4-9-2004

A framework for policy based coordinated adaptation in mobile systems

Saurabh Agrawal
Florida International University

DOI: 10.25148/etd.FI13101538

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

A FRAMEWORK FOR POLICY BASED COORDINATED ADAPTATION IN MOBILE SYSTEMS

A thesis submitted in partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE

in

TELECOMMUNICATIONS AND NETWORKING

by

Saurabh Agrawal

2004

To: Dean Vish Prasad
College of Engineering

This thesis, written by Saurabh Agrawal, and entitled A Framework for Policy Based Coordinated Adaptation in Mobile Systems, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this thesis and recommend that it be approved.

Chi Zhou

Jian Wang

Niki Pissinou. Co-Major Professor

Kia Makki, Major Professor

Date of Defense: April 09, 2004

The thesis of Saurabh Agrawal is approved.

Dean Vish Prasad
College of Engineering

Dean Douglas Wartzok
University Graduate School

Florida International University, 2004

DEDICATION

I dedicate this thesis to my family for their support and encouragement, without which the completion of this work would not have been possible.

ACKNOWLEDGMENTS

It gives me great pleasure to acknowledge and thank my advisor and co – advisor of my thesis committee, Dr. Kia Makki and Dr. Niki Pissinou, for their patience, dedication and technical talents throughout the course of my thesis work. I am deeply indebted to them for their professional guidance, encouragement and friendship. They introduced me to the field of Adaptation Services, suggested the topic and supplied the initial work. My discussions with them have been extremely rewarding because of their insight, knowledge and quick grasp of essentials of the problems. Their excellent suggestions, criticisms and their prompt careful reading of early drafts of this thesis were enormously helpful in its clarification and development.

I would also like to thank the other members of my thesis committee, Dr. Jian Wang and Dr. Chi Zhou for their advice and feedback. Also, my special thanks to Mr. Christos Christophi, Mr. Ravi Kailat and Ms. Dingding Wang for their support and help in completing my thesis.

And last but not the least I thank all the people in the lab, and all my friends who have been encouraging and supportive at all times. This research was partially supported by two grants from The National Science Foundation (NSF), contract number ANI-0123950 and CCR-0196557.

ABSTRACT OF THE THESIS

A FRAMEWORK FOR POLICY BASED COORDINATED ADAPTATION IN MOBILE SYSTEMS

by

Saurabh Agrawal

Florida International University, 2004

Miami, Florida

Professor Kia Makki, Major Professor

Adaptation is an important requirement for mobile applications due to the varying levels of resource availability that characterizes mobile environments. However without proper control, multiple applications can each adapt independently in response to a range of different adaptive stimuli, causing conflicts or sub optimal performance. In this thesis we presented a framework, which enables multiple adaptation mechanisms to coexist on one platform. The key component of this framework was the 'Policy Server', which has all the system policies and governs the rules for adaptation. We also simulated our framework and subjected it to various adaptation scenarios to demonstrate the working of the system as a whole. With the help of the simulation it was shown that our framework enables seamless adaptation of multiple applications.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 BACKGROUND

Recent years have witnessed exponential growth in the field of computer networks and systems, particularly in mobile computing environment. Modern networks are extremely complex, varying both statically and dynamically. This complexity and dynamism are greatly increased when the network contains mobile elements [4]. The mobile environment also introduces another complication; heterogeneity in the communicating devices. Cell phones, personal digital assistants, palmtop computers, digital pagers, digital cameras and portable computers all have different capabilities and different requirements [10].

It is desirable that these devices receive the best quality of service at the same time utilizing their limited resources efficiently. To enable this the devices must adapt to the changing network conditions and contextual information. Mobile systems must be able to detect their transmission environment and exploit knowledge about its current situation to improve the quality of communications.

1.1.1 NEED FOR ADAPTATION

As seen from the tables 1 and 2 modern computing devices and networks vary dramatically. Bandwidths currently provided by networking hardware in daily use range from a few tens of kilobits per second up to thousands of megabits per second. Similarly, bit error rates of commonly used network devices span orders of magnitude. Latencies

can range from nanoseconds to large fractions of a second. Mobile devices have varying computing capabilities along with several types of screen sizes, resolutions and colors. We also demand far more of our networks than ever before. Not only is the total volume of traffic increasing at an alarming rate, but new applications put new kinds of stipulation on the network. Web browsing, video conferencing, and Internet telephony have very different network requirements than the mature Internet applications like electronic mail and file transfer.

Networks that contain mobile elements tend to experience a wide range of these characteristics, accompanied with rapid changes. In the future it is expected that many billions of tiny embedded devices worldwide will have some networking capabilities. Such high numbers makes any form of static planning or optimization of network operations difficult to imagine.

Other issues such as security, accessibility and quality of service, also complicate the problem. The existing networking infrastructure that we have inherited was not designed with commercial use in mind; as a result, performing efficient, safe business transactions over that network infrastructure using the modern devices is challenging.

Moreover, the existing network protocols that have enabled the Internet revolution are not perfectly suited to the environment they themselves have created. TCP has its own shortcomings, for example, it does not work well on noisy links (e.g., many wireless links) and often behaves poorly over satellite links due to long latencies.

Researchers have altered some protocols to handle such problems, but it is highly impossible to guarantee a design of network infrastructure or protocols that will function well in all probable network conditions. Even if we could develop such protocols, we

would face the challenge of converting the enormous installed base of today's network infrastructure. The Internet is distributed, decentralized, ever expanding, vast, therefore, an intuitive solution of complete replacement of that existing infrastructure is scary. But still there is no guarantee that even the deployment of a complete replacement will eliminate all the problems.

It is imperative to deal with larger, more variable, more complex, rapidly growing networks that must meet ever increasing demands, yet rely largely on existing networks and protocols.

Hence making the applications adapt to the changing network conditions seems to be a very viable solution. Such a solution allows hardware or software to alter the protocols or the data content being transmitted to provide a better quality of service to users and efficient functioning of the devices and the network.

1.1.2 TYPES OF ADAPTATION

Over the past several years numerous researchers have come up with different ideas and architectures to support network and application adaptation. Some have even developed prototypes and models to support their claims. In general the flow of data over a network could be adapted in a number of different ways. The essential protocol can be adapted to handle difficult situations like the Berkeley snoop protocol. Data can be altered in a loss less way. Various systems allow compression or encryption across links with poor connectivity, with the involvement of applications. Lossy adaptation could be used to obtain data faster over a congested link. Data could be converted to formats better suited to the end systems or the transitional network.

The general division of adaptive solutions to network problems attracts a very large number of distinct and interesting variations. After a close examination of a broad spectrum of adaptation solutions, we can classify any adaptive machinery to fall under one or more of the following genres.

Application Transparent vs. Application aware: In some cases application could be conscious that adaptation is taking place and perhaps is expected to offer an application level response, as oppose to the situation in which the underlying system completely screens the adaptation procedure from it.

General vs. Application–specific adaptation: The system could be able to support adaptation for all the application and assist in efficient operation of the system as a whole. On the other hand adaptation could help only in optimizing a specific application or a targeted narrowly defined class of applications.

Location of adaptation machinery: There could be possibly three places where adaptation mechanism could reside. It could be located on the client side where all the data is received normally and then altered for efficient presentation. On the contrary the adaptation mechanism could be residing on the server and depending on the clients, data could be altered to enable better quality of service. The third spot where adaptation mechanism could reside is in the middle tier. In this case neither the client is altered the server, but there could be an inclusion of a proxy in the middle, which handles adaptation. In some cases the adaptation machinery could be partially located on two or more of these places.

1.2 OBJECTIVE OF THESIS

Many researchers have developed various adaptation systems over the past years. Many of which are being used to optimize system performance. Sometimes it is necessary to deploy more than one adaptation mechanism on a system. In such cases there could be a conflict between different adaptation mechanisms leading to sub-optimal system performance. In order for adaptation to take place without any clashes amongst its peers it is necessary to govern the process of adaptation with certain rules or policies.

[11] Rightly identifies the need for a policy driven architecture to render cooperative adaptation when multiple adaptation mechanisms coexist on a system. The validity of this point is illustrated with a system that consists of two different kinds of independent adaptation mechanisms, power adaptation mechanism to reduce the system's power consumption and the network adaptation mechanism. To overcome this issue [11] proposes a policy language based on event calculus [13].

In our research on this topic we discovered that the system model used in [11] to address the problem of conflicting adaptation is not powerful enough to service all the complex situations that could arise in such conditions. In this thesis we identify the drawbacks of the current system models used for the purpose of adaptation and then suggest a framework for policy based coordinated adaptation in mobile systems. While designing a framework for mobile systems, it is important that the mobile devices are subjected to the least amount of overhead. Hence mobility and thin clients are given utmost significance.

1.3 OVERVIEW OF APPROACH

In order to formulate an adaptive framework it is first necessary to identify our system requirements. We start our research by studying the different adaptive systems and logic formulation languages and identifying the pros and cons of each of them. We then present our system design giving prime importance to mobility and thin clients. All the necessary system components are then studied along with their interactions with one another. We support our claim by presenting certain scenarios in which the previous approaches fail, but our framework succeeds. We demonstrated the working of the framework by simulating an adaptation scenario and subjecting it to various adaptive stimuli.

1.4 ORGANIZATION OF CHAPTERS

The rest of the thesis is organized as follows. Chapter 2 consists of the previous work done on adaptation services. Chapter 3 shows the detailed design of the framework and algorithms used by its various components. Chapter 4 demonstrates the framework simulation software and test cases. Chapter 5 discusses the future work and concludes the study.

CHAPTER II

RELATED WORK

2.1 INTRODUCTION

Over the past fifteen years a lot of research has been done on adaptation services directly or indirectly. The need for adaptation was felt not only with the introduction of mobile computing but much before that. In this section we survey previous research done on this topic and also discuss some of the existing adaptive and context aware systems.

2.2 ADAPTIVE SYSTEMS

2.2.1 TRANSEND

University of California Berkeley's TranSend Web accelerator proxy [9] was one of the earliest projects to investigate adaptation proxies aggressively. TranSend captures HTTP requests from standard Web clients and applies datatype-specific lossy compression whenever possible, for example, images can be scaled down in quality to reduce the file size, long HTML pages can be broken up into a series of short pages, etc. TranSend's was developed to provide network adaptation for users of slow network links, such as UC Berkeley's dial-up modems or the Metricom Ricochet service [33].

TranSend supports a wireless vertical handoff mechanism [34]. When a client having several wireless interfaces switches between wireless networks, the client-side vertical handoff software (which is absolutely independent of TranSend) produces a notification packet enclosing some essential information (e.g., estimated expected

throughput) of the new network. This packet is then sent to a special UDP port on TranSend where the notification is processed and stored in the profile of that particular client. TranSend would then process future requests from that client in accordance with the new network type, for example, very aggressive image down sampling was performed for clients connecting over Ricochet with an expected throughput of 15-25 Kb/s, whereas compression was much less aggressive (and in some cases disabled) for WaveLAN clients connecting at about 1 Mb/s.

TranSend eventually evolved into a general system for deploying adaptive applications in a scalable and fault-tolerant fashion [10]. One such application, Top Gun Wingman [31], provides the ability to browse the Web from extremely thin clients such as the USR PalmPilot handheld device. Although built in the line of TranSend, Wingman provided an additional service know as a network adapter. TranSend uses HTTP to communicate with clients and servers, but the PalmPilot's limited capabilities advocate a simpler and a less chatty protocol. A simple datagram-based client-to-adapter protocol was crafted for Wingman, which also encapsulates security and encryption. Wingman's proxy-side adapter provides translation between this protocol and HTTP, giving Wingman the ability to access existing Web servers. When Wingman evolved into a PalmPilot implementation of the shared whiteboard [32], the network adapter was augmented to tunnel multicast to the PalmPilot over a unicast TCP connection, to compensate for the PalmPilot's inability to handle multicast directly, which is nothing but network adaptation.

2.2.2 ODYSSEY

Odyssey is a system built at Carnegie Mellon University to support challenging network applications on portable computers [3]. A particular focus of Odyssey is resource management for multiple applications running on the same machine. Odyssey was designed primarily to run in wireless environments characterized by changing and frequently limited bandwidth, but the model is sufficiently general to handle many other kinds of challenging resource management issues, such as battery power or cache space.

The goal of the system is to provide all applications on a portable machine with the best quality of service, consistent with available resources and the needs of other applications. Odyssey is an application-aware approach to adaptation intended primarily to assist client/server interactions. The Odyssey system consists of a viceroy, an operating system entity in charge of managing the limited resources for multiple processes; a set of data type-specific wardens that handle the intercommunications between clients and servers and applications that negotiate with Odyssey to receive the best level of service available.

Applications request the resources they need from Odyssey, specifying a window of tolerance required to operate in a desired manner. If resources within that window are currently available, the request is granted and the client application is connected to its server through the appropriate warden for the data type to be transmitted. Wardens can handle issues like caching or pre-fetching in manners specific to their data type to make best use of the available resource. If resources within the requested window are not available, then the application is notified and has a choice of requesting a lower window of tolerance and corresponding level of service. As conditions change and previously

satisfied requests can no longer be met or conditions improve radically, the viceroy uses upcalls registered by the applications to notify them that they must operate in a different window of tolerance, causing them to alter their behavior.

2.2.3 CONDUCTOR

The UCLA Conductor system allows deployment of cooperating adaptive agents at specially enabled nodes throughout a network [7]. Conductor is an application-transparent adaptation mechanism. Conductor doesn't require any recoding of applications, nor does the applications have to request its services explicitly. Instead, the underlying system is configured to specify what kinds of data flows Conductor is capable of assisting, and the Conductor system automatically traps and adapts those data flows.

Conductor also handles issues of arranging adaptations in support of a single flow at multiple nodes. Conductor determines the characteristics of the data path from source to destination and determines if the path will meet the needs of the applications using it. If not, Conductor will automatically deploy adapters at one or several of the available nodes along the path to adapt the data flow to network conditions, allowing better application-visible network behavior.

Conductor plans the cooperative behavior of the agents and handles problems of transient or long-term failure of particular adapter nodes. Conductor is designed to handle general-purpose adaptations, including both lossy and lossless adaptations. Handling lossy adaptations is especially challenging when trying to provide reliable behavior, because a lossy adapter may drop a part of the data or may shrink several data packets into fewer packets. If an adapter or its node fails, some of the adapted packets could be

delivered while others could not. Without the lossy adapter's state to determine which original packets were dropped or coalesced, the system may find it difficult to resume transmission without either duplicating information that has already been received or failing to deliver required information. Unaware applications are generally unprepared for either problem, so Conductor hides these problems from such applications. Conductor attaches numbers to pieces of semantic content that do not vary when adapted. For example, if every other packet is dropped, the undropped packets are renumbered to include the dropped packets. This scheme allows the system to determine which information has and has not been delivered despite failures.

2.2.4 SMILEY

Smiley is an intelligent agent real-time program developed at University of California Los Angeles to enhance Web browsers [35]. It has two main components:

1. A dynamic Graphical User Interface that hints users regarding the nature of the links they may choose to select on a web page.
2. A transparent agent that prefetches carefully selected links.

The GUI provides users a measure of the quality of connectivity available between themselves and the servers they contact to obtain web pages [35], and of the nature of the data residing behind those links. It was designed especially to handle the types of limited links that are encountered frequently in mobile computing, but also to handle general connectivity and bandwidth problems in the overall network. Smiley's GUI provides user feedback, in the form of augmentations to the links shown on a web page, that allow the user to predict the likely effect of clicking on a particular link. This

feature allows a user to avoid requesting a page that is unavailable or will take a long time to retrieve. Moreover, Smiley prefetches web pages intelligently to allow users to browse more effectively over limited and variable links. A prefetch threshold algorithm is used to decide when it is worthwhile to prefetch a web page the user hasn't yet asked for. Smiley includes models that consider different users associated with different time and bandwidth costs, trying to minimize the average cost for each request in the entire system.

2.2.5 RUTGERS ENVIRONMENTAL AWARE API

To cope with the sudden changes in resource availability, applications must augment any underlying adaptation capabilities by robustly continuing to work, though at a lower commitment. Such application adaptation implies that applications must be designed to receive notifications about any important changes in the environmental state and to react appropriately. The complex changes in the network environment requires that the method of interacting with adaptive applications deal with the possibility of a large number of environmental conditions, sources and probable reactions.

The Rutgers Environment Aware API addresses this problem. This API is based on a flexible mechanism for asynchronous event delivery. Environmental changes are modeled as asynchronous events that are delivered to mobile computing applications over an entity called Event Channel. This entity implements the event delivery mechanism. The events are organized as an extensible type hierarchy, and the architecture itself can be configured and extended. This extensibility enables support for a new condition to be easily incorporated into an existing system. A novel feature of the API is its ability to

utilize event type information not only to filter out uninteresting events, but also to handle an event at an appropriate level of abstraction. An application that chooses to be environmentally aware creates a handler for that event type. The application specific response to the new situation is encoded in this handler and is invoked when the appropriate event is delivered.

2.3 ADAPTIVE CONTEXT AWARE SYSTEMS

Context-aware computing is a mobile computing paradigm in which applications can discover and take advantage of contextual information (such as user location, time of day, nearby people and devices, and user activity)[18]. There is a strong interdependence and similarity between adaptive systems and context aware systems. The design and implementation of both these types of systems face similar obstacles. Hence in order to make our policy formulation flexible enough to fit in even context aware applications it is important to study and analyze these systems also. Following are few examples of some implementations of context aware systems.

2.3.1 SHOPPING ASSISTANT

This application was developed by AT&T Bell Laboratories [19] to provide store customers a better shopping experience. The context in this case is the Customer's location within the store. The device can guide the shoppers through the store, provide details of items, help locate items, point out items on sale, do a comparative price analysis, and so forth. There is a privacy concern since the store maintains the customer profiles. As a consequence, customers are divided into two classes: regular customers

who shop anonymously without profiles in store, and store customers who signed up with a store and will get additional discounts in exchange for sacrificing their privacy.

2.3.2 LOCATION BASED REMINDERS

The ComMotion project [20] at MIT media laboratory makes use of both location and time context. ComMotion creates reminder message with every location, and when the intended recipient disembarks at that location, the message is delivered via voice synthesis without requiring user to hold the device and read the message on screen. The Rome project at Stanford also demonstrates a similar reminder application [21]. CybreMinder [22] at Georgia Tech augments the reminder tool with more complex context, such as nearby people, and current weather conditions. Researchers at TecO went further, and built a device called MemoClip [23] to support this kind of active reminder application.

2.3.3 CONFERENCE ASSISTANT

Future Computing Environments (FCE) developed the Conference Assistant at the Georgia Institute of Technology. The assistant uses a variety of context information to help conference attendees [27]. The assistant examines the conference schedule, topics of presentations, user's location, and user's research interests to suggest the presentations to attend. Whenever the user enters a presentation room, the Conference Assistant automatically displays the name of the presenter, the title of the presentation, and other related information. Available audio and video equipment automatically record the slides of current presentation, comments, and questions for later retrieval.

2.3.4 OFFICE ASSISTANT

The office assistant [28] developed by MIT Media Laboratory is an agent that interacts with guests at the office door and administers the office owner's schedule. The assistant is activated when a visitor approaches, which is sensed by two pneumatic mats placed on both side of the office door, and it will adapt its behavior to such contextual information as the identity of the visitor, the office owner's schedule status and busy status, and the owner's willingness to see the current visitor. It is intrusive, however, to recognize visitor's id by a name asking process. An alternative approach to this system is the Active Floor [29] or Smart Floor [30].

2.3.5 TELEPORTING

Teleporting (sometimes referred to as "follow me" computing) is a tool developed by Olivetti Research Ltd. (ORL) to dynamically map the user interface onto the resources of the surrounding computer and communication facilities. Also based on the Active Badge system, this tool can track the user location so that the application follows the user while they move around. A new version of the Teleporting system developed at AT&T Laboratories uses a new location tracking system called the Bat3, which is based on both ultrasonic and radio signals [25]. Composite Device Computing Environment (CDCE) [26] is also working on a project, aimed to supplement resource-poor PDA with surrounding computing resources such as PCs, workstations, TVs, and telephones.

2.4 POLICY BASED COORDINATED ADAPTATION

Adaptation is an important requirement for mobile applications due to the varying levels of resource availability that characterizes mobile environments. As we have seen in the previous section a lot of work has been done in this area to augment mobile computing. Very soon it will be common to have more than one type of adaptation mechanisms on a system. However without proper control, multiple applications can each adapt independently in response to a range of different adaptive spurs. As a result of this uncontrolled independent adaptation there could be conflicts in the underlying system, which may result in its sub optimal or questionable performance.

The validity of this point is illustrated with a system that consists of two different kinds of independent adaptation mechanisms, power adaptation mechanism to reduce the system's power consumption and the network adaptation mechanism [11]. Let us consider a scenario when the power adaptation mechanism reduces the use of network interface in order to conserve the system's overall power and hence to increase its uptime. Now this state of the system is sensed as increase in bandwidth by the network adaptation mechanism tempting it increases the use of network interface in order to utilize the excess bandwidth. This results in negating the primary purpose for which adaptation was done. It is clear from this example that there should be some kind of coordination between adaptation mechanisms for the proper functioning of the system. This is could be accomplished by defining policies for the system.

C. Efstratiou et. al. [1][8] were the first to address this problem of multiple adaptation mechanisms coexisting on a system, which they further improved and detailed in [11]. The system is based on based on a policy language derived from the

Event Calculus logic programming formalism [13]. The following section briefly describes their system model as well as identifies some issue that their system fails to address.

2.4.1 AVAILABLE SYSTEM MODEL

[11] Presents a middleware platform that has been designed specifically to provide support for adaptation both within and between mobile applications. Coordinated behavior is achieved on a system-wide level by separating policy from mechanism through the use of a host specific policy-driven adaptation controller. In other words they decouple the adaptive mechanisms supported by each application and their adaptation policies and hence controlling when and how adaptation takes place. All adaptation policies of each host are handled by an adaptation control module based on a policy language derived from the Event Calculus [13].

In order to realize the system requirements outlined above, they design of a platform in, which adaptive mechanisms and policies are decoupled. Furthermore, mechanisms must be 'exposed' or externalized in order to enable coordinated control by an 'adaptation controller'. Figure 1 shows the relationship between the main components of their platform. The functionality of the platform is split into two main areas:

1. The sharing of application state information and their available adaptive mechanisms.
2. The coordination of the behavior of the applications according to the adaptation policies.

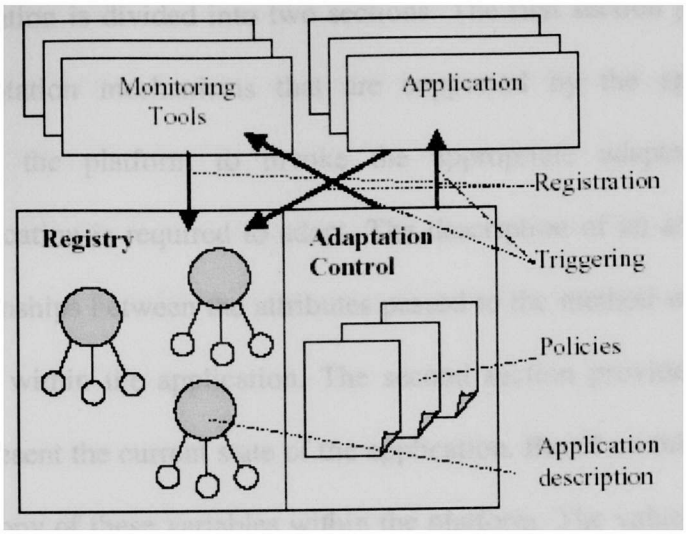

Figure 2.1: Policy Based Coordinated Adaptation

The platform introduces applications and tools that monitor changes in the systems' environment providing a description of their functionality including a set of state attributes containing information on the environment/application and adaptive actions that they can perform. These descriptions as well as a set of state variables that provide information about each application's current state are handled by the platform's registry. The adaptation control is the component that is responsible for making decisions about the appropriate adaptive actions that the applications must take. These decisions are made based on the set of policies exported by the applications or established by the user.

Each adaptive application that is running on a host must register with the platform's registry. The application registry holds a description of each application's available adaptation mechanisms as well as a set of state variables that provide information about each application's current state. The registration information is provided by the application in XML (eXtensible Markup Language). Their structure of the registration information is influenced by the design of the UPnP protocol [36]. The

registration information is divided into two sections. The first section provides a list of the available adaptation mechanisms that are supported by the applications. This description allows the platform to invoke the appropriate adaptation mechanism whenever the application is required to adapt. The description of an adaptation method may specify relationships between the attributes passed to the method when invoked and the state variables within the application. The second section provides a list of 'state variables' that represent the current state of the application. Based on this information the registry creates a copy of these variables within the platform. The values of the variables are updated by the application by sending events to the platform whenever a state variable changes or the application feels the need to change it. This allows the platform to monitor all applications within the system and consider their state before taking adaptation decisions.

They introduce an adaptation control module, which is responsible for monitoring the behavior of all applications on a system and triggering them to adapt according to adaptation policies set with the platform. The behavior of the adaptation control module is controlled by events fired by the applications when the values of their state variables change. These events act as triggers for the adaptation control module in order to apply an adaptive mechanism as described by the adaptation policies. Applications register default policies with the platform that specify their default adaptive conduct. The user could also provide modification to active policies. Part of the adaptation control module's functionality is to provide a dynamic policy evaluator for the adaptation policies.

They use Kowalski's event calculus for their policy language formulation. In our research in this area we discovered that the above system model is very well designed but

there is a significant scope of improvement. This thesis is targeted towards the improvement in policy language, used to model the above system. The following section describes in brief some policy formulations including event calculus as well as identifies their shortcomings. We feel it is essential to study the available logic formulations and identify their drawbacks in order to come up with a policy language that will suit our purpose.

2.5 LOGIC PROGRAMMING

2.5.1 EVENT CALCULUS

The event calculus was introduced by Kowalski and Sergot [13] as a logic programming formalism for reasoning about events and change. It is a formal language for representing and reasoning about dynamic systems. Since its original introduction a number of variations and updates are demonstrated in [37] [38] [39]. In this section we briefly explain the concept of event calculus.

Table 2.1: Event Calculus Predicates

Base Predicates	
Initiates (A,B,T)	Event A initiates fluent B for all time > T
Terminates (A,B,T)	Event B terminates fluent B for all time >T
Happens(A,T)	Even A happens at time point T
HoldsAt(B,T)	Fluent B holds at time T. This predicate is useful for defining static rules
InitiallyTrue (B)	Fluent B is initially true
InitiallyFalse(b)	Fluent B is initially false
Auxiliary Predicates	
Clipped (T1,B,T2)	Fluent B is terminated sometimes between time point T1 and T2
deClipped (T1,B,T2)	Fluent B is initiated sometimes between time point T1 and T2

The event calculus provides a theoretical framework where it is possible to reason about events and their effects in an event-driven system. 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 'bandwidth is low'. When the system under consideration gets into that specific condition the fluent is considered to be valid (it holds). The states of fluents are 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. More specifically the basic predicates and axioms defined in event calculus are shown in Table 2.1.This is a classical form of event calculus where theories are written using Horn clauses.

2.5.2 DRAWBACKS OF EVENT CALCULUS

The main intended applications of Kowalski and Sergot's event calculus [13] were updating databases and narrative understanding. Hence it is inherently incapable of formulating the complex situations that may take place when multiple adaptation mechanisms coexist. For instance the event calculus just takes into consideration events that take place at specific time points and fluents that represent the effects of the events.

The system assumes that if a particular fluent happens, the fluent will be terminated in a predictable fashion and there will be no exception to the final termination of the fluent. This will lead to enormous complexity. As a result trying to combine solutions to all the possible scenarios using event calculus into a single unified formalism

is remarkably complex, even when remaining focused on carefully hand-tailored sample problems. The chances for integrating such complex situations into event calculus and making the network on the whole achieve its final goal seem remote [12].

Apart from this the system proposed by [11] does not consider the effect of other systems present in the network, which may influence its behavior. The policies residing on the system under observation should also be aware of the world (network) it is situated in. It should consider the fact that not only the applications local to the system can influence its' behavior but also the applications in the network or some remote applications can affect it. When such is the scenario it is desirable to have a system, which has at least some provision of security and access control. Even though the event calculus is very efficient, but due to the issues just mentioned above we cannot use it directly to form a policy language based solely on it.

2.5.3 POLICY DESCRIPTION LANGUAGE

The Policy Description Language (PDL) [14] defines policy as a function that maps a series of events into a set of actions. PDL is a simple and expressive language developed to specify policies. The design of PDL has been strongly influenced by the action languages in [41] [42]. PDL is based on the event-condition-action rule of active databases. The PDL assumes that there is an intermediate service present between the policy server and the environment that continuously poles the environment for changes, which are nothing but events and then sends these events to the policy server for appropriate action. PDL takes into account the following 4 situations:

1. A policy must be enforced if two events e_1 and e_2 occur simultaneously.
2. A policy must be enforced if and event e does not occur.
3. A policy must be enforced if an event e_2 immediately follows an event e_1.
4. A policy must be enforced if an event e_2 occurs after an event e_1 occurs.

PDL also introduces a concept of simultaneous events i.e. one or more primitive events occurring at the same time called as epoch.

2.5.4 DRAWBACKS OF POLICY DESCRIPTION LANGUAGE

Even though the PDL seems to be a good option to base our policy language for coordinated adaptation it has certain shortcomings. PDL does not define any entities representing time; it also does not provide any explicit specification of time relationships. The PDL is not designed to handle conflict very well. The policies written in PDL are not prevented from executing conflicting actions. Like event calculus PDL also lacks the security and access control functionality, which could be desirable in adaptive systems.

2.5.5 PONDER POLICY SPECIFICATION LANGUAGE

The Ponder policy specification language [15] defines a full-scale policy management system. It supports the specification of authorization, delegation, information filtering, refrain and obligation policies over system specific policy domains. It identifies the following requirements:

1. Support for security policies for access control, and delegation to cater for temporary transfer of access rights to agents acting on behalf of a client as well as policies to express management activity.

2. Structuring techniques to facilitate the specification of policies relating to large systems with millions of objects. This implies the need for policies relating to collections of objects rather than individual ones.
3. Composite policies which allow the basic security and management policies relating to roles, to organizational units and to specific applications to be grouped. Composite policies are essential to cater for the complexity of policy administration in large enterprise information systems.
4. It must be possible to analyze policies for conflicts and inconsistencies in the specification. In addition it should be possible to determine which policies apply to an object or what objects a particular policy applies to. Declarative languages make such analysis easier.
5. Extensibility is needed to cater for new types of policy that may arise in the future and this can be supported by inheritance in an object-oriented language.
6. The language must be comprehensible and easy to use by policy users.

2.5.6 DRAWBACKS OF PONDER POLICY SPECIFICATION LANGUAGE

Time relationships, as defined in Ponder are quite limited; used mainly for the specification of constraints over the time a policy should be considered active. It does not relate time with events, which is of vital importance when designing a policy language for coordinated adaptation. Ponder is not a logic based language and does not provide direct support for formal reasoning methods or for expressing general models of system behavior. Therefore, Ponder cannot account for the effect of policies on system state and

cannot be used directly for policy analysis [16], which is one of the major requirements in our case. Apart from this Ponder is very detailed and focuses more towards network management and access control aspect of policy formulation.

2.6 ANALYSIS

As seen above all the three policy formulation languages have some or the other drawback and are not suited for our requirements directly. In order to achieve the desired results and overcome the shortcomings of these logical specifications, we present certain criteria, which must be fulfilled while writing policies.

When writing system policies, it is important that it is expressive enough to represent the systems being modeled and that the language is based on solid theoretical foundations. Event calculus is a good starting point for a formal language for specifying policy-based systems because it has direct support for representing the events used in these systems. Moreover, it is a well-researched area of logic programming that supports most modes of logical reasoning and provides a number of theoretical results and tools [16].

An important requirement of any policy management system or policy language is that it should support the specification of authorization, delegation, information filtering, refrain and obligation policies over system specific policy domain. The Ponder policy language [15] defines a full-scale policy management system. But for our requirements Ponder is very detailed and focuses more towards network management and access control aspect of the policy formulation. For this reason we will just make use of the postulates, that we think will suffice in formulating our policy language.

Policies may depend on several events or the lack of certain events happening, or even the events that happened in the past. For coordinated adaptation between several applications it is important that the policy language takes into account the four criterion discussed in Policy Description Language mentioned in the previous chapter.

Moreover, for a policy based system to work properly it should have some mechanism to detect and resolve conflicts. These conflicts may not only arise due to coexistence of multiple adaptation mechanism but also due to the contradicting polices present in the policy description itself. The framework described in this thesis, not only enables the detection and resolution of conflicts but also reasons for the conditions under which conflicts may arise and take a preventive action.

CHAPTER III

FRAMEWORK FOR POLICY BASED ADAPTATION

3.1 APPROACH

In the previous chapter we saw that there are a number of systems available for adaptation. But most of them are either targeted towards a specific application or have some serious issues. In the coming future it is expected to have multiple adaptation mechanisms coexisting on one system, which could result in system conflicts and sub optimal performance, if not handled properly. [11] Suggests a model to over come this problem by introducing policies for adaptation based on event calculus formalism. But their model cannot cope up with the increasing demand of network resources that a mobile environment stipulates. To take care of this problem we think it is necessary to have a framework that will enable the coexistence of multiple adaptation mechanisms on a system in harmony. In this section we present our framework along with the guidelines that are necessary to write policies for such a system that governs the rules for adaptation to occur.

3.2 CASE STUDY

In order to better understand our framework, we will demonstrate its functioning and interaction with other system components with the help of following scenario.

Scenario: There are three applications: Web Browser, Streaming Video and Enterprise Application residing on an adaptive system. Initially all three applications share equal

amount of network bandwidth. The adaptive system can increase the bandwidth of an application upon request if there is availability or if some other application is willing to reduce its assigned quota or is not fully utilizing it. Suddenly the Web Browser and the Streaming Video application feel the need to extend their network operations for which they will require more bandwidth. Both of these applications request the system for some extra bandwidth simultaneously. This situation will be very difficult to model with the existing systems discussed in the previous chapter. But by using the policy framework presented in this thesis the above task seems very simple, intuitive and easy to model.

3.3 OUR SYSTEM ARCHITECTURE

In order to have seamless coordinated adaptation in a system, which could have multiple adaptation mechanisms and applications, we present an adaptation framework. The framework proposed is fully distributed and facilitates feedback among all the components to enable smooth functioning of the system as a whole. The system model is general enough to fit in any kind of existing adaptation mechanisms. It is also designed keeping in mind that in the future the need might arise for stricter security policies concerning access control. In our system the policies are not hard coded into the system itself but they reside on a policy server that is external to the system under consideration. Our model also introduces a new component called the 'Event monitor and log', which proves very helpful in governing the system's performance. Unlike any other system our system has a required amount of intelligence and decision making ability available in all the components. Figure 3.1 shows the schematic of our proposed system model.

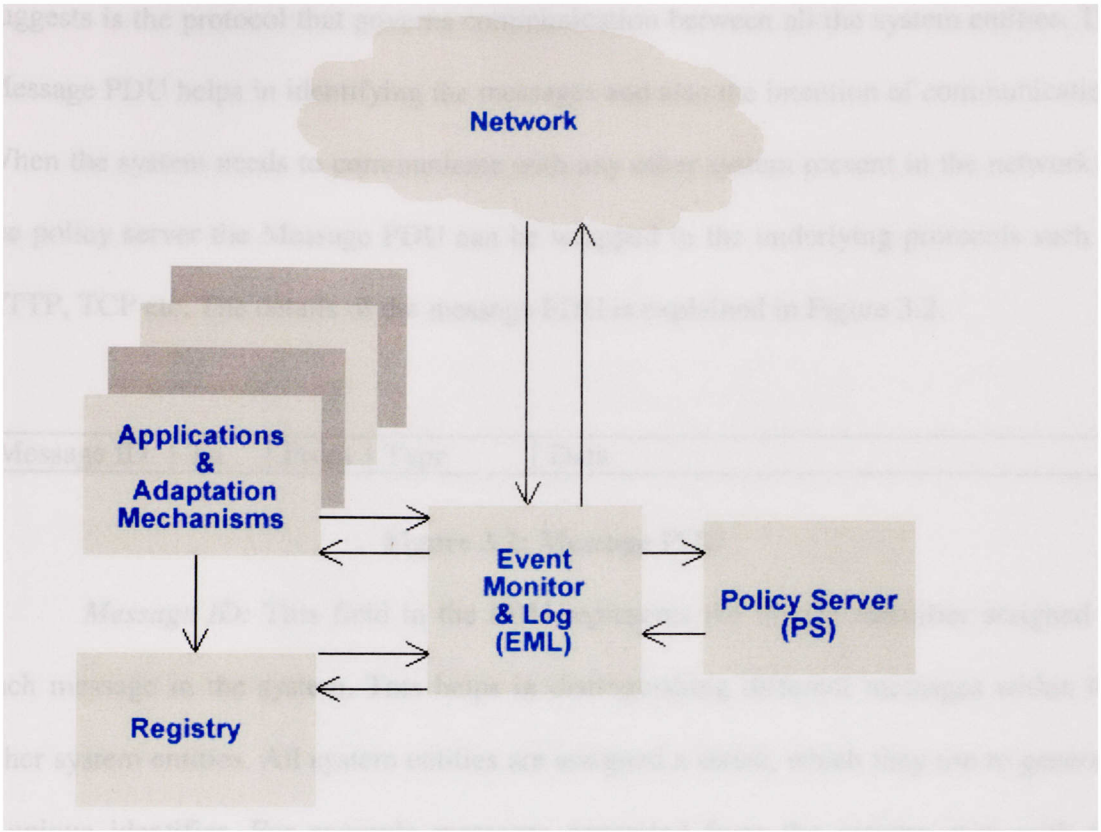

Figure 3.1: Framework for Policy Based Adaptation

From the design overview of the framework we can see that the system is made of five components: Applications, Registry, Event Monitor and Log (EML), Policy Server (PS) and Message PDU. In the following section we will discuss the internal working and the algorithms associated with each of these system components.

3.3.1 MESSAGE PROTOCOL DATA UNIT (PDU)

As we have seen from the design overview that all the system entities continuously communicate with each other. Therefore it is important to have a communication standard or protocol to simplify this task. The Message PDU, as the name

suggests is the protocol that governs communication between all the system entities. The Message PDU helps in identifying the messages and also the intention of communication. When the system needs to communicate with any other system present in the network or the policy server the Message PDU can be wrapped in the underlying protocols such as HTTP, TCP etc. The details of the message PDU is explained in Figure 3.2.

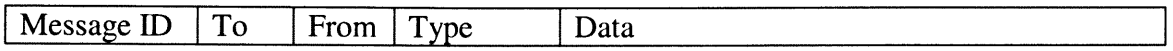

Figure 3.2: Message PDU

Message ID: This field in the PDU represents the unique identifier assigned to each message in the system. This helps in distinguishing different messages within the other system entities. All system entities are assigned a series, which they use to generate a unique identifier. For example messages generated from the registry start with the alphanumeric character 'R' followed by a sequence of numeric characters. This method makes sure that all the system entities generate unique message identifier.

To: This field contains the identity of the receiver to whom the message is addressed. Table 3.1 shows the list of value that it can hold.

Table 3.1: To and From Details

Name	Value
Application	0
Registry	1
EML	2
Policy Server	3
External Network	4

From: This field contains the identity of the. The table above shows that values it can hold.

Type: This field represents the type of message being communicated. Table 3.2 shows the list of value that it can hold.

Table 3.2: Type Details

Name	Value
Registration	0
Request	1
Response	2
Update	3
Query	4
Event	5
Fluent	6
System	7
Action	8
Other	9

Data: With the help of the data field entities send the data related to the type of message being communicated. For example when the message type is registration, the data filed will contain the details of the component registering including the exposed state variables and methods.

Let us consider a scenario in which an application tries to register itself with the system registry. The message being sent in this case will look like the one shown in Figure 3.3.

A77876	1	0	0	(Exposed methods and properties)

Figure 3.3: PDU Example

3.3.2 REGISTRY

The design of the system registry is influenced by [11] but the way it interacts with other system components is completely different than [11]. All applications and adaptation mechanisms running on the system must register on the system registry. When registering themselves the applications and adaptation mechanisms provide the system the list of methods and properties they are willing to expose to the rest of the system. The properties of applications and adaptation mechanisms also contain the information on the their state variables. Whenever a new candidate registers itself on the system registry, the registry notifies about this event to the Event Monitor and Log (EML), which further consults the Policy Server (PS) for the necessary set of actions if any. If there are certain actions to be taken then the PS notifies the EML, which executes the necessary actions and also updates the registry. This kind of feedback between the EML and the Registry keeps the system updated at all times.

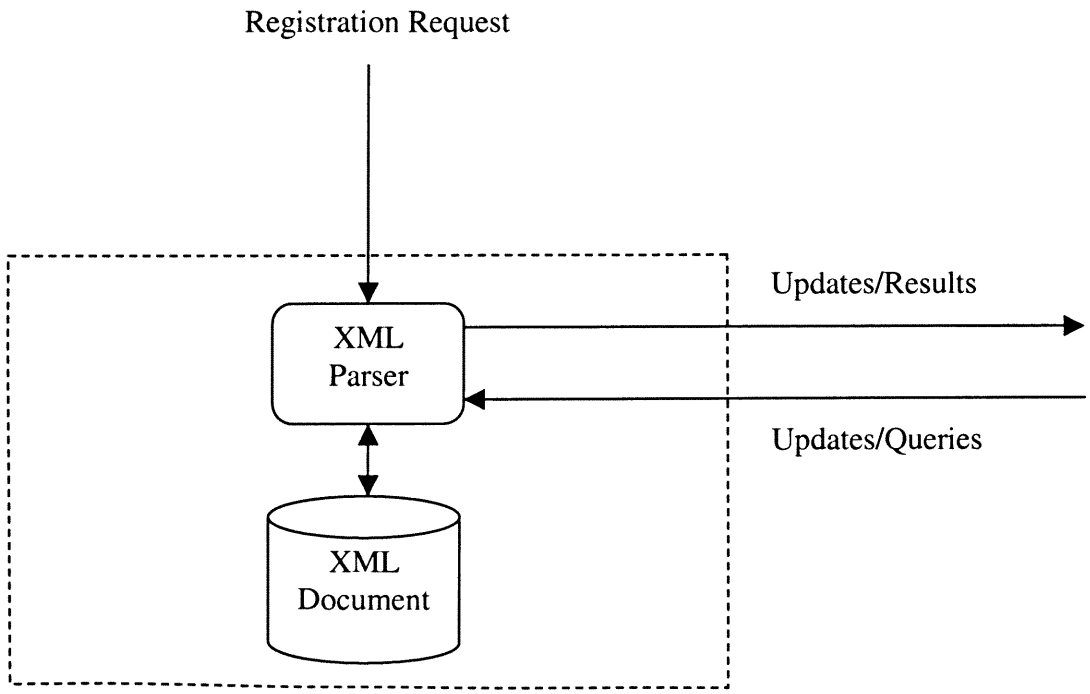

Figure 3.4: Registry Details

The system registry is made of two main components XML parser and XML documents. The XML parser handles the communication aspect of the registry. It is responsible to read and write data from the registry. It also communicates with the EML to notify it about the registration event and also responds to the EML's request messages. Figure 3.4 shows the details of the registry component.

XML Parser: The XML parser is responsible to read and write data to the system registry. It communicated with other system entities such as EML to notify events and the values of state variables. The XML Parser uses the algorithm shown in Figure 3.5 to communicate with the XML Documents, applications and the EML.

```
// Read incoming message
if Message.Type = "Registration" then
      if Application already registered then
            // Message EML about repeat registration attempt
            Exit Sub
      else
            // Write the application data in XML document
            // Notify the EML about new registration
      end if
else if Message.Type = "update" then
      // Update the requested state variables in XML Docs
      // Notify the EML
else if Message.Type = "Request" then
      // Retrieve the requested information form the XML Docs
      // Send the results over to the EML
end if
```

Figure 3.5: XML Parser Algorithm

XML Document: The system registry stores its data in an XML document. The reason we choose XML to store registration information is due to its simplicity of design and accessibility. An application's state variables and the methods are stored as elements in the XML document. This information can be easily accessed with the help of the XML

Parser. Storing registration data in this format keeps the system registry very light, which helps in conserving the limited recourses that characterize mobile devices. Figure 3.6 shows the XML schema that we use for registering applications on our system with the help of an example from the case study described previously.

```
<registry>
  <application id="452">
      <name>app1</name>
      <timeStamp>03/12/2004 17:30:13</timeStamp>
      <property id="0">
        <name>priority</name>
        <value>8</value>
      </property>
      <property id="1">
        <name>type</name>
        <value>application</value>
      </property>
      <property id="2">
        <name>TimeGranularity</name>
        <value>12</value>
      </property>
     <property id="3">
        <name>bandwidth</name>
        <value>56</value>
      </property>
      <method id="0">
        <name>execute</name>
        <parameter />
        <return />
      </method>
      <method id="1">
        <name>stop</name>
        <parameter />
        <return />
      </method>
      <method id="2">
        <name>requestBandwidth</name>
        <parameter>bandwidth</parameter>
        <return />
      </method>
    </application>
  </registry>
```

Figure 3.6: XML Document

3.3.3 EVENT MONITOR AND LOG (EML)

The Event Monitor and Log (EML) is a very important component of the system. It is responsible to keep track of all the events that occur in the system and also has the ability to perform the necessary actions. EML is the only component in the system, which has the liberty to interact with all the entities of the system and in some cases even the entities that are external to the system. Some adaptation decisions could be very critical and could require a history or sequence of events to take the appropriate action. The EML also keeps a log of all the events taking place in the system as well as the actions and the effects they have on the state variables. Whenever an application decides to change its present behavior, it starts by communicating with the EML; the EML then logs the event in its database and also notifies the registry if any state variables are affected. Before the EML takes the action what is being requested by the application it consults the PS for suggestions. The PS processes the request made by the EML (on behalf of the application actually) by looking for any existing polices for that particular scenario. The result of the lookup is notified to the EML, which takes the appropriate actions and also updates its event log and the system registry. The EML is also capable of monitoring events that occur in the environment in which the system is present and provides an interface of the system to the outside world. This outside interface could also be used to sense context information generated by a context server, present in the underlying network if any. This context sensing mechanism leaves a room for scalability and extensibility of the applications residing on the system.

The EML consists of three elements: Message Handler, Binary Converter and Event Log in the form of a binary file. Figure 3.7 shows details of the EML.

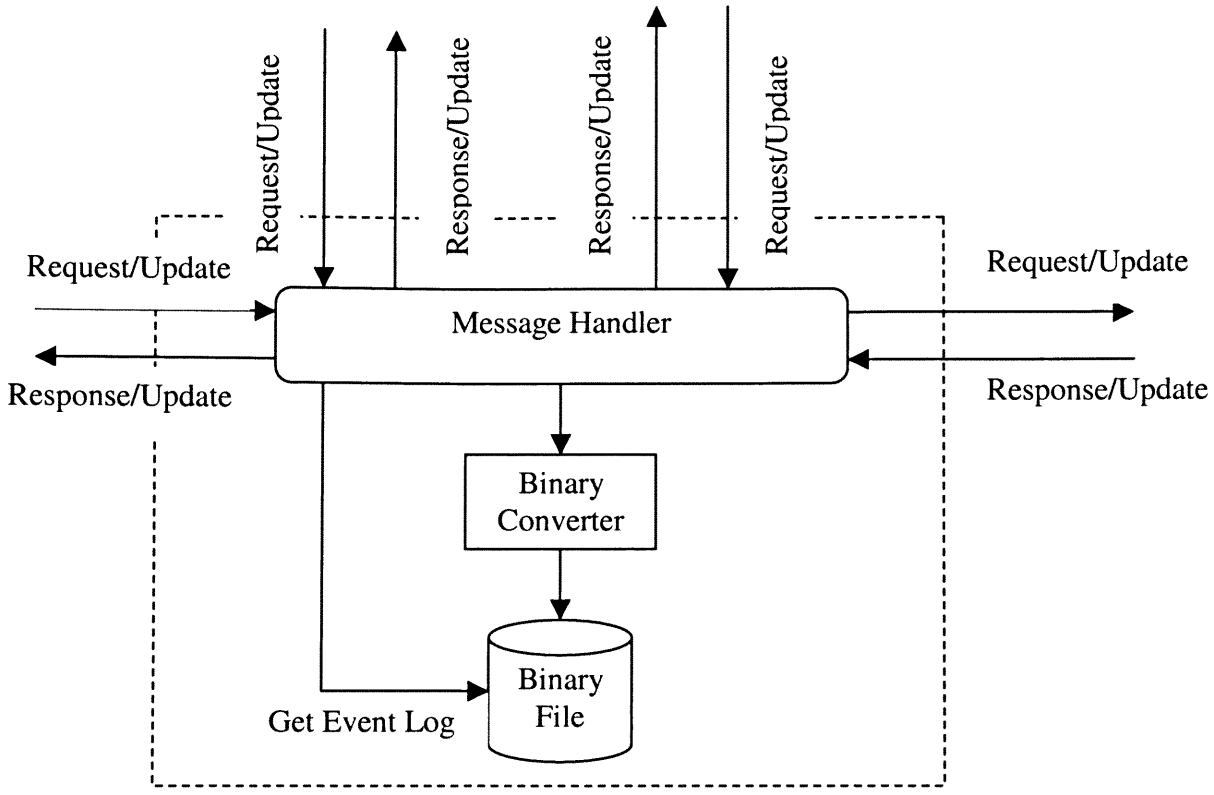

Figure 3.7: EML Details

Message Handler: All the in-bound and out-bound communication to the EML passes through the Message Handler. It is responsible for communicating with the Policy Server to get system polices and executing the appropriate action. The Message Handler also documents all the system events in the event log. After a predefined interval the Message Handler sends the event log file to the Policy Server for data analysis and processing. The Message Handler works on the algorithm shown in the Figure 3.8.

```
// Read incoming message
if Message.From = "Policy Server" then
      if Message.Type = "Request" then
      // Call Procedure GetAppInfo by passing
      // Message.Data as a parameter to it
      // Call Procedure WriteLog
      else if Message.Type = "Response" or Message.Type = _
      "Action" or Message.Type = "Update" then
            // Call Procedure ExecuteAction by passing
            // Message.Data as a parameter to it
            // Call Procedure WriteLog
      end if
else if Message.From = "Application" or Message.From = _
"External Network" or Message.From = "Registry" then
      // Call Procedure GetPolicy by passing
      // Message.Data as a parameter to it
      // Call Procedure WriteLog
end if

Procedure SendLog()
      // This procedure sends the event log file to the PS
      // after a fixed predetermined interval of time
      // for analysis and processing of events
end Procedure

Procedure GetAppInfo()
      // This procedure gets the state information
      // of applications from the system registry
end Procedure

Procedure ExecuteAction()
      // This procedure performs the required action
      // which may range from communicating with the
      // applications to setting the values of state variables
end Procedure

Procedure GetPolicy()
      // This procedure checks with the Policy Server
      // for any policies present for the current state of
      // events
end Procedure

Procedure WriteLog()
      // This procedure helps the EML to document all the
      // events happening in the system. With the help of this
      // procedure the Message Handler passes the event
      // information to the Binary converter which writes the
      // event in the event log in the form of a binary file
end Procedure
```

Figure 3.8: Message Handler Algorithm

Binary Converter: The Binary Converter is responsible to convert the incoming event messages from the Message Handler into binary format, which is then logged in the event log. The reason we store the event log in binary format is due to the size of a binary file and the inherent interoperability.

Binary File: This file contains the details of all the events that occurred in the system. The message Handler sends this Binary File over to the Policy Server after every fixed interval of time or on-demand for further analysis and processing. Figure 3.9 shows the data fields that are logged in this Binary File, which are self explanatory.

Event ID	Event Type	Event Source	Time	Date	Detail

Figure 3.9: Binary File

3.3.4 POLICY SERVER (PS)

Policy Server is the place where the system policies reside. The PS interacts with the EML guiding the operation of the EML and hence the entire system. The isolation of PS from the rest of the system plays a significant role in the overall performance of the system. This helps in future modification and updates of the system policies. The PS could also be centrally managed and located external to the system in the underlying network for other network management operations. It can also be integrated to the existing network management system only with a few minor changes to accommodate the adaptations policies. The Policy Server has four main components: Policy Handler, Compiler, Policy Modules and Policies. Figure 3.10 shows the details of Policy Server.

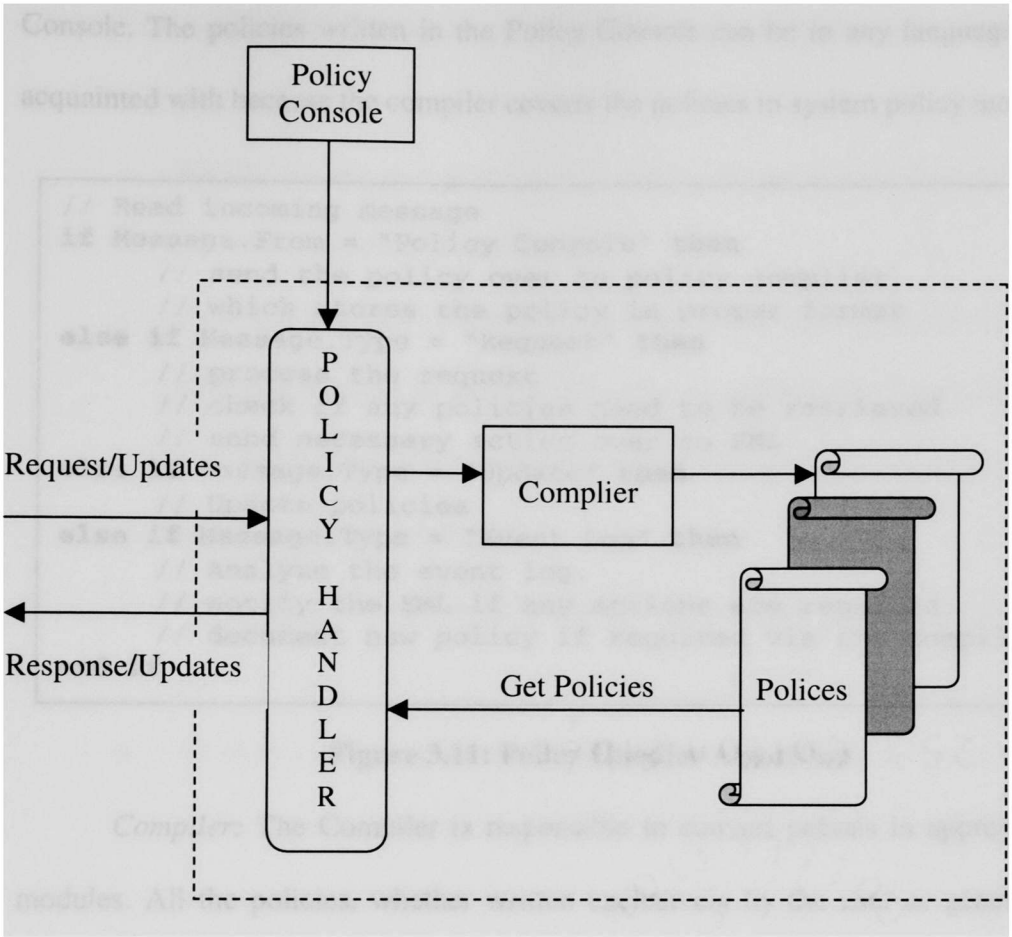

Figure 3.10: Policy Server Details

Policy Handler: The Policy Handler is responsible for all the in-bound and out-bound communication with the Policy Server. It analyzes polices and guides the EML for the appropriate action. It also processes the periodic event log obtained from the EML to update system polices and take proper actions if required. The Policy Handler works on the algorithm shown in Figure 3.11.

Policy Console: The Policy Console is the user interface supplied with the policy server to write, edit and read system policies. Any changes to the existing policies or addition of new policies are written in the policy server with the help of the Policy

Console. The policies written in the Policy Console can be in any language the user is acquainted with because the compiler coverts the policies in system policy modules.

```
// Read incoming message
if Message.From = "Policy Console" then
      // send the policy over to policy complier
      // which stores the policy in proper format
else if Message.Type = "Request" then
      // process the request
      // check if any policies need to be retrieved
      // send necessary action over to EML
else if Message.Type = "Update" then
      // Update policies
else if Message.Type = "Event Log" then
      // Analyze the event log.
      // notify the EML if any actions are required
      // document new policy if required via the compiler
end if
```

Figure 3.11: Policy Handler Algorithm

Compiler: The Compiler is responsible to convert polices in appropriate policy modules. All the policies, whether written exclusively by the user or generated by the Policy Handler (After processing the event log) pass through the compiler for documentation.

Policies: Polices are stored as modules in the Policy Server. All policy objects have a mapping (one-to-one, one-to-many, many-to-many) to the applications or system entities they influence. All policies have a unique identifier, which helps the Policy Handler to identify and retrieve the appropriate policy.

The language used for writing system policies must be based on object-oriented programming foundation. We have not included any discussion on the concepts of object oriented programming as it is out of scope for this thesis. It is assumed that the reader has sufficient background on computer programming and is well versed with the

fundamentals of object-oriented programming. Guidelines for writing system policies are the following:

1. All the entities in a given system should be treated as objects belonging to one or more classes (inheritance);
2. Policies are separated from the implementation of a system permitting it to be modified in order to dynamically alter the strategy for managing the system and hence modifying the behavior of the system, without changing its underlying implementation;
3. An interface to an object is provided by the means of methods and properties; and
4. All objects have the following mandatory properties and method. In addition to the mandatory properties and methods the objects can have user defined properties and methods also.

Mandatory Properties are the following:

1. *Priority:* It is an integer value ranging from 1 to 10 to determine the priority of an object in case of simultaneous events caused by the peers or contradicting objects, where 1 being the lowest priority and 10 being the highest. This property proves very useful to resolve system conflicts;
2. *Type:* This describes the type of object, which could be application, adaptation mechanism, event, fluent, or any user defined type. This property also helps in determining the methods exposed by its parent object; and
3. *TimeGranularity:* In order to handle simultaneous events, it is first necessary to have a clear-cut definition of the time granularity of the system under

consideration. For one system, events could be considered simultaneous if they occur at the same exact infinitesimally small instant of time and for the other the events could be simultaneous if they occur within a time frame of ten minutes. Hence it becomes important to define the time granularity of a system before one writes the system's policies. Even though this declaration is very important, surprisingly is not talked about much. Therefore, TimeGranularity is one of the mandatory properties of all the objects present in the system.

Mandatory Methods are the following:

1) *Execute:* This method triggers the execution of the object; and
2) *Stop:* This method stops the execution of the object.

The Policy Module could be coded and implemented in any of the modern object oriented languages such as JAVA, C++, Visual Basic etc. Figure 3.12 shows an example policy for the case study discussed previously. In the following example it is assumed that the application objects have a property 'bandwidth' and a method 'RequestBandwidth' in addition to the mandatory properties and methods.

```
If (appA.RequestBandwidth AND appB.RequestBandwidth AND
appC.RequestBandwidth) then
      If (appA.Priority = appB.Priority = appC.Priority)
      then
      //Equally divide the available bandwidth among the
      //three applications
      Else
      //Grant the bandwidth requested to the
      //applications in the ascending order of their
      //priority values
      End if
End if
```

Figure 3.12: Policy module example

CHAPTER IV

SIMULATION OF THE FRAMEWORK

4.1 SCENARIO AND ASSUMPTIONS

In order to prove the feasibility of our framework we simulated the case study discussed in Chapter 3. The programming language used to write the simulation software was Microsoft Visual Basic 6.0. Most of the components of this simulation software were written keeping in mind reusability for future implementation. The only simulated parts in this software are the adaptive applications.

We demonstrate the framework, starting with registration of applications on the system followed by their interactions with the system components. We also show the working of the EML and the role of Policy Server in governing the behavior of the system. Our simulated framework consists of two parts: Mobile Client and Policy Server, which reside on separate machines to demonstrate the distributed nature of the framework. The protocol used for communication between the two machines was TCP/IP and the machines were networked using wireless LAN technology 802.11b.

4.2 MOBILE CLIENT

The Simulated mobile client consisted of three applications namely the Web Browser, Streaming Video and the Enterprise Application. Figure 4.1 show a snapshot of the graphical user interface (GUI) of the mobile client. As seen from the snapshot the mobile client has a text box that displays the system registry (fetched form an XML file) and another test box that displays the event log of the system. User can generate adaptive

stimuli by clicking on the "Request Bandwidth" button listed under each application. On clicking on this button user is prompted to enter the desired bandwidth of operation. All communications to, from and within the mobile client were handled by the EML exactly as proposed in the framework.

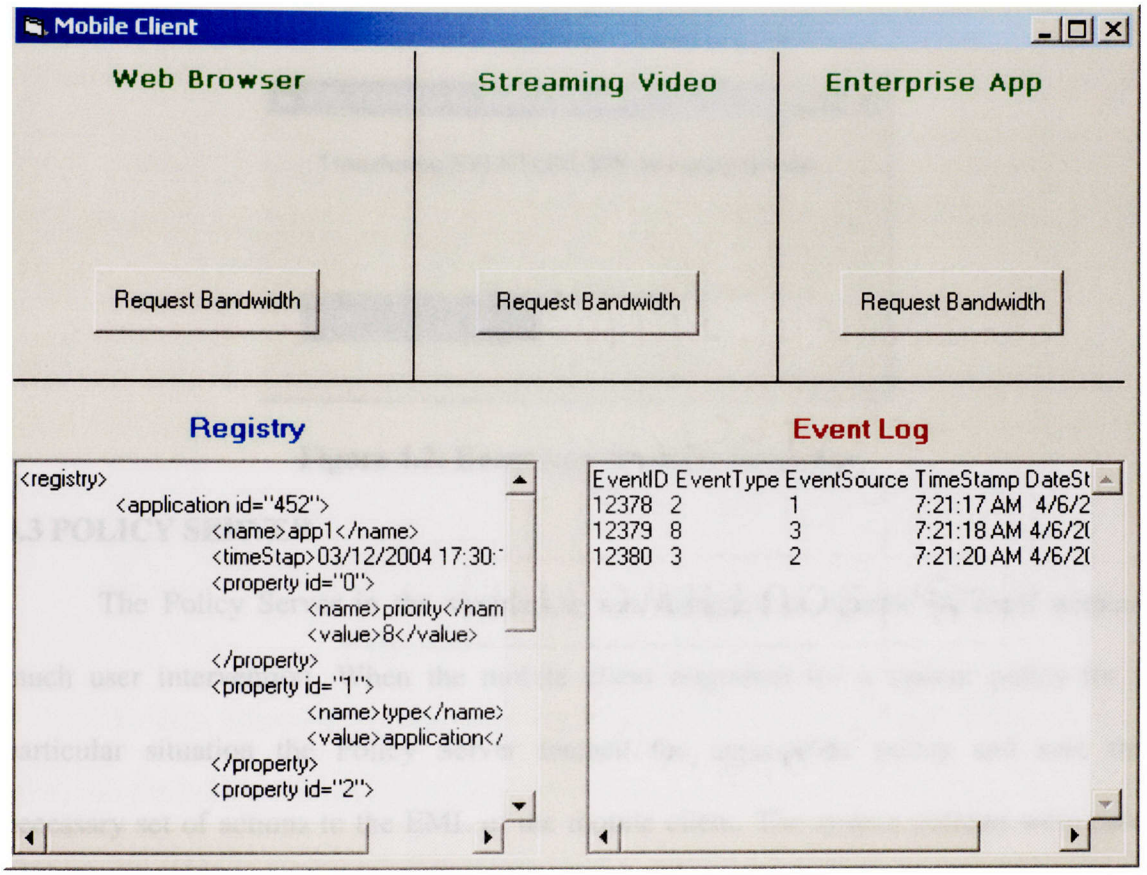

Figure 4.1: Mobile Client Snapshot

4.2.1 EVENT LOG FILE

The EML logs all the system events occurring on the mobile client and writes them to a binary file. This operation of logging events in a binary file was implemented exactly like it were to work in a real system as compared to a simulation. The use of

binary data storage while implementing the event log proved robust as predicted by the framework proposal. Figure 4.2 shows the GUI of the binary event log being transferred from the mobile client to the Policy Server. This event log is received by the Policy Server and then processed by the event log analyzer present in it, to determine if any actions are needed.

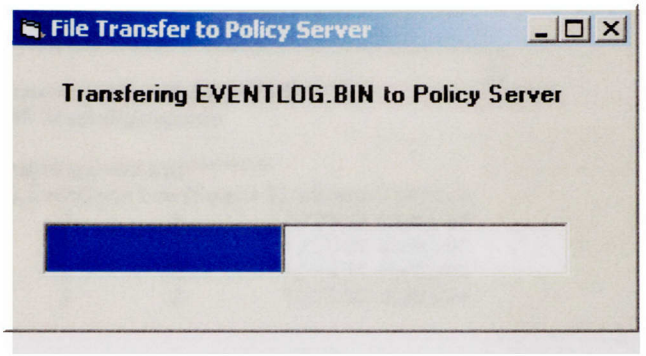

Figure 4.2: Event Log Transfer Snapshot

4.3 POLICY SERVER

The Policy Server in the simulation was designed to operate by itself without much user intervention. When the mobile client requested for a system policy for a particular situation the Policy Server fetched the appropriate policy and sent the necessary set of actions to the EML of the mobile client. The system policies were hard coded in the system by the user, and were programmatically modifiable. The Policy server, when implemented for a fully functional system need not have a fancy GUI cause it should be designed to function in the background. The only GUI required is the one to manage it, which includes a Policy Console to edit existing system policies or to write new ones. But in order to demonstrate the complete functioning of the framework, in our

simulation we developed a GUI for the Policy Server to reflect its internal working. Figure 4.3 shows a snapshot of the GUI for the Policy Server.

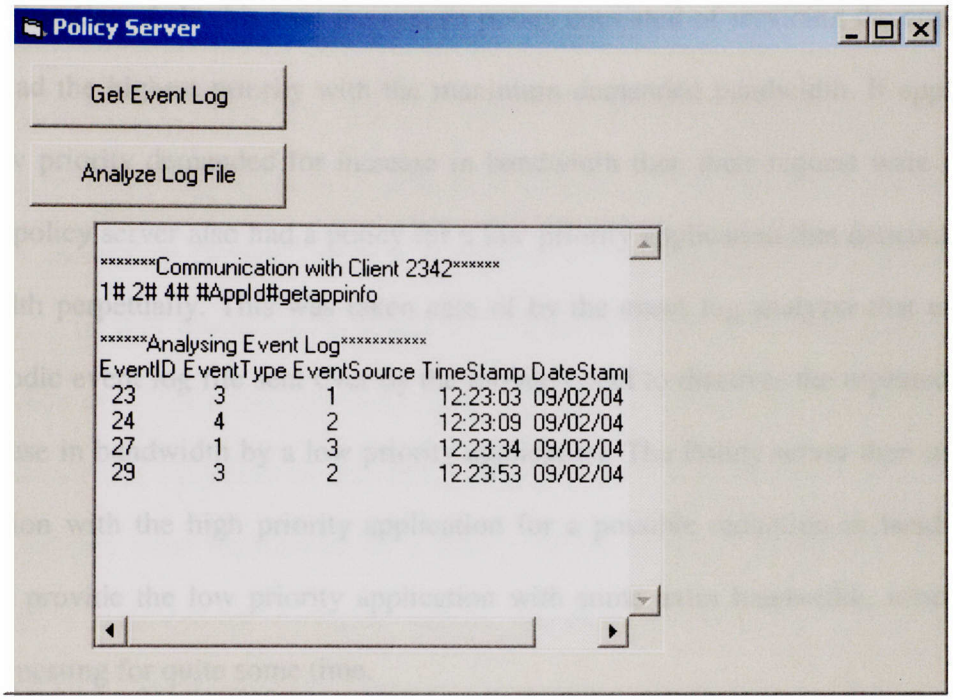

Figure 4.3: Policy Server Snapshot

We also implemented the event log analyzer for our case study. When the user clicks on the button "Analyze Log File", the event log analyzer processes the binary file sent by the mobile client and draws some useful conclusions based on the algorithm in the policy handler about the system and also takes the necessary action.

4.4 TEST CASES

Having simulated our framework we then subjected it to various kinds of adaptive stimuli to demonstrate its complete working. As shown in Chapter 3 all the applications have the common state variables: Priority and Bandwidth. To test the framework we

initially registered all the applications with different priorities and equal bandwidths with the system registry. We then observed the working of the framework by changing system policies for different test cases.

Test Case 1: In this case the system policy consisted of servicing the application, which had the highest priority with the maximum demanded bandwidth. If applications with low priority demanded for increase in bandwidth then their request were rejected. But the policy server also had a policy for a low priority application that demanded more bandwidth perpetually. This was taken care of by the event log analyzer that examined the periodic event log file sent over by the mobile client to discover the repeated request of increase in bandwidth by a low priority application. The Policy server then initiated a negotiation with the high priority application for a possible reduction in bandwidth in order to provide the low priority application with some extra bandwidth, which it had been requesting for quite some time.

Test Case 2: In this case the system policy consisted of serving the applications on first come first served bases. The policies were written to grant bandwidth to applications upon request depending on the availability. The state variable 'priority' was not taken into consideration by the system policies in this test case.

There is a number of combinations possible when writing system policies, but for the efficient functioning of the system it is very important to have a clear definition of system policies and the desired objective.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

As more and more new types of mobile devices are introduced in the market, there will be a very high demand of adaptation services in the coming future. To cope up with this increasing demand there is a requirement of a truly distributed adaptation framework. In this thesis we designed an adaptation framework that ensures system performance even when multiple adaptation mechanisms coexist. In this framework the mobile device was subjected to, bear the minimum processing and data storage overhead. All the policies were maintained on a Policy Server, which is external to the mobile device and is dedicated for managing policies. We also introduced an Event Monitor and Log (EML) in our framework, which is responsible for logging system events and sending them to the Policy Server for further analysis and processing. None of the previous approaches make use of system events in this manner. We demonstrated the working of the framework by simulating an adaptation scenario and subjecting it to various system conditions. With the help of the simulation we can conclude that the framework is well suited for adaptation.

5.2 FUTURE WORK

In this thesis we presented a framework for policy based coordinated adaptation in mobile systems. A very detailed study of the design of the framework was conducted and

to support our propositions, the framework was also simulated. Even though most of the components of the simulation software could be used directly for implementation, it would be important to implement the entire framework rather than simulating certain parts of it. By implementing the framework in a real-time mobile environment we can then study the performance of our framework as compared to other adaptation models proposed by various researchers.

REFERENCES

[1] Keith Cheverst, Christos Efstratiou, Nigel Davies and Adrian Friday "Architectural Ideas for the Support of Adaptive Context-Aware Applications", Distributed Multimedia Research Group, Lancaster University, 2000.

[2] Guanling,Chen and David Kotz "Asurvey of context aware mobile computing reseach", Dartmouth Computer Science Technical Report TR2000-381.

[3] Brian Noble, M. Satyanarayanan, D. Narayanan, James Tilton, Jason Flinn, Kevin Walker, "Agile Application-Aware Adaptation for Mobility," Symposium on Operating System Principles, Nov. 1997.

[4] B. Badrinath, Armando Fox, Leonard Kleinrock, Gerald Popek, Peter Reiher, and M. Satyanarayanan, "A Conceptual Framework for Network and Client Adaptation," *Mobile Networks and Applications*, Vol. 5, No. 4, 2000.

[5]Jay Kistler and M. Satyanarayanan, "Disconnected Operation in the Coda File System," ACM Transactions on Computers, Vol. 10, No. 1, Feb. 1992.

[6] Brian Noble, M. Satyanarayanan, D. Narayanan, James Tilton, Jason Flinn, Kevin Walker, "Agile Application-Aware Adaptation for Mobility," Symposium on Operating System Principles, Nov. 1997.

[7] Mark Yarvis, Peter Reiher, and Gerald Popek, "Conductor: A Framework for Distributed Adaptation," Proc. Seventh Workshop on Hot Topics in Operating Systems (HotOS-VII), March 1999.

[8] C. Efstratiou, A. Friday, N. Davies, and K. Cheverst, "A Platform Supporting Coordinated Adaptation in Mobile Systems" C, in proc. WMCSA 2002, New York, U.S. pp 128-137, June 2002.

[9] Armando Fox, Steven D. Gribble, Eric A. Brewer and Elan Amir. "Adapting to Network and Client Variability via On-Demand Dynamic Distillation". Proc. Seventh Intl. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII), Cambridge, MA, Oct. 1996.

[10] A. Fox, S. Gribble, Y. Chawathe and E. A. Brewer "Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives" *IEEE Personal Communications*, Special Issue on Adaptation, August 1998.

[11] C. Efstratiou, A. Friday, N. Davies and K. Cheverst "Utilising the Event Calculus for Policy Driven Adaptation in Mobile Systems" Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002), Monterey,

Ca., U.S., J. Lobo, B. J. Michael and N. Duray (eds.), IEEE Computer Society, pp. 13-24, June, 2002.

[12] J. Allen, G. Ferguson, "Actions and Events in Interval Temporal Logic", Journal of Logic and Computation, special issue on Actions and Processes, 1994.

[13] R. Kowalski, "A Logic Based Calculus of Events", Journal of Logic Programming, 4:67-95,1986.

[14] J. Lobo, R. Bhatia, and S. Naqvi. "A policy description language". In Proceedings of AAAI, Orlando, FL, July 1999.

[15] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. "The Ponder policy specification language". In Proceedings of Policy Workshop, Bristol, UK, January 2001.

[16] A. Bandara, E. Lupu and A. Russo, "Using event calculus to formalize policy specification and analysis". The Proceedings of 4th IEEE workshop for distributed systems and networks, Lake Como, Italy, June 2003.

[17] J. Chomicki,J. Lobo, and S. Naqvi, "Conflict Resolution Using Logic Programming",IEEE Transactions on knowledge and data engineering, vol. 15, no. 2, March/April 2003.

[18] A K. Dey and G D. Abowd "Towards a Better Understanding of context and context-awareness". Technical Report GIT-GVU-99-22,Computing, June 1999.

[19] A. Asthana, M. Cravatts, and P. Krzyzanowski. "An indoor wireless system for personalized shopping assistance". In Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, pages 69-74, Santa Cruz, California,December 1994.

[20] N. Marmasse and C. Schmandt. "Location-aware information delivery with ComMotion". In Proceedings of Second International Symposium on Handheld and Ubiquitous Computing, HUC 2000,pages 157-171, Bristol, UK, September 2000. Springer Verlag.

[21] A C. Huang, B C. Ling, S Ponnekanti, and A Fox. "Pervasive computing: What is it good for?" In Proceedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access, pages 84-91, Seattle, WA, August 1999. ACM Press.

[22] A K. Dey and G D. Abowd. "CybreMinder: A context-aware system for supporting reminders". In Proceedings of Second International Symposium on Handheld and ubiquitous Computing,HUC 2000, pages 172-186, Bristol, UK, September 2000. Springer Verlag.

[23] M .Beigl. "MemoClip: A location-based remembrance appliance". Personal Technologies, 4(4):230-233, September 2000.

[24] F .Bennett, T .Richardson, and A .Harter. "Teleporting - making applications mobile." In Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, pages 82-84, Santa Cruz, California, December 1994. IEEE Computer Society Press.

[25] Andy Harter, Andy Hopper, Pete Steggles, Andy Ward, and Paul Webster. "The anatomy of a context-aware application". In Proceedings of the Fifth Annual ACM/IEEE International conference on Mobile Computing and Networking, pages 59-68, Seattle, WA, August 1999. ACM press.

[26] Thai-Lai Pham, Georg Schneider, and Stuart Goose. "Exploiting location-based composite devices to support and facilitate situated ubiquitous computing". In Proceedings of Second International Symposium on Handheld and Ubiquitous Computing, HUC 2000, pages 143-156, Bristol, UK, September 2000. Springer Verlag.

[27] Anind K. Dey, Masayasu Futakawa, Daniel Salber, and Gregory D. Abowd. "The Conference Assistant: Combining Context-Awareness with Wearable Computing". In Proceedings of the 3rd International Symposium on Wearable Computers (ISWC '99), pages 21-28, San Francisco, CA, October 1999. IEEE Computer Society Press.

[28] Hao Yan and Ted Selker. "Context-aware office assistant". In Proceedings of the 2000 International Conference on Intelligent User Interfaces, pages 276- 279, New Orleans, LA, January 2000. ACM Press.

[29] Michael D. Addlesee, Alan Jones, Finnbar Livesey, and Ferdinando Samaria. "The ORL Active Floor". IEEE Personal Communications, 4(5):35-41, October 1997.

[30] Robert J. Orr and Gregory D. Abowd. "The Smart Floor: A mechanism for natural user identification and tracking". In Proceedings of the 2000 Conference on Human Factors in Computing Systems (CHI 2000), The Hague, Netherlands, April 2000.ACM Press.

[31] Armando Fox, Ian Goldberg, Steven D. Gribble, David C. Lee, Anthony Polito, and Eric A. Brewer. "Experience With Top Gun Wingman, A Proxy-Based Graphical Web Browser for the USR PalmPilot". Proc. IFIP Middleware 98, Lake District, UK, Sept. 1998.

[32] Yatin Chawathe, Steve Fink, Steven McCanne, and Eric A. Brewer. "A proxy architecture for reliable multicast in heterogeneous environments". Proc. IFIP Middleware 98, Lake District, UK, Sept.
1998.

[33] Metricom Inc. Ricochet wireless modem service. http://www.ricochet.net

[34] Mark Stemm and Randy H. Katz. "Vertical handoffs in wireless overlay networks". ACM Mobile Networking (MONET) Special Issue on Mobile Networking in the Internet, Fall 1997.

[35] Zhimei Jiang and Leonard Kleinrock, "An Adaptive Pre-fetching Scheme". Journal of Selected Areas in Communications, Vol. 16, No. 3, pp. 358-368, 1-11, April 1998.

[36] Microsoft Corporation. Universal Plug and Play device architecture, http://www.upnp.org. Version 0.91, March 2000.

[37] R. Kowalsky. "Database updates in event calculus". Journal of Logic Programming, 12:121–146, 1992.

[38] R. Miller and M. Shanahan, "Some alternative formulations of the Event Calculus", in A. C. Kakas and F. Sadri (eds.): Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II. Lecture Notes in Computer Science 2408,ISBN 3-540-43960-9, pages: 452-490, Springer 2002.

[39] A. Russo, R. Miller, B. Nuseibeh, and J. Kramer, "An Abductive Approach for Analysing Event-Based Requirements Specifications," presented at 18th Int. Conf. on Logic Programming (ICLP), Copenhagen, Denmark, 2002.

[40] H. Geffner, B. Bonet, "High level planning and control with incomplete informationusing POMDP's", in working notes of the AAAI fall symposium on Cognitive Robotics 1998.

[41] M. Gelfond and V. Lifschitz, "Representing asction and change by logic program", JPL 17: 301-321, 1993.

APPENDIX A

CODE: MOBILE CLIENT

```
Option Explicit
Dim WebBrowser As Application
Dim StreamVideo As Application
Dim EnterApp As Application
Dim intEventCount As Integer
Dim logArray() As EventLog
Dim intEventId As Integer
Dim intEMLMessID As Integer
Dim intAppMessID As Integer
Dim xDoc As New DOMDocument

Private Sub Command1_Click()
'Dim message As MessagePDU
'message.MessageID = 12
'message.Data = 242
'message.To = 2
'message.Type = 1
'message.Data = "40.98"
'Call Talk2EML(message)
Call makeBinFile
Transfer.Show
End Sub

Private Sub Command2_Click(Index As Integer)
Dim message As MessagePDU
Dim BD As String
BD = InputBox("How much bandwidth do you want to Request?", "Request Bandwidth")
Select Case Index
 Case Is = 0 ' for the webbrowser
   message.From = WebBrowser.appId
 Case Is = 1 ' for streamin video
   message.From = StreamVideo.appId
 Case Is = 2 ' for enterprise application
   message.From = EnterApp.appId
End Select
message.MessageID = intAppMessID
intAppMessID = intAppMessID + 1
message.To = 2
message.Type = 1
message.Data = BD
Call Talk2EML(message)
```

```
End Sub

Private Sub Form_Click()
WinsockSend.RemoteHost = "131.94.127.165" 'PS IP address
WinsockSend.RemotePort = 3001
WinsockSend.Connect

End Sub

Private Sub Form_Load()
'**** connecting to the PS*******
WinsockReceive.LocalPort = 2001
WinsockReceive.Listen
'************
intEventCount = 0
intEventId = 0
WebBrowser.appId = 111
WebBrowser.Priority = 8
WebBrowser.TimeGranularity = 10
WebBrowser.Bandwidth = 33.3
WebBrowser.Type = "App"

StreamVideo.appId = 222
StreamVideo.Priority = 6
StreamVideo.TimeGranularity = 10
StreamVideo.Bandwidth = 33.3
StreamVideo.Type = "App"

EnterApp.appId = 333
EnterApp.Priority = 9
EnterApp.TimeGranularity = 10
EnterApp.Bandwidth = 33.3
EnterApp.Type = "App"
Text2.Text = "EventID EventType EventSource TimeStamp DateStamp Details" +
vbCrLf
xDoc.Load ("c:\registry.xml")
Text1.Text = xDoc.xml
End Sub

Private Sub Talk2EML(message As MessagePDU)
Dim MessageOut As MessagePDU
Dim strData As String
'**********logging event*****************
intEventId = intEventId + 1
ReDim Preserve logArray(intEventCount)
```

```
logArray(intEventCount).EventId = intEventId
Text2.Text = Text2.Text & "  " & logArray(intEventCount).EventId
logArray(intEventCount).EventType = message.Type
Text2.Text = Text2.Text & "          " & logArray(intEventCount).EventType
logArray(intEventCount).EventSource = message.From
Text2.Text = Text2.Text & "              " & logArray(intEventCount).EventSource
logArray(intEventCount).TimeStamp = Time()
Text2.Text = Text2.Text & "               " & logArray(intEventCount).TimeStamp
logArray(intEventCount).DateStamp = Date
Text2.Text = Text2.Text & "  " & logArray(intEventCount).DateStamp
logArray(intEventCount).Details = message.Data
Text2.Text = Text2.Text & "  " & logArray(intEventCount).Details & vbCrLf
'Text2.Text = Text2.Text + vbCrLf + Str(logArray(intEventCount).EventId) +
Str(logArray(intEventCount).EventType) + Str(logArray(intEventCount).EventSource) +
Str(logArray(intEventCount).TimeStamp) + Str(logArray(intEventCount).DateStamp) +
Str(logArray(intEventCount).Details)
intEventCount = intEventCount + 1
intEMLMessID = intEMLMessID + 1

'********** Processing message***********
If message.Type = 1 Then ' for Requests
   MessageOut.MessageID = intEMLMessID
   MessageOut.From = 2 ' From EML
   MessageOut.To = 3   ' To Policy Server
   MessageOut.Type = 4 ' For Quering the PS for any Policies
   MessageOut.Data = Str(message.From) & "#" & message.Data
   Call Talk2PS(MessageOut)
End If

If message.Type = 2 And message.From = 3 Then  ' for Response from the PS
   If Left(message.Data, 10) = "getAppInfo" Then
      strData = getAppInfo(Mid(message.Data, 11, Len(message.Data) - 10)) ' to keep
track of which app requested for change in bandwidth
      MessageOut.MessageID = intEMLMessID
      MessageOut.From = 2 ' From EML
      MessageOut.To = 3   ' To Policy Server
      MessageOut.Type = 3 ' For updating the prior request by PS
      MessageOut.Data = strData
      Call Talk2PS(MessageOut)
   End If
End If

If message.Type = 8 And message.From = 3 Then ' for action from the PS
   Call AlterBandwidth(Right(message.Data, 6))
End If
```

```
End Sub

Private Sub Talk2PS(message As MessagePDU)
Dim strSend As String
strSend = Str(message.MessageID) + "#" + Str(message.To) + "#" + Str(message.From)
+ "#" + Str(message.Type) + "#" + message.Data
'Form2.Label1.Caption = strSend
'Form2.Show
If WinsockSend.State = sckConnected Then
   WinsockSend.SendData strSend
Else
   MsgBox "Unable to connect to the Policy Server"
End If

End Sub

Private Function getAppInfo(strOrigin As String) As String
Dim strData As String
strData = Str(WebBrowser.appId) & "#" & Str(WebBrowser.Priority) & "#" &
WebBrowser.Type & "#" & Str(WebBrowser.TimeGranularity) & "#" &
Str(WebBrowser.Bandwidth)
strData = strData & "#" & Str(StreamVideo.appId) & "#" & Str(StreamVideo.Priority) &
"#" & StreamVideo.Type & "#" & Str(StreamVideo.TimeGranularity) & "#" &
Str(StreamVideo.Bandwidth)
strData = strData & "#" & Str(EnterApp.appId) & "#" & Str(EnterApp.Priority) & "#" &
EnterApp.Type & "#" & Str(EnterApp.TimeGranularity) & "#" &
Str(EnterApp.Bandwidth)
getAppInfo = strData & strOrigin
End Function

Private Sub AlterBandwidth(change As String)
Dim i As Integer
Dim id As Integer
Dim BD As Long
id = CInt(Mid(change, 1, 3))
BD = CDbl(Mid(change, 5, Len(change) - 4))
If WebBrowser.appId = id Then WebBrowser.Bandwidth = BD
If StreamVideo.appId = id Then StreamVideo.Bandwidth = BD
If EnterApp.appId = id Then EnterApp.Bandwidth = BD
MsgBox BD
End Sub

'******** this sub resolves incoming mesage from outside the system
```

```
Private Sub ResolveMessage(strReceive As String)
Dim message As MessagePDU
Dim pos1 As Integer
Dim pos2 As Integer
strReceive = Trim(strReceive)
pos1 = 1
pos2 = 1
pos2 = InStr(pos1, strReceive, "#")
message.MessageID = (Trim(Mid(strReceive, pos1, pos2 - pos1)))
pos1 = pos2 + 1

pos2 = InStr(pos1, strReceive, "#")
message.To = CInt(Trim(Mid(strReceive, pos1, pos2 - pos1)))
pos1 = pos2 + 1

pos2 = InStr(pos1, strReceive, "#")
message.From = CInt(Trim(Mid(strReceive, pos1, pos2 - pos1)))
pos1 = pos2 + 1

pos2 = InStr(pos1, strReceive, "#")
message.Type = CInt(Trim(Mid(strReceive, pos1, pos2 - pos1)))
pos1 = pos2 + 1


message.Data = Mid(strReceive, pos1, Len(strReceive) - pos1 + 1)

Call Talk2EML(message)
End Sub

Private Sub WinsockReceive_ConnectionRequest(ByVal requestID As Long)
If WinsockReceive.State <> sckClosed Then WinsockReceive.Close
WinsockReceive.Accept requestID
End Sub

Private Sub WinsockReceive_DataArrival(ByVal bytesTotal As Long)
Dim Data As String
WinsockReceive.GetData Data
Text1.Text = Text1.Text + vbCrLf + Data
Call ResolveMessage(Data)
End Sub

Private Sub WinsockSend_Error(ByVal Number As Integer, Description As String,
ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal
HelpContext As Long, CancelDisplay As Boolean)
MsgBox "Connection Failed to the Policy Server"
```

```
End Sub

Public Sub makeBinFile()
Dim i As Integer
Dim intBinaryFile As Integer
i = UBound(logArray)
intBinaryFile = FreeFile
Open "c:\EventLog.Bin" For Binary As intBinaryFile
Put intBinaryFile, 1, i
Put intBinaryFile, , logArray
Close intBinaryFile
End Sub
```

Code: Event Log Transfer

```
Dim strSize
Dim strBlock As String
Dim FileClosed As Boolean
Dim KBPS

Private Sub Command1_Click()
   WS.LocalPort = 1001
   WS.Listen
End Sub

Private Sub Command2_Click()
   WS.RemoteHost = "131.94.127.165"
   WS.RemotePort = 1001
   WS.LocalPort = 0
   WS.Connect
End Sub

Private Sub Form_Load()
WS.RemoteHost = "131.94.127.165"
   WS.RemotePort = 1001
   WS.LocalPort = 0
   WS.Connect
End Sub

Private Sub KbpsTimer_Timer()
  Text3.Text = Mid$(Str$(KBPS / 1024), 1, InStr(1, Str$(KBPS / 1024), ".",
vbTextCompare) + 2)
'  KBPS = 0
End Sub
```

```
'Private Sub Text1_Change()
'   If Len(Text1.Text) > 1000 Then Text1.Text = Right$(Text1.Text, 900)
'   Text1.SelStart = (Len(Text1.Text) - 1)
"   Text1.SelLength = 1
'   Text1.SelLength = 0
'End Sub

Private Sub Timer1_Timer()
   KBPS = 0
End Sub

Private Sub WS_ConnectionRequest(ByVal requestID As Long)
   WS.Close
   WS.Accept requestID
   DoEvents
   WS.SendData "sendfile"
End Sub

Private Sub WS_DataArrival(ByVal bytesTotal As Long)
   Dim strData As String
   WS.GetData strData

   If strData = "sendfile" Then
      CD.Filter = "*.*|*.*"
      CD.ShowOpen
      Open CD.FileName For Binary As #1
      FileClosed = False
      KbpsTimer.Interval = 1000
      strSize = LOF(1)
      WS.SendData "name" & CD.FileTitle & ":" & strSize
      DoEvents
   End If

   If strData = "okok" Then
      PBar.Max = strSize
      If Not FileClosed Then
         If strSize - Loc(1) < 2040 Then ' last chunk of data
            strBlock = Space$(strSize - Loc(1))
            Get 1, , strBlock
            WS.SendData "file" & strBlock
            DoEvents
            'Text1.Text = Text1.Text & strBlock
            PBar.Value = PBar.Value + Len(strBlock)
            WS.SendData "stop"
            DoEvents
```

```
            KBPS = KBPS + Len(strBlock)
            Close #1
            FileClosed = True
            KbpsTimer.Interval = 0
            DoEvents
            MsgBox "File sent"
            'Unload (Transfer)
        Else 'Not the last chunk
            strBlock = Space$(2040)
            Get 1, , strBlock
            WS.SendData "file" & strBlock
            DoEvents
            KBPS = KBPS + Len(strBlock)
            'Text1.Text = Text1.Text & strBlock
            PBar.Value = PBar.Value + Len(strBlock)
        End If
    End If
End If

If Left(strData, 4) = "file" Then
    strData = Mid$(strData, 5)
    Put 1, , strData
    'Text1.Text = Text1.Text & strData
    PBar.Value = PBar.Value + Len(strData)
    WS.SendData "okok"
    DoEvents
    KBPS = KBPS + Len(strData)
End If

If strData = "stop" Then
    Close #1
    MsgBox "file recieved"
    KbpsTimer.Interval = 0
    'PBar.Value = 0.001
End If

If Left(strData, 4) = "name" Then
    CD.Filter = "*.*|*.*"
    strData = Mid$(strData, 5)
    x = InStr(1, strData, ":", vbTextCompare)
    strSize = Mid$(strData, x + 1)
    CD.FileName = Mid$(strData, 1, x - 1)
    CD.ShowSave
    Open CD.FileName For Binary As #1
    KbpsTimer.Interval = 1000
```

```
        PBar.Max = strSize
        WS.SendData "okok"
        DoEvents
    End If
End Sub
```

Code: Module

```
Public Type EventLog
EventId As Integer
EventType As Integer
EventSource As Integer
TimeStamp As String * 10
DateStamp As String * 10
Details As String * 10
End Type

Public Type Application
appId As Integer
Priority As Integer
Type As String
TimeGranularity As Long
Bandwidth As Long
End Type

Public Type MessagePDU
MessageID As String
To As Integer
From As Integer
Type As Integer
Data As String
End Type
```

APPENDIX B

CODE: POLICY SERVER

```
Option Explicit
Dim appInfo As ForappInfo
Dim intPolMessId As Integer
Dim appArray(2) As Application
Dim extractLogArray() As EventLog

Private Sub ResolveMessage(strData As String)
Dim message As MessagePDU
Dim strIdData As String
strData = Trim(strData)

message.Type = (Mid(strData, 10, 1))
message.Data = Mid(strData, 13, Len(strData) - 12)

'MsgBox "type is" & " =" & message.Type
'MsgBox "data is" & " =" & message.Data
If message.Type = 3 Then ' for updates from the eml
   'code for updating appinfo variable
   ' and also update the application structure array
   ' use a method called as updateTemps (str as string)
   strIdData = updateTemps(message.Data)
   Call Policy(CInt(Mid(strIdData, 1, 3)), Mid(strIdData, 5, Len(strIdData) - 4))
End If


If message.Type = 4 Then ' query from the EML
   Call Policy(CInt(Mid(message.Data, 1, 3)), Mid(message.Data, 5, Len(message.Data) -
4))
End If

End Sub



Private Sub Policy(appId As Integer, BD As String)
Dim strSend As String
Dim i As Integer
Dim intOur As Integer
Dim intOther1 As Integer
Dim intOther2 As Integer
```

```
If (appInfo.Available) And (appInfo.From = appId) Then
   Call WhichIsOur(appId, intOur, intOther1, intOther2)
' if demanding less bandwidth then just grant it
   If CDbl(BD) < appArray(intOur).Bandwidth Then
   ' write code for action method and send it to EML
   Else
   ' when reducing and increasin send action twice seperately
      If appArray(intOur).Priority > appArray(intOther1).Priority And
appArray(intOur).Priority > appArray(intOther2).Priority Then
         If appArray(intOther1).Priority < appArray(intOther2).Priority Then
            'reduce for intOther1 and increace for intOur
            strSend = intPolMessId & "#" & "2" & "#" & "3" & "#" & "8" & "#" &
appArray(intOther1).appId & "#" & appArray(intOther1).Bandwidth
            Call Send2EML(strSend)
            strSend = intPolMessId & "#" & "2" & "#" & "3" & "#" & "8" & "#" &
appArray(intOur).appId & "#" & BD 'appArray(intOur).Bandwidth
            Call Send2EML(strSend)
         Else
            'reduce for intOther2 and increace for intOur
            strSend = intPolMessId & "#" & "2" & "#" & "3" & "#" & "8" & "#" &
appArray(intOther2).appId & "#" & appArray(intOther2).Bandwidth
            Call Send2EML(strSend)
            strSend = intPolMessId & "#" & "2" & "#" & "3" & "#" & "8" & "#" &
appArray(intOur).appId & "#" & BD 'appArray(intOur).Bandwidth
            Call Send2EML(strSend)
         End If
      End If
   End If

'
'check the criteria and compare
'
'
Else
   '*** to get app info
   strSend = intPolMessId & "#" & "2" & "#" & "3" & "#" & "2" & "#" & "getAppInfo"
& "#" & appId & "#" & BD
   Call Send2EML(strSend)
End If
End Sub

Private Sub Command1_Click()
'Call ResolveMessage("cas")
Dim strData As String
Dim i As Integer
```

```
strData = Trim(Label1.Caption)
For i = 1 To Len(strData)
Text1.Text = Text1.Text + vbCrLf + Str(i) + "=" + Mid(strData, i, 1)
Next i

End Sub

Private Sub Command2_Click()
'MsgBox updateTemps(Text2.Text)
Call Send2EML(Text2.Text)
End Sub

Private Sub Send2EML(strSend As String)
' write here teh winsock senddata commmand and pass strsend as parameter
If WinsockSend.State = sckConnected Then
   WinsockSend.SendData strSend
Else
   'Call Form_Click
  ' WinsockSend.SendData strSend
   MsgBox "Failed to send message to the Client"
End If
End Sub

Private Function updateTemps(strData As String) As String
strData = Trim(strData)
Dim pos1 As Integer
Dim pos2 As Integer
Dim i As Integer
pos1 = 1
pos2 = 1
For i = 0 To 2
pos2 = InStr(pos1, strData, "#")
appArray(i).appId = CInt(Trim(Mid(strData, pos1, pos2 - pos1)))
pos1 = pos2 + 1


pos2 = InStr(pos1, strData, "#")
appArray(i).Priority = CInt(Trim(Mid(strData, pos1, pos2 - pos1)))
pos1 = pos2 + 1

pos2 = InStr(pos1, strData, "#")
appArray(i).Type = (Trim(Mid(strData, pos1, pos2 - pos1)))
pos1 = pos2 + 1

pos2 = InStr(pos1, strData, "#")
```

```
appArray(i).TimeGranularity = CDbl(Trim(Mid(strData, pos1, pos2 - pos1)))
pos1 = pos2 + 1

pos2 = InStr(pos1, strData, "#")
appArray(i).Bandwidth = CDbl(Trim(Mid(strData, pos1, pos2 - pos1)))
pos1 = pos2 + 1
'Text2.Text = Text2.Text + vbCrLf + Mid(Text1.Text, pos1, pos2 - pos1)
'pos1 = pos2 + 1
Next i
pos2 = InStr(pos1, strData, "#")
appInfo.From = CInt(Trim(Mid(strData, pos1, pos2 - pos1)))
'pos1 = pos2 + 1
appInfo.Available = True
' to return in the form of id data i.e 222#12.34
updateTemps = Mid(strData, pos1, Len(strData) - pos1 + 1)
' remember to extract tye appinfo the id and set available before returning
End Function

Private Sub WhichIsOur(ByVal appId As Integer, ByRef intOur As Integer, ByRef
intOther1 As Integer, ByRef intOther2 As Integer)
If appArray(0).appId = appId Then
   intOur = 0
   intOther1 = 1
   intOther2 = 2
End If

If appArray(1).appId = appId Then
   intOur = 1
   intOther1 = 0
   intOther2 = 2
End If

If appArray(2).appId = appId Then
   intOur = 2
   intOther1 = 1
   intOther2 = 0
End If
End Sub

Private Sub Command3_Click()
Dim intEventCount As Integer
Call readBinFile
Text1.Text = Text1.Text + vbCrLf + "EventID EventType EventSource TimeStamp
DateStamp Details" + vbCrLf
For intEventCount = 0 To UBound(extractLogArray)
```

```
Text1.Text = Text1.Text & "  " & extractLogArray(intEventCount).EventId

Text1.Text = Text1.Text & "          " & extractLogArray(intEventCount).EventType

Text1.Text = Text1.Text & "              " &
extractLogArray(intEventCount).EventSource

Text1.Text = Text1.Text & "             " & extractLogArray(intEventCount).TimeStamp

Text1.Text = Text1.Text & "  " & extractLogArray(intEventCount).DateStamp

Text1.Text = Text1.Text & "  " & extractLogArray(intEventCount).Details & vbCrLf
Next intEventCount
End Sub

Private Sub Command4_Click()
frmTransfer.Show
End Sub

Private Sub Form_Click()
WinsockSend.RemoteHost = "131.94.126.158"
WinsockSend.RemotePort = 2001
WinsockSend.Connect
Text1.Text = ""
End Sub

Private Sub Form_Load()
'***communication  to the Client*****
WinsockReceive.LocalPort = 3001
WinsockReceive.Listen
End Sub


Private Sub WinsockReceive_ConnectionRequest(ByVal requestID As Long)
If WinsockReceive.State <> sckClosed Then WinsockReceive.Close
WinsockReceive.Accept requestID
End Sub

Private Sub WinsockReceive_DataArrival(ByVal bytesTotal As Long)
Dim Data As String
WinsockReceive.GetData Data
Text1.Text = Text1.Text + vbCrLf + Data
Call ResolveMessage(Data)
End Sub
```

```
Private Sub WinsockSend_Error(ByVal Number As Integer, Description As String,
ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal
HelpContext As Long, CancelDisplay As Boolean)
MsgBox "Unable to Connect to the Client"
End Sub

Private Sub readBinFile()
Dim i As Integer
Dim extractfile As Integer
extractfile = FreeFile
Open "c:\EventLog.Bin" For Binary As extractfile
Get extractfile, 1, i
ReDim extractLogArray(i)
Get extractfile, , extractLogArray
Close extractfile
End Sub
```

Code: Event Log Transfer

```
Dim strSize
Dim strBlock As String
Dim FileClosed As Boolean
Dim KBPS

Private Sub Command1_Click()
   WS.LocalPort = 1001
   WS.Listen
End Sub

Private Sub Command2_Click()
   WS.RemoteHost = Text2.Text
   WS.RemotePort = 1001
   WS.LocalPort = 0
   WS.Connect
End Sub

Private Sub Form_Load()
WS.LocalPort = 1001
   WS.Listen
End Sub

Private Sub KbpsTimer_Timer()
   Text3.Text = Mid$(Str$(KBPS / 1024), 1, InStr(1, Str$(KBPS / 1024), ".",
vbTextCompare) + 2)
   KBPS = 0
```

```
End Sub

Private Sub Text1_Change()
   If Len(Text1.Text) > 1000 Then Text1.Text = Right$(Text1.Text, 900)
   Text1.SelStart = (Len(Text1.Text) - 1)
   Text1.SelLength = 1
   Text1.SelLength = 0
End Sub

Private Sub Timer1_Timer()
   KBPS = 0
End Sub

Private Sub WS_ConnectionRequest(ByVal requestID As Long)
   WS.Close
   WS.Accept requestID
   DoEvents
   WS.SendData "sendfile"
End Sub

Private Sub WS_DataArrival(ByVal bytesTotal As Long)
   Dim strData As String
   WS.GetData strData

   If strData = "sendfile" Then
      CD.Filter = "*.*|*.*"
      CD.ShowOpen
      Open CD.FileName For Binary As #1
      FileClosed = False
      KbpsTimer.Interval = 1000
      strSize = LOF(1)
      WS.SendData "name" & CD.FileTitle & ":" & strSize
      DoEvents
   End If

   If strData = "okok" Then
      PBar.Max = strSize
      If Not FileClosed Then
         If strSize - Loc(1) < 2040 Then ' last chunk of data
            strBlock = Space$(strSize - Loc(1))
            Get 1, , strBlock
            WS.SendData "file" & strBlock
            DoEvents
            Text1.Text = Text1.Text & strBlock
            PBar.Value = PBar.Value + Len(strBlock)
```

```
            WS.SendData "stop"
            DoEvents
            KBPS = KBPS + Len(strBlock)
            Close #1
            FileClosed = True
            KbpsTimer.Interval = 0
            DoEvents
            MsgBox "File sent"
        Else 'Not the last chunk
            strBlock = Space$(2040)
            Get 1, , strBlock
            WS.SendData "file" & strBlock
            DoEvents
            KBPS = KBPS + Len(strBlock)
            Text1.Text = Text1.Text & strBlock
            PBar.Value = PBar.Value + Len(strBlock)
        End If
    End If
End If

If Left(strData, 4) = "file" Then
    strData = Mid$(strData, 5)
    Put 1, , strData
    Text1.Text = Text1.Text & strData
    PBar.Value = PBar.Value + Len(strData)
    WS.SendData "okok"
    DoEvents
    KBPS = KBPS + Len(strData)
End If

If strData = "stop" Then
    Close #1
    MsgBox "file recieved"
    'Unload (frmTransfer)

    KbpsTimer.Interval = 0
    'PBar.Value = 0.001
End If

If Left(strData, 4) = "name" Then
    CD.Filter = "*.*|*.*"
    strData = Mid$(strData, 5)
    x = InStr(1, strData, ":", vbTextCompare)
    strSize = Mid$(strData, x + 1)
    CD.FileName = Mid$(strData, 1, x - 1)
```

```
        CD.ShowSave
        Open CD.FileName For Binary As #1
        KbpsTimer.Interval = 1000
        PBar.Max = strSize
        WS.SendData "okok"
        DoEvents
    End If
End Sub
```

Code: Module1

```
Public Type EventLog
EventId As Integer
EventType As Integer
EventSource As Integer
TimeStamp As String * 10
DateStamp As String * 10
Details As String * 10
End Type

Public Type Application
appId As Integer
Priority As Integer
Type As String
TimeGranularity As Long
Bandwidth As Long
End Type

Public Type MessagePDU
MessageID As String
To As Integer
From As Integer
Type As Integer
Data As String
End Type

Public Type ForappInfo
Available As Boolean
From As Integer
End Type
```