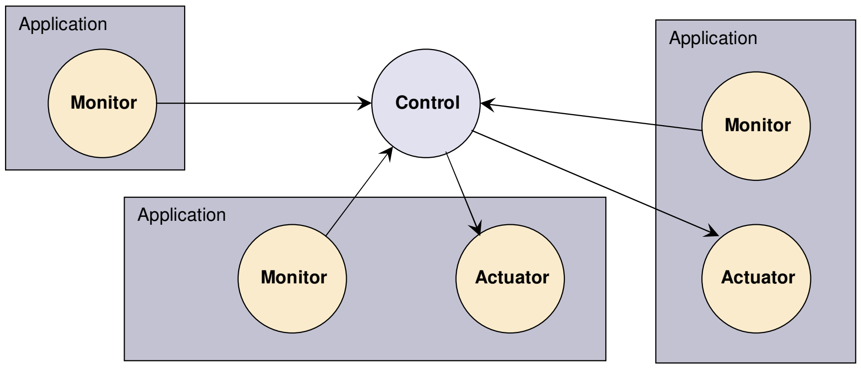


Figure 4.4: Externalise application adaptive mechanisms

invocation as described by the application interface.

The final requirement for supporting adaptive context-aware applications is to support a mechanism that enables adaptation control to be modified by the end user. This means that the actual decision taking policies implemented by the adaptation controllers should not be hard coded by a developer but rather provided in a way where the decision logic can be inspected by the end user. Since applications require default adaptation control functionality when they are installed, it should be possible for the end user to have access to this default adaptation control policy and to be able to modify it as they wish. Apart from support for user involvement, this requirement is also the basis for supporting extensibility and coordinated adaptation. Both of these features include adaptation controllers that either receive monitoring information from multiple applications or invoke actions on multiple applications. It is not realistic to expect such controllers to be defined by an application developer as default adaptation control policies. Indeed, application developers do not have any knowledge about the existence of other applications in the end system. Therefore, default adaptation policies that are specified by the developer do not perform coordinated adaptation (invoking actions to other applications in the system) or receive monitoring information from other applications. However, the end user, being aware of the configuration of the end system, is capable of modifying these default policies in order to achieve coordinated adaptation. The design requirements derived from these observations are that the adaptation control mechanism should be based on a design where the end user can:

- Inspect the adaptation policies employed by existing adaptation controllers.
- Modify the decision logic of existing adaptation controllers.
- Extend existing controllers with the inclusion of new monitoring information.
- Extend existing controllers with the inclusion of new targets for adaptive actions.
- Add their own adaptation controllers with all the above features (multiple monitors, multiple actions).

Technologies that provide the aforementioned features have already been used in other domains of computer science. A particular approach that has been employed in systems where reconfiguration without re-implementation is needed is the use of policy management systems. As E. Lupu and M. Sloman define in [Lupu99]:

"A Policy is information which can be used to modify the behaviour of a system. Separating policies from the managers which interpret them permits the modification of the policies to change the behaviour and strategy of the management system without re-coding the managers. The management system can then adapt to changing requirements by disabling policies or replacing old policies with new ones without shutting down the system."

The features of policy based systems described in this definition appear to cover a significant part of the requirements presented earlier for supporting coordinated adaptation and in particular, the ability to adapt the behaviour of a system without the need to re-code the management system itself. The extent to which a policy based system can support all of the aforementioned requirements depends to some extend on the specific characteristics of the policy language used.

4.3 Architectural Overview

Based on the design features described in the previous section a high level architecture is proposed for a platform for supporting coordinated adaptation of adaptive context-aware applications (Figure 4.5). This architecture describes an adaptation support platform that cooperates with running adaptive applications in order to provide re-configurable coordinated adaptation. The main role of the platform is to act as the adaptation control entity for all adaptive applications running in the system. From an application's point of view the platform is the point where they report any changes in their state or environment monitoring information and from which they expect requests to perform adaptation. Features described in section 4.2 such as coordination, extensibility and user involvement are realised by the platform without any re-implementation of the applications.

The platform builds on the control of policies, realising and utilising a policy based management system for controlling adaptation. Adaptation policies are described through a human readable policy language specifying the conditions that can trigger adaptation and the actions that need to be performed. The specification of the conditions that can trigger adaptation are related to the information that is reported by applications. For example, a web browser application may report that it is currently downloading a large file. The system manager component that is responsible for evaluating the policy rules

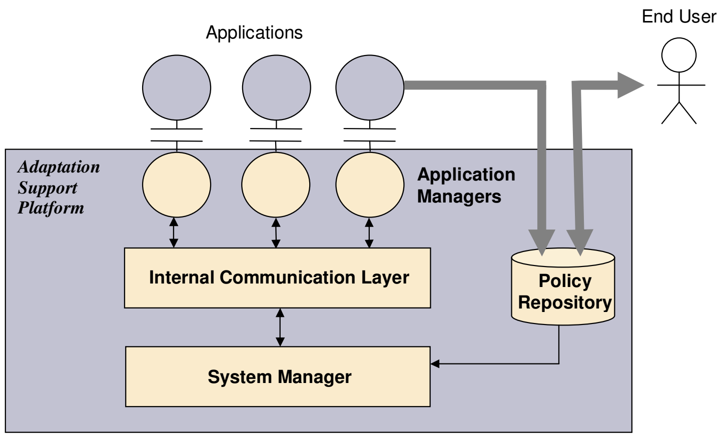


Figure 4.5: Architecture for supporting coordinated adaptation

active in the platform may have a specific policy rule that will take this fact into account and possibly request an adaptive response by one or more applications running in the system. The adaptation actions that need to be performed are translated into method invocations on applications' adaptation control interfaces. These adaptation methods represent the application's implementation of an actuator. For example, a web browser may have a method that can switch the downloading stream from raw data to compressed data. A policy rule in the system manager may use that method as part of a request for adaptation.

In order for such interactions between the platform and the running applications to be realised, applications need to define an interface that describes the information that they can export and the methods that can be called by the platform. The platform uses this interface specification in order to dynamically create an application manager component. This component is responsible for handling all information exchange with the application and all method invocations requested by the system manager. Moreover, the application managers act as caches for the information reported by the applications. Specifically, when the system manager requires the value of a specific variable reported by an application, the application manager returns the last update of that value.

As a glue between the application managers and the system manager, the internal communication layer handles all communications between the platform components. The internal communication layer maintains an ordered delivery of application information to the system manager as well as invocation requests from the system manager to the application managers.

User involvement is achieved by allowing the user to access the policy repository where adaptation policies are installed. The user can inspect and modify existing policies or add new ones. The policy repository is also available to new applications installed in the system. New applications need to install default adaptation policies as specified by the application developer. As discussed earlier, the policy rules that implement the adaptation controller are affected by the information exposed by the applications and affect the activities of the applications by invoking adaptation actions. Therefore, to allow the user to understand properly the logic of the policy rules in the policy repository it is necessary to offer the means for a comprehensive description of the application interfaces involved. In more detail, for the end user to understand the meaning of a policy rule that triggers a web browser to switch the raw data downloading stream to a compressed data stream it is necessary to provide a human readable description of the semantics of the action implemented by the application. In practice, this means that the application interface exposed by an application should include a human readable description of its functionality comprehensible to the end user.

In the following section we discuss the critical aspects of this design in more detail. In particular, the issues discussed include the application interface and the mechanism for application-platform communication, the design of the adaptation manager component, the design of the internal communication layer and the system manager component.

4.4 Application Interface and Communication

Before looking into the design of a mechanism that supports communication between adaptive applications and the adaptation support platform, a brief overview of existing technologies that can support such functionality is investigated. Following this, the design of the adaptation interface is discussed and possible design approaches are presented.

4.4.1 Background

As discussed in section 4.3, the architecture for supporting adaptive context-aware applications requires applications to export an interface where they specify the information they can offer and the adaptation actions that can be invoked. This interface can be used by the platform to invoke adaptation methods or monitor the state of the application.

The issue of application interface specification has always been an integral part of the design of distributed middleware platforms, such as CORBA, Java RMI and Web Services. The following sections provide a description of the mechanisms provided by these platforms.

4.4.1.1 CORBA

CORBA (Common Object Request Broker Architecture) is OMG[1]'s open specification for supporting distributed object oriented applications [OMG01]. One of the characteristics of CORBA is its support for interoperability across different hardware platforms and programming languages.

CORBA applications are composed of objects that may be located on a number of distributed hosts. In order for these objects to interact with each other, each one defines an interface in OMG IDL (Interface Definition Language). The IDL specification describes the external boundary of the object through whitch all network communication is performed. Any client that wants to invoke an operation on an object must use this IDL interface to specify the operation it wants to perform, and to marshall the arguments that it sends. When the invocation reaches the target object, the same interface definition is used to un-marshall the arguments so that the object can perform the requested operation.

The IDL interface definition specifies the operation that can be performed on a given object, and all of the input and output parameters with their types. The actual interface definition is independent of programming language. In the typical (static invocation) scenario, the IDL definition is compiled through an IDL compiler in order to generate the client's stub code and the server's skeleton code. Stubs and skeletons serve as proxies for clients and servers, respectively (Figure 4.6). This interaction method, called *static invocation*, requires the IDL of the remote object to be known during the development of the client.

CORBA also supports a *dynamic invocation* method where a client can invoke operations on a remote object without compile time knowledge of the remote object's IDL. In more detail, the IDL compiler generates type information for each method in an inter-

[1]Object Management Group

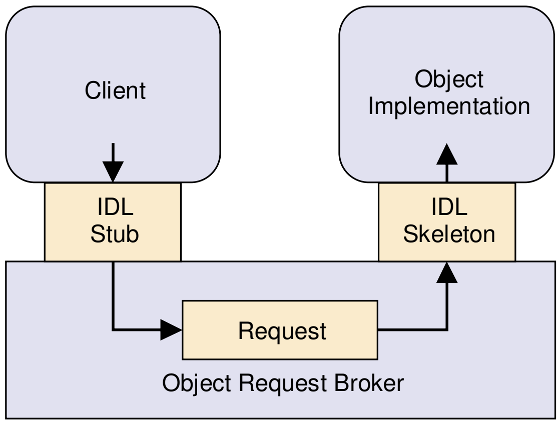


Figure 4.6: A request passing from client to object implementation.

face and stores it in the Interface Repository (IR). A client can thus query the IR to get run-time information about a particular interface and then use that information to create and invoke operations on the remote CORBA object dynamically through the Dynamic Invocation Interface (DII). On the server side, the Dynamic Skeleton Interface (DSI) handles the dynamic client invocations.

In summary, the interface description mechanism provided by CORBA uses a programming language independent description language for specifying the operations provided by a CORBA object. In the common scenario, this description should be available to the client during compile time. However, CORBA offers a mechanism for discovering an object's interface during run-time and dynamically invoking operations on this interface.

4.4.1.2 Java/RMI

Java/RMI [Wollrath96] is the remote method invocation mechanism for distributed Java objects. Unlike CORBA, Java/RMI requires both client and server to be implemented in Java. Each Java/RMI Server object defines an interface which can be used to access the server object from a remote client. A client can locate a remote server object using the RMIRegistry: a Java/RMI specific naming service.

Java/RMI uses Java language constructs to define a server object's interface. In particular, the interfaces are .java files that are compiled along with the object's implementation. In a fashion similar to CORBA, the typical Java/RMI interaction requires the interface of a java object to be available to the client during compile time.

Dynamic invocation of a remote java object without any prior knowledge of the object's interface is available through java's Reflection mechanism. In more detail, the java.lang.reflect package, allows a client to discover at run-time the class of a remote java object, examine the class to discover what methods are available and invoke these methods with dynamically created arguments.

4.4.1.3 Web Services

Web services [W3C01] were designed to offer interoperability between different applications. The communication interfaces provided by the web services are language and platform independent. In more detail, a web service offers an interface that describes a collection of operations/methods that can be accessed through the web using XML messages (SOAP protocol [W3C00]). This description hides the implementation details of a web service but offers all the information necessary to interact with the service. This implementation transparency allows the use of a web service independent of the platform or language used to develop the service.

The Web Service Description Language (WSDL) is the language used for describing the interfaces of web services. The WSDL defines an XML grammar for the structured description of the services and the operations they offer. The XML document with the description of a web service consists of all the information required to discover and interact with a web service. In particular, the information in a WSDL document includes the name of the service, the operations that can be called as well as the location of the service where operation invocations should be directed. A client can use this WSDL document to discover the functionality of a service and how to access the service.

The main focus of Web Services is on dynamic discovery and use of services. A client can dynamically discover a web service (i.e. using directory services) and access its methods.

4.4.2 Application Interface Design

The previous section gave an overview of the most prevalent existing technologies that support application interface specification. From this presentation it is clear that all of the technologies discussed offer the means for distributed applications to describe an interface (and in particular their adaptation interface) through a predefined interface definition language. Moreover, all technologies support the dynamic inspection of this interface and the dynamic invocation of methods exported through the interface.

As discussed in section 4.3, applications that communicate with the adaptation support platform discussed here should also provide one additional feature through their interface specification, i.e. an adaptation interface specification should include human readable descriptions of the semantics of the interface. In more detail, an adaptation interface should offer mechanisms to allow the retrieval of textual descriptions of the application itself, the functionality implemented by the adaptation methods and the meaning of state variables reported by the application.

As seen in section 4.4.1, none of the existing technologies support this functionality by default. However, it is possible to use these technologies to support such a feature. In more detail, apart from the adaptation interface that applications expose, applications could be required to implement a description interface. A possible description interface for adaptive application is shown bellow:

```
interface IDescription
{
    int GetAppDescription(out string sDescription);
    int GetMethodDescription(in string sMethod,
                                      out string sDescription);
    int GetVarDescription(in string sVarName,
                                      out string sDescription);
}
```

The method GetAppDescription returns the description of the application, GetMethod-Description returns the description of the requested method and GetVarDescription return the description of a variable defined in the application interface. This interface could be queried by a user support module in order to give human readable descriptions of the application's interface.

A possible alternative approach can be applied in the case of an XML based interfaces specification language, such as the one used by the Web Services architecture. One of the characteristics of XML is the fact that an existing XML language can be extended with additional tags without breaking backwards compatibility. In more detail, a WSDL definition of a message sent to a web service has the following format:

```
<message name="SetBandwidth">
   <part name="bandLimit" type="xsd:integer"/>
</message>
```

Representing the invocation of the method SetBandwidth(int bandLimit). Based on the backwards compatibility feature of XML it is possible to extend this definition with an additional tag without breaking the support for standard web service clients:

```
<definitions  ....  xmlns:ad="http://www.comp.lancs.ac.uk/wsdl−adapt−schema/" />
    ....
<message name="SetBandwidth">
    <part name="bandLimit" type="xsd:integer"/>
    <ad:description>
        Sets the upper bandwidth limit for the network traffic......
    </ad:description>
</message>
```

With this definition the application can communicate with the platform using the standard Web Service infrastructure, while a user support module can query the application interface and present it to the user (possibly passing it through an XSL filter).

Summarising the discussion on existing technologies, it is clear that existing technologies can support the requirements for interface definition for adaptive context-aware applications. In particular, the use of an XML based approach seems appealing as it allows a more elegant incorporation of user readable descriptions of the applications' interfaces.

This design chapter does not make specific claims about the technologies that should be used for the communication between adaptive applications. Instead, in the following section we present a platform-independent interface description language that can either be used as it is or taken as a guideline for the implementation of a custom interface definition mechanism.

4.4.2.1 Service Interface Definition

According to the discussion in section 4.3, the features that the interface specification mechanism should have are:

- Allow dynamic inspection of the application's interface.
- Allow dynamic invocation.
- Support human readable descriptions of the interface's semantics.

In this section we present an XML based interface description language that meets these requirements. In particular the application exports an XML document that is roughly divided into three parts (Figure 4.7).

The first part of the interface description includes application specific information. In particular, the interface defines the name of the applications as well as a unique id that allows the distinction between multiple instances of the same application. It is the application's responsibility to make sure the id is unique among multiple instances of the application. This can be achieved trivially by creating an id using the current host's MAC address and the process id of the running application.

The second part of the application's interface specification involves the identification of all adaptation methods implemented by the application. The identification of the adaptation methods is indicated by a string representing the name of the adaptation method, and a set of parameters that can be passed as in or out arguments by the platform. This information can be used to construct the invocation event that will trigger the application to execute the requested adaptation method.

The third part of the application's interface specification involves the identification of a set of state variables that represent the current state of the application. These state variables are identified by a name and a basic type such as integer, string, etc.

Each of these parts includes a <description> tag that provides a textual description of the interface's semantics. Moreover, the definition of an adaptation method can include the indication of related state variables that are affected by attributes passed by the invocation. This indication of a related state variable allows the identification of possible dependencies between actions and state variable. Referring back to section 3.2.5, this dependency is an indication of the possible relationship between an actuator and a monitoring entity. This related variable specification can be used by the end user to help better understand the application's behaviour and the dependencies of adaptation actions and state variables.

The interface description presented is intended to allow adaptive applications to export their adaptive interface and allow the platform to control their adaptive behaviour. However, in a typical adaptive system in addition to the adaptive applications there are system monitoring components responsible for retrieving information about the system's environment. For example, such monitoring components might include a network interface monitoring tool, a power monitoring tool or a location monitoring tool. Ex-

```
<?xml version="1.0" encoding="ISO−8859−1"?>
<application>
    <name>WebBrowser</name>
    <uniqueId>1234</uniqueId>
    <description>
        ...
    </description>
    <methodList>
        <method>
            <name>SetBandwidth</name>
            <description>
                ...
            </description>
            < attributeList >
                < attribute >
                    <name>bandLimit</name>
                    <type>Integer</type>
                    <relatedVariable>Bandwidth</relatedVariable>
                </ attribute >
            </ attributeList >
        </method>
    </methodList>
    < stateVariableList >
        <stateVariable>
            <name>Bandwidth</name>
            <type>Integer</type>
            <description>
                ...
            </description>
        </stateVariable>
    </ stateVariableList >
</application>
```

Figure 4.7: Sample XML description of an adaptive web browser

isting platforms supporting adaptive applications tend to incorporate such monitoring functionality within the platform itself [Noble98, Friday96]. In contrast, the platform presented here follows a different approach. In order to support extensibility of the system, monitoring components are treated as first class system components that collaborate with the platform to support application adaptation. The approach proposed by this design is for the same application interface mechanism to be used by both system monitoring components and applications. Although conceptually monitoring components are only sources of information while adaptive applications are receivers of adaptation triggering, the design of the adaptation platform does not make any distinctions between the two. The reason for such an approach is twofold:

- Monitoring components usually correspond to a specific device within the system. This means that they can support hardware specific adaptation. For example a component responsible for the wireless network card could set that card to sleeping mode when triggered.
- Application state variables can be useful for the controlling adaptation in other applications in the system. For example, the fact that a particular application is currently using the network may be an important factor for the adaptation policy of other applications in the system.

This design approach offers a greater level of flexibility in the design of an adaptation support platform. In particular, the platform can be extended with the incorporation of new monitoring tools as and when required. Moreover, adaptation policy rules can include information about both applications running in the system and monitoring tools and can trigger adaptation not only to adaptive applications but also to the system's devices.

Summarising, the application interface that was discussed in this section is used by the platform for the registration of adaptive applications and system monitoring tools. This interface describes the state information offered by the application and the adaptive methods that can be invoked by the platform. Using this specification the platform constructs an application manager component that handles all communication with the application.

4.4.3 Application Manager

The application manager is the component that is responsible for handling communication between the platform and the applications. The platform consists of a number of dynamically created application managers that communicate directly with individual applications (one per application instance). The functionality that the application manager should provide is to retrieve application state information and to invoke adaptation methods. The design of the application state monitoring can follow a passive approach using an event based mechanism or an active approach where the manager polls the application for updated state information. In the first case, technologies such as Jini and Web Services offer the infrastructure for event registration and notification. In the latter case, the application should provide a state query interface that the application manager

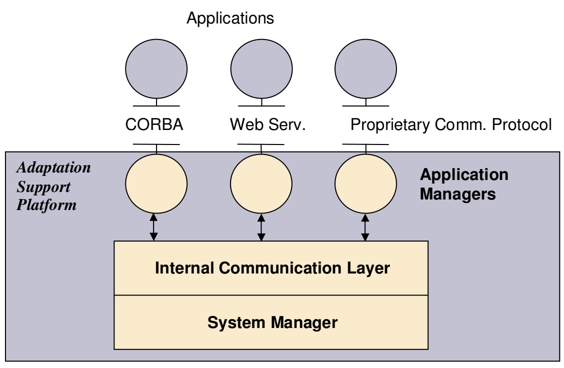

Applications
CORBA
Web Serv.
Proprietary Comm. Protocol
Adaptation Support Platform
Application Managers
Internal Communication Layer
System Manager

Figure 4.8: Application Managers for multiple communication protocols.

can use. However, since adaptation is based on the reactive response of the system on state changes, the event based approach has substantial benefits compared to a polling approach. In more detail, as the application is the first entity in the system to know that something has changed its state, it should be the application that initiates the state information update for the application manager. Therefore an event notification mechanism is much more fitting for allowing the application to notify the platform about their state changes.

Considering a design that is based on an event notification approach, the application manager acts as a cache for the application state changes reported by application events. In more detail, the application manager is a container holding the values of the last updates of the applications's variables. Thus the application manager can report to the rest of the platform what is the overall state of an application without re-querying the application.

In terms of method invocation there are no special design requirements to satisfy: any existing technologies supporting remote method invocation can be used. As a result, the design of the platform can incorporate application managers that are based on alternative communication technologies (Figure 4.8). It should be noted that in order to avoid diverting the focus of this thesis it is assumed that the system operates in a secure environment where no malicious applications are allowed to operate. Obviously in a real world scenario proper security and monitoring mechanisms should be employed to ensure that the application behaviour is acceptable. Possible approaches for implementing such an environment could include the use of certifications as guarantees for non-malicious applications.

4.5 Internal Communication Layer

The previous sections presented the mechanisms that allow adaptive applications to communicate with the adaptation support platform. For the platform this communication is handled by the application managers. Internally, application managers are required to notify the system manager about changes in applications' state. Moreover, the system manager is required to notify application managers when an adaptive method should be invoked. The internal communication layer is the component that lies between the application managers and the system manager and handles communication between these components. In the following sections we investigate existing technologies that can be used for realising the internal communication layer. Following on from this, we discuss the design issues related to the internal communication layer and in particular we present the internal communication layer in the form of an event management component.

4.5.1 Background

In the following sections an investigation of some existing technologies that support event notification are presented. In particular, the discussion includes examples of systems that follow two different communication paradigms: a subscription-notification paradigm (Jini, Elvin, CEA) and a tuple space paradigm (L^2imbo, Event Heap).

4.5.1.1 Jini

Jini is a distributed system supporting service discovery and interaction developed by Sun Microsystems [Waldo99]. The Jini system extends the Java application development environment offering tools for the implementation of network services and the applications that can discover and interact with those services. The key features supported by Jini are:

Lookup Service The lookup service allows clients in a network to discover a specific service. The lookup service maps service interfaces requested by clients into objects that implement those interfaces. In terms of implementation the lookup service is based on IP multicast. The clients multicast a service lookup request and lookup service responde with the matching services.

Java Remote Method Invocation Jini uses the Java remote method invocation (RMI) as the main mechanism for interacting with a remote service. RMI is the standard remote procedure call mechanism used by Java (Sec 4.4.1.2).

Events A service can allow clients to register interest in its events. The service then sends notifications to the registered clients when these events take place. The basic protocol uses unicast notification messages to report events. However, there are third party objects that support notification mutliplexing to reduce the network traffic.

4.5.1.2 Elvin

Elvin was first introduced as an event messaging service following the publish-subscribe notification approach [Fitzpatrick99]. One of the main drives for the development of Elvin was the complete separation between the generation and the consumption of notifications. Specifically, Elvin allows the delivery of *unaddressed* notification messages. This is achieved by using content based event delivery, that is event consumers receive event notifications based on the content of the notifications. In particular, an event subscription includes a set of named and typed data elements that the client is interested in. The notification server evaluates incoming notifications against the client subscriptions. If a subscription matches a notification the related client receives a copy of the notification message.

In terms of design, Elvin uses a server acting as a notification router between multiple connected clients. Clients can be both the sources and the sinks of event notifications. The notification router is responsible for routing notifications from event sources to the interested event sinks. Obviously the use of a central notification router limits the scalability potentials. However, Elvin addresses this issue by supporting the operation of multiple notification servers in the form of a *federation*. In more detail, multiple notification servers can work together appearing to the clients as one single notification server.

Elvin supports APIs for a variety of programming languages (e.g. C, C++, Java, Python). A number of applications have been developed using Elvin, mostly related to computer supported cooperative work (CSCW).

4.5.1.3 Cambridge Event Architecture (CEA)

The Cambridge Event Architecture [Pietzuch04, Pietzuch03] is a publish-subscribe based event management platform. One of the important characteristics of this work is the special consideration for composite events. Many existing publish-subscribe systems restrict subscriptions to single events only, and thus lack the ability to express interest in the occurrence of patterns of events. The Cambridge Event Architecture allows the registration for event patterns that will result in a notification if the specified pattern is met. Specifically, the event patterns supported by the architecture are:

Atoms: individual events similar to the traditional single-event notification platforms.

Concatenation: detects the follow up of two events with possible overlapping.

Sequence: detects the occurrence of an event after another without overlapping.

Iteration: Detects any number of occurrences of event expressions.

Timing: Detects the occurrence of events within a specified time interval.

Parallelisation: Detects two events in parallel and succeeds if both are detected. No requirement for sequence or overlapping is expressed.

In terms of implementation the event architecture utilises finite state automata for the monitoring of event expressions. The FSAs are driven by atomic events and can generate new events if the composite expression they monitor is satisfied. FSAs can be cascaded using the generated events as input to higher level FSAs for the support of complex event compositions.

In summary the design of the Cambridge Event Architecture offers a powerful mechanism for the monitoring of composite events. Considering the design of the policy language described in section 4.7.3 this platform could be considered as a potential candidate for an event management module. In particular the composite event expressions can assist the evaluation of the Event Calculus policy rules.

4.5.1.4 L^{2}imbo

L^{2}imbo is a distributed platform developed at Lancaster University [Davies98b]. L^{2}imbo does not follow the client-server paradigm, proposing an alternative communication approach especially designed to address the requirements of mobile communication. Specifically, L^{2}imbo is based on the tuple space paradigm formerly used in parallel computing (e.g. Linda [Ahuja86]) and allows applications to communicate using the tuple space API.

Tuples are data structures that consist of a collection of typed data fields. Tuples can be dynamically inserted in and removed from a tuple space. Tuple spaces are shared between applications allowing access to the tuples contained within the tuple spaces. Considering this communication approach in a distributed environment it is clear that applications do not interact directly with each other. Each application interacts with the tuple space only and inter-application communication is achieved via the tuple space. As tuple spaces contain persistent tuple objects communication does not break when connection between applications is lost for a period of time. Disconnected applications can continue to send tuples to the tuple spaces and retrieve tuples after reconnection.

In terms of implementation, L^{2}imbo is based on IP multicast where each tuple space is modeled as a multicast group. Each host in the distributed system maintains a local replica of the tuple space. Whenever a new tuple is inserted in the tuple space a multicast message updates the local replicas of the tuple space with the new tuple. If one of the hosts looses connection with the rest of the group, L^{2}imbo allows disconnected communication. In more detail, applications can insert tuples to their local replica of the tuple space and retrieve tuples from the local replica. Upon reconnection L^{2}imbo updates local replicas with the changes that took place while disconnected.

4.5.1.5 Event Heap

The Event Heap is a coordination platform developed at Stanford University and is also based on the tuple space communication model [Johanson02]. Although the Event Heap uses the same communication model as L^{2}imbo it differs in the level of specialisation of its use. L^{2}imbo is a general purpose platform for wireless communication while the Event Heap aims at supporting the specific communication requirements of interactive workspaces.

In more detail, the Event Heap is designed to support a prototype interactive workspace called the iRoom. The iRoom is a ubiquitous computing environment where people can collaborate and interact with the devices in a meeting room, such as touch screens, bottom projected tables, etc. In addition, the room has wireless LAN coverage which allows laptops or PDA's to communicate with the other machines in the room. The Event Heap is the communication platform that allows applications running in different devices in the iRoom to coordinate their activities. In order to support the specific needs of iRoom, the Event Heap extends the tuple space model with additional features as required by the project.

Self-describing Tuples: The tuples in Event Heap consist of named typed fields instead of typed fields. This means that every field in a tuple has a meaningful name and thus it is possible for a user to browse the tuple space and understand the meaning of the tuples.

Flexible Typing: The tuples in Event Heap do not require the fields to have a specific sequence or even specific number of fields. With named fields applications can retrieve the fields irrespective of their order.

Tuple Sequencing: Event Heap ensures that on a 'read' or 'in' operation receivers always get the earliest matching tuple they haven't seen yet. With sequencing, applications placing requests that match multiple tuples, will get each tuple once and in source order.

Tuple Expiration: Tuples are given a 'TimeToLive' field that specifies how long they will persist in the tuple space before they are destroyed.

In terms of implementation the Event Heap is built on top of TSpaces from IBM [Wyckoff98], a Java based tuple space system. The TSpaces system is based on a client-server architecture with the actual tuple space stored on a server machine.

4.5.2 The Design of the Event Manager

The previous sections gave an overview of examples of existing technologies that support distributed communication and coordination. Technologies like L^{2}imbo and the Event Heap were especially designed to meet the needs of mobile systems and ubiquitous computing respectively while Jini is targeted at the domain of service discovery and interaction and Elvin and CEA are general purpose event notification platforms. The main role of the adaptation platform's internal communication layer is to allow the platform to receive notifications about changes in applications' state and to invoke adaptation actions. The requirement for a mechanism to support state variable notification messages suggests the need for an event-based platform. As seen in section 4.5.1 platforms supporting event notifications can follow different communication approaches. In particular, Jini and CEA follow a directed subscription-notification approach. Elvin, though still following the subscription-notification design, is trying to break the directed dissemination of events. L^{2}imbo and the Event Heap offer undirected communication as this is a primary characteristic of tuple space based design.

Considering the design of the application manager described in section 4.4.3 we can identify an interesting feature of the tuple space mechanism that is appealing in the design of the internal communication layer. In particular, the application manager is required to act as a cache for the application state variables. One of the features of the tuple space mechanism is the persistence of tuples. Specifically, a tuple that is put in the tuple space will remain there until it is explicitly remove by an application. Mapping this functionality to the application manager this means that the application state information received by the applications can be inserted in the tuple space in the form of tuples. Therefore, when the system manager requires the value of a particular state variable, this value can be retrieved from the tuple space without requesting the application manager to fetch that information.

Following this discussion we propose as the mechanism for the internal communication layer an event dissemination mechanism (event manager) that will deliver notification messages from the application managers to the system manager and adaptation triggering requests from the system manager to the application managers. As a favourable approach the tuple space paradigm appears to offer some benefits in terms of communication and information persistence.

In terms of internal design the event manager is required to maintain an ordered delivery of notification messages to the system manager. Specifically, in the process of evaluating adaptation policies it is important that the application state changes are reported in chronological order — as it will be shown in section 4.7.3 the sequence that changes take place are of importance when adaptation decisions are taken. Considering the design of existing technologies, the CEA offers a flexible mechanism for specifying event notifications with respect to event ordering. Moreover, the mechanisms supported by CEA allow the expression of specific relationships on the occurrences of events. Considering the design of the policy language described in section 4.7.3 the CEA platform could be considered as a possible event manager that can collaborate with the system manager in the evaluation of event-driven policy rules. In particular, the composite event expressions supported by CEA can be used for sharing part of the policy evaluation process with the event management module. The Event Heap offers a tuple sequencing feature allowing tuples to be retrieved on e source-order basis. However, full support for time-ordered delivery of events is a much more complicated issue and in particular, the problem of dealing with late notifications. For example, how should a system respond to a notification that reports that the user has left the building but was delivered a day later? One approach (e.g. followed by the Event Heap) is to discard such late messages. A different approach is to accept these messages and using the timestamp that they took place evaluate them as if they were delivered on time. The actual choice of the most appropriate approach depends highly on the system that is implemented. In an active environment such as iRoom where the system interacts with users and therefore should respond fast to environmental changes, such late notifications can be discarded.

Summarising this discussion it is clear that each of the aforementioned systems have their own benefits. The requirements for the design of the event management component are modest and can be met by most of the existing systems. The choice of the most appropriate system is considered an issue related to the particular implementation.

4.6 System Manager Design

The system manager is the component that decides when adaptation actions should be invoked according to the changes reported by the application managers. As it was discussed in section 4.3, the system manager is based on a policy management system where adaptation policies are expressed through a policy language. In the following sections an overview of prevalent policy specification languages is presented. Following this background section, a discussion about the design of the system manager as a policy management component is given.

4.6.1 Background

Policy Management systems have been widely used in the areas of network and system management. In the following section two popular policy languages that were designed as general purpose policy language are presented.

4.6.1.1 Ponder

The Ponder policy language [Damianou01], developed at Imperial College London, is a declarative, object-oriented language for specifying security and management policies for distributed object systems. It defines a set policy classes with different characteristics. Specifically Ponder provides *authorisation*, *delegation*, *information filtering*, *refrain* and *obligation* policies.

The general assumptions for all policies in Ponder is that they all refer to objects with interfaces defined in terms of methods using an interface definition language. The model assumed by Ponder includes *subject* objects (users, principals or automated manager components) that have management responsibility and *target* objects (resources or service providers) that are accessed by the subjects. *Domains* provide a means of grouping objects to which policies apply and can be used to partition the objects in a large system according to geographical boundaries, object type, responsibility and authority or for the convenience of human managers.

Authorisation policies define what activities a subject object can perform on the set of target objects. A positive authorisation policy defines the actions that subjects are permitted to perform on target objects. A negative authorisation policy specifies the actions that subjects are forbidden to perform on target objects.

Information filtering policies define the type of information transformations that should be performed based on the characteristics of the subject object. For example, a location service might only permit access to detailed location information, such as whether a person is in a specific room, to users within the department.

Refrain policies define actions that subject objects must not perform on a subject even if they are actually permitted to perform the action (based on authorisation policies). The main difference between a refrain policy and a negative authorisation policy is that the former are implemented by the subjects themselves rather than a policy management component. Refrain policies are used for situations where negative authorisation policies are inappropriate because the targets are not trusted to enforce the policies.

Delegation policies are used in order to support the temporary transfer of access control rights from one object to another. These policies are required in order for this transfer of access rights to be managed by the policy management system.

Obligation policies specify the actions that must be performed by managers within the system when certain events occur and provide the ability to respond to changing conditions. Obligation policies are event-triggered and define the activities subjects must perform on objects in the target domain. Events can be simple, i.e. an internal timer event, or an external event notified by monitoring service components.

In addition to this set of policies the Ponder policy language defines policy constraints: a set of conditions that specify which policies are valid. These constraints can either be basic policy constraints that apply to specific policies or meta-policies that apply to a group of policies.

With this extensive set of policy types Ponder can support a wide range of management and security systems. Concrete examples have been presented for the use of Ponder in distributed network management, storage systems, application and service management and enterprise-wide security polices [Lupu99].

4.6.1.2 PDL

The PDL (Policy Description Language) is a domain independent declarative policy language [Lobo99, Chomicki00]. In PDL there are no assumptions about the underlying system that should be managed by the specified policies. The policy rules defined in PDL follow the event-condition-action scheme:

$$event\ \textbf{causes}\ action\ \textbf{if}\ condition$$

Intuitively a rule of this form says that if the *event* occurs at a time when the *condition* is true the *action* should be performed.

PDL consists of three basic classes of symbols: primitive event symbols, action symbols and function symbols. The primitive event symbols include system defined event symbols and user defined event symbols. Action and function symbols are predefined and are given to the user that defines the policies.

The aforementioned classes of symbols can be better described by example [Lobo99]: consider the case of an Internet provider that has a pool of modems that accept dial-up connections. In this example the internet provider wants to limit the number of simultaneous connections for a specific customer (i.e. with the phone number 5559991) to 5 connections during the night. The event that should be monitored for such a policy is time. A symbol called *CoarseTimeEvent* is associated with this event. This particular symbol is defined to have an attribute *Time* with the enumerated type "morning", "noon", "evening", "midnight". The policy rule in PDL that implements such a policy is:

CoarseTimeEvent
causes *ModemPoolAssignment*(5559991, 5)
if (*CoarseTimeEvent.Time* = "morning")

PDL also defines how simple events can be combined through logical expressions of the form $e_1 \& e_2 \& \ldots \& e_n$ or $e_1 | e_2 | \ldots | e_n$ to define composite events that should or should not take place at the same time. PDL does not specify how events or actions should be defined. These are considered to be system dependent features.

4.6.2 Policy Manager

As discussed in section 4.3 the system manager is the component that evaluates adaptation policies and triggers adaptation when required. Here a policy manager component, realising the functionality of the system manager is presented.

The policy manager is the component responsible for deciding when adaptation is required using a set of adaptation policy rules. In more detail, the decision mechanism is driven by a set of policy rules including the default policies installed by the applications (possibly modified by the user) and any new policy rules defined by the user.

Specifically, when an application is installed on the system a set of default policies are added to the policy repository. These policies are parsed by the policy manager and are used for handling adaptation for the running applications. The policy repository is always available to the current user for modifications and addition of new rules. This way the active set of policy rules can be modified by the user to meet their personal needs.

The evaluation of policy rules is driven by events delivered by the event manager. Each of these events is related to the values of the state variables of the running applications (Section 4.7.3).

In order to identify the policy language that best satisfies the requirements of this platform it is necessary to further analyse the specifics of this platform and identify the features of the policy language that should be used. Specifically, the policy language that will be used should satisfy the following requirements:

1. The policy language should be able to operate in an event-driven environment. The input given to the system manager is the set of state variables reported by applications. As described in section 4.4.3, as the values of these state variables change, the corresponding applications fire events that notify the platform about their new values.

2. The platform is required to handle the conditions under which an adaptive reaction should take place in a uniform manner, irrespective of the type of adaptation. More specifically, the decision mechanism should be a general purpose mechanism that will handle adaptation policies relating to a variety of adaptation types such as network based adaptation or physical context related adaptation.

3. The specification of policy rules should be flexible enough to allow the specification of fine grained temporal relationships between events. In most cases conflicts or instabilities in adaptive systems occur due to time dependencies between changes that take place or the time between adaptive mechanisms being invoked. The policy language should allow the fine tuning of adaptation mechanisms to allow the resolution of such types of conflicts.

Section 4.7.1 discusses the applicability of existing policy languages with respect to these requirements and section 4.7.3 proposes a new language based on the Event Calculus logic programming formalism.

4.7 Policy Language

This section provides a detailed description of the policy language used in the prototype platform. This policy language is based on the event calculus logic programming formalism.

4.7.1 Choosing a Policy Language

Section 4.6.1 presented a brief overview of existing policy languages such as Ponder and PDL. With the support of the refined set of requirements for a policy language presented in section 4.6.2 it is possible to identify the features that a policy language supporting adaptation in adaptive context-aware applications should have.

Conflicts or instabilities in adaptive systems occur due to time dependencies between changes that take place or the time between adaptive mechanisms being invoked. Furthermore, context-related conditions like “if I enter my office after leaving from John’s office” require a language that would allow the expression of temporal relations as in that particular example, the sequence of events. In adaptive systems it is quite common to have conditions like “if the system is running on low power” that clearly indicate situations (‘running on low power’) that have a certain duration rather than momentarily events. Therefore, a policy specification language that would support the expression of temporal relations between events and support the definition of entities that express duration would certainly offer more flexibility for defining adaptation policies.

The event-condition-action pattern used by most policy languages, meets the needs for an event-driven policy based system. However, the event-condition-action model is not intended for supporting complex temporal relationships between events. A simple case of a policy rule that requires the specification of temporal relationships between events is a rule that is triggered when a certain sequence of events takes place. Such a rule, for example, would be to require an application to adapt if the available battery power gets below a threshold after the user has moved outdoors (thus being away from any power source). This example would, for example, require the adaptation rule to be triggered when the event “battery low” is fired after the event “exit home”. Considering the event-condition-action model, such a rule should be triggered by event “battery low” as this is fired when the application’s action should take place. However, the condition body evaluating the rule should be able to check the occurrence of event “exit home” that took place earlier in time. This fact implies a mechanism where the policy rule can maintain state information concerning the occurrence of multiple events.

In principle, the event-condition-action model considers policy rules that are stateless. Specifically, a policy rule is evaluated considering the state of the system at the time the event triggering the rule took place. A possible approach for implementing stateful policy evaluation using the event-condition-action model would require the combination of multiple rules for defining a policy that is based on a sequence of events. Considering the example at hand, two policy rules are required that are triggered by the two specified events:

Rule 1: **event** “exit home”
causes *save state “outdoors”*

Rule 2: **event** “battery low”
causes *adaptation action*
if *state “outdoors” is valid*

This approach however, has certain limitations: It requires the implementation of necessary extensions to the policy evaluation engine that will allow storing of state information as shown in the previous example. Moreover, the implementation of a single policy would require the definition of multiple policy rules. In cases of more complex policies the number of rules involved can increase significantly making it hard for the policy author to identify the overall condition that is implemented by the combination of these rules.

Following from this discussion it is clear that using a policy language that is based on the event-condition-action model for specifying policies that involve temporal relationships between events would result to a policy set that is hard to manage. Moreover, it would be difficult for a policy author/administrator to identify what is the intended condition being implemented by a set of such inter-related policy rules. In order to avoid such shortcomings a policy language that embodies the notion of time dependencies between events would be more appropriate for the specification of adaptation policies. In more detail, the required policy language should allow the definition of multiple events that are involved in a policy’s condition, and the specification of conditions that relate the time points that these events took place. Moreover, a feature that would improve the expressiveness of such language would be to provide support for defining entities that represent duration. These entities can allow conditions of the form “when I am at home” where the phrase “I am at home” represents a situation that has duration.

In order to meet the need for such a policy language the Event Calculus Policy Language was defined. This language was derived from the specifications of the Event Calculus logic programming formalism. The following section gives a brief overview of the Event Calculus as described by Kowalski and Sergot [Kowalsky86].

4.7.2 The Event Calculus

The event calculus was introduced by Kowalski and Sergot [Kowalsky86] as a logic programming formalism for reasoning about events and change. The work presented here is based on a simplified version of the event calculus that was presented later by Kowalski [Kowalsky92].

The event calculus provides a theoretical framework where it is possible to reason about events and their effects in an event-driven system. In more detail, the event calculus is defined over a set of entities, namely *events* that take place at specific time points and *fluents* that represent the effects of the events. A fluent represents a specific situation that has a timed duration, for example a state like "battery is low". When the system under consideration gets into that specific condition the fluent is considered to be valid (it is said to *hold*). The state of fluents is defined according to events that can initiate or terminate them.

Along with the basic entities of events and fluents, the event calculus defines a set of predicates that allow the specification of propositions about when specific events take place and what the state of fluents are. The basic predicates defined in Event Calculus are:

$$
\begin{aligned}
\text{Initiates}(e,f,t) &: \text{ Fluent } f \text{ is initiated by event } e \text{ at time } t.\\
\text{Terminates}(e,f,t) &: \text{ Fluent } f \text{ is terminated by event } e \text{ at time } t.\\
\text{Happens}(e,t) &: \text{ Event } e \text{ occurs at time } t.
\end{aligned}
$$

By using these predicates we can ask about the validity of some fluents at particular time points. The simplified event calculus defines the following additional predicates:

$$
\begin{aligned}
\text{HoldsAt}(f,t) \Leftarrow\ & \exists e,t_1[\ \text{Happens}(e,t_1)\ \wedge\\
& \text{Initiates}(e,f,t_1)\ \wedge\\
& \neg\text{Clipped}(f,t_1,t)\ ]\ \wedge\ t_1 \leqslant t\\
\text{Clipped}(f,t_1,t_2) \Leftarrow\ & \exists e,t[\ \text{Happens}(e,t)\ \wedge\\
& t_1 < t \leqslant t_2\ \wedge\ \text{Terminates}(e,f,t)\ ]\\
\text{Declipped}(f,t_1,t_2) \Leftarrow\ & \exists e,t[\ \text{Happens}(e,t)\ \wedge\\
& \text{Initiates}(e,f,t)\ \wedge\ t_1 \leqslant t < t_2]
\end{aligned}
$$

The *HoldsAt* rule states that a fluent is valid at a specific time point t if an event e exists that initiated this fluent at an earlier time and this fluent has not been terminated during this time. The *Clipped* and *Declipped* rules state that a fluent has been terminated or initiated respectively by an event that took place within a time period.

Based on this small set of rules the event calculus allows us to define an event based system that changes as events take place. In addition, we can use the available rules to ask about the validity of specific conditions of the system and the times that these conditions are valid.

4.7.3 The Event Calculus Policy Language

As discussed in section 4.6.2 the policy language used for coordinated adaptation should satisfy a set of design requirements. Specificaly, it should allow the specification of event-driven policy rules, support the specification of temporal relations and it should be general enough to allow the specification of policy rules for a wide range of adaptive applications. As seen from the previous section the Event Calculus offers a basis for designing a language that can support all these requirements. In particular, the event calculus by definition embodies the eventing mechanism within its specification, it is general enough to allow specification of rules for any type of event based system and one of the fundamental elements of the event calculus is allowing a high level of flexibility in the specification of temporal relationships. It should be noted that apart from the Event Calculus there are other calculi that satisfy the aforementioned requirements (e.g. the Situation Calculus [Turner97, Kowalski94]). This thesis does not claim that the Event Calculus is the only appropriate formalism to be used as a policy language defining adaptation policies. However, the expressiveness and the comprehensiveness of the Event Calculus's predicates make it an appropriate choice for such use.

Based on the specifications of the Event Calculus we define the event calculus policy language [Efstratiou02b] in which policy rules are formulated as event-fluent-condition-action sets, in a form similar to policies specified in PDL [Lobo99].

Specifically, each policy rule is comprised of a set of system specific event definitions, a set of fluents controlled by the events, a condition body and an action body. The basic operation of a rule is to perform the actions defined in the action part if the condition part evaluates to true. The condition part consists of a logical expression involving the occurrence of events or the current state of fluents. Each fluent expresses a specific situation that the rule is interested in. The situations expressed by fluents are directly controlled by the defined events.

For example, let's consider a policy specifying that the network connection should switch to GSM when the user is outdoors. An informal way to describe this is:

Events LeftHome, LeftOffice, EnterHome, EnterOffice
Fluent Outdoors :
    initiated by events: LeftHome, LeftOffice
    terminated by events: EnterHome, EnterOffice
Condition :
    Initiated(Outdoors)
Action:
    Switch network to GSM

As described in this example the fluent outdoors is controlled by the events denoting when the user leaves or enters areas that the network connection should not be GSM. The condition part evaluates to true at the time the fluent is initiated and the action part is executed.

In more detail, the policy language allows the user to define conditions using Event Calculus predicates (as in the previous example: *Initiates*). The policy manager would evaluate the policy rules based on the notifications received by the application managers. This evaluation procedure will try to determine the time points for which the events that took place allowed the condition to be valid. In the example the policy management system would try to determine the time the fluent *Outdoors* was initiated. When the whole condition is found to be valid, the action is executed.

Formally speaking, we define an event calculus policy rule to be an expression of the form:

$event\ definition_1$

$\ldots$

$event\ definition_n$

$fluent\ definition_1$

$\ldots$

```
    $fluent\ definition_m$
    **condition** { $condition$ }
    **action** {
            $action_1$
            ...
            $action_k$
    }
```

Definition 3: *An event symbol e represents the occurrence of an event as described by the event definition. The event definition is an expression of the form:*

event *e :- l*

where e is an event symbol and l is a system specific logical expression. The logical expression is of the form $p_1 \theta p_2$ *where*

1. θ *is a Boolean operator from the set {**and**, **or**} and* p_1, p_2 *are logical expressions as well, or*
2. θ *is a relation operator from the set {=, <>, <, <=, >, =>},* p_1 *is a system specific attribute and* p_2 *is a constant of the same type. It is assumed that the user has access to the set of available system attributes that can be used for the definition of the logical expression.*

As highlighted in Definition 3, the user is assumed to have access to the set of system attributes that can be used for the definition of events. In our system these attributes are the application state variables reported by the adaptive applications running on the system during registration with the platform (described in section 4.4.2). The specification of such an attribute is represented by an expression of the form:

$$a.v$$

where a represents the application running on the system and v is one of its state variables. An event, for example, specified to mark the time the network bandwidth is between 19.2Kbps and 64Kbps is defined as:

event normBand:- (NetworkInterface.Bandwidth $>$ 19200)
and (NetworkInterface.Bandwidth $<$ 64000)

Definition 4: *The occurrence of an event is defined through the predicate happens*(e,t) $\rightarrow$ *{**true**, **false**} where e is an event symbol and t is a time point. Predicate happens evaluates to true iff t is the time point at which the logical expression l specified by the event definition transits from false to true.*

The *happens* predicate should be interpreted as "the logical expression defined for event *e* has changed its value from false to true at time point *t* causing the event to take place".

Definition 5: *A time point is a positive integer that represents a specific point in time.*

In our system, time points are considered to represent time in seconds. However, the granularity for the representation of time within a policy system is an issue that depends on the requirements of each implementation. It should be noted that within the specification of a policy rule it is required to specify time points as symbols. The actual values for these time points will be set by the policy evaluation engine. In particular, as events are delivered to the policy manager the time points specified in the policy rule will receive their values according to the semantics of the predicates they are members of (Section 5.3.5.1).

Definition 6: *A fluent symbol f represents the state of a fluent as described by the fluent definition. The fluent definition is an expression of the form:*

fluent *f {*

$init_1$

$\ldots$

$init_n$

$term_1$

$\ldots$

$term_m$

}

where f is a fluent symbol and each $init_i$ is an expressions of the form **initiates**(e) *where e is an event symbol representing the event that initiates the specific fluent; and each $term_i$ is an expressions of the form* **terminates**(e) *where e is an event symbol representing the event that terminates the specific fluent.*

A fluent is considered to hold for the time period between its initiation and termination including the initiation time and it does not hold for the time period between *termination and initiation including the termination time.*

A fluent in the policy language does not relate to any value within the platform itself. It is an abstract entity that can be defined according to the policy author's requirements. The purpose of a fluent is to represent entities that have time duration and their state changes according to the occurrence of events. In practice a fluent usually represents a real situation of the system's behaviour (like for example operating in a low bandwidth state as shown in figure 4.9).

As Definition 6 describes, the state of a fluent is controlled by the events that initiate or terminate the fluent.

Definition 7: *The condition is a logical expression of the form*

1. $p_1 \theta p_2$ *where* θ *is a Boolean operator from the set {**and**, **or**} and* p_1, p_2 *are condition expressions as well, or*
2. *a predicate proposition of initiates, terminates, holdsat, happens, clipped, declipped and their negations, or*
3. *a logical expression of the form* $t_1 \theta t_2$ *where* θ *is a relation operator from the set {=, <>, <, <=, >, =>},* t_1 *is a time variable and* t_2 *is a time variable or an expression representing a time point.*

The body of a condition specifies the logical expression that should be evaluated in order for the action part to be executed. Within the condition body a policy rule may include combinations of predicate propositions and time relationships.

Definition 8: *The initiates/terminates proposition is an expression of the form:*

$$\textbf{initiates}(e, f, t) \ / \ \textbf{terminates}(e, f, t)$$

where e is an event symbol or the literal '', f is a fluent symbol and t is a time variable. If e is an event symbol then this proposition is true iff* **initiates**(e)/**terminates**(e) *is part of fluent's f definition,* **happens**(e,t) *is true and the fluent does not hold/hold at time t. In the case where e is the literal symbol '*' then the truth value of the proposition is defined as follows: The proposition is true iff, there is an event e for which* **initiates**(e, f, t)/**terminates**(e, f, t) *is true.*

These predicates allow the specification of queries in relation to the initiation/ termination of a fluent. They should be interpreted as "the event *e* initiated/ terminated fluent f at time t". The special keyword "*" is used to denote the initiation/termination of a fluent by any event that can initiate/terminate the fluent. We have to make clear the distinction between the statements $\mathbf{initiates}(e)$ and $\mathbf{terminates}(e)$ defining a fluent from the predicates $\mathbf{initiates}(e, f, t)$ and $\mathbf{terminates}(e, f, t)$ evaluating if a fluent was initiated/terminated by an event at a given time.

Definition 9: *The* **holdsat** *proposition is an expression of the form:*

$$holdsat(f,t)$$

where f is a fluent symbol and t is a time variable. This proposition is true iff there is an event e_1 for which $\mathbf{initiates}(e, f, t_1)$ is true and $t_1 \leqslant t$ and for every event e_2 and time point t_2, $t_1 \leqslant t_2 < t$, $\mathbf{terminates}(e_2, f, t_2)$ is false.

The *holdsat* predicate allows the specification of queries in relation to the actual state of a fluent. The predicate should be interpreted as "fluent f holds at time t".

Definition 10: *The clipped/ declipped proposition is an expression of the form:*

$$\mathbf{clipped}(f,t_1,t_2)/\mathbf{declipped}(f,t_1,t_2)$$

where f is a fluent symbol and t_1, t_2 are time points and $t_1 < t_2$. This proposition is true iff there is an event e for which $\mathbf{happens}(e,t)$ is true and $t_1 < t \leqslant t_2$ and $\mathbf{terminates}(e, f, t)$ /$\mathbf{initiates}(e, f, t)$ is true.

The *clipped/declipped* predicates are used for specifying queries about the initiation or termination of a fluent within a specific time range. The predicates should be interpreted as "fluent f has been terminated/initiated sometime within $(t_1,t_2]$"

Definition 11: *An action is a statement of the form:*

$$a(p_1,\ldots,p_n)$$

where a is an action symbol with n arguments and each p_i is a parameter of the appropriate type.

An *action statement* represents a call to a specific adaptation method of an application as defined by the applications by their registration. An action call triggers an application to adapt when the condition part of the policy evaluates to true. In the definition of the Event Calculus Policy Language we assume that an action that should be

```
event lowBand :– NetworkInterface.availableBandwidth < 19200
event normBand:– NetworkInterface.availableBandwidth >= 19200
fluent inLowBand {
    initiates(lowBand)
    terminates(normBand)
}
condition {
    initiates(lowBand, inLowBand, t1) and
    not clipped(inLowBand, t1, t2) and
    t2 > t1 + 30
}
action {
    WebBrowser.LowBand()
}
```

Figure 4.9: A sample policy rule

taken when a condition is true consists of a set of adaptation method calls to the application interfaces. However, it is possible to expect more complex action procedures for certain cases. Therefore it should be noted that this definition does not consider the action body of a policy rule as a strict sequence of method calls. Specifically, certain implementations may require a more powerful way to express actions that should be invoked when a policy rule is true. Possible approaches to realise this would be to use a scripting language (e.g. JavaScript, Python) or a pre-compiled action module as the body of a policy rule.

Looking at the presented policy language definition in a more informal way, each rule of the policy language consists of two main parts: a condition and an action. The condition is a logical expression that can evaluate to true or false. When this condition evaluates to true the action body is executed.

Each condition is further divided into two parts: the declaration part and the condition body. The declaration part defines the events and fluents that participate within the body of the condition. The body itself consists of a logical expression combining Boolean operations (and, or, not) and the predicates specified by the event calculus.

The declaration of an event specifies when an event is considered to have occurred in relation to the values of specific application state variables. For example, as shown in figure 4.9 the event lowBand is considered to have taken place when the state variable availableBandwidth of the application NetworkInterface has taken a value below 19.6Kbps. The declaration of a fluent includes all the events that can initiate or terminate the particular fluent.

The condition body consists of a logical expression using the event calculus predicates. This logical expression can use predicates to evaluate the time specific events take place or whether a fluent holds or does not hold. Moreover, the condition body can include time relationships between time variables (e.g. t1 < t2). This way the policy author can specify not only the events and fluents that will enable the condition to be true, but also the time relationships between these predicates. As presented in figure 4.9 the body of that condition specifies that it will evaluate to true only if the fluent inLowBand has been initiated at a time t_1 and has remained valid until time $t_2 > t_1 + 30$. In essence, this rule specifies that it evaluates to true if the systems' available bandwidth has remained below 19.6Kbps for more than 30 seconds.

The last part of a policy rule is the list of actions. Within the list of actions the policy author has to specify a sequence of adaptation methods that should be invoked by the platform when the condition of the rule evaluates to true.

4.7.4 Examples

In order to better illustrate how the Event Calculus Policy Language can be used in practice, this section presents a list of examples where adaptation in adaptive context-aware applications is handled by Event Calculus policy rules. Please note that in the following examples the application interfaces are purely theoretical. Moreover, the interfaces are simplified in order to give more emphasis to the policy rules. Real world examples will require more complex interfaces (Chapter 6).

In the first scenario we are considering a mobile device equipped with a network interface capable of switching between a GSM connection and a Wireless LAN connection. Moreover, the device has a location monitoring module that can report the current location in terms of labels, such as “Home”, “Office”, etc.

The first adaptation rule specifies that the network interface should switch to a GSM connection when the user leaves their home. First the appropriate events and fluents are defined:

```
event LeftHome :– Location.label<>"Home"
event EnterHome :– Location.label="Home"
fluent Outdoors {
    initiates(LeftHome)
    terminates(EnterHome)
}
```

Here the fluent Outdoors is initiated when the user leaves their home and is terminated when they enter their home. For the condition definition the only check is to see when the fluent is initiated and trigger the appropriate action on the network interface:

```
condition {
    initiates(∗, Outdoors, t1)
}
action {
    NetworkInterface.UseGSM()
}
```

A similar rule is used for switching the network interface back when the fluent is terminated.

```
condition {
    terminates(∗, Outdoors, t1)
}
action {
    NetworkInterface.UseWLan()
}
```

An interesting observation here is that this rule can be easily extended by simply modifying the definition of the fluent. For example, if the same reaction is needed when the user enters and leaves their office the fluent definition can be modified as follows:

```
event LeftHome :– Location.label<>"Home"
event EnterHome :– Location.label="Home"
event LeftOffice :– Location.label<>"Office"
event EnterOffice :– Location.label="Office"
fluent Outdoors {
    initiates(LeftHome)
    initiates(LeftOffice)
    terminates(EnterHome)
    terminates(EnterOffice)
}
```

No modifications are required for the condition and action bodies.

Next we assume that the location monitoring module is capable of switching between alternative location mechanisms. In more detail, we assume that that the location can either use the GPS device, built in the mobile device, or use GSM positioning. Considering that the GPS module consumes extra power, in times when the GSM connection is active and power saving is required, it is preferred for the system to switch into GSM positioning and turn off the GPS device.

In order to define the rules that implement this adaptation policy, we first have to define the events and fluents involved. Specifically, we need to define two fluents, one expressing the situation "running with GSM connection" and the other expressing the situation "running in low power":

```
event powerLow :− Power.percent < 10
event powerNorm :− Power.percent>= 10
event gsmActive :− NetworkInterface.CurrentMode = "GSM"
event gsmInactive :− NetworkInterface.CurrentMode <> "GSM"

fluent inLowPower {
    initiates(powerLow)
    terminates(powerNorm)
}

fluent inGSM {
    initiates(gsmActive)
    terminates(gsmInactive)
}
```

The condition that will trigger the adaptive reaction will have to match the overlapping of the two situations. In particular, the condition should be triggered when one situation is initiated while the other is active:

```
condition {
    (initiates(∗, inGSM, t1) and
     holdsat(∗, inLowPower, t1)) or
    (initiates(∗, inLowPower, t1) and
     holdsat(∗, inGSM, t1))
}
action {
    Location.DisableGPSPositioning()
    Location.EnableGSMPositioning()
}
```

An additional rule to switch the location module back to GPS positioning is:

```
condition {
    terminates(∗, inGSM, t1)
}
action {
    Location.DisableGSMPositioning()
    Location.EnableGPSPositioning()
}
```

In the next scenario we assume an office environment where the system can control the room lighting. The rule that is presented is controlling the automatic switching off of the room lights. The condition that we are trying to achieve is to turn the lights off only if:

- the lights were switched on after the user entered the room (otherwise this is a room that lights usually stay on).
- the user left the room and the lights were left on.
- 15mins have passed after the user left the room and he/she hasn't returned.

First we define the appropriate events and fluents to specify two situations: 'user in the office' and 'room lights are on':

```
event LeftOffice :− Location.label<>"Office"
event EnterOffice :− Location.label="Office"
event SwitchOnLights :− RoomLights.State = "On"
event SwitchOffLights:− RoomLights.State = "Off"
fluent inOffice {
    initiates(EnterOffice)
    terminates(LeftOffice)
}
fluent RoomLightsOn {
    initiates(SwitchOnLights)
    terminates(SwitchOffLights)
}
```

Next we define the condition:

```
condition {
    initiates(∗, inOffice, t1) and
    initiates(∗, RoomLightsOn, t2) and
    t2>t1 and
    terminates(∗, inOffice, t3) and
    not clipped(inOffice, t1, t3) and
    t3 > t2 and
    not clipped(RoomLightsOn, t2, t4) and
    t4 = t3 + 900 and
    not declipped(inOffice, t3, t4)
}
```

The first line of this condition checks if the user entered the room. The second and third lines check if the room lights were turned on after the user entered the room. The next three lines check if the user left the room sometime after turning the lights on. The final three lines ensures that the lights are still on 15mins after the user left the room and that the user has not returned within these 15mins. Finally we define the action body which is a simple:

```
action {
    RoomLights.Off()
}
```

The scenarios presented here illustrate the expressiveness of the Event Calculus Policy Language but are not intended as real world examples. Chapter 6 provides specific real world examples that were implemented as part of the evaluation of the adaptation support platform.

4.8 Summary

This chapter presented the design of a platform to support coordinated adaptation for adaptive context-aware applications. Specifically, the architecture presented ensures that the requirements for decoupling of adaptation policies and mechanisms, applications externalising their state, applications externalising adaptation mechanisms and support for modification of adaptation policies are all satisfied. Moreover, existing technologies for interface specification, event management and policy management are investigated and appropriate solutions are proposed. Finally, the chapter presented the Event Calculus Policy Language that was designed in order to meet the explicit requirements of a policy language supporting adaptive context-aware applications. The next chapter describes a prototype implementation of this design.

CHAPTER V

Implementation

Contents

5.1 Overview

This chapter presents a prototype implementation of the architecture described in chapter 4. The first sections of this chapter offer a discussion about possible configurations for the implementation of the architecture. Following this discussion the prototype implementation is presented as a system application supporting coordinated adaptation for the applications running on a single host. Each of the prototype's components are presented in detail, followed by the presentation of the policy engine for the evaluation of Event Calculus policy rules.

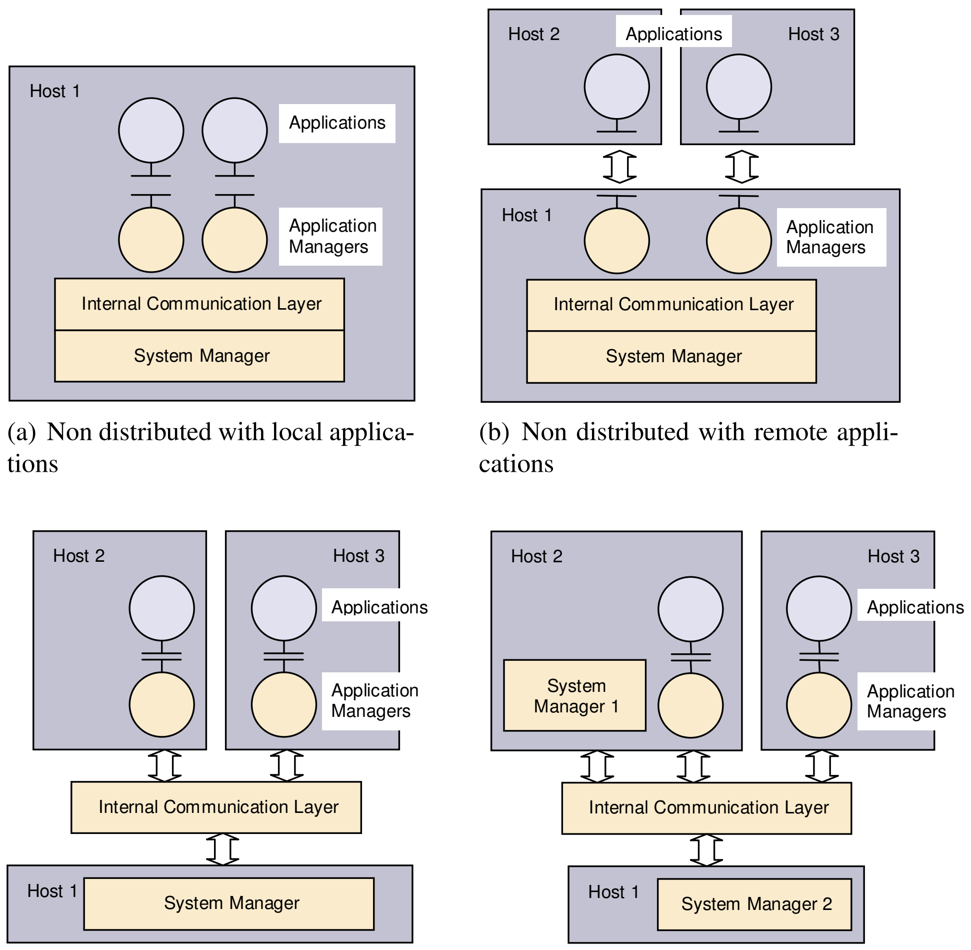
5.2 Platform Configuration

The high level architecture presented in chapter 4 does not make any statements about the location of each individual component of the platform. However, the level of distribution chosen for the system has implications for the implementation of the platform. In this section we examine the possible configurations for an implementation of a platform supporting adaptive context-aware applications.

The platform configurations presented here follow an increasing level of distribution. Specifically, the discussion begins with the configuration of a system where all applications and the platform are located in the same host and finishes with a configuration where applications and platform components are distributed across multiple hosts.

5.2.1 Non Distributed with Local Applications

The single host configuration assumes a system where all platform components and applications are located on the same host. (Figure 5.1(a)). This configuration has minimal requirements in terms of communication. Specifically, there is no need for a network protocol to be used between the applications and the platform or between the platform components. In particular, both, communication between applications and application managers and the internal communication layer, can be implemented using an inter-process communication mechanism such as shared memory. Moreover, the time-ordered delivery of events from the application managers to the system manager will not face the delay issues experienced by a distributed configuration.



(a) Non distributed with local applications

(b) Non distributed with remote applications

(c) Partially distributed platform

(d) Fully distributed platform

Figure 5.1: Platform configurations

5.2.2 Non Distributed with Remote Applications

The second configuration assumes a system where applications can be distributed across different hosts while the platform is located on a single host (Figure 5.1(b)). This configuration implies a requirement for the use of a network protocol for the communication between applications and application controllers. In more detail, this configuration requires the employment of a technology for remote process invocation and event dissemination. Technologies like Corba, Java/RMI and Web Services (Section 4.4.1) would be appropriate candidates for an implementation of this system. In terms of the internal communication, both the application managers and the system manager are located in the same host. Therefore the communication between these components can be based on inter-process communication as described in the previous paragraph. However, since there are delays between the time state variables change and the time these changes are reported to the platform, an appropriate mechanism to ensure timely delivery of these notifications should be implemented.

5.2.3 Partially Distributed Platform

The third configuration assumes a system where both applications and application controllers can be distributed across different hosts (Figure 5.1(c)). In this case the burden of network communication is pushed away from the application managers and into the internal communication layer. Specifically, application managers can be located on the same hosts as the remote applications. Thus the network traffic concerns the communication between the application manager and the system manager. This particular configuration would require the employment of a middleware platform handling the exchange of notification events from the application managers. Platforms such as L^{2}imbo, the Event Heap and CEA are examples of plausible choices (Section 4.5.1).

5.2.4 Fully Distributed Platform

In a system where applications are distributed across multiple hosts, having a single central system manager can significantly reduce the responsiveness of the system, especially where high latency or low bandwidth networks are used. Having system managers located on the same hosts as the applications can improve the responsiveness of the system. The fourth configuration is where the system manager is distributed across different hosts. (Figure 5.1(d)). In more detail, adaptation polices that are related to a particular application can be located on a system manager running on the same host. Thus, the adaptation reaction related to that particular application should have faster responses as compared to a remote system manager. This configuration requires the implementation of a policy management system that allows the distribution of policy rules across different system managers. As presented in section 4.7.3, the Event Calculus Policy Language allows the specification of policy rules that are complete and have no requirements regarding the co-existence of other policy rules. In particular, each policy rule contains the definitions of all entities required for its evaluation (events, fluents). Therefore, it is possible to distribute policy rules across hosts allowing their management by distributed policy managers.

However, such a configuration imposes additional requirements on the implementation of the internal communication layer. In particular, state variable changes must be delivered to all system managers that are handling policy rules related to the state variables in question. The use of a tuple space mechanism offers a significant benefit. As state variable notifications are shared among all components in the system, system managers can retrieve the notifications they require for the sub-set of policy rules they are handling. Therefore, there is no need for the internal communication layer to employ mechanisms to maintain the delivery of the notifications to the appropriate system managers. Moreover, the approach of a replicated tuple space (such as the one used by L^{2}imbo, section 4.5.1.4) can reduce the delay overhead for the requests of application state information. In particular system managers can retrieve information about the state of distributed application controllers using their local replica of the tuple space. Considering the fact that the implementation of the replicated tuple-space is based on multicast, this means that the total network traffic would be considerable smaller, compared to a mechanism based on point-to-point communication where each system manager would retrieve information from each distributed application controller.

5.3 Prototype

In the previous section a list of possible configurations for the implementation of the adaptation support platform were presented. Here we consider the most appropriate configuration for a prototype implementation of such a platform. In particular, the aim of this prototype implementation is to:

- Illustrate that the design presented in chapter 4 describes a feasible system that can be implemented.
- Evaluate whether the design presented in chapter 4 allows the implementation of a system that supports coordinated adaptation for adaptive context-aware applications.
- Investigate possible strengths and/or weaknesses in the design presented in chapter 4.

The requirements for the design of this platform as presented in chapter 3 are:

1. To decouple the adaptation control mechanism and the application's implementation

2. To externalise application state
3. To make applications' adaptation interfaces accessible to other components
4. To allow the modification of the adaptation control mechanism.

The design of the platform presented in chapter 4 was directly derived from this set of requirements. It is evident that the design features that are directly related to the target of this thesis are actually unrelated to the level of distribution of the platform's components. All of the features described in the design chapter can be illustrated through any of the aforementioned configurations. Specifically, irrespective of the level of distribution, the issues of policy and method decoupling, application interface externalisation and modification of the controlling mechanism follow the same design guidelines.

In order to achieve the implementation goals stated above, this chapter presents the implementation of a prototype that is based on the non-distributed configuration discussed in section 5.2.1. The implementation of this prototype allows the experimentation with and evaluation of the design characteristics of this platform without the overhead of distribution related issues that are unrelated to the main aims of this thesis and that might make the identification of the platform's effects less clear and harder to evaluate.

This prototype is intended to support adaptation on a mobile device, controlling adaptation for the applications running on the same device. The prototype was developed using Microsoft Visual C++ (approximately 8,000 lines of code for the platform and 1,300 lines of code for the application stub) and was compiled for the operating system Microsoft Windows. The prototype operates as a Windows application using a text-based interface and communicates with running applications through shared memory. In the next sections we present in detail the components that comprise the prototype implementation of the platform.

5.3.1 Component Overview

The implemented prototype reflects the design guidelines presented in chapter 4. The overall operation of the prototype is illustrated in figure 5.2. The prototype consists of a set of components that are bundled in to a single system-support application. The components that comprise the adaptation support platform are:

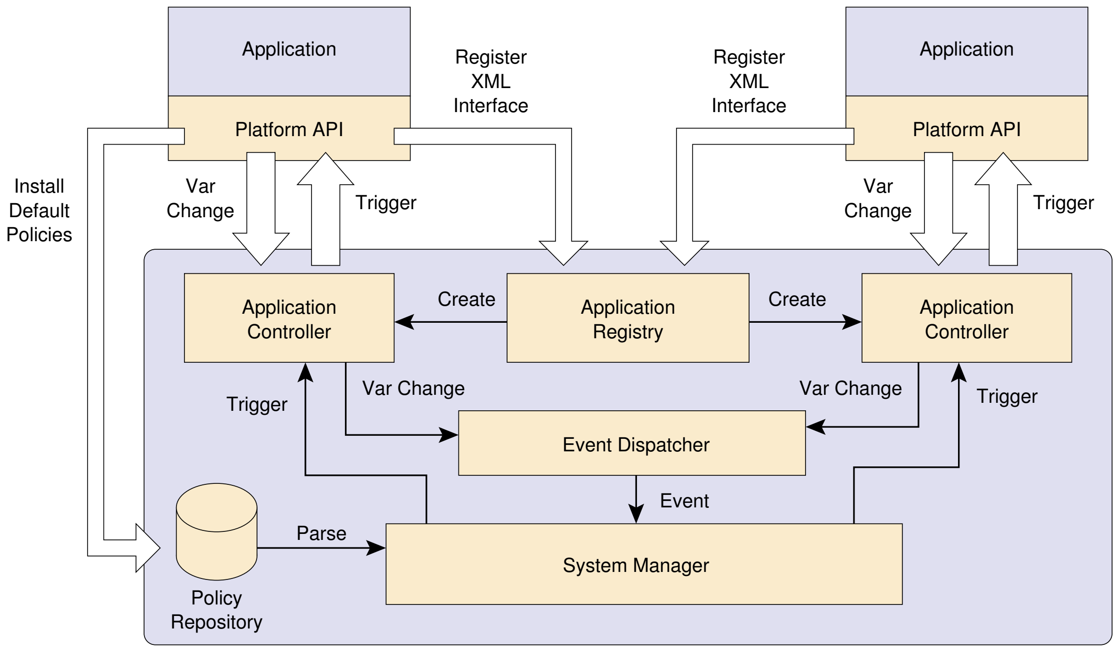


Figure 5.2: Platform component overview

Application Registry: The application registry accepts registration information from adaptive applications running in the system. Using this information it creates application controllers that handle all communication between the platform and individual applications.

Application Controllers: Each application controller handles the communication between the platform and a specific application running in the system. The application controller forwards application notifications to the event dispatcher and receives adaptation requests from the policy manager.

Event Dispatcher: The event dispatcher implements the internal communication layer for the adaptation platform. It receives state change notifications from the application controllers and delivers them to the system manager.

System Manager: The system manager component decides when adaptation reactions are required by specific applications. These decisions are based on a set of policy rules specified by the user and/or the applications. When, according to the policy rules, adaptation is required a request for adaptation is forwarded to the corresponding application controller.

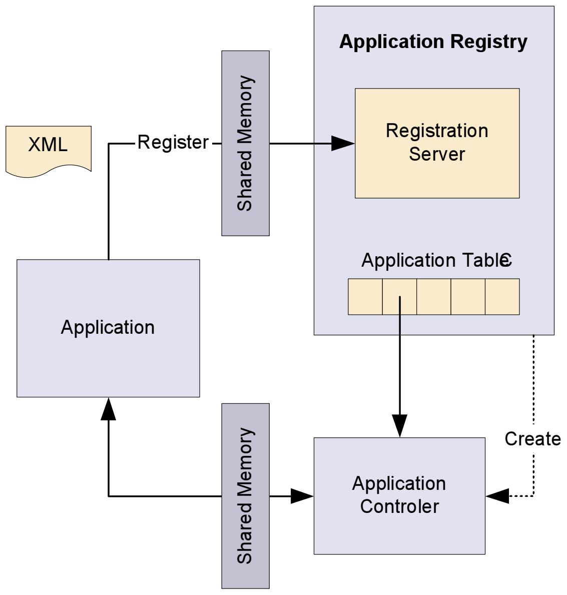


Figure 5.3: Application registration

The following sections provide a detailed description of these components.

5.3.2 Application Registry

The Application Registry is the first contact point for every application that uses the system. Applications are required to connect to the Application Registry and submit a registration document that describes their adaptive interface. The prototype implementation uses the XML-based interface specification language that was described in section 4.4.2.1. In terms of communication, there is no requirement for a network-based communication protocol to be used. Applications communicate with the application registry through shared memory (Figure 5.3). In more detail, when the application registry is initiated it creates a named shared memory space along with a set of global semaphores to control access to the shared memory. The name of this shared memory space and the global semaphores are predefined and they are known to all running applications. When a running application is initiated it opens a handle to that shared memory space and communicates with the platform by passing raw data through the memory space.

All communication through the shared memory is handled by the Registration Server sub-component. When the Registration Server receives an XML registration document from an application it parses it and uses the Document Object Model (DOM) tree to extract the registration information. In particular, the DOM tree contains details about the application, the list of adaptation methods that can be invoked as well as the state variables that the application externalises. The Application Registry uses that information to create and initialise an Application Controller component that will handle all communication with the specific application.

In addition to handling registration requests the Application Registry component holds pointers to all Application Controllers active in the system. In more detail, the Application Registry maintains a list of pointers of all the Application Controllers that it creates. Moreover it maintains two hashing indices used for locating application controllers based on the application name and the application unique id respectively. This way the Application Registry can locate an Application Controller and return a pointer to that controller when requested by other components in the platform. This functionality is primarily used by the Policy Manager. Specifically, as policy rules contain actions that are related to specific applications, when such actions need to be invoked, the Policy Manager locates the corresponding Application Controller through the Application Registry and forwards the invocation request to the controller.

5.3.3 Application Controller

The application controller is the component that handles all application communication after the application has registered with the system. In terms of communication, Application Controllers communicate with applications through shared memory. When an Application Controller is initialised it creates a shared memory space that is used only by the corresponding application. This space represents a dedicated point-to-point communication channel between the Application Controller and its corresponding application. After the creation of the shared memory space, the corresponding application receives a pointer to the newly created shared memory through the Registration Server.

After creation the Application Controller is initialised with the registration information submitted by the application. Specifically, the DOM tree (i.e. the parsed XML document submitted by the application) that was generated by the Application Registry is used by the Application Controller to create the table of adaptation methods and the table of state variables exported by the application. The table of adaptation methods is used for generating adaptation trigger messages that the Application Controller passes to the corresponding application. The table of state variables acts as a cache for application state variables. When a state variable value changes the application controller stores the new value in its local variable table. This local copy of the state variable value is returned to the rest of the platform component when it is requested.

The design of the architecture presented in chapter 4 does not give any details about when an application should send notifications to the application controller. It is considered an implementation issue to specify whether an application should report all state changes to the platform or not. Considering the different configuration options presented in section 5.2, it is clear that these decisions depend on the actual nature of the underlying system. A distributed system would require a mechanism where changes in state would only be reported if they have some significance for the system manager. In contrast, a single host implementation would not require such a mechanism. However, in order to improve the performance of this prototype and to avoid unnecessary communication through the shared memory a mechanism that minimises the number of state notifications was implemented.

In order gain a better understanding of the internals of this mechanism we need to revisit the design of the policy rules as presented in section 4.7.3. The definition of a policy event is related to the values of state variables through a logical expression:

```
event lowBand :– NetworkInterface.availableBandwidth < 19200
event highBand :– NetworkInterface.availableBandwidth >= 512000
event powerNorm :– Power.percent>= 10
```

When the policy rules are parsed all the expressions related to a state variable are combined into a set of *interesting changes*. This set consists of a list of boolean expressions relating the value of a state variable to constant values in accordance to the policy rules' definitions. This list of *interesting changes* is sent to the application after it connects to the Application Controller. The message sent to the application is an XML document of the form:

```
<?xml version="1.0" encoding="ISO–8859–1"?>
< varNotifications >
    <stateVariable>availableBandwidth</stateVariable>
    <conditionList>
        <condition>
            <value>19200</value>
            <operation>LT</operation>
        </condition>
        <condition>
            <value>512000</value>
            <operation>GE</operation>
        </condition>
    </ conditionList >
</ varNotifications >
```

Where LT corresponds to *less than* and GE corresponds to *greater or equal* (other keywords used are EQ: equal, LE: less or equal, NEQ: not equal, etc.). The message includes the name of the state variable along with a list of expressions that the platform is interested in. The application is responsible for notifying the Application Controller when any of these expressions changes truth value. Specifically the application should send a notification every time one of these expressions change from true to false and vice versa. The description of the application stub component that implements this functionality (Section 5.3.6) includes a discussion about the way this message specifying interesting values for a state variable can be used to efficiently discover when notifications are required. In certain cases the platform can request the application to report all changes related to a state variable. This approach is necessary if the definition of a policy event includes multiple state variables. In this case it is not possible to specify constant boundaries that the platform is interested in. For example, an event definition of the form:

```
event lowBand :− WebBrowser.bandwidth < VideoPlayer.bandwidth
```

is one such case where the platform cannot specify constant boundaries for the notification of changes. The applications should therefore report all changes related to these variables.

After the initialisation phase is complete the Application Controller enters a communication loop where it receives notification messages from the application and forwards adaptation requests to the application as requested by the policy manager. The messages exchanged between the Application Controller and the application are XML documents. Specifically, a message that is sent by the application to inform about a change of value for a state variable has the form:

```
<?xml version="1.0" encoding="ISO−8859−1"?>
<varChange>
    <name>bandwidthInUse</name>
    <value>1024</value>
</varChange>
```

The message includes the name of the variable and the new value. A message sent from the Application Controller to the application to invoke a particular adaptation method has the form:

```
<?xml version="1.0" encoding="ISO−8859−1"?>
<invokeMethod>
    <name>SetBandwidth</name>
    < attributeList >
        < attribute >
            <name>bandLimit</name>
            <value>1024</value>
        </ attribute >
    </ attributeList >
</invokeMethod>
```

In terms of communication with the rest of the platform an Application Controller supports the following interactions:

- An outbound call to the method PostEvent offered by the Event Dispatcher in order to notify the event dispatcher that the value of a state variable has changed. This method is called as part of the controller's communication loop.
- An inbound call to the method Trigger offered by the Application Controller. This method is called by the System Manager when application adaptation is required.

5.3.4 Event Dispatcher

The Event Dispatcher is the component that interconnects a system's application controllers and the system manager. The main aims of the Event Dispatcher are:

- Fast propagation of state variable changes to the appropriate policy rules.
- Ensure that state variable changes are processed in a *first-come-first-served* order.
- Ensure that only one state variable change is processed by the system manager at any time.

The first issue is mainly a performance requirement. Specifically, considering that in an adaptive system with a large number of applications the number of policy rules controlling the application can be very large. Therefore, the platform should be able to forward event changes to the appropriate rules with minimum performance overhead despite the possibly large number of policy rules.

The second issue is a translation of the requirement for ordered delivery of events to the system manager in the context of a non distributed implementation. In more

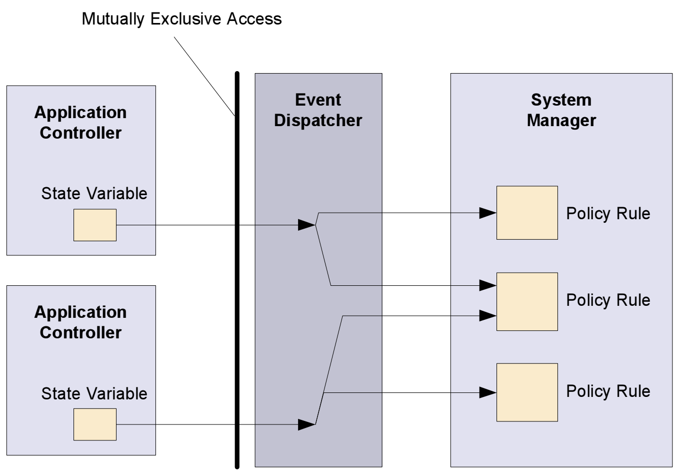


Figure 5.4: Forwarding notification events trough the Event Manager

detail, in a system where all platform components are located on a single host there are no significant communication delays in the propagation of state variable changes. Therefore, the actual state change propagation mechanism is only required to ensure a *first-come-first-serve* policy for the variable change notifications. Moreover, in order to maintain a deterministic behaviour of the system manager no state variable changes should be forwarded to the system manager before the previous change has been fully processed.

Fast propagation of state variable changes is achieved by linking each state variable to a list of policy rules that are affected by this variable. Specifically, each event definition in the Event Calculus Policy Language contains one or more state variables as part of a boolean expression (Section 4.7.3). Using this relationship between state variables and policy rules a list of pointers to policy rules is constructed for each state variable. When a state variable change is reported by the Application Controller, the corresponding list of policy rules is used to forward the variable change notification. The notification forwarding does not include any searches through the table of policy rules and therefore is not affected by its size. Indeed, this approach eliminates the negative effect that a large number of policy rules might have on the performance of the platform.

In order to ensure that only one state variable is processed by the system manager at any time, the Event Dispatcher ensures a mutually exclusive access to its PostEvent method forwarding a state variable change to the System Manager. However, as the Application Managers report state variable changes as part of their communication loop, it is not acceptable to block the Application Manager on a mutex semaphore awaiting to gain access to the Event Dispatcher. In order to avoid such behaviour, the Event Dispatcher handles each state variable notification on a separate thread, detached from the Application Manager. In more detail, when the Application Manager calls the PostEvent method in order to report changes of one of its state variables, the Event Dispatcher starts a new thread that handles the variable change and the method returns immediately. This behaviour allows the Application Managers to continue their communication with the application without any delays. The new thread is then queued in order to gain exclusive access to the method forwarding events to the system manager.

In terms of communication with the rest of the platform the Event Dispatcher supports the following interactions:

- An inbound call to the method PostEvent called as part of the application controllers' communication loop in order to report a variable change.
- An outbound call to the method VarChange offered by a Policy rule. This method call allows the System Manager to process the variable change and potentially trigger an adaptation action.

5.3.5 System Manager

The System Manager is the component responsible for taking decisions on whether adaptation is required by the applications registered with the platform. The decision taking mechanism is handled by a set of Event Calculus Policy rules. In this prototype the policy repository is implemented as a text file. The textual description of the policy rules can be modified by the user through a common text editor. Upon the platform's start up the System Manager reads the policy file and parses the policy rules. The policy rule parser has been implemented using the Parser Generator [Bee00], a port of the YACC/LEX tools for the Windows platform. The policy parser constructs a parse tree of the policy code specified in the policy file. This parse tree is then used to construct the necessary structures that are used for the evaluation of the policy rules (Section 5.3.5.1).

In terms of communication with the rest of the platform, the System Manager does not have a single entry point for the delivery of notification events coming from the Event Dispatcher. Instead the communication with the rest of the platform is delegated to the individual policy rules handled by the system manager. Specifically, each policy rule is represented by an individual CRule object. Each CRule object includes the constructs and functionality for the evaluation of a single policy rule. As described in section 5.3.4 each state variable is related to a list of policy rules that are affected by the changes of the particular variable. When an event is dispatched by the Event Dispatcher this event is delivered to the appropriate CRule object through a call to the object's CheckEvent method. Within the body of CheckEvent the specific event notification is processed. At some point after the delivery of a number of events, the condition body of a policy rule may become true. Then the CRule object processes the action body of the policy rule and triggers the adaptation methods specified. The triggering process involves calls to the method Trigger of the corresponding Application Controllers.

The following section presents a detailed description of the algorithm implemented for the evaluation of the policy rule conditions.

5.3.5.1 Evaluation of Policy Rules

A characteristic of the policy language presented in section 4.7.3 is that the condition body of a policy rule describes a pattern of events that should take place in order for the rule to be considered as true. Therefore the evaluation of a policy rule requires the occurrence of a number of events that may take place at different time points. This characteristic implies that the evaluation of policy rules should take place progressively as the values of state variables change over time and trigger related events. One reason for such an approach is that there is no easy way to discover when all necessary information is available in order to evaluate a rule in one step. Rather, the evaluation of a rule must take place incrementally as information about the state variables become available. More specifically, as events take place, some predicates within the body of a policy rule condition may evaluate to true, whilst others are false, awaiting future events that may change their value. Even while all predicates may have evaluated to true at specific time points, the time relationships between the time points the events occurred, may still not be satisfied. Therefore, the policy evaluation mechanism should progress over time as events take place and allow the execution of the action body only when the whole condition body has been satisfied.

Before describing how the policy evaluation engine works it is necessary to see in abstract terms what an evaluation mechanism for these rules should produce. As seen in section 4.7.3 one characteristic of this policy language is that each policy rule includes a set of time variables that represent certain time points related to the occurrence of events (e.g. $initiates(e, f, t)$, the time point t is related to the occurrence of event e). From the policy author's point of view these time variables do not have any specific value but represent the time that these events take place. In practice, these time variables receive specific values through the evaluation process. For example, the predicate $initiates(e, f, t)$ will allow the time variable t to receive a specific value when the event e takes place and the fluent f is initiated. Based on this observation, we define the evaluation engine as:

A mechanism that is able to find a solution for the condition body given a set of events that take place during run-time. The solution includes the values for the time variables involved in the condition body that allow the condition body to be true. This solution should be the latest solution relative to the current time.

This last statement is necessary in order to ensure that the evaluation engine will re-evaluate a condition even after it has already been found to be true before, thus allowing the continuous re-evaluation of the rules throughout the system's life-time.

In order to identify the mechanism that can find a solution for a given condition body it is necessary to look in more detail at how time variables receive their values from the predicates they are members of. One particularly important feature of the policy language is the fact that not all Event Calculus predicates allow the specification of a single value for the time variables involved. For example, the predicate $holdsat(f, t)$ does not indicate a single specific time point for t. In practice, this predicate requires: $a \leq t < b$ where a is the time fluent f was initiated and b the time f was terminated. This implies that the evaluation of a predicate does not result in the specification of single values for a time point but rather the specification of certain constraints for the value of the time points involved in that predicate. To explain this by example, the predicates $happens$, $initates$, $terminates$ set constraints for their related time variables in the form: $t = a$ (a is the time point the related event took place), while the predicates $holdsat$, $clipped$, $declipped$ set constraints of the form $t > a$ or $t < a$ (a is the time point the related fluent is initiated or terminated).

Another observation that is derived from the specification of the policy language is the fact that time variables can be attributes to more than one predicate. For example, a condition of the form:

```
initiates(event1,fluent1,t) and
holdsat(fluent2, t)
```

implies that the time variable t should satisfy the constraints imposed by both predicates:

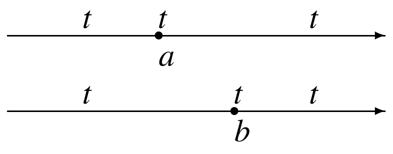
$$t = t_e \quad \wedge \quad t \geq t_{fi} \quad \wedge \quad t < t_{ft}$$

where t_e is the time the event event1 took place, t_{fi} is the time point the fluent fluent2 was initiated and t_{ft} is the time point the fluent was terminated. However, this fact implies that it is possible to have conflicts within the constraints imposed by a time variable. In the previous example, if the event event1 takes place before the initiation of fluent1 then the constraints are:

$$t = t_e \;\; \wedge \;\; t \geq t_{fi} \;\; \wedge \;\; t_e < t_{fi} \;\; \Longrightarrow \;\; t < t_{fi} \;\; \wedge \;\; t \geq t_{fi} \quad (conflict)$$

In such cases of conflicts in the constraints it is necessary for the evaluation engine to resolve and discard the constraints that are irrelevant to the evaluation of the full condition body. Following the example at hand, the event involved in the initiates predicate that took place before the initiation of fluent2 cannot be part of the condition's solution. Specifically, as this event took place in the past and this occurrence did not satisfy the other constraints imposed later by the holdsat predicate, then this event is not part of the condition's solution and can be safely discarded.

In order to define the mechanism for resolving such conflicting constraints we should consider the general case where a time variable t is related to two time points a, b. Assuming that $a \leq b$, all the possible relationships between t and the two time points (e.g. $t < a$, $t = a$, $t > a$ etc.) are illustrated in the following diagram:



where the two arrows represent time and the relative position of the variables represent relationships of the form $t < a$, $t = a$, etc. Through this illustration it is obvious that the following conflicting situations cannot be satisfied:

$$t \leq a \wedge t \geq b \wedge a < b$$

$$t < a \wedge t \geq b \wedge a \leq b$$

$$t \leq a \wedge t < b \wedge a \leq b$$

The way to resolve these conflicts is to discard the relationship related to time point a (where $a \leq b$). The rationale behind this approach is the following: The relationship between t and a can either be part of a solution for the condition defined in a policy rule or not. If the relationship is not part of a solution, discarding the related constraint is valid. If the relationship is part of a solution, considering that it conflicts with time point b that is later in time, then this relationship was part of a previously evaluated solution that has already been handled. Therefore, in both cases discarding the constraint related to a is acceptable for the evaluation of the most recent solution. This mechanism implies that the evaluation engine is a progressive procedure that evaluates conditions as events take place. This evaluation includes the discarding of constraints that have been part of a previous solution or that do not match with the latest events.

Based on these observations the implementation of the evaluation engine consists of a mechanism that receives event notifications in terms of a tuple (e,t) — where e is the event symbol and t is the time the event took place — and discovers the set of constraints for the time variables involved in a condition that allows the condition body to be true. The processing of the events in the evaluation engine includes the specification of constraints according to the semantics of the related predicates. The constraints are checked for possible conflicts and based on the approach described above, the appropriate constraints are discarded.

In more detail, the actual implementation of the evaluation engine uses finite state automata to represent the state of each predicate. With the exception of the *happens* predicate, all other predicates are related to a single fluent. Therefore the FSAs that represent these predicates consist of two states corresponding to the *holding* and *not holding* states of the fluent. The FSAs transit from one state to the other when the events that initiate or terminate the related fluent take place. The transition of the FSAs between states triggers the specification of constraints on their related time variables. The specific constraints imposed by the FSAs depend on the particular predicate they represent. For example the $holdsat(f,t)$ predicate will impose the constraint $t > a$ when the fluent f is initiated at time point a. The same predicate will later impose the constraint $t < b$ when the fluent f is terminated at time point b. The predicate $clipped(f,t_1,t_2)$ will impose the constraints $t_1 < b$ and $t_2 > b$ when the predicate is terminated. For the evaluation of the policy rules FSAs are defined for all the Event Calculus predicates as well as their negations (i.e. *not happens*, *not initated*, etc.). The defined FSAs can be seen in Table 5.1 along with the constraints imposed when there is

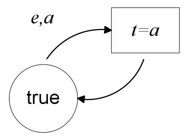
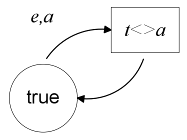
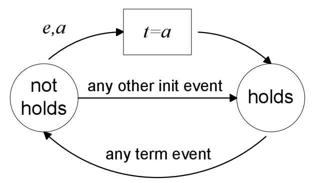
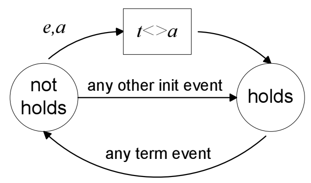
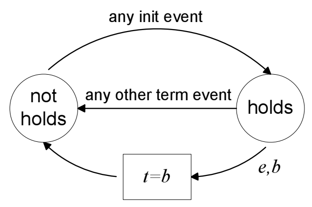
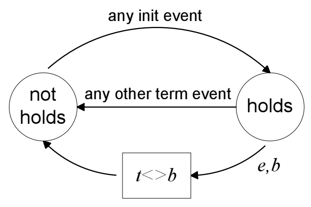
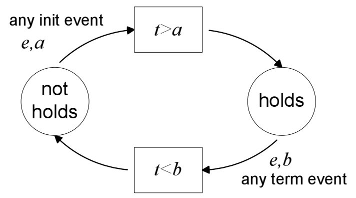
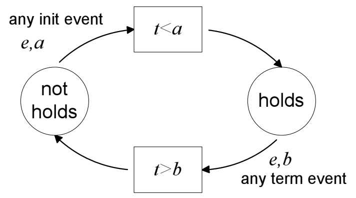
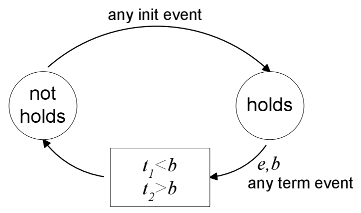
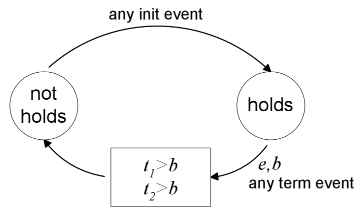
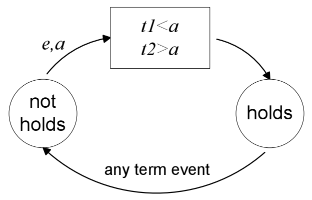
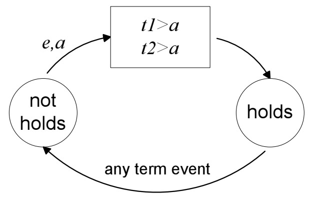
Predicate	Affirmation FSA	Negation FSA
$happens(e,t)$		
$initiates(e,f,t)$	 e,a t=a not holds any other init event holds any term event	 e,a t<>a not holds any other init event holds any term event
$terminates(e,f,t)$	 any init event not holds any other term event holds t=b e,b	 any init event not holds any other term event holds t<>b e,b
$holdsat(f,t)$	 any init event e,a t>a not holds holds t<b e,b any term event	 any init event e,a t<a not holds holds t>b e,b any term event
$clipped(f,t_1,t_2)$	 any init event not holds holds t1<b t2>b e,b any term event	 any init event not holds holds t1>b t2>b e,b any term event
$declipped(f,t_1,t_2)$		

Table 5.1: Finite State automata representing Event Calculus predicates. The first column specifies the FSAs for the predicates in their affirmative form and the second column in their negated form (e.g. not happens, not initiates).

a transition between the states.

When the policy evaluation engine is initiated for a specific policy rule, a set of FSAs is created that correspond to the predicates specified in the condition body of the rule. In addition to the FSAs, a table of all the time variables specified in the condition is constructed. For each time variable a list of constraints is created that will receive a value from the predicates. Each of these constraints holds the type of the constraint (e.g. $t > value$) and is marked as invalid until it is given a value by the related predicate.

As discussed earlier each time variable may participate in more than one predicate. Therefore the list of constraints may contain constraints that are related to multiple predicates.

Once the operation of the system is started and events start arriving the evaluation engine passes these events to the corresponding FSAs, allowing these to transit from one state to the next. During these transitions, the FSAs assign values to the specified time variable constraints. Whenever a new value is assigned to a time variable constraint, the engine checks this new value against all previously validated constraints. During this check some of the constraints that do not validate against each other are discarded based on the approach described earlier.

With this procedure the evaluation engine processes all the incoming events and sets the required constraints until a solution for the condition body is found. This solution consists of a set of constraints for all the time variables defined in the condition, where these constraints allow all the predicates to be satisfied. If such a situation can be found, the condition is considered to be true. In essence this means that a combination of events has taken place at the specific time points that match the situation described in the condition body.

5.3.5.2 Policy Evaluation Example

Consider the policy rule presented in section 4.7.3:

```
event lowBand :— NetworkInterface.availableBandwidth < 19200
event normBand:— NetworkInterface.availableBandwidth >= 19200
fluent inLowBand {
    initiates(lowBand)
    terminates(normBand)
}
condition {
    initiates(lowBand, inLowBand, t1) and
    not clipped(t1, inLowBand, t2) and
    t2 = t1 + 30
}
action {
    WebBrowser.LowBand()
}
```

In this policy rule there are two time variables: t1 and t2. When the rule is initiated no constraints are expressed for the time variables and all the predicates are false.

Event	Fluent	Constraints	Resolved
lowBand, 1	inLowBand : holds	t1=1	t1=1
normBand, 15	inLowBand : not holds	t1=1 t1>15 t2>15	t1>15 t2>15
lowBand, 20	inLowBand : holds	t1=20 t1>15 t2>15	t1=20 t1>15 t2>15

timerEvent, 50	inLowBand : holds	t1=20
		t1>15
		t2=50
		t2>15
			t1=20
			t1>15
			t2=50
			t2>15

Table 5.2: Evaluation walk through for a sample policy rule

Assume that the event lowBand is fired at time 1. The fluent inLowBand moves into *holding* state (Figure 5.1). Based on the definition of the FSA for the initiates predicate, the time variable t1 gets a constraint: t1=1 (Table 5.2). At the same time a timer event is scheduled to be fired at time 1+30 = 31. This timer event will be used to evaluate the expression t2 = t1 + 30. The transition of the not clipped FSA does not impose any constraints on the time variables.

Assume that the event normBand is fired at time 15. The predicate initiates' FSA transits to *not holds* but it does not impose any constraints. The transition of the not clipped FSA imposes constraints for both variables: t1>15 and t2>15. The constraint for t1 conflicts with the previously set constraint: t1=1 and t1>15. In order for this conflict to be resolved the constraint t1=1 should be discarded. This means that from that point on in order for the predicate to be true, the time variable t1 should have a value larger than 15. This change also causes the predicate initiates not to be true any more, since the value for t1 does not satisfy the predicate. Moreover, the timer event scheduled for time 31 is canceled as it was initiated when the variable t1 took the value 1.

Next assume another lowBand event at time 20. This event imposes the constraint t1=20. This constraint does not conflict with the previous constraint t1>15 and therefore no additional actions are necessary. A timer event is also scheduled for 20+30=50. Assume that no other application events are fired for the next 30 seconds. This means that the next event would be the timer event fired at time 50. This timer event would impose the constraint t2=50 which complies with the previously set constraint t2>15. At that point all time variables have received a valid set of constraints that satisfy all predicates in the condition body. As a result the policy manager considers the condition to be true and executes the action body of the rule.

5.3.6 Application Stub

In order to assist the creation of adaptive context-aware applications that will collaborate with the adaptation support platform, a platform stub library was developed. This library supports the application side operations that are related to the platform. In particular, the library supports registration and communication with the platform, notifications for state variable changes and callback functionality for the invocation of adaptation methods.

The stub was developed as a C++ library that can be statically linked with an application. In order to allow developers to implement their own application stub (if, for example, support for other programming languages is required) a detailed description of the stub's functionality is presented. In more detail, the operation of the stub can be divided into two sections, the registration section and the communication loop (Fig 5.5). During the registration phase the stub connects to the shared memory space handled by the Registration Server and sends out the XML description of the application's interface. Next it waits for a response in the form of the name of the shared memory space that was created by the Application Controller. The graph of *interesting values* is received after the stub has connected to the application controller. This step completes the registration phase. The communication loop consists of a thread handling adaptation requests sent by the application controller and another thread that forwards variable

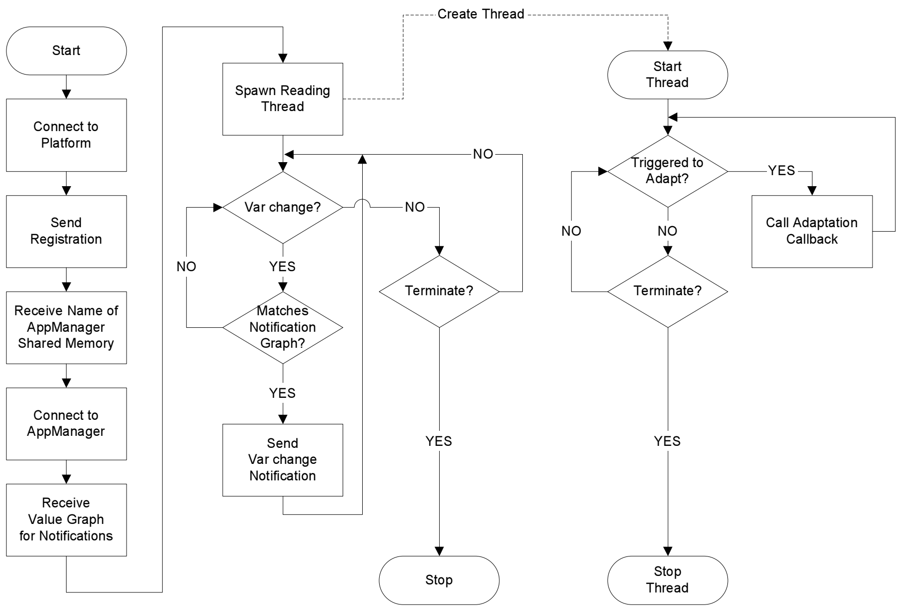


Figure 5.5: Application stub

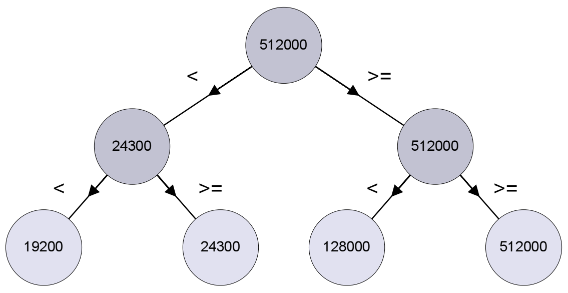


Figure 5.6: Value Tree used for matching variable values against the interesting values reported by the platform

change notifications to the platform.

Handling variable change notifications involves the use of a filtering mechanism that checks whether the value change reported by the application should be forwarded to the platform. As discussed in section 5.3.3 the Application Controller sends out an XML description of the values that are considered interesting for the adaptation platform. This description is parsed by the Application Stub in order to construct a tree of value change borders that when passed a notification event should be sent to the platform. For example, a description of the values the platform is interested in for the variable availableBandwidth may contain the following conditions:

```
availableBandwidth <19200
availableBandwidth = 24300
availableBandwidth > 128000
availableBandwidth >= 512000
```

Using these conditions a binary tree is constructed that allows fast searching of values (Figure 5.6). In particular, for every variable change reported by the application the stub stores its position in the value tree. If the next variable change has a value that results at a different position in the value tree then this change should be reported to the platform.

5.3.6.1 Application API

For the purpose of simplifying the development of adaptive applications that can collaborate with the platform, the developers can use the object-oriented application programming interface (Figure 5.7). The API consists of a collection of classes that take care of all communication with the platform. The main classes defined by the API are:

```
//---- Client side support for adaptation ---
class AD_Client {
public:
    AD_Client();
    virtual ~AD_Client();
    void VarChange(LPCTSTR varName, void* varValue);
    void Stop();
    void Start(AD_ApplicationInfo *p_Application);
protected:
    void CommunicationLoop();
};

//--- Application information container ---
class AD_ApplicationInfo {
public:
    AD_ApplicationInfo(LPCTSTR pAppName, LPCTSTR pAppId);
    virtual ~AD_ApplicationInfo();
    bool AddNewMethod(AD_MethodInfo *newMethod);
    bool AddNewVariable( AD_VariableInfo *newVar);
    void CreateXML(char* out);
    void SetDescription(const char* sDescription);
    AD_VariableInfo * GetStateVariable( const char* pVarName);

    CString m_sAppName;
    CString m_sAppId;
protected:
    SVarList *m_lstStateVariables;
    SMethodList *m_lstMethods;
};

//--- Method information container ---
class AD_MethodInfo {
public:
    AD_MethodInfo(const char* pName, AD_Attribute* attribList,
        int (__cdecl *p_TriggerCallback)(void*));
    ~AD_MethodInfo(void);
    void CreateXML(char* out);
    void SetDescription(const char* sDescription);

    CString m_sName;
    AD_Attribute *lstAattributes;
    AD_ApplicationInfo *m_pParentApplication;
};

//--- State variable information container ---
class AD_VariableInfo {
public:
    AD_VariableInfo(const char* vName, AD_VarType vType);
    void SetDescription(const char* sDescription);
};

//--- Attribute information container ---
class AD_Attribute {
public:
    AD_Attribute(const char* vName, AD_VarType,
                        AD_VariableInfo *relVar = NULL);
};
```

Figure 5.7: Application stub API

```
// Create the ApplicationInfo object
AD_ApplicationInfo *application =
            new AD_ApplicationInfo("Test","10001");

// Add a state variable
AD_VariableInfo *var = new  AD_VariableInfo("bandwidth", vInteger);
application->AddNewVariable(var);

// Add an adaptive method
var = application->GetStateVariable("bandwidth");
AD_Attribute * attrib = new AD_Attribute("newBand", vInteger, var);
AD_MethodInfo * method =
            new AD_MethodInfo("SetBandwidth", attrib, CallbackFunc);
application->AddNewMethod(method);

// Create the adaptClient object
AD_Client adaptClient;
adaptClient.Start(application);

// Notify about the value change of a state variable
int nBandValue = 52030;
adaptClient.VarChange("bandwidth", (void*)&nBandValue);
```

Figure 5.8: Sample code: using the application stub

class AD_Client The main component responsible for all interactions with the platform. It constructs the XML definition of the application's interface and passes it to the platform during registration. This action is performed through the `Start(appInfo)` method call. The `VarChange(varName, varValue)` method call is used to notify the `AD_Client` object that a state variable value has changed. This notification may be passed to the platform, if required.

class AD_ApplicationInfo Holds all information about the application. It is a container for all definitions of state variables and methods as defined by the method calls `AddNewMethod(methodInf)` and `AddNewVariable(variableInfo)` respectively.

class AD_MethodInfo Holds information about the definition of an application's method. Apart from the details required for the application registration (i.e. method name, list of attributes, etc.) the specification of an AD_MethodInfo object includes the definition of a callback function that will be invoked whenever that method is triggered by the platform.

class AD_VariableInfo Holds information about the definition of an application's state variable.

In order to illustrate how this API can be used by an application, figure 5.8 shows a simple example of its use. Specifically, the application uses the AD_ApplicationInfo to construct a container that holds the application specification. This container is loaded with AD_VariableInf objects representing the application's state variables and AD_MethodInfo objects representing the application's adaptation methods. The Start method call performs the application registration and initiates the communication loop. Whenever an adaptation request is sent from the Application Controller, a callback function is called containing the name of the method and a list of AD_Attribute objects with the attribute values.

5.4 Platform Operation

Based on the descriptions of the platform components presented, this section offers a description of the actions taken by each component during operation.

5.4.1 Platform Initialisation

During initialisation, the policy manager loads the set of policy rules from the policy repository. The actual policy repository is a file containing the active policy rules. If new policy rules have been installed, the policy manager merges the existing policy rules with the newly installed set. The set of policy rules that are loaded include both the policy rules that are inserted by the applications during their installation and the policy rules added by the user.

Using the event specifications defined in the policy rules, the policy manager constructs a relationship table linking the application state variables with the policy rules that are affected by their change. This relationship table is passed to the event manager. The event manager uses this table for directing events regarding state variable changes to the specific rules affected by these changes.

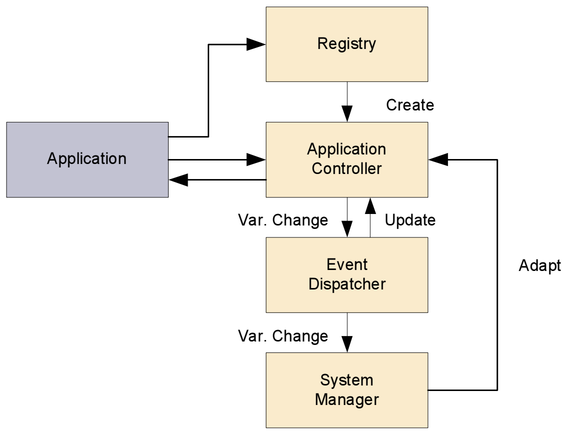


Figure 5.9: Operation of the coordinated adaptation platform

5.4.2 Application Initialisation

When an application is initiated it connects to the adaptation platform using a well known communication point (a named shared memory buffer). The application can then send out the XML document with the description of the adaptation interface it implements. The Application Registry component parses the XML document and creates an Application Controller component dedicated to that particular application.

5.4.3 State Change Notification

When a value of a state variable changes, the application is responsible for notifying the platform about the new value. This notification is sent to the Application Controller. The Application Controller requests access to the Event Manager. If another Application Controller is reporting a notification, it is put into a FIFO queue. When the Application Controller is granted access to the Event Manager, the Event Manager signals the Application Controller to update the value of the application's state variable with the new value. This late update of the Application Controller's values is necessary in order to make sure that the values of state variables are updated in the right order and a previous evaluation of a policy rule will not use newer state variable values. Next the Event Manager forwards the notification message to the Policy Manager. The Policy Manager uses the notification message to partially evaluate the policy rules affected by the value change.

5.4.4 Adaptation

As the variable change notifications reach the Policy Manager the evaluation of some rules will result into a sequence of adaptation actions that need to be performed. The policy manager sends the action messages specified in the body of the policy rules to the corresponding application controllers. The application controllers marshal the adaptation triggering message and send the message to the application.

5.5 Summary

This chapter presented the implementation of a prototype platform supporting adaptive context-aware applications. The particular prototype is based on a “single host” configuration where both the platform and the adaptive applications are located on the same host. The discussion focused on particular aspects of the implementation that have significant impact on the efficiency of the platform. In particular, the mechanism for filtering notification messages, the implementation of the event manager and the implementation of the policy evaluation mechanism were aimed at creating an efficient platform that imposes minimal overhead to the operation of adaptive applications running in the system. The efficiency of this prototype is investigated in chapter 6.

CHAPTER VI

Evaluation

Contents

6.1 Overview

The previous two chapters presented the design and implementation of a prototype platform that was developed following the requirements presented in chapter 3. This chapter presents an evaluation of the developed platform. The platform evaluation has two parts: a qualitative evaluation that investigates the characteristics of the platform (and in particular the platform's support in relation to the criteria established in chapter 2), and a quantitative evaluation that measures the performance and scalability characteristics of this platform. The aim of the performance evaluation is to be able to draw general conclusions beyond the scope of this prototype implementation, about the design of a platform that supports coordinated adaptation.

6.2 Qualitative Evaluation

In this section we evaluate the features offered by the platform that, as discussed in chapter 3, are considered important for the support of adaptive context-aware applications. In particular, in this section we investigate the behaviour of the platform in terms of support for coordination, conflict resolution, extensibility and user involvement.

For this qualitative evaluation a set of test applications were developed to allow experimentation with the platform. The applications developed were a video streaming player, a web browser and an e-mail client. The actual selection of the particular applications was based firstly on popularity (applications that are commonly used on a computer system) and secondly on their diversity in terms of functionality. In particular, the video player uses a data streaming communication protocol and has high resource requirements in terms of CPU and power; the web browser's traffic follows a pattern where bursts of data downloads are followed by periods of inactivity and, as an interactive application, requires fast responses to user requests; the e-mail client is, most of the time, working in the background, its network demands are periodic and may occasionally require the download of large amounts of data (i.e. e-mails with attachments). These diverse characteristics allow the evaluation of the platform's ability to support different applications and to allow effective collaboration between diverse applications.

In addition to these applications a set of system monitoring components were also implemented. Specifically, a network interface module that monitors and controls the

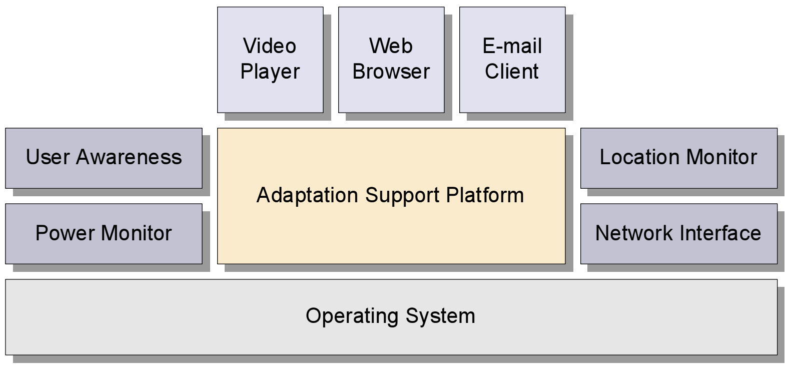


Figure 6.1: System setup for the evaluation of the adaptation platform

network interface, a power monitoring module that reports the state of the battery power and a simulated location module that reports the current location of the device. The following sections present these applications and system components in more detail.

6.2.1 Applications and Monitoring Tools

6.2.1.1 Video Player

The adaptive video player is based on the Real Player video player [Real03]. The Real Player is an RTP/RTCP compliant video player that supports adaptation based on the quality of the network connection. More specifically, the Real Player can switch between different versions of streaming video and/or audio that correspond to varying levels of quality, in response to the changing quality of the network connection. In order to make the Real Player compliant with the adaptation support platform, we developed an RTCP proxy that can filter the RTCP messages sent to and from the remote video server. Moreover, the RTCP proxy can inject RTCP commands to the client-server channel in order to cause adaptation on demand. This configuration allows the RTCP proxy to instruct the video server to start or stop the video streaming and switch to alternative

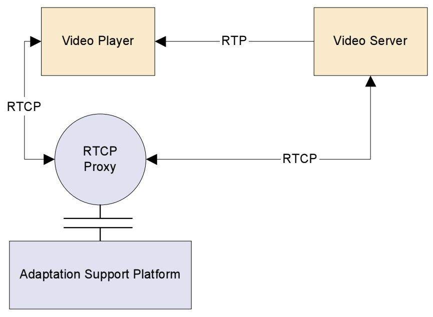


Figure 6.2: Adaptive video player through an RTCP proxy

Video Player State Variables	
string state	Reports the current state of the video player. Possible values: ‘idle’, ‘playing’,‘paused’, ‘congested’, ‘buffering’
int streamBandwidth	Bitrate of the current stream in bps.
int videoBandwidth **int audioBandwidth**	Bitrate of the current video/audio stream in bps.
int highVideoBandwidth **int highAudioBandwidth**	The next higher video/audio bitrate from the alternative streams that the player can switch to.
int lowVideoBandwidth **int lowAudioBandwidth**	The next lower video/audio bitrate from the alternative streams that the player can switch to.

Video Player Methods	

Start()
Stop()
	Starting, stopping the video and audio play-out.

StartVideo()
StartAudio()
StopVideo()
StopAudio()
	Starting, stopping the video or the audio play-out.

VideoBandUp()
VideoBandDown()
	Switch to a video stream with higher/lower bitrate.

AudioBandUp()
AudioBandDown()
	Switch to an audio stream with higher/lower bitrate.

Table 6.1: Video Player: Adaptation Interface

video or audio streams.

Collaboration of the video server with the adaptation platform is achieved through the RTCP proxy. In particular, the RTCP proxy, representing the video player application, registers with the platform. The interface exposed by the proxy includes state variables reporting the current state of the player, information about the current video stream and information about alternative video streams available. The RTCP proxy retrieves that information from the RTCP messages exchanged between the player and the server. In particular, the initialisation messages exchanged when the player connects to the server contains details about the alternative video and audio streams and the consecutive messages include details about the current state of the video stream.

The application interface that is used by the RTCP proxy to connect to the adaptation platform also includes a set of methods that allow the platform to start or stop the video streaming and request the player to switch into an alternative video or audio stream. Details of the video player's interface are presented in Table 6.1.

The default policies that were implemented for the adaptive video player allow the switching between higher/lower bandwidth streams in response to network QoS changes. In particular, the default policies use the NetworkInterface monitoring module to determine the available bandwidth on the local network connection and, if enough, switch into a higher video stream. Moreover, the lack of available bandwidth would require the video player to switch to a lower quality video stream. A sample rule that handles the switching of the video stream is:

```
event videoBandAvail :− NetworkInterface.availableBandwidth >
    VideoPlayer.highVideoBandwidth − VideoPlayer.streamBandwidth
event noVideoBandAvail :− NetworkInterface.availableBandwidth =<
    VideoPlayer.highVideoBandwidth − VideoPlayer.streamBandwidth
fluent availVideoBand {
    initiates(videoBandAvail)
    terminates(noVideoBandAvail)
}
condition{
    initiates(videoBandAvail, availVideoBand, t1) and
    not clipped(availVideoBand, t1, t2) and
    t2 = t1 + 10
}
action {
    VideoPlayer.VideoBandUp()
}
```

This rule instructs the video player to switch to a higher bitrate/higher quality stream when the available network bandwidth is enough to handle the new video stream. The video player will switch to the new video stream if that condition stays valid for more than 10 seconds. Similar rules are used for reducing the stream bit rate of either video or audio stream.

6.2.1.2 Web Browser

The adaptive web browser was based on a standard non adaptive web browser (e.g. Internet Explorer) with the support of a web proxy pair implementing the necessary adaptive behaviour (Figure 6.3). Specifically, two proxies are responsible for handling

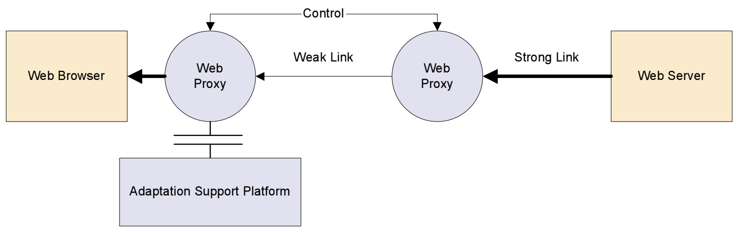


Figure 6.3: Adaptive web browser based on a pair of proxies

the web traffic over a weak link (e.g. a wireless link). The first web proxy is located on the same host as the web browser and the second proxy on a remote host that is part of the fixed network and has a high speed connection with the web server. The two proxies communicate in order to collaborate and perform the necessary adaptive functionality. In particular, adaptive behaviour supported by the proxy pair include: specification of the bitrate at which data should be delivered to the web browser, compression of text/html data, image conversion to black-and-white and removal of images from a web page.

In terms of communication with the adaptation support platform, the local proxy represents the web browser as an adaptive web browser. This means that all adaptation requests are from the platform to the local proxy and the local proxy collaborates with the remote proxy to implement them. In terms of state variables, the web proxy exports state information such as the URL being fetched, the bitrate that the current stream is being downloaded at, and the use of a specific image filtering technique, etc. One interesting aspect of the behaviour of the web proxy is that when the web browser requests the retrieval of a specific URL, the proxy submits that information to the platform as a *requested URL* and awaits for an invocation of the *Fetch* method in order to carry out the request. This feature allows the platform to perform necessary adaptation actions before a specific URL is retrieved. Details of the web browser's interface are presented in Table 6.2.

The default policies of the adaptive web browser handle common URL requests and network related adaptation. For the adaptive web browser the main policy that handles all web requests is the following:

```
event :- WebBrowser.requestURL <> ""
condition {
    happens(requestURL)
}
action{
    WebBrowser.Fetch(WebBrowser.requestURL)
}
```

This rule lets the web browser fetch the URL that was requested by the user. This is an unconditional fetching rule that can be modified by the user to perform any custom adaptations required when a specific URL is requested.

One of the policy rules that handles the behaviour of the web browser under low

Web Browser State Variables	
string state	Reports the current state of the web browser. Possible values: (i.e. ‘idle’, ‘downloading’)
string requestURL	The URL that was requested.
int bandwidth	The bitrate that the data stream is delivered.
int setBandwidth	The set bitrate that the data stream should delivered.
bool compressed	Boolean variable to specify if the text compression is applied.
bool imageBW	Boolean variable reporting if the images are converted to black and white.
bool imageNo	Boolean variable reporting if the images are filtered out of the delivered data.

Web Browser Methods	
Fetch(*url*)	Fetches the specified URL.
SetBandwidth(*band*)	Sets the bitrate that the data should be delivered.
SetImageBW(*boolVar*)	Toggles the black-and-white image filtering.
SetImageNo(*boolVar*)	Toggles the no-image filtering.
SetProxy(*ip*, *port*)	Sets the address of the corresponding proxy pair.

Table 6.2: Web Browser: Adaptation Interface

bandwidth conditions is:

```
event lowBand :– NetworkInterface.availableBandwidth > 19200
event highBand :– NetworkInterface.avilableBandwidth <= 19200
fluent inLowBand {
    initiates(lowBand)
    terminates(highBand)
}
condition {
    initiates(lowBand, inLowBand, t1) and
    not clipped(inLowBand, t1, t2) and
    t2 = t1 + 30
}
action {
    WebBrowser.SetBandwidth( 0.9 ∗ NetworkInterface.availableBandwidth)
    WebBrowser.SetImageBW(true)
}
```

This rule is triggered by the low available bandwidth. When the low bandwidth state is active for more than 30 seconds the platform will request the pair of web proxies to convert all delivered images to smaller black-and-white images.

6.2.1.3 E-mail client

The E-mail client is an application that was developed from scratch in order to allow a high level of external control of its behaviour. The client is using the Post Office Protocol [J.Myers96] and the Simple Mail Transfer Protocol [Postel82] to communicate with the e-mail server. The application is registered with the name *Email*. The registration information includes state variables like: the state of the application and information related to the current email being send or retrieved. The control methods supported by the e-mail client include methods for checking for new e-mails, fetching e-mails and controlling the network usage of the client (Table 6.3).

For the e-mail client the default policy rules handle the periodic checking for new e-mails and the fetching of e-mails:

```
event echeck :– Email.state = "emailChecking"
event necheck:– Email.state <> "emailChecking"
fluent noEmailCheck {
    initiates(necheck)
    terminates(echeck)
}
condition {
    initiates(noEmailCheck, necheck, t1) and
    not clipped(noEmailCheck, t1, t2) and
    t2 = t1+300
}
action {
    Email.CheckMail()
```

E-mail State Variables	
string state	Reports the current state of the e-mail client. Possible values: (i.e. ‘idle’, ‘emailChecking’, ‘emailFetching’, ‘emailSending’, ‘fetchReq’, ‘sendReq’)
int currentEmailSize **string currentEmailFrom** **string currentEmailTo** **string currentEmalSubject**	State variables related to the current e-mail either the one being fetched or the one being sent.

E-mail Methods	
Check()	Retrieves the headers of the new e-mails from the server.
FetchEmail()	Retrieves the current email from the server.
SuspendNet()	Suspends all network activity.
ResumeNet()	Resumes network activity continuing the operation that was stopped by the *SuspendNet* call.

Table 6.3: E-mail: Adaptation Interface

Network Interface State Variables	
string state	String variable that reports the current state of the network interface (i.e. ‘idle’,‘sleeping’,‘suspended’).
int netBandwidth	The bandwidth of the network connection.
int availableBandwidth	The presently available bandwidth.

Network InterfaceMethods	
Sleep()	Set the network interface into sleep mode.
Suspend()	Set the network interface into suspended mode.
Wake()	Resumes from sleep or suspended mode.

Table 6.4: Network Interface: Adaptation Interface

```
}
```

This rule triggers the Email client to check for new e-mails 5mins after the last check.

```
event fetchEmail :– Email.state = "fetchReq"
condition {
    happens(fetchEmail)
}
action {
    Email.FetchEmail()
}
```

This rule lets the e-mail client fetch the body of the requested e-mail. This is an unconditional fetching rule that can be modified by the user to perform any custom adaptations when fetching an e-mail.

6.2.1.4 Network Interface

The network interface module is a system application that reports and controls the state of the local network connection. This module registers with the platform as a *Network-Interface* application. The network interface module reports the bitrate of the existing network connection and an estimation of the available bandwidth. For the estimation of the available bandwidth the network interface module periodically retrieves the number of bytes received and transmitted by the network interface. The estimation of the used bandwidth is based on the calculation of the weighted average usage of the network

Power Monitor State Variables	
string state	String variable that reports the current state of the Power Monitor (i.e. 'charging','battery').
int percent	The battery status in terms of percentage being full.

Table 6.5: Power Monitor: Adaptation Interface

card.

$$avBand_k = \frac{\sum_{i=k}^{n} A_{i-k} b_i}{T \sum_{i=k}^{n} A_{i-k}} \qquad \forall i,j \quad i > j \Rightarrow A_i > A_j$$

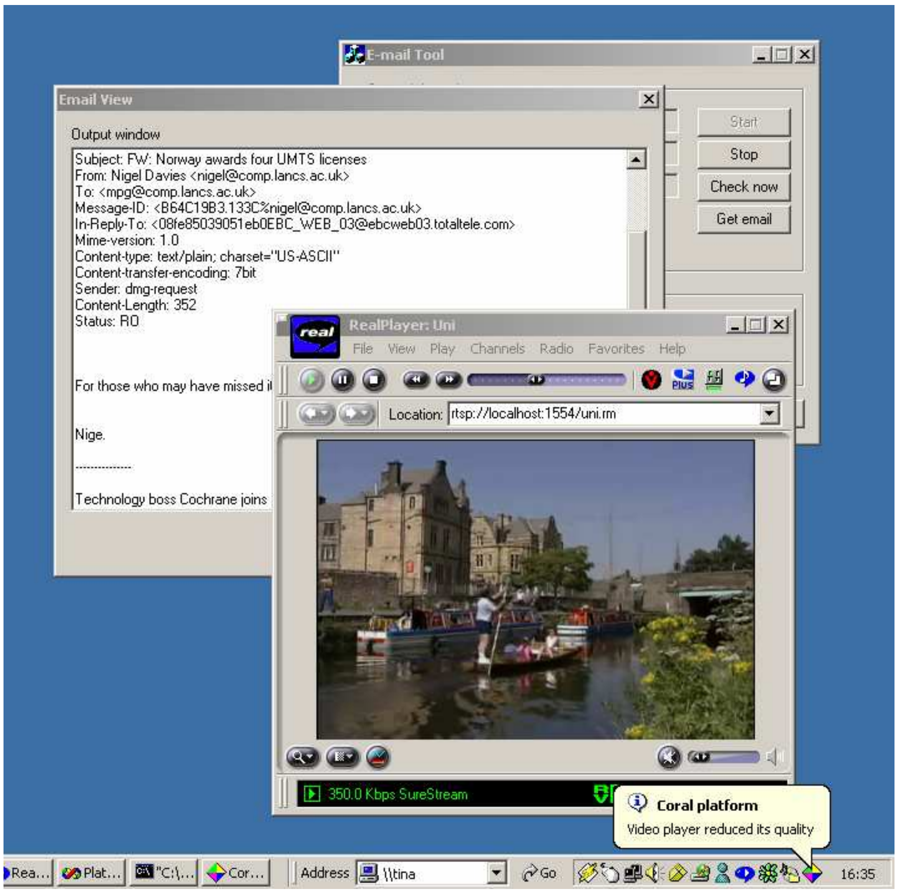
where n is the total number of samples, k is the oldest sample used for the calculation of the weighted average, A_i $(i = 0, \ldots, n-k)$ is the weight used for the calculation of the average, b_i is the sampled traffic in bytes and T is the time period between samples. The weights $A_0 \ldots A_{n-k}$ follow a linear increase of values. This is a very simple way to estimate the average bandwidth usage on the network card. It is not intended for general use and it is here only as a simple mechanism that can support network related adaptation in the context of this evaluation.

6.2.1.5 Power Monitor

The power monitor is a system application that reports the state of the battery power in the system. The interface of the power monitor includes a state variable reporting the percentage that the battery is full and the current state of the power source (i.e. running on batteries or charging).

Location Monitor State Variables	
string state	String variable that reports the current state of the Location Monitor (i.e. 'active','stopped').
string locationLabel	String variable that returns the label of the current location(i.e. 'office', 'home').

Table 6.6: Location Monitor: Adaptation Interface


```
Subject: FW: Norway awards four UMTS licenses
From: Nigel Davies <nigel@comp.lancs.ac.uk>
To: <mpg@comp.lancs.ac.uk>
Message-ID: <B64C19B3.133C%nigel@comp.lancs.ac.uk>
In-Reply-To: <08fe85039051eb0EBC_WEB_03@ebcweb03.totaltele.com>
Mime-version: 1.0
Content-type: text/plain; charset="US-ASCII"
Content-transfer-encoding: 7bit
Sender: dmg-request
Content-Length: 352
Status: RO


For those who may have missed i


Nige.

---------------

Technology boss Cochrane joins
```

Location: rtsp://localhost:1554/uni.rm

350.0 Kbps SureStream

Coral platform
Video player reduced its quality

Figure 6.4: Notification message from the User Awareness Module

6.2.1.6 Location Monitor

The location monitor module is an application that simulates a location tracking service. The design of the location module is influenced by location-aware systems based on location advertising beacons (e.g. [Cheverst00]) mostly used in cellular networks where location is identified by a network cell id. Specifically the location module returns the current location of the system in terms of a string label. This string label contains the tag of the current location, such as ‘home’, ‘office’, ‘corridor’, etc.

6.2.1.7 User Awareness Module

The user awareness module is a simple application that notifies the user about adaptation actions that the platform is performing. The module works in the background and can present notification messages to the user in the form of balloon pop-up messages or modal dialog boxes. The user is able to set the level of intrusion for the notification messages. In more detail, on the lowest level of intrusion setting the user awareness module will not show any messages to the user and simply logs the messages for later review. Medium level of intrusion allows the awareness module to show balloon pop-up message over the system task bar. High level of intrusion allows the awareness module to notify the user through modal dialog boxes. The interface of the awareness module includes a simple method Notify that an adaptation policy rule can call in order for a particular message to be presented:

```
condition {
    ......
}
action{
    ......
    Awareness.Notify("Video player reduced the quality of the video")
}
```

6.2.1.8 Applications Summary

The previous sections presented the set of adaptive applications that were developed as part of this evaluation procedure. In particular, existing applications were modified in order to collaborate with the prototype adaptation platform as well as new applications and monitoring components that were implemented to work with the platform. The following section presents the qualitative evaluation of the adaptation platform. This qualitative evaluation includes the use of the presented applications and the definition of adaptation policies that perform coordinated adaptation, conflict resolution or extend the triggers that applications can respond to.

6.2.2 Coordination

One of the main subjects of criticism for existing adaptation support platforms presented in chapter 2 is the lack of efficient coordination between applications. Specifically, many adaptation systems (e.g. Puppeteer, Laissez-Fair applications) tend to treat applications in isolation from the rest of the system. Systems based on open architectures break this isolation but rely on the application developer to use information about other applications and achieve coordinated adaptation. There are platforms that were developed in order to support coordination. However, these platform tend to look at coordination from a limited point of view. For example, coordination support platforms such as the Event Heap considers the support for the the exchange of notification messages between applications. Such approach relies on the applications themselves to use these notifications and coordinate their activities. Middleware platforms consider coordination as a mechanism to achieve balanced resource sharing between applications. In Odyssey in particular, allowing applications to specify their requirements in terms of resources and maintaining resource sharing according to these requirements is considered a form of coordination.

In this thesis we consider coordination as the ability of the system to coordinate the actions taken by adaptive applications based on a set of specified rules. The aforementioned approaches are either special cases of this approach (e.g. resource sharing) or supporting technologies (e.g. event exchange mechanisms).

In order to look at the support for coordination offered by the platform presented in this thesis, a simple coordination scenario was implemented. This scenario shows how two applications (a Web browser and an E-mail client) can coordinate the use of the network in order to improve the delivery time of web content. In particular, using the adaptation interfaces described in section 6.2.1 a set of policy rules were defined that control the two application to trigger the e-mail client to suspend the use of the network when a web page is downloaded. This can be a required behaviour for a system connected over a weak link. In particular, as the web browser is an interactive application, disrupting the loading of a page in order to check for new e-mails might be undesirable for the user. In order to achieve this effect two additional rules were added to coordinate the e-mail client with the web browser:

```
event webDownload :– WebBrowser.state = "downloading"
condition {
    happens(webDownload)
}
action {
    Email.SuspendNet()
}
```

This rule causes the e-mail client to suspend all network activity when the web browser starts to download a page. An additional rule is specified that will trigger the email client to resume the network usage when the page has been downloaded.

```
event webDownload :– WebBrowser.state = "downloading"
event webNotDownload :– WebBrowser.state <> "downloading"
fluent webNotDownloading {
    initiates(webNotDownload)
    terminates(webDownload)
}
condition {
    initiates(webNotDownloading, webNotDownload, t1) and
    not clipped(webNotDownloading, t1, t2) and
    t2 = t1+10
}
action {
    Email.ResumeNet()
}
```

This rule triggers the e-mail client to resume network activity when the web browser has stopped downloading pages for more than 10 seconds.

As we can see from this example the actual approach that is used by the platform presented in this thesis is quite different from the adaptation approach used by existing middleware applications. In more detail, existing systems require the applications to specify their resource requirements and possible coordination is performed implicitly by allowing sharing of the resources. However, here the adaptation rules control the actual actions that the applications are required to take. In more detail, the notion of coordination as it is approached by this work relates to activities and resources used.

An implication that is derived from this approach is that the adaptation mechanism is not related to a specific resource. In more detail, coordinated actions can be specified for applications regardless of the involvement of resource sharing or not. Indeed, in a context-aware environment coordination exceeds the boundaries of resource sharing. Applications may require to coordinate their actions simply because that is what the user wants. These coordinated actions may be related to a resource (as presented in the previous scenario) or it can be a requirement of the user. For example, one scenario that falls in this category is to coordinate applications in relation to the location of the user. Specifically, a rule that can switch off the audio from the video player when the user enters the corridor of the building is the following:

```
event corridorIn :— Location.locationLabel = "corridor"
event corridorOut :— Location.locationLabel <> "corridor"
fluent inCorridor {
    initiates(corridorIn)
    terminates(corridorOut)
}
condition {
    initiates(inCorridor, corridorIn, t1)
}
action {
    VideoPlayer.StopAudio()
}
```

By extending the rule body this rule can be used to coordinate other applications that should be triggered when the user enters the corridor:

```
action {
    VideoPlayer.StopAudio()
    WebBrowser.SetProxy(10.10.10.1, 8080)
}
```

As discussed in the previous paragraphs, an existing approach in resource management is to allow applications to specify resource requirements and rely on the system to satisfy their requirements. Although the design of this platform does not rely on resource reservation mechanisms to control resource sharing, the actual design of the platform does not prevent this. In more detail, the adaptation interfaces of the application can include state variables that express the resource requirements of the application. One such example is the video player. The state variables representing the existing bit rate of the video stream along with the bit rates of lower and higher bit rate streams are actually indications of resource windows that the video can switch to. The default policy rules that were described in section 6.2.1.1 control how the application can adapt if the network resources are either enough to support a higher quality video stream or not enough for the current stream and the player should switch to a lower quality one. Nevertheless even in this case the policy rules that control such an application are based on actions that should be taken instead of an explicitly resource related adaptation approach. Specifically, the rules include the actions that the application should take in order to adapt or collaborate in a coordinated adaptation. This characteristic allows the platform to offer a general purpose coordination mechanism.

One observation that is derived from the video example and the coordination scenario presented earlier is that there is a clear relationship between the adaptation interfaces exposed by the applications and the degree to which coordination can be achieved. In more detail, in the web browser-email client coordination scenario the fact that the e-mail client implements an adaptation interface with the method calls SuspendNet and ResumeNet is vital to achieve the specific coordinated action. Generalising this observation it is clear that the level of flexibility offered by this platform is directly linked to the level of control applications offer to the platform. Considering a system where applications follow an open approach (i.e. a Reflective approach) in their design might allow the platform to have a greater degree of control over the actions of the applications.

Summarising the discussion on coordination, the adaptation support platform discussed here follows an approach where adaptation is not related to resource sharing between applications according to the requirements expressed by applications. Rather the approach followed is related to the actual actions that the applications are required to take in order to achieve resource related coordination or any other type of coordination. Specifically, this approach is general enough to allow coordination for any context related information that applications should respond to (e.g. location). One observation that is derived from this investigation is that the degree of flexibility in achieving coordination between applications is directly related to the adaptation interfaces exposed by the applications. In particular the more control applications offer to the platform, the more flexibility is possible to coordinate adaptive applications.

6.2.3 Conflict Resolution

Before we investigate the features of the adaptation platform discussed here in relation to conflict resolution it is first necessary to define what a *conflict* is. Let's consider an obvious case of a conflict. Consider the following adaptation policies defined for the email client:

```
event officeEvent :− Location.locationLabel = "office"
condition{
    happens(officeEvent, t1)
}
action{
    Email.SuspendNet()
}

event officeEvent:− Location.locationLabel = "office"
condition{
    happens(officeEvent, t1)
}
action{
    Email.ResumeNet()
}
```

It is clear that the two rules are identical with the only difference in their action body. Specifically, both rules are triggered when the user enters their office and the one triggers the e-mail client to suspend all network activity while the other triggers the client to resume network activity. Obviously this is a conflicting situation. When these rules are used the actual result depends on the sequence that the rules are evaluated. Both are activated but the outcome after their execution is the one caused by the rule executed last. Now, if we modify the first rule as follows:

```
event officeEvent :– Location.locationLabel = "office"
condition{
    happens(officeEvent, t1)
}
action{
    Email.Check()
}
```

This rule triggers the email client to check for new e-mails when the user enters their office. This new rule does not conflict with the second rule described previously. Indeed, the result is that the e-mail client resumes network activity when the user enters their office and checks for new e-mails.

The examples presented here offer an interesting observation. One factor that is directly related to conflict detection is the actual semantics of the actions that are executed when an application is triggered to adapt. In particular, the diference between the conflicting rules and the non confliction rules is that the action SuspendNet and ResumeNet can not take place at the same time, while the actions ResumeNet and Check can. Therefore, the actual reason for this conflict is the dependency between the two actions and in particular the fact that they both affect the network connectivity of the application in contradicting ways.

Moving one step on, consider the two conflicting rules presented earlier and consider the case where the first rule has the following form:

```
event officeEvent :– Location.locationLabel = "office"
condition{
    happens(officeEvent, t1) and
    t2 = t1+1
}
action{
    Email.SuspendNet()
}
```

This rule is similar to the one presented with the only difference that it is activated one second after the user enters the office. With this rule the actual sequence of actions is now predictable and moreover, in strict terms this is not a conflict since the two rules are triggered by different conditions. However, in practice the user experience is the same as before. In particular, from the user's point of view the two rules should not coexist since they don't make any practical sense.

Generalising the aforementioned observations in a context-aware environment where the primary aim for the system is to enhance the user experience, the concept of a conflict is best described as an "undesirable behaviour". Indeed the previous example is not a conflict in strict terms but it is an undesirable behaviour of the system. Another similar example is the case of the web browser and the video player described in section 6.2.1 using their default policies. When the two applications run together an interesting behaviour is observed: when the available network bandwidth is low the web browser tends to reduce its demand for network in favour of the video player.

The reason for that behaviour lays in the set of default policies. As seen earlier the video player is constantly trying to deliver the best possible video quality that can be supported by the available network bandwidth. If the web browser stays inactive for more than 10 seconds (something that quite often happens) the video player takes over the available bandwidth in order to improve the delivered video. After that, the browser can never recover. According to the policy rules, the browser will maintain its delivery bitrate within the limits of the new available bandwidth.

This behaviour can be considered a conflict or not, depending on the requirements of the user. In particular, if the user requires the web browser to have fast responses regardless of any video player active in the system then this case is certainly a conflict. However, if the user requires the video player to use the best video quality possible, then this behaviour is not a conflict. Referring back to the first example discussed the actual cause of this, possibly, undesirable behaviour is again the dependency of the two applications on the network interface and the fact that the actions taken by the applications affect the available bandwidth. Moreover, this change of available bandwidth has an impact on the policy rules that check the available bandwidth in order to take their decisions.

Summarising this discussion the following observations can be made: in order to identify clear and undisputed cases of conflicts (e.g. the first presented example) it is necessary for the system to have an understanding of the semantics of the actions that applications can perform and in particular the dependencies between the actions. It should be noted that the semantic understanding of application behaviour is beyond the scope of this thesis.

Apart from these cases, in a context-aware environment most cases that can be considered conflicts depend highly on the user requirements. In particular, certain situations such as the example of a conflict between the web browser and the video player can be considered as undesirable or conflicting for some users and a desirable system behaviour for others. The author supports the notion that such perceptual conflicts are best detected and resolved by the user themselves.

One approach used by this platform is to rely on the user to identify the dependencies between adaptation actions and possible conflicts that may occur. In particular, the descriptions of the application interfaces along with the access to the set of policy rules active in the system allow the user to investigate the behaviour of the system and discover the reasons that a certain undesirable behaviour is taking place.

One important feature of this platform is that when conflicts or undesirable behaviours are identified the user has the ability to modify the system in order to resolve such cases. This power is derived from the fact that the adaptation decisions are based on modifiable policy rules. Therefore the user has the power to modify the policy rules that cause conflicts and resolve such cases. For example, if we consider the video player and the web browser running in the system using their default policies a user may require for the web browser not to reduce its network usage in favour of the video player. In order to resolve such a situation the default policy rules were modified. The aim of this change was to achieve coordination between the two applications.

The first step in specifying the policy rules for resolving this conflict is to define a fluent that will monitor the existence of the web browser in the system:

```
event webBrowserOn :− isRunning(WebBrowser) = true
event webBrowserOff :− isRunning(WebBrowser) = false
fluent webBrowserRunning {
    initiates(webBrowserOn)
    terminates(webBrowserOff)
}
```

Next the default policy rules of the video player are modified so that the they will only be used if the web browser is not running in the system:

```
condition {
    not holds(webBrowserRunning, t1) and
    initiates(videoBandAvail, availVideoBand, t1) and
    not clipped(availVideoBand, t1, t2) and
    t2 = t1 + 10
}
```

Finally a set of new rule is defined to specify the behaviour of the video player when the web browser is running. The actual body of the policy rule depends on the preferences of the user. A possible approach could be to trigger the video player to pause the video streaming (without changing the quality of the stream) when a web page is requested, so that the web page can be uploaded faster.

```
condition {
    happens(requestURL, t1)
}
action {
    VideoPlayer.Pause()
}
```

A similar condition can be used to start the video streaming when the page upload is finished. Looking at this example closely it is clear that the initial reason for this conflict is the fact that the default policy rules implemented by the developer were not aware of other applications and other adaptation policies that may cause this effect. Indeed, developers can not be expected to know the configuration of the target system and thus undesirable or conflicting situations like this is possible to occur. Moreover, in order for this conflict to be resolved there were two main requirements. First, that the decision logic must be able to be modified after the applications were installed in the end system and, second, that the decision logic can have access to information about both conflicting applications. Thus, resolving such a conflict requires the user to modify the decisions that lead to the conflict. Moreover, when more than one application is involved the modification may include information from all related applications. As presented in this example, the platform described in this thesis meets these two requirements.

Summarising this discussion, the aim of the platform discussed here is not to provide mechanisms for conflict detection. The approach followed by this prototype is to rely on the user in order to discover possible conflicts or undesirable behaviour. This is based on the fact that automated conflict discovery would require active participation of the application developer in order to allow the platform to understand the applications' semantics and identify possible dependencies between applications. Moreover, a wide range of possibly conflicting situations are actually related to the user requirements and therefore should include the user in the process of conflict discovery. When conflicts or undesirable behaviour is identified, the user can resolve these conflicts by modifying the adaptation policy rules that cause the conflicts. As seen in one of the examples, in certain cases the solution to conflicts between multiple applications running in the system is to coordinate the adaptive actions so that the conflict can be overcome.

6.2.4 Extensibility

The concept of extensibility in the context of a platform supporting adaptive context-aware applications is related to the degree to which the platform allows the incorporation of adaptation triggers that the applications were not initially designed to support. In more detail, an application developer typically makes assumptions about the configuration of the end system. For example, the developer of a web browser assumes that the end system has a network connection that the web browser can use. However, these assumptions should be kept to the minimum in order to achieve greater level of portability. Therefore, it is possible that an application will not utilise special purpose monitoring components that are not expected for common computer systems. In particular, a component that reports the location of a mobile device is not commonly expected to exist on all end systems. Moreover, as certain context-aware systems may rely on technologies that are tightly coupled with the actual working/living environment of the user. For example, the location component that was developed as part of this evaluation reports the location of the mobile device in the form of labels that represent certain locations. Such a component cannot be expected to be utilised by an application by default.

These observations support the conclusion that in adaptive context-aware systems certain applications will not be able to utilise all available monitoring components. However, from the end-user's point of view the coordination of the running applications in relation to all monitoring technologies available is definitely desirable. For example, let's consider the case of the web browser described in section 6.2.1.2. This web browser uses a pair of web proxies that control the data stream over a wireless link. Considering that the end device is a mobile device, it is obvious that using a statically configured remote proxy can degrade the performance of the communication. Specifically, as the end device moves in different locations the path between the server-remote proxy-local proxy will not always be the optimal path for fast delivery of data. Therefore a desirable feature for the operation of the web browser would be to dynamically switch to alternative remote proxies when the location of the end system changes. In essence this means that the web browser should become *location aware*. In order to achieve this a set of new rules were added to the system:

```
event inOffice :– Location.label = "offIce"
condition {
    happens(inOffice)
}
action {
    WebBrowser.SetProxy("10.2.3.4", 5123)
}

event inHome :– Location.label = "home"
condition {
    happens(inHome)
}
action {
    WebBrowser.SetProxy("10.3.2.1", 5123)
}
```

This rule is triggered when the mobile device enters the user's office. The action includes the specification of a new proxy that is assumed to be closer than the one previously used.

It is simple to consider similar examples where applications can become location-aware. One example described in section 6.2.2 is the switching off of the video player's audio stream when the user is walking down a corridor of the building. The main idea behind this adaptation policy is to turn off the audio so that the user will not disturb other people working in their offices as he/she passes by their door.

When considering such policy rules an interesting observation is that the policy rules can be easily enhanced in order to include much more precise conditions. Specifically, it is possible to define rules such as "trigger an action that is invoked when the user enters their office after passing from the kitchen":

```
event kitchenIn :– Location.label = "kitchen"
event kitchenOut :– Location.label <> "kitchen"
event officeIn :– Location.label = "office"
event officeOut :– Location.label <> "office"
fluent inKitchen {
    initiates(kitchenIn)
    terminates(kitchenOut)
}
fluent inOffice {
    initiates(officeIn)
    terminates(officeOut)
}
condition {
    initiates(inOffice, officeIn, t1) and
    holdsat(inKitchen, t2) and
    t2 < t1
}
action {
    ...
}
```

This rule is activated if the fluent *inKitchen* was active before the fluent *inOffice* is initiated. A practical example would probably have time limit between the two situations e.g. t2 $<$ t1 **and** t2 $>$ t1 + 600.

Considering the aforementioned examples it is clear that the ability of the platform to extend existing applications, adding awareness for additional context triggers is based on the fact that the adaptation control mechanism is completely decoupled from the applications' adaptation actions. In more detail, the actions that the platform triggers can be based on any possible adaptation rules, including information from any application or monitoring module available to the system. Moreover, this mechanism is further enhanced by the fact that the platform itself does not make any assumptions about the types of monitoring modules installed in the system or the applications running. This means that the adaptation control mechanism — based on policy rules — is a general purpose controlling mechanism that can be extended to use additional information as and when needed.

6.2.5 User Involvement

As presented in the previous scenarios most of the features supported by the platform under consideration require the active involvement of the user. In particular, coordination, conflict resolution and extensibility require the user to modify or insert additional policy rules that can realise these operations. As discussed in detail, this user involvement is mainly a requirement for these features because all these three characteristics are related to the configuration of the end system that the application developers can not be assumed to have any knowledge about. However, as discussed in chapter 3 the involvement of the user is also an important requirement for systems working in a context-aware environment. Indeed, a user working/living in a context aware environment should be able to specify how their computer system should operate in relation to their context.

Most of the scenarios presented in the previous paragraphs are mainly user focused. Specifically, the implementation of coordinated behaviour, conflict resolution and extendability are all related to the actual requirements of the end user. Considering this user focused approach it should be noted that a possible drawback of this approach is the fact that the user needs to be able to understand both the way applications work in the system and how their behaviour can be modified through the use of the Event Calculus Policy Language. For most users we accept that this will be a specialist skill (e.g. the role of an administrator).

The issue of user understanding of the system's behaviour is covered to some extent by requiring applications to expose a comprehensive description of the semantics of their adaptation interfaces. This description is offered as a simple mechanism that can assist the user in order to understand the functionality implemented by the applications. Moreover, the support of the *awareness module* can allow the user to understand how the platform is behaving and if certain situations cause adaptive responses by the platform. However, as adaptation interfaces can occasionally be very complex additional mechanisms should be provided for the end user in order to allow the investigation in a more user friendly and comprehensive way. A subject of future work (Chapter 7) is to investigate possible mechanisms that will assist the end user in understanding the behaviour of an adaptive system. Moreover, the possible specification of application interdependencies discussed in section 7.3.1 could be used in order to represent graphically how the actions of one application affect the behaviour of another, offering a starting point for this work.

In terms of the user, the Event Calculus Policy Language offers a comprehensible vocabulary for specifying situations that adaptation is required. In particular, the fact that Event Calculus fluents can be specified to represent real world situations, such as "system in low battery", "user in the office", "low availability in network bandwidth" can offer substantial support in allowing the user to understand existing policy rules and modify or add new ones. However, the user involvement in the specification of policy rules can be greatly improved with the support of a user tool for the specification of policy rules. It is the authors belief that the characteristics of the Event Calculus Policy Language are well suited for the design of such a user interface. In more detail, the graphical representation of fluents and event can allow the user to see how an adaptation condition is related to possible overlapping of fluents or time period durations between to occurrence of events.

One final observation deriving from the aforementioned examples is the fact that certain policy rules are specified in order to satisfy special cases for the operation of the system while other policy rules are considered to support the general behaviour of the system. One such example is the conflict resolution between the video player and the web browser. In a real world situation the end user would require automatic switching between predefined policy rules according to environmental changes or their own requirements. This observation advocates the implementation of a full policy management system on top of the Event Calculus Policy Language. In more detail, a policy management system should include mechanisms where policy rules are grouped in policy sets that should be activated under certain conditions. This requirement is discussed further in chapter 7.

6.2.6 Qualitative Evaluation Summary

Section 6.2 presented a qualitative evaluation of the prototype adaptation support platform discussed in chapter 5 was discussed in the previous section. In particular set of adaptation applications was implemented to illustrate how applications can work in collaboration with the platform. Subsequent sections discussed in detail the issues of coordination, conflict resolution, extensibility and user involvement as these were the main motivation for this work. The findings of this evaluation can be summarised as follows:

- Coordination support is directly related to the approach followed by this platform in supporting adaptation. In particular, adaptation is not related to resource sharing but to the specific actions that applications are required to take. This fact allows the specification of policy rules that can coordinate actions regardless of any resources involved in the decisions.

- Conflict resolution is related to the involvement of the user in modifying the behaviour of the system through the modification of the adaptation policies. In more detail, conflicting situations are in general related to the actual user requirements where certain users may consider a situation as a conflict while another may consider it as an acceptable situation. This platform offers a policy based mechanism where the user can actively modify the system's behaviour and overcome possible undesirable behaviour. Coordination of multiple application is, in most cases, required in order to resolve a conflicting situation.

- Extensibility is related to the ability of the platform to incorporate information from any application or system monitoring component in a uniform way. In more detail, the adaptation control mechanism is general enough to allow the specification of policy rules that can include additional information sources. As a result applications can become aware of additional adaptation or context triggers.
- User involvement is related to the provision of user comprehensible descriptions of the adaptation interfaces provided by the applications and the use of a policy language that they can use to specify how the system should behave. It is noted that user involvement should be enhanced with the use of a graphical user interface that will support the user in understanding how the system behaves and the implementation of a policy management system that will allow the grouping of policy rules into replaceable policy sets.

6.3 Performance Evaluation

In a system where the adaptive behaviour of applications is controlled or assisted by an adaptation support platform both the applications and the platform affect the performance of the system. The performance of platform components involved in the adaptation mechanism are of particular interest as they affect the behaviour of all applications in the system. A performance evaluation of the platform is therefore necessary in order to identify its behaviour both under normal conditions and as the number and characteristics of the applications involved increase.

The primary functionality of the presented platform involves the handling of state variable changes reported by running applications, the evaluation of defined policy rules and the possible triggering of applications to perform an adaptive action as specified by the policy rules. In the measurements presented here we are trying to identify the performance overhead imposed by this chain of actions whenever a state variable change is reported by an application. In addition, a series of measurements have been conducted in order to identify the scalability factors that have a significant impact on the performance of the platform. For this scalability evaluation we define a set of variables that may affect the performance of the platform:

- Number of adaptive applications registered with the platform.
- Number of policy rules loaded in the platform's rule table.
- Number of rules affected by a single variable change reported by the application.
- Complexity of the policy rules defined.

For each of these variables a series of measurements was performed to identify their significance.

6.3.1 Methodology

All experiments used the same hardware and software configuration: a single 730 MHz Pentium III workstation with 640MB of memory running Microsoft Windows XP (SP1). The performance measurements where taken using Intel's VTune Performance Analyzer[1] [Int03]. For each of the experiments the reported results include the average time spent by the platform to process a reported variable change and the break-down of this time to individual platform components, namely: the time spent by the Application Controller receiving the application message, the time spent by the Policy Manager evaluating the related policy rule(s) and the time spent by the Application Controller triggering an adaptive reaction. All tests were conducted using a test application that was developed for the needs of this analysis. The test application allows the user to specify the state variables and methods reported to the platform, execute a series of variable changes according to a given script and report the adaptation triggers received by the platform.

In order to specify a baseline reference point we measured the platform's performance under optimum conditions. These conditions refer to an environment where only one application is registered with the platform, only one policy rule is defined and this policy rule has a very simple condition body. The particular rule used has a single "happens" predicate checking for the occurrence of an event:

```
event testEvent :– Test.testVar = "fired"
condition {
    happens(testEvent, t1)
}
```

[1]The particular technique used was the *Call Graph*. This technique reports, among others, the time spent for the execution of each function and the number of times each function has been called.

```
action {
    Test.Adapt()
}
```

Using this setup we performed an initial set of experiments to identify the minimum overhead imposed by the platform:

Minimum overhead per event

Policy Manager	2.05 milliseconds
Var Change Message	0.56 milliseconds
Adapt Trigger	0.27 milliseconds
Other	0.07 milliseconds
Total time spent	2.97 milliseconds

This measurement served as a reference point for all the subsequent experiments performed. For comparative reasons, in all the graphs presented here this measurement appears as the first test column.

6.3.2 Number of Applications

This section presents the measurements that investigate the platforms behaviour in relation to the number of adaptive applications. A series of experiments was conducted

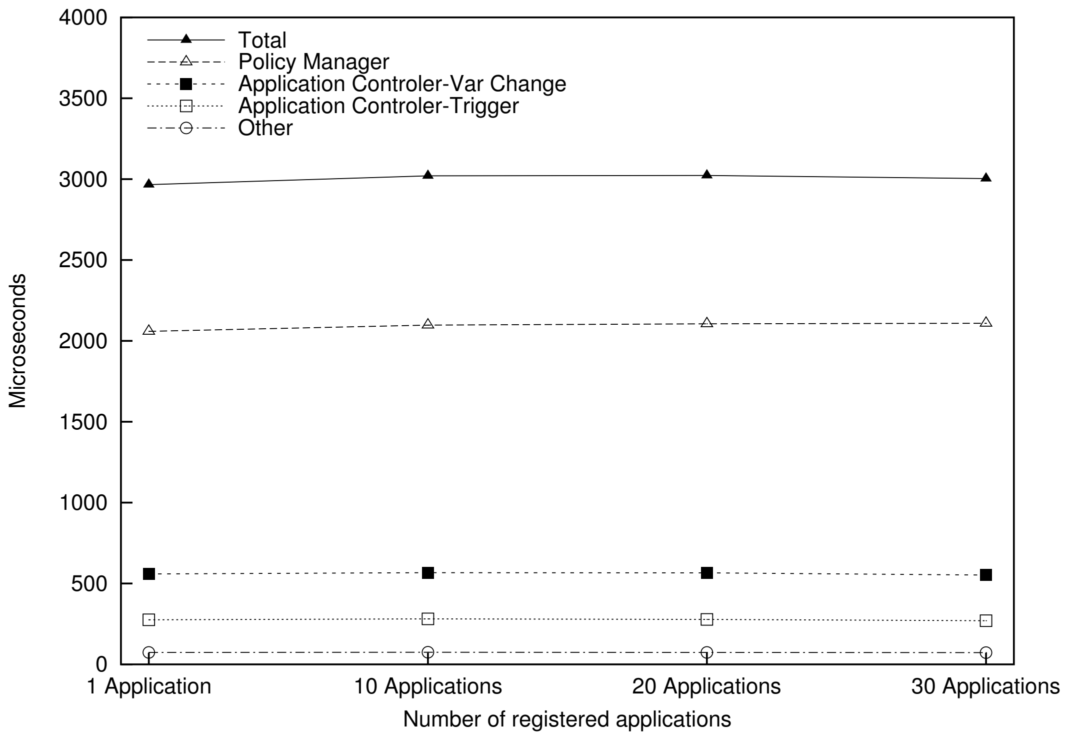


Figure 6.5: Platform overhead in relation to the number of registered applications.

with increasing number of applications registered with the platform. All other parameters (number of rules, rules' complexity) where kept to their minimum. The results of these measurements showed that the number of applications had no significant impact on the performance (Figure 6.5).

This is a reasonable result considering the design of the platform. Each application communicates with the platform through a dedicated Application Controller. During the parsing of policy rules the platform constructs a table of rules affected by each state variable and attaches it to the definition of that particular state variable. Therefore, a state variable change reported by an application leads directly to the evaluation of the corresponding policy rule without being affected by the number of applications registered with the platform. During the evaluation of a policy rule the Policy Manager is required to search through the registered applications when there are references to other state variables or when an adaptive triggering is required. However, this search is performed through a hash table which has a constant overhead. The case of a more complex policy rule with references to variables of several applications is discussed in section 6.3.4.

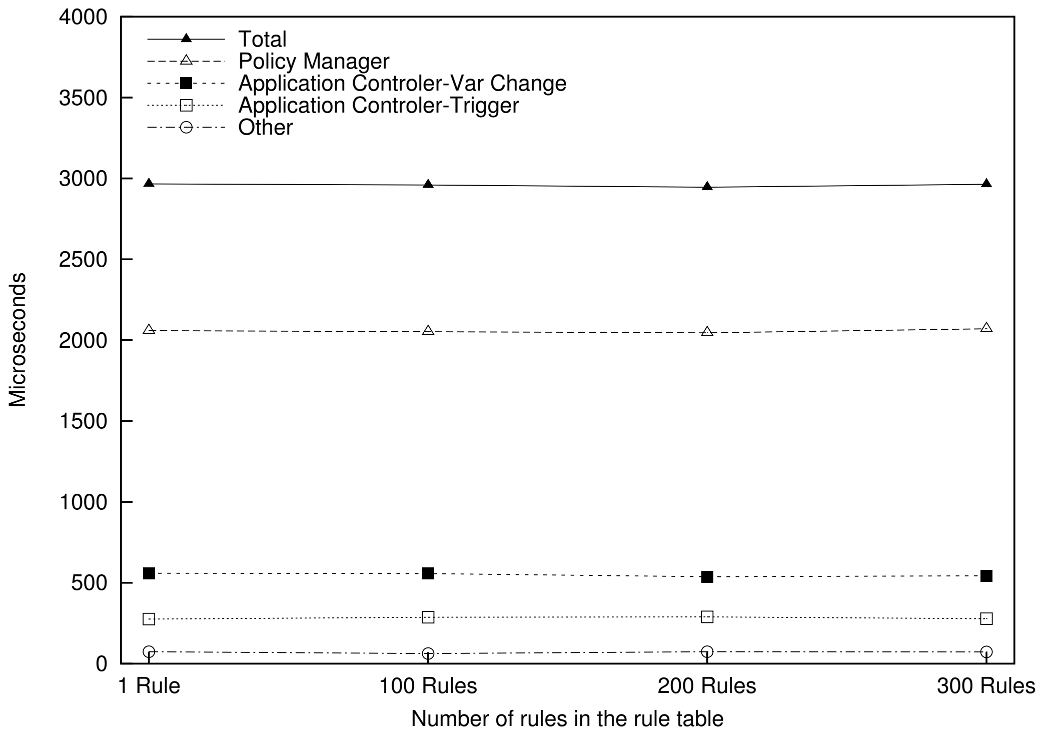


Figure 6.6: Platform overhead in relation to the number of rules in the platform's rule table.

6.3.3 Number of Rules

In order to identify the platform's behaviour in relation to the number of policy rules, a series of experiments was performed with an increasing number of policy rules loaded in the platform's rule table. All other parameters (number of applications, rules' complexity) were kept to their minimum. The results of these measurements showed that the number of rules within the platform had no impact on the performance of the platform (Figure 6.6).

As described in the previous paragraph, during parsing of the policy rules the platform constructs a table of rules affected by each state variable and attaches it to the definition of that particular state variable. As a result the total size of the rule table has no impact on the time spent processing a single state variable change. The particular case of a state variable affecting more than one rule is discussed in 6.3.5.

6.3.4 Rule complexity

The term *rule complexity* used here has a vague meaning that can not be mapped directly into a quantitative attribute. In order to specify a way to measure the complexity of a policy rule we will refer to the evaluation mechanism described in section 5.3.5.1. As described there, all predicates are mapped into two state FSAs. The evaluation procedure involves the feeding of these FSAs with the corresponding events allowing them to move from one state to the next. This similarity among predicate evaluation allow us to consider the overhead to be the same no matter which particular predicate is evaluated. Therefore for this particular set of experiments we will consider the complexity of a rule as the number of individual predicates specified in the rules condition no matter what types of predicates are defined.

However, even though the actual type of the predicates involved in the evaluation of a rule may not have any significance we did make sure that a variety of predicate were involved in the construction of the policy rules used in these experiments. In more detail, the rules were constructed using a pattern of cascading fluents encapsulating a single event. The starting condition had a body of the form:

initiates(eventA, fluentA, t1) **and**
not clipped(fluentA, t1, t2) **and**
holdsat(fluentA, t2) **and**

happens(testEvent, t2)

Testing that the event testEvent took place while the fluent fluentA was holding during the period (t1,t2). Using this body as a starting point the condition was enriched with additional fluents that were required to hold during this event:

initiates(eventA, fluentA, t2) **and**
not clipped(fluentA, t2, t1) **and**
holdsat(fluentA, t1) **and**
happens(testEvent, t1) **and**
initiates(eventB, fluentB, t3) **and**
not clipped(fluentB, t3, t1) **and**
holdsat(fluentB, t1) **and**
initiates(eventC, fluentC, t4) **and**
...

Using this pattern we conducted a series of experiments with rules of increasing complexity. The results of these experiments showed that the complexity of the policy rules had no impact on the performance of the platform (Figure 6.7).

This result is a direct consequence of the semantics of the Event Calculus Policy Language. As described in section 4.7.3 the Event Calculus Policy Language describes rules that correspond to a sequence of events that take place at different time points.

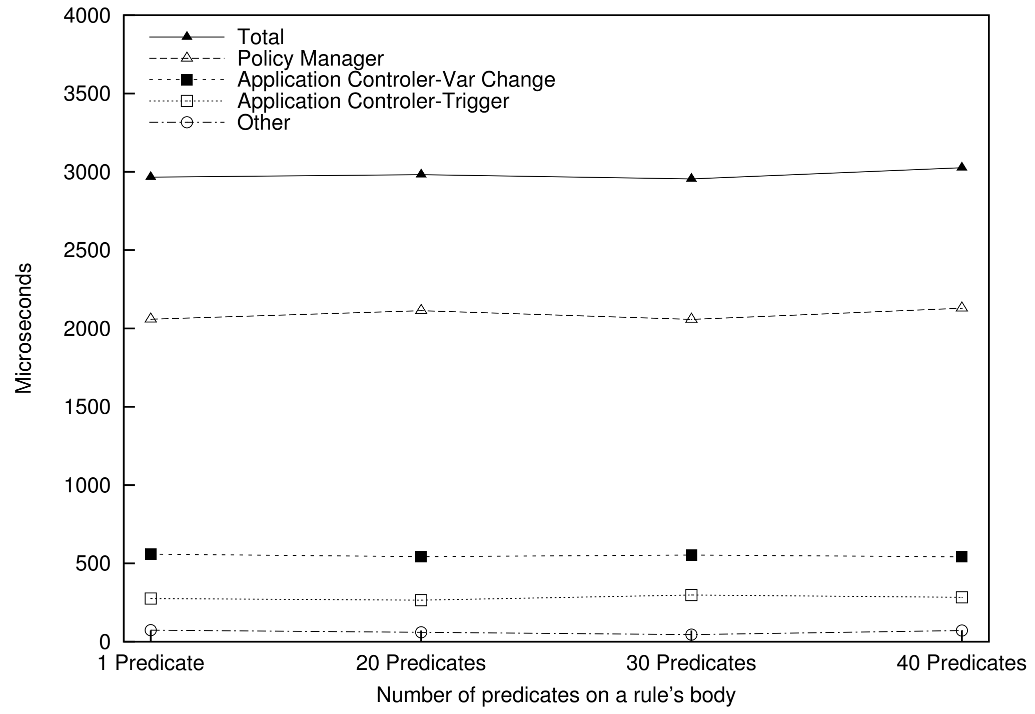


Figure 6.7: Overhead in relation to the complexity of the rules' definitions. The complexity is measured according to the number of predicates appearing in a rule's body.

As a result each event leads to the evaluation of only the particular predicates that it is involved with. Therefore the total overhead of a evaluating the whole condition body is spread over all the individual events that need to take place in order for the condition to become true. Thus the size (complexity) of the condition body has no impact on the average overhead per event. Referring back to the issue of multiple applications it is clear that even in the case of a condition body where several applications are referred, each of these references can only relate to one event. Therefore the evaluation of the predicates related to that event will include only one search through the application registry. So the average cost per event is again not related to the number of applications referred in the condition body.

6.3.5 Rules per event

The final set of measurements conducted was related to the number of rules in the rule table that are triggered by a state variable change. For this set of tests the platform was loaded with 100 simple rules (similar to the one presented in 6.3.1). For each individual set a number of these rules were modified so that they were triggered by the same event. The results of these experiments showed a linear increase of the platform's overhead

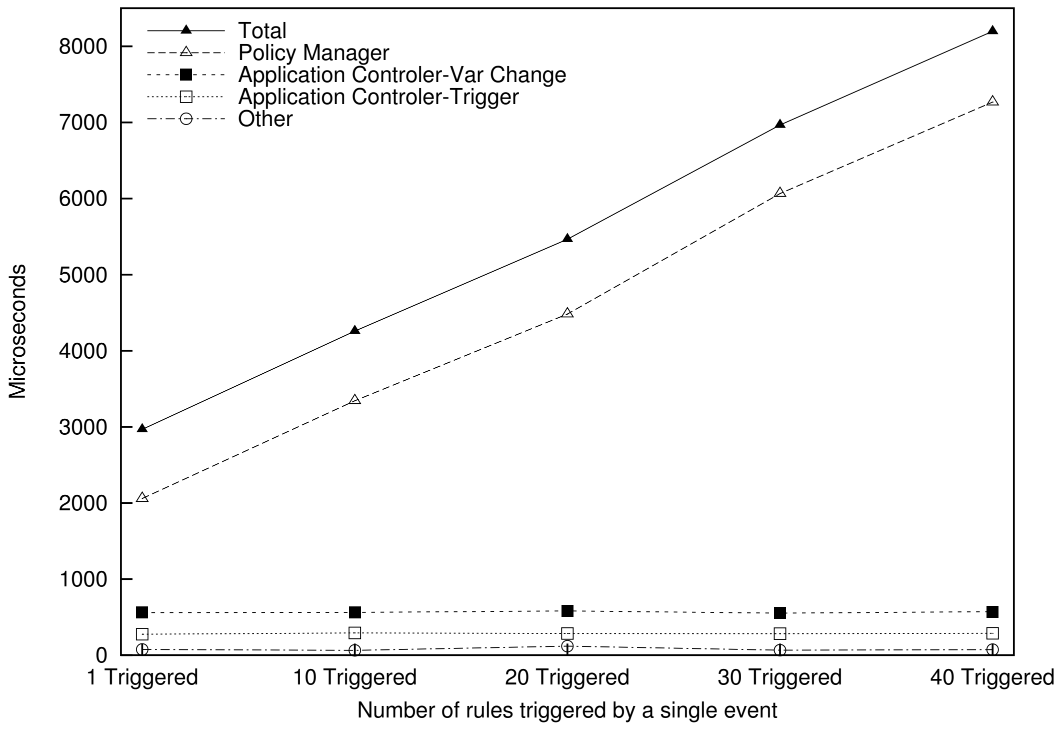


Figure 6.8: Platform overhead in relation to the number of rules triggered by a single event.

(Figure 6.8).

The linear increase of processing time is justified by the fact that more policy rules are required to be processed for each state variable change. An interesting result of this measurement is that the average time cost per policy rule is considerably smaller compared to the cost of a single rule being triggered by a single variable change. In other words the total time cost for the processing of a single policy rule triggered by one state variable change includes intra-platform function calls and evaluation of event expressions that impose additional possessing time while the time for the actual evaluation of the rule is relatively small. Therefore, in this set of measurements the additional time cost is limited (one state variable change) and, spread over a number of policy rules, leads to a smaller average time per policy rule.

This particular observation allows the specification of a certain methodology for defining policy rules that can increase the overall performance of the system. Considering cases where state variables with high granularity (e.g. a variable that reports the available bandwidth of the network connection) there is a high probability that a number of applications may define rules that refer to the same (semantically) events but with different triggering values. For example, a web browser may define a rule where the state of low bandwidth is triggered by an event when the available bandwidth drops below 24Kbps while an e-mail client may have a similar event that is triggered when the bandwidth drops bellow 20Kbps. In that case modifying these rules so that they are triggered by the same conditions (e.g. when the bandwidth drops bellow 22Kbps) can improve the overall performance of the system. However, such a modification can only be performed by the end user and therefore this is an additional case were allowing the user to modify the systems behaviour can improve the system's functionality.

6.3.6 Performance Summary

Most of the performance benefits derive from the semantics of the policy language used. The rules specified with the Event Calculus Policy Language identify the sequence of events that will lead to an adaptive response. This allows the design of a policy evaluation engine that can evaluate policy rules progressively as these events take place. Therefore the overhead of evaluating a policy rule is spread over the events taking place. As a result the average overhead per event remains constant. This fact means that the whole platform can offer predictable response time that is not affected by scalability factors such as the number of applications, the number of rules and the complexity of the rules.

6.4 Summary

The evaluation of the prototype adaptation support platform was presented in this chapter. The qualitative evaluation investigated the behaviour of the platform in terms of support for coordination, conflict resolution, extensibility and user involvement. Specifically, the platform offers the necessary functionality to achieve all four of the aforementioned features. Specific limitations concerning the particular prototype implementation include the limited support for proper tools to assist the user in understanding the behaviour of the adaptive applications.

The performance evaluation measured the performance of the platform against a set of scalability factors. In particular, the performance of the platform was tested in response to an increasing number of applications, number of rules, rules' complexity and rules triggered by a single event. The performance measurements showed that the platform can offer predictable response time that is not affected by the aforementioned scalability factors.

CHAPTER VII

Conclusions

Contents

7.1 Overview

This thesis presented an investigation of the issues concerning support for coordinated adaptation for context-aware adaptive applications. The particular problems of conflict resolution, reconfiguration and user involvement were the main motivation for coordinated adaptation. This thesis proposed that existing systems fail to provide reconfigurable coordinated adaptation supporting user involvement. It argues that support for coordination requires applications to delegate their adaptation control mechanism to an entity that can retrieve state information from multiple applications and invoke adaptation actions on multiple applications. Moreover, the adaptation control mechanism should allow modifications by the end user. The design of an architecture and the implementation of a prototype illustrate that the aforementioned requirements can actually support coordinated adaptation.

The first chapter of this thesis establishes the target domain of this work. The identification of common characteristics of traditional adaptive applications and context-aware systems concludes with the proposition that a common approach for supporting adaptation is possible for both of these classes of applications. The chapter defines the target of this thesis as the provision of adaptation support for context-aware adaptive applications. Moreover, the issue of dependencies between adaptive behaviour of multiple applications is highlighted and the need for coordination is presented as a prime requirement for supporting multiple co-existing context-aware adaptive applications.

The second chapter consists of an investigation of existing adaptive and context-aware systems. This investigation is focused on the level of support provided by existing systems for coordination, reconfiguration, extensibility and user involvement. The results of this investigation indicate that existing systems offer limited support for coordination and furthermore none of the examined systems offers support for all of the targeted characteristics.

The third chapter of this thesis presents an analysis of the possible limitations of current approaches for supporting adaptation. Through a set of theoretical scenarios the shortcomings of existing designs are highlighted. In particular, the approach of coupling adaptation mechanisms and adaptation control and the lack of a mechanism for reconfiguration of the systems behaviour through the involvement of the user are considered as the main reasons for the limited support for coordinated adaptation. The chapter concludes with a set of design requirements for supporting coordinated adaptation for context-aware adaptive applications. These requirements are to:

- Decouple adaptation policies and adaptation mechanisms.
- Require applications to externalise their adaptation mechanisms.
- Require applications to externalise information about their state or environmental attributes they monitor.
- Provide a mechanism where adaptation control entities can be modified without the need for re-implementation of the applications or the system.

Based on these requirements, the fourth chapter of this thesis presents the design of an architecture for the support of coordinated adaptation. The discussion that leads to this design illustrates how the aforementioned requirements are sufficient for achieving coordinated adaptation. Specifically, coordination is considered the ability of an adaptation support system to retrieve information from multiple applications and monitoring components and trigger adaptation to multiple applications. The first three of the design requirements allow the design of such a system. Furthermore, the consideration that applications should not be expected to have any knowledge about the characteristics of co-existing applications leads to the conclusion that coordination is not a feature that applications can provide by default. Thus the requirement for reconfiguration and the involvement of the user allows the design of a platform where coordinated behaviour can be specified based on the configuration of the end system. The design of the platform presented in chapter 4 consists of a policy based adaptation control mechanism. In order to satisfy the adaptive requirements of context-aware adaptive applications a new policy language is defined derived from the Event Calculus logic programming formalism. The main feature of the Event Calculus Policy Language is the support for policy rules where the condition body can include temporal relationships between events and fluents (i.e. entities that express duration).

Following the design of the platform a prototype implementation is presented in chapter 5. The prototype is a Microsoft Windows application that can control the adaptive behaviour of applications running on the same host. The chapter includes the implementation details of the prototype as well as a detailed discussion about the evaluation engine for the Event Calculus Policy Language.

The sixth chapter of this thesis presents an evaluation of the prototype. A qualitative evaluation considers the level of support for features such as coordination, conflict resolution, extensibility and user involvement. The evaluation concludes that this prototype does offer support for all these characteristics. Possible limitations of the prototype are identified (i.e. limited support for user-friendly interaction with the platform, requirement for a high level policy management mechanism) however these limitation are related to the particular implementation. The performance evaluation considers the performance characteristics of the particular prototype adaptation support platform. The performance measurements conducted reveal that the use of the prototype for controlling adaptation on a single host imposes limited performance cost and the prototype scales well in terms of number of applications controlled, number of policy rules and complexity of policy rules.

7.2 Contributions

This section reviews the main contributions of the work described in this thesis. The sequence in which the following sections are presented is based on the order they appeared in this thesis and it does not imply any ranking of importance.

C1. The Problem of Uncoordinated Adaptation

Contribution C1: *Identification of the limitations of existing systems in supporting coordinated adaptation, reconfiguration and user involvement. Identification of the design characteristics of these systems that lead to these limitations: coupling of adaptation control with either the adaptive method or the monitoring entity.*

This thesis examined the design characteristics of existing adaptive and context-aware applications with respect to the level of support for coordinated adaptation. This investigation was conducted by:

1. A criteria-based survey of existing adaptive and context-aware application.
2. An analysis by example of issues concerning coordination, conflict resolution and user involvement in the specification of the system's behaviour.

3. An analysis of the design approach followed by existing applications and the potential problems imposed by their design approach.

Based on this investigation this thesis provided the following results:

R1 Showed that existing systems perform poorly in terms of support for coordinated adaptation, conflict resolution and user involvement (Chapter 2 and 3).

R2 Identified key architectural properties of existing systems that lead to limited support for coordination. In particular, the coupling of adaptation policies with either the adaptation mechanisms or the monitoring entities does not allow coordination and extensibility. Moreover, acknowledging that application developers cannot have knowledge about the configuration of the end system, coordination can only be performed by allowing the reconfiguration of the adaptive behaviour with the involvement of the end user. Finally this thesis identifies the lack of consideration for the user requirements by existing systems with respect to the adaptive behaviour of the applications (Chapter 3).

R3 Identified the common characteristics between traditional, resource-driven adaptation and a class of context-aware applications concerned with adaptation triggered by changing context. This thesis argued that a common adaptation support approach can be used for both classes of applications (Chapter 1).

R4 Identified the importance of interdependencies between adaptive applications and in particular their adaptive actions. This thesis described that lack of consideration for such interdependencies can lead to certain undesirable effects such as conflicts, instabilities, etc.

The aforementioned results concerning the shortcomings of existing systems have been published in [Efstratiou00]. These results have been cited by a number of researchers [Loke02, Blair01].

C2. An Architecture for Supporting Coordinated Adaptation

Contribution C2: *Specification of design requirements for supporting coordinated adaptation for adaptive context-aware applications: decoupling adaptation control* *and implementation, externalisation of applications' adaptation interface and modification of the adaptation control mechanism. Presentation of an overall architecture for coordinated adaptation based on these design requirements.*

This thesis presented a set of design requirement for future platforms supporting context-aware adaptive applications. Moreover, this thesis proposed an overall architecture supporting coordinated adaptation derived from the presented requirements. In more detail, the results concerning this architecture are:

R5 This thesis presented a set of design requirements for supporting coordinated adaptation based on the analysis of the limitations of existing systems. In particular, future system designs should be based on the decoupling of adaptation policies and adaptation mechanisms and the externalisation of the application's adaptation interfaces. This particular requirement has been proposed in the past in the context of distributed computing (e.g. [Marzullo91]). This thesis transfers this design requirement to the domain of adaptive and context-aware systems. Moreover, the requirement for modification of the adaptation control entity without the need for re-implementation allows the design of systems where the user can actively specify how applications should behave (Chapter 3).

R6 This thesis presented the design of an overall architecture for a platform that supports coordinated adaptation. The design is derived from the aforementioned set of requirements. This design does not make any assumptions about the level of distribution of the system (Chapter 4).

R7 This thesis explored the issues of distribution in the design of a platform supporting coordinated adaptation. Possible technologies for realising both distributed and non-distributed configurations of the platform were presented (Chapter 5).

The set of requirements and the design of this platform has been published in [Efstratiou01, Efstratiou02a]. These publications have influenced to some extent the work of a number of researchers in the wider area of mobile and adaptive systems [Indulska03, Popovici02, Rakotonirainy01, Yuan04, Riva03, Costa03]

C3. A Policy Language Supporting Temporal Relationships

Contribution C3: *Specification of a new policy language derived from the Event Calculus logic programming formalism. This new language allows the specification of policies based on temporal relationships between events and entities that express duration.*

This thesis presented the definition of a policy language that was designed in order to support the specification of policy rules where temporal relationships between events are considered important. The Event Calculus Policy Language was derived from the semantics of the Event Calculus logic programming formalism. The policy rules specified in this language include conditions where the occurrence of events and the state of fluents is expressed through Event Calculus predicates. The detailed results of this thesis concerning the Event Calculus policy language are:

R8 This thesis identified the limitations of existing policy languages that follow the event-condition-action model for the support of conditions where the temporal relationships between multiple events is considered important. Specifically, this thesis acknowledges that the particular policy specification model is not intended for the expression of policy rules with temporal relationships between events. This thesis identifies as a limitation of this model the lack of support for entities that express duration. Such entities are considered important in a context-aware environment where situations like "user in their office" are entities that express duration (Chapter 4).

R9 This thesis identified the Event Calculus as a candidate starting point for a policy language that allows the specification of temporal relationships between events. Specifically, this thesis considered the use of a programming formalism for the description of event-based systems as a candidate starting point for the definition of a policy language that supports the specification of temporal relationships between events. The Event Calculus was chosen as one formalism that satisfies these requirements and offers a comprehensive vocabulary for the specification of event-based conditions (Chapter 4).

R10 This thesis demonstrated the expressiveness of this language for specifying a wide range of adaptation policies. This demonstration included a set of examples of adaptation policy rules for adaptive and context-aware applications (Chapter 4).

R11 This thesis identified this policy language as a possible candidate control mechanism that can be applied to the wider area of ubiquitous computing. The demonstration of this policy language revealed, in a certain extent, that this language is flexible enough to be applied to other domains of ubiquitous computing, such as home automation, intelligent environments, etc.(Section 7.3.3).

The specification of the Event Calculus Policy Language has been published in [Efstratiou02b] and cited in, for example, [Bandara03, Reiff-Marganiec04].

C4. Feasibility of Coordinated Adaptation

Contribution C4: *A prototype implementation of the architecture supporting coordinated adaptation. Demonstration of the prototype's ability to support coordination, reconfiguration, conflict resolution and user involvement.*

This thesis demonstrated the feasibility of coordinated adaptation in a non-distributed adaptive system. This demonstration consisted of the creation and evaluation of a prototype based on the architectural design for supporting coordinated adaptation. The detailed results concerning the feasibility of coordinated adaptation are:

R12 This thesis presented a prototype implementation of the architecture for supporting coordinated adaptation. This prototype was implemented for a non-distributed configuration where multiple applications running on the same host are controlled by a centralised platform (Chapter 5).

R13 The thesis demonstrated by example that application coordination can be achieved with the support of the prototype platform. This demonstration revealed that coordinated adaptation can improve the support for user needs that relate to the behaviour of multiple applications and the coordination of their actions (Chapter 6).

R14 The performance evaluation of this prototype showed that the use of a platform controlling adaptation based of adaptation policies can offer the benefits of coordinated adaptation with relatively small overhead. Moreover, this thesis has demonstrated that the performance of this prototype does not degrade when the number of applications, the number of policy rules and the complexity of the rules increase (Chapter 6).

R15 As part of the evaluation process this thesis demonstrated the feasibility for augmenting common applications with an API for coordinated adaptation. Although this thesis does not specify a uniform approach for augmenting existing application it does present example applications that have been extended in order to allow coordinated adaptation (Chapter 6).

The results of the evaluation of this prototype implementation have been submitted for publication.

7.3 Future Work

There are a number of issues related to this work that can become the basis for further research. Some of the most significant elements are considered in the following sections.

7.3.1 Support Conflict Detection

Dealing with conflicts in adaptive systems is a two-step process: conflict detection and conflict resolution. This thesis demonstrates that support for reconfigurable coordinated adaptation can offer the mechanisms for conflict resolution. Though beyond the main focus of this thesis, chapter 6 offered a discussion of the problem of conflict detection. In particular, the fact that adaptive methods can have side-effects or depend on other applications in the system is highlighted as one of the main causes of conflicts. The dependencies between applications' actions are generally related to the semantics of the applications and in particular the adaptive methods they implement. One particularly interesting aspect of conflict detection is the issue of user perception in the identification of conflicts as discussed in this thesis (Section 6.2.3).

Considering these observations, future research in the area of conflict detection in adaptive systems should consider both the inter-dependencies of multiple applications and the involvement of the user in the identification of conflicts. In more detail, conflict detection should combine both a mechanism for identifying *potential* conflicts and a mechanism where the user can identify the reasons the system exhibits certain undesirable behaviour. Both of these mechanisms should include the identification of dependencies between applications and their adaptive behaviour.

A possible approach for supporting conflict detection is to require the assistance of the application in the identification of dependencies. Specifically, applications should be able to express their dependencies, either in abstract terms (e.g. in terms of resources or types of services) or explicit dependencies on certain applications and functionality. With the use of these dependency declarations a platform supporting conflict detection should be able to construct a dependency graph that illustrates how adaptation actions performed by one application can have side-effects or depend on other actions and/or applications. This approach for identifying dependencies between applications can allow the investigation of mechanisms for the detection of possible conflicts and potentially suggest solutions for overcoming these conflicts. Furthermore, the dependency graph can be a useful tool for the user to comprehend how different applications interact with each other and what policy modifications are necessary in order to achieve a specific user goal.

7.3.2 Policy Management

The design of the platform presented in this thesis identifies the use of policy based mechanisms for the specification of adaptive behaviour. A particularly interesting research issue is the design of a policy management system on top of the existing platform that can allow flexible management of policies.

Existing policy management systems [Damianou01] define different classes of policy rules (e.g. obligation, authorisation). The Event Calculus Policy Language allows the specification of obligation policy rules only as required for the specification of adaptation actions. An extension of this language with the inclusion of more policy rule classes (e.g. authorisation policies) would allow the construction of a much more flexible policy management system and provide mechanisms for avoiding conflicts within the specification of the policy rules. Furthermore, an interesting feature that a policy management system could provide is the introduction of *meta policies*. Using the syntax of the Event Calculus Policy Language, a certain class of policy rules could be defined that will allow the dynamic management of existing policy rules. These meta policies could be used to enable or disable particular sets of rules based on either system conditions or user preferences. An example use of meta policies would be to dynamically modify the active policy rules when the system gets into low power mode.

7.3.3 Application to Ubiquitous Computing

The Event Calculus Policy Language was defined in order to satisfy the requirements of context-aware adaptive applications. However, the particular characteristics of this language in terms of support for temporal relationships between events can be applicable to other application domains. Specifically, environments where coordination of multiple entities is of importance can be considered possible candidate domains for the use of the Event Calculus Policy Language.

A particular domain that the author considers as a possible target for the use of this policy language is the area of active environments. Active environments require the coordination of multiple applications and devices in response to changes in the environment. Current work in the domain of active environments can be informally classified in the following categories:

1. Systems that provide support for the exchange of information between applications/devices but rely on the applications themselves to coordinate their actions (e.g. [Johanson02, Brumitt00, Kindberg01]).
2. Systems that use computer learning in order to make the system understand the requirements of the users and coordinate the applications/devices in an active environment accordingly (e.g. [Mozer98]).
3. Systems that use a rule based mechanism allowing the user to specify how the active environment should behave (e.g. [Román03]).

The architecture presented in this thesis and the Event Calculus Policy Language can be considered as potential candidates for a system falling into the third category of active environments. The use of the Event Calculus Policy language for the specification of temporal relationships between events and the specification of entities with duration (i.e. fluents) can be a significant tool in an environment where applications and devices should be coordinated according to user actions and social situations. Such policy rules can allow the user to express abstract situations such as “having a meeting” through environmental state variables such as “number of people in a room”, “volume of the speaker’s voice”, etc. A clear benefit that arises from this approach is that the user has a clear understanding of the conditions that trigger the system’s behaviour and therefore can intervene to modify the system’s behaviour if it is not according to their requirements.

7.4 Concluding Remarks

Mobile environments are tightly coupled with the notion of *change*. Change can occur in the level of resources, such as quality of the network connection, or the external context of the mobile system or the user, such as the physical location. Future mobile systems are expected to consist of a collection of applications that demonstrate adaptive behaviour in response to both of these types of changes.

The work in this thesis investigates the adaptation support for applications capable of adapting to both resource and context changes. In particular, coordination of the adaptive behaviour of applications is considered an important feature for a system that can support the user requirements and overcome conflicts. This thesis identifies the limitations of existing approaches in adaptation and proposes a set of requirements for supporting coordinated adaptation. Furthermore, an overall design for a platform supporting adaptation is presented utilising a policy based mechanism for controlling adaptation. As a proposed policy language that meets the requirements for context-aware adaptive applications the Event Calculus Policy Language is defined. The evaluation of a prototype implementation reveals the feasibility of the approach.

The author hopes that this work will influence the design of future mobile adaptive systems and allow the design of adaptation support systems with improved support for coordination and consideration of the user needs.

References

[ACP99] Advanced Configuration and Power Interface Specification, Revision 1.0. Intel/Microsoft/Toshiba http://www.acpi.info/. 1999.

[Adams95] Adams, P. and Wall, N. Global System for Mobile Communications: The Development of the GSM Standard. *British Telecommunications Engineering*, 14(1):pp. 38–45. 1995.

[Ahuja86] Ahuja, S., Carriero, N. and Gelernter, D. Linda and Friends. *IEEE Computer*, 19(8):pp. 26–34. 1986.

[Amir95] Amir, E., Balakrishnan, H., Seshan, S. and Katz, R. Efficient TCP over Networks with Wireless Links. In *Proceedings of the 5th IEEE Workshop on Hot Topics in Operating Systems (HotOS-V)*, pp. 35–40. IEEE Computer Society Press, Rosario Resort, Orcas Island, Washington, U.S. 1995.

[Apple] Apple. QuickTime. http://www.apple.com/quicktime.

[Badrinath00] Badrinath, B., Fox, A., Kleinrock, L., Popek, G., Reiher, P. and Satyanarayanan, M. A Conceptual Framework for Network and Client Adaptation. *IEEE Mobile Networks and Applications*, 5(4):pp. 221–231. 2000.

[Bakre95] Bakre, A. and Badrinath, B. I-TCP: Indirect TCP for Mobile Hosts. In *Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS)*, pp. 136–143. IEEE Computer Society Press, Vancouver, British Columbia. 1995.

[Bandara03] Bandara, A. K., Lupu, E. C. and Russo, A. Using Event Calculus to Formalise Policy Specification and Analysis. In *Proceedings 4th IEEE Workshop on Policies for Distributed Systems and Networks (Policy 2003)*, pp. 26–45. Lake Como, Italy. 2003.

[Bee00] Bumble-Bee Software. Parser Generator. 2000. http://www.bumblebeesoftware.com/.

[Blair00] Blair, G. S., Coulson, G., Andersen, A., Blair, L., Clarke, M., Costa, F. M., Duran, H. A., Parlavantzas, N. and Saikoski, K. B. A Principled Approach to Supporting Adaptation in Distributed Mobile Environments. In *Proceedings of International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE 2000)*, pp. 3–12. IEEE Computer Society, Limerick, Ireland. 2000.

[Blair01] Blair, L., Blair, G., Pang, J. and Efstratiou, C. 'Feature' Interactions outside a Telecom Domain. In *Workshop on Feature Interactions in Composed Systems, ECOOP2001*. Budapest. 2001.

[Bluetooth99a] Bluetooth. Specification of the Bluetooth System: Volume 1. Technical report version 1.0 b, Bluetooth Consortium. 1999.

[Bluetooth99b] Bluetooth. Specification of the Bluetooth System: Volume 2. Technical report version 1.0 b, Bluetooth Consortium. 1999.

[Brown95] Brown, P. J. The Stick-e Document: A Framework for Creating Context-aware Applications. *Electronic Publishing Origination, Dissemination, and Design*, 8(2/3):pp. 259–272. 1995.

[Brumitt00] Brumitt, B., Meyers, B., Krumm, J., Kern, A. and Shafer, S. A. EasyLiving: Technologies for Intelligent Environments. In *Proceedings of Handheld and Ubiquitous Computing, Second International Symposium, (HUC 2000)*, Lecture Notes in Computer Science, pp. 12–29. Springer. 2000.

[Caceres94] Caceres, R. and Iftode, L. The effects of mobility on reliable transport protocols. In *Proceedings of the 14th Intl. Conf. on Distributed Computing Systems*, pp. 12–20. IEEE Press, Poznan, Poland. 1994.

[Campbell94] Campbell, A. and Coulson, G. A Quality of Service Architecture. *ACM Computer Communications Review*, 24(2):pp. 6–27. 1994.

[Cen97] Cen, S. *A software feedback toolkit and its applications in adaptive multimedia systems*. Ph.D. thesis, Oregon Graduate Institute of Science and Technology. 1997.

[Cheverst00] Cheverst, K., Davies, N., Mitchell, K. and Friday, A. Experiences of Developing and Deploying a Context-Aware Tourist Guide: The GUIDE. In *Proceedings of the 6th ACM International Conference on Mobile Computing (MOBICOM) 2000*, pp. 20–31. ACM Press, Boston. 2000.

[Chomicki00] Chomicki, J., Lobo, J. and Naqvi, S. A Logic Programming Approach to Conflict Resolution in Policy Management. In *International Conference on Principles of Knowledge Representation and Reasoning*, pp. 121–132. Brechenridge, Corolado. 2000.

[Costa03] Costa, P. D., Filho, J. G. P. and van Sinderen, M. Architectural Requirements for Building Context-Aware Services Platforms. In *9th Open European Summer School IFIP Workshop on Next Generation Networks (EUNICE2003)*. Hungary. 2003.

[Damianou01] Damianou, N., Dulay, N., Lupu, E. and Sloman, M. The Ponder Policy Specification Language. In *Proceedings of Policy Workshop*, Lecture Notes in Computer Science, pp. 18–38. Springer, Bristol, UK. 2001.

[Davies94a] Davies, N., Pink, S. and Blair, G. Services to Support Distributed Applications in a Mobile Environment. In *Proceedings of the 1st International Workshop on Services in Distributed and Networked Environments (SDNE'94)*, pp. 84–89. Prague, Czech Republic. 1994.

[Davies94b] Davies, N., Wade, S. P. and Blair, G. S. Services to Support Distributed Applications in Mobile Environments. In *Proceedings of the 1st International Workshop on Services in Distributed and Networked Environments (SDNE '94)*, pp. 84–89. Praguw, Czech Republic. 1994.

[Davies98a] Davies, N., Finney, J., Friday, A. and Scott, A. Supporting Adaptive Video Applications in Mobile Environments. *IEEE Communications Magazine*, 36(6):pp. 138–143. 1998.

[Davies98b] Davies, N., Friday, A., Wade, S. and Blair, G. L^{2}imbo: A distributed systems platform for mobile computing. *ACM Mobile Networks and Applications (MONET) Special Issue on Protocols and Software Paradigms of Mobile Networks*, 3(2):pp. 143–156. 1998.

[Davies98c] Davies, N., Wade, S., Friday, A. and Blair, G. L^{2}imbo: a tuple space based platform for adaptive mobile applications. *ACM Mobile Networks and Applications (MONET): Special Issue on Protocols and Software Paradigms of Mobile Networks*, 3(2):pp. 143–156. 1998.

[Davies99] Davies, N., Cheverst, K., Mitchell, K. and Friday, A. Caches in the Air: Disseminating Information in the Guide System. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '99)*, pp. 11–19. IEEE Computer Society Press, New Orleans, Louisiana. 1999.

[deLara01] de Lara, E., Wallach, D. S. and Zwaenepoel, W. Puppeteer: Component-based Adaptation for Mobile Computing. In *Proceedings of the 3rd USENIX Symbosium on Internet Technologies and Systems*, pp. 159–170. USENIX Press, San Francisco, California. 2001.

[Demers94] Demers, A., Petersen, K., Spreitzer, M., Terry, D., Theimer, M. and Welch, B. The Bayou Architecture: Support for Data Sharing among Mobile Users. In *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications*, pp. 2–7. IEEE Computer Society Press, Santa Cruz, California. 1994.

[Dey00] Dey, A., Abowd, G. and Salber, D. A Context-Based Infrastructure for Smart Environments. In *Proceedings of the 2000 Conference on Human Factors in Computing Systems*, pp. 114–128. 2000.

[Dey01] Dey, A. K. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):pp. 4–7. 2001.

[Diot95] Diot, C., Huitema, C. and Turletti, T. Multimedia Applications should be Adaptive. In *Proceedings of the 3rd IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS'9)*, pp. 23–25. Mystic, Connecticut. 1995.

[Efstratiou00] Efstratiou, C., Cheverst, K., Davies, N. and Friday, A. Architectural Requirements for the Effective Support of Adaptive Mobile Applications. Work in progress paper presented in Middleware2000, (USA:New Yort). 2000.

[Efstratiou01] Efstratiou, C., Cheverst, K., Davies, N. and Friday, A. An Architecture for the Effective Support of Adaptive Context-Aware Applications. In *Proceedings of 2nd International Conference in Mobile Data Management (MDM'01)*, vol. 1987 of *Lecture Notes in Computer Science*, pp. 15–26. Springer, Hong Kong. 2001.

[Efstratiou02a] Efstratiou, C., Friday, A., Davies, N. and Cheverst, K. A Platform Supporting Coordinated Adaptation in Mobile Systems. In *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'02)*, pp. 128–137. IEEE Computer Society, Callicoon, New York, U.S. 2002.

[Efstratiou02b] Efstratiou, C., Friday, A., Davies, N. and Cheverst, K. Utilising the Event Calculus for Policy Driven Adaptation in Mobile Systems. In Lobo, J., Michael, B. J. and Dulay, N. (eds.), *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002)*, pp. 13–24. IEEE Computer Society, Monterey, Ca., U.S. 2002.

[Fitzpatrick99] Fitzpatrick, G., Mansfield, T., Kaplan, S., Arnold, D., Phelps, T. and Segall, B. Instrumenting the Workaday World with Elvin. In *Proceedings ECSCW'99*, pp. 431–451. Kluwer Academic Publishers, Copenhagen, Denmark. 1999.

[Flinn99] Flinn, J. and Satyanarayanan, M. PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications. In *Proc. of the Second IEEE Workshop on Mobile Computing Systems and Applications*, pp. 23–30. IEEE Computer Society, New Orleans, Louisiana. 1999.

[Friday96] Friday, A., Davies, N., Blair, G. and Cheverst, K. Developing Adaptive Applications: The MOST Experience. *Journal of Integrated Computer-Aided Engineering*, 6(2):pp. 143–157. 1996.

[Fuggetta98] Fuggetta, A. and G. P. Picco, a. G. V. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):pp. 342–361. 1998.

[Glass99] Glass, G. Overview of Voyager: ObjectSpace's Product Family for State-of-the-Art Distributed Computing. Tech. rep., ObjectSpace. 1999.

[Gray96] Gray, R. S. Agent Tcl: A flexible and secure mobile-agent system. In Diekhans, M. and Roseman, M. (eds.), *Fourth Annual Tcl/Tk Workshop (TCL 96)*, pp. 9–23. Monterey, CA. 1996.

[Havinga99] Havinga, P. J. M. and Smit, G. J. M. Octopus: Ebracing the Energy Efficiency of Handheld Multimedia Computers. In *Proceedings of the Fifth Annyal (ACM/IEEE) International Conference on Mobile Computing and Networking (MOBICOM'99)*, pp. 77–87. ACM Press, N.Y. 1999.

[IEEE97] IEEE. Local and Metropolitan Area Network Standards Committee. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE std 802.11-1997, The Institute of Electrical and Electronics Engineers, New York, New York. 1997.

[Indulska03] Indulska, J., Robinson, R., Rakotonirainy, A. and Henricksen, K. Experiences in Using CC/PP in Context-Aware Systems. In *In Proceedings 4th International Conference on Mobile Data Management, MDM'03*, vol. 2515 of *Lecture Notes in Computer Science*, pp. 247–261. Springer, Melbourne. 2003.

[Int03] VTune Performance Analyzer, Version 7.0. http://www.intel.com/software /products/vtune/vpa/. 2003.

[ITU] Narrow-band visual telephone systems and terminal equipment. ITU-T Recommendation (1999).

[Jacobson88] Jacobson, V. Congestion Avoidance and Control. In *Proceedings of the ACM Symposium on Communications Architectures and Protocols SIGCOMM '88*, pp. 314–329. ACM Press, Stanford, CA. 1988.

[Jacobson94] Jacobson, V. and McCanne, S. Visual Audio Tool. Available on the Internet at http://www-nrg.ee.lbl.gov/vat/. 1994.

[J.Myers96] J.Myers and Rose, M. Post Office Protocol - Version 3. Internet RFC 1932. 1996.

[Johansen97] Johansen, D., Sudmann, N. P. and van Renesse, R. Performance Issues in TACOMA. In *3rd Workshop on Mobile Object Systems, 11th European Conference on Object-Oriented Programming*. Jyväskylä, Finland. http://www.tacoma.cs.uit.no/ papers/ECOOP.tacoma.ps. 1997.

[Johanson02] Johanson, B. and Fox, A. The Event Heap: A Coordination Infrastructure for Interactive Workspaces. In *4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002)*, pp. 83–93. IEEE Computer Society, Callicoon, NY, USA. 2002.

[Joseph97] Joseph, A., Tauber, J. and Kaashoek, F. Mobile Computing with the Rover Toolkit. *IEEE Transactions on Computers: Special issue on Mobile Computing*, 46(3):pp. 337–352. 1997.

[Katz94] Katz, R. Adaptation and Mobility in Wireless Information Systems. *IEEE Personal Communications*, 1(1):pp. 6–17. 1994.

[Kindberg01] Kindberg, T. and Barton, J. A Web-based nomadic computing system. *Computer Networks (Amsterdam, Netherlands)*, 35(4):pp. 443–456. 2001.

[Kistler91] Kistler, J. and Satyanarayanan, M. Disconnected Operation in the Coda File System. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles (SOSP)*, vol. 25, pp. 213–225. ACM Press, Asilomar Conference Center, Pacific Grove, U.S. 1991.

[Kistler92] Kistler, J. and Satyanarayanan, M. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1):pp. 3–25. 1992.

[Kokar99] Kokar, M., Baslawski, K. and Eracar, Y. Control Theory-Based Foundation of Self-Controlling Software. *IEEE Intelligent Systems*, pp. 37–45. 1999.

[Kounavis01] Kounavis, M. E., Campbell, A. T., Ito, G. and Bianchi, G. Design, Implementation and Evaluation of Programmable Handoff in Mobile Networks. *Mobile Networks and Applications*, 6(5):pp. 443–461. 2001.

[Kowalski94] Kowalski, R. and Sadri, F. The situation calculus and event calculus compared. In *In Proceedings of International Logic Programming Symposium (ILPS 94)*, pp. 539–553. MIT Press, Ithaca, NY. 1994.

[Kowalsky86] Kowalsky, R. A Logic-Based Calculus of Events. *New Generation Computing*, 4:pp. 67–95. 1986.

[Kowalsky92] Kowalsky, R. Database Updates in Event Calculus. *Journal of Logic Programming*, 12:pp. 121–146. 1992.

[Leboux99] Leboux, T. OpenCORBA: A Reflective Open Broker. In *Proceedings of Reflection '99*, vol. 1616 of *Lecture Notes in Computer Science*, pp. 197–214. Springer-Verlag, St. Malo, France. 1999.

[Lobo99] Lobo, J., Bhatia, R. and Naqvi, S. A Policy Description Language. In *Proceedings of Innovative Applications of Artificial Intelligence (IAAI '99)*, pp. 291–298. MIT Press, Orlando, FL. 1999.

[Loke02] Loke, S. W. Modelling Service-Providing Location-Based E-communities and the Impact of User Mobility. In *Distributed Communities on the Web, 4th International Workshop, DCW 2002, Sydney, Australia, April 3-5, 2002, Revised Papers*, vol. 2468 of *Lecture Notes in Computer Science*, pp. 266–277. Springer. 2002.

[Long96] Long, S., Kooper, R., Abowd, G. and Atkenson, C. Rapid Prototyping of Mobile Context-Aware Applications: The Cyberguide Case Study. In *Proceedings of the 2nd ACM International Conference on Mobile Computing (MOBICOM)*, pp. 97–107. ACM Press, Rye, New York. 1996.

[Lupu99] Lupu, E. C. and Sloman, M. Conflicts in Policy-Based Distributed Systems Management. *IEEE Transactions on Software Engineering*, 25(6):pp. 852–869. 1999.

[Marzullo91] Marzullo, K., Cooper, R., Wood, M. and Birman, K. Tools for Distributed Application Management. *IEEE Computer*, 24(8):pp. 42–51. 1991.

[McCanne95a] McCanne, S. and Jacobson, V. vic : A Flexible Framework for Packet Video. In *ACM Multimedia*, pp. 511–522. ACM Press. 1995.

[McCanne95b] McCanne, S. and Jacobson, V. vic: A Flexible Framework for Packet Video. In *Proceedings of ACM Multimedia '95*, pp. 511–522. San Francisco, CA. 1995.

[Meng00] Meng, A. On Evaluation Self-Adaptive Software. In *Proceedings of the First International Workshop on Self-Adaptive Software (IWSAS2000)*, pp. 65–74. Springer, Oxford, UK. 2000.

[Microsoft03] Microsoft. Windows Media Player. Available on the Internet at http://www.microsoft.com/mediaplayer/. 2003.

[Mouly92] Mouly, M. and Pautet, M. B. *The GSM System for Mobile Communications*. Published by the authors, 4 rue Elisée Reclus, F-91120 Palaiseau, France. 1992.

[Mozer98] Mozer, M. C. The neural network house: An environment that adapts to its inhabitants. In *Proceedings of the American Association for Artificial Intelligence Spring Symposium on Intelligent Environments*, pp. 110–114. AAAI Press, Menlo, Park, CA. 1998.

[Mummert95] Mummert, L., Ebling, M. and Satyanarayanan, M. Exploiting Weak Connectivity for Mobile File Access. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP)*, vol. 29, pp. 143–155. ACM Press, Copper Mountain Resort, Colorado, U.S. 1995.

[Noble95] Noble, B., Satyanarayanan, M. and Price, M. A Programming Interface for Application-Aware Adaptation in Mobile Computing. In *Proceedings of the second USENIX Symposium on Mobile and Location-Independent Computing: April 10–11, 1995, Ann Arbor, Michigan, USA*, pp. 57–66. USENIX, Berkeley, CA, USA. 1995.

[Noble97] Noble, B. D., Satyanarayanan, M., Narayanan, D., Tilton, J. E., Flinn, J. and Walker, K. R. Agile Application-Aware Adaptation for Mobility. In *Sixteen ACM Symposium on Operating Systems Principles*, pp. 276–287. Saint Malo, France. 1997.

[Noble98] Noble, B. *Mobile Data Access*. Ph.D. thesis, School of Computer Science, Carnagie Mellon University, Pittsburgh. 1998.

[OMG01] OMG. The Common Object Request Broker: Architecture and Specification revision 2.5. OMG Document formal/01-09-01. 2001.

[Peine97] Peine, H. and Stolpmann, T. The Architecture of the Ara Platform for Mobile Agents. In *Proceedings of the First International Workshop on Mobile Agents MA'97*, no. 1219 in Lecture Notes in Computer Science, pp. 50–61. Springer Verlag. 1997.

[Pietzuch03] Pietzuch, P. R., Shand, B. and Bacon, J. A Framework for Event Composition in Distributed Systems. In Endler, M. and Schmidt, D. (eds.), *Proc. of the 4th ACM/IFIP/USENIX Int. Conf. on Middleware (Middleware '03)*, pp. 62–82. Springer, Rio de Janeiro, Brazil. 2003. Best paper award.

[Pietzuch04] Pietzuch, P. R., Shand, B. and Bacon, J. Composite Event Detection as a Generic Middleware Extension. *IEEE Network*, 18(1):pp. 44–55. 2004.

[Popovici02] Popovici, A. and Alonso, G. Ad-Hoc Transactions for Mobile Services. In *Technologies for E-Services, Third International Workshop, TES 2002, Hong Kong, China, August 23-24, 2002, Proceedings*, vol. 2444 of *Lecture Notes in Computer Science*, pp. 118–130. Springer. 2002.

[Postel82] Postel, J. B. Simple Mail Transfer Protocol. Internet RFC 821. 1982.

[Rakotonirainy01] Rakotonirainy, A., Indulska, J., Loke, S. W. and Zaslavsky, A. Middleware for Reactive Components: An Integrated Use of Context, Roles, and Event Based Coordination. *Lecture Notes in Computer Science*, 2218:pp. 77–86. 2001.

[Real03] Real. Real Player. Available on the Internet at http://www.real.com/. 2003.

[Reiff-Marganiec04] Reiff-Marganiec, S. and Turner, K. J. Feature Interaction in Policies. Submitted for publication in Elsevier Computer Networks Journal. 2004.

[RFCa] RTP Payload Format for H.261 Video Streams. IETF-RFC 2032 (1996).

[RFCb] RTP Payload Format for MPEG1/MPEG2 Video. IETF-RFC 2250 (1998).

[RFCc] RTP Payload Format for MPEG-4 Audio/Visual Streams. IETF-RFC 3016 (2000).

[Riva03] Riva, O. Middleware for Context-Aware Applications. Seminar on Research Themes in Context-Aware Computing. Department of Computer Science, University of Helsinki. 2003.

[Román00] Román, M., Mickunas, D., Kon, F. and Campbell, R. H. LegORB and Ubiquitous CORBA. In *Proceedings of the IFIP/ACM Middleware'2000 Workshop on Reflective Middleware*, pp. 1–2. ACM/IFIP, Palisades, NY. 2000.

[Román03] Román, M. and Campbell, R. H. A Middleware-Based Application Framework for Active Space Applications. In *Proceedings of ACM/IFIP/USENIX International Middleware Conference (Middleware 2003)*, vol. 2672 of *Lecture Notes in Computer Science*, pp. 433–454. Springer, Rio de Janeiro, Brazil. 2003.

[Salber99] Salber, D., Dey, A. K. and Abowd, G. D. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In Williams, M. G., Altom, M. W., Ehrlich, K. and Newman, W. (eds.), *Proceedings of the Conference on*

Human Factors in Computing Systems (CHI-99), pp. 434–441. ACM Press, New York. 1999.

[Satyanarayanan85] Satyanarayanan, M., Howard, J., Nichols, D., Sidebotham, R., Spector, A. and West, M. The ITC Distributed File System: Principles and Design. In *Proceedings of the 10th ACM Symposium on Operating Systems Principles (SOSP)*, pp. 35–50. ACM Press, Orcas Island, Washington, U.S. 1985.

[Satyanarayanan90] Satyanarayanan, M., Kistler, J., Kumar, P., Okasaki, M., Siegel, E. and Steere, D. Coda: A Highly Available File System for a Distributed Workstation Environment. *IEEE Transactions on Computers*, 39(4):pp. 447–459. 1990.

[Schilit94a] Schilit, B., Adams, N. and Want, R. Context-Aware Computing Applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pp. 85–90. IEEE Computer Society, Santa Cruz, CA. 1994.

[Schilit94b] Schilit, B. and Theimer, M. Disseminating Active Map Information to Mobile Hosts. *IEEE Network*, 8(5):pp. 22–32. 1994.

[Schmidt98] Schmidt, D. C., Levine, D. L. and Mungee, S. The design of the TAO real-time object request broker. *Computer Communications*, 21(4):pp. 294–324. 1998.

[Schulzrinne96] Schulzrinne, H., Casner, S., Frederick, R. and Jacobson, V. RTP: A Transport Protocol for Real-Time Applications. Network Working Group RFC 1889. 1996.

[Tai99] Tai, H. and Kosaka, K. The Aglets project. *Communications of ACM*, 42(3):pp. 100–101. 1999.

[Tennenhouse97] Tennenhouse, D. L., Smith, J. M., Sincoskie, W. D., Wetherall, D. J. and Minden, G. J. A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1):pp. 80–86. 1997.

[Terry95] Terry, D., Theimer, M., Petersen, K. and Demers, A. J. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proceedings of the 15th ACM Symposium on Operating System Principles*, pp. 172–183. ACM, Copper Mountain Resort, Colorado. 1995.

[Turner97] Turner, H. Representing Actions in Logic Programs and Default Theories: A Situation Calculus Approach. *Journal of Logic Programming*, 31(1–3):pp. 245–298. 1997.

[W3C00] Simple Object Access Protocol (SOAP) 1.1. W3C note, http://www.w3.org/TR/SOAP. 2000.

[W3C01] Web Services Description Language (WSDL) 1.1. http://www.w3.org/tr/wsdl. 2001.

[Waldo99] Waldo, J. The Jini Architecture for Network-centric Computing. *Communications of the ACM*, 42(7):pp. 76–82. 1999.

[Walpole97] Walpole, J., Koster, R., Cen, S., Cowan, C., Maier, D., McNamee, D., Pu, C., Steere, D. and Yu, L. A Player for Adaptive MPEG Video Streaming Over The Internet. In *Proceedings of Applied Imagery Pattern Recognition AIPR-97, SPIE*, pp. 249–258. Washington DC. 1997.

[WAP99] WAP. Wireless Application Protocol - White Paper. Wireless Internet Today. 1999.

[Weiser93] Weiser, M. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, 6(7):pp. 75–84. 1993.

[White94] White, J. E. Telescript technology: The foundation for the electronic marketplace. Tech. rep., General Magic Inc., CA. 1994.

[Wollrath96] Wollrath, A., Riggs, R. and Waldo, J. A distributed object model for the Java System. In *2nd Conference on Object-Oriented Technologies & Systems (COOTS)*, pp. 219–232. USENIX Association. 1996.

[Wyckoff98] Wyckoff, P., McLaughry, S., Lehman, T. and Ford, D. T Spaces. *IBM Systems Journal*, 37(3):pp. 454–474. 1998.

[Yeadon96] Yeadon, N. *QoS Filtering for Multipeer Communications*. Ph.D. thesis, Computing Department, Lancaster University, Lancaster, United Kingdom. 1996.

[Yuan04] Yuan, W. and Nahrstedt, K. Process group management in cross-layer adaptation. In *Proceedings of SPIE/ACM Multimedia Computing and Networking Conference (MMCN'04)*, vol. 5305, pp. 55–68. Santa Clara, CA. 2004.