University of Twente
Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)

Telematica Instituut

Policy-based Handoffs across Intermediary Multimedia Content Providers in the Wireless Internet

Malohat Ibrohimovna Kamilova

Thesis for a degree of
Professional Doctor of Engineering

GRADUATION COMMITTEE:

Prof. Dr. ir. E. Huizer (UT)
Dr. ir. I. Widya (UT)
ir. C. Hesselman (Telematica Instituut & UT)
Dr. ir. H. Eertink (Telematica Instituut)

UNIVERSITY OF TWENTE

Design and Analysis of Communication Systems
Architecture and Services of Network Applications
Faculty of EEMCS

TELEMATICA INSTITUUT

Enschede, the Netherlands
October 2004

Abstract

In the near future, the fringes of the Internet will consist of different types of wired and wireless networks that are operated by different administrative authorities. Additionally, the growing number of services available on the Internet enable a mobile host to receive these services via multiple networks simultaneously, for instance when they roam into a hotspot.

In this research we consider the distribution of multimedia content through multiple content aggregators over wireless Internet to mobile hosts. Aggregator is an intermediary content provider that forwards the multimedia content from a source (e.g. cnn.news) to mobile hosts in a user-friendly manner, i.e. different quality levels and prices. At some locations, such us in hotspots, a mobile host can choose from different alternatives (i.e. quality, price) of this multimedia content forwarded by different aggregators through different networks. These alternatives might change for instance when user roams, which requires a user be able to handoff between aggregators.

We propose an application level policy-based control mechanism that enables mobile hosts to deal with (changing) alternatives automatically, therefore without burdening the user. The novelty of this mechanism lies in the use of application-level policies, which determine when and how to adapt the reception of multimedia content to changes in available (network) resources or user preferences. Being equipped with these policies, the control mechanism enables the user to roam across heterogeneous networks and aggregators, and receive the multimedia content in an uninterrupted way.

We apply a policy-based control model, which is inspired by the IETF policy framework, in the design and the prototype development of the control mechanism for the mobile host. We use existing components of the mobile host that monitor the changes in the previously mentioned multimedia distribution environment. On command of the developed control mechanism, these components initiate discovery and to handoff to one of the aggregators in the vicinity of the mobile host. The prototype demonstrates the flexibility of policy based control and to a certain extent validates the proposed policy model.

Preface

This research is a joint TWAIO project between the University of Twente and Telematica Instituut and is embedded in the PhD research project QUADAPT on adaptive distribution of live broadcast in an Internet environment supported by wireless networks 'beyond-3G'. QUADAPT develops a generic model for adaptive distribution of live Internet broadcasts, for example radio, TV and e-cinema distribution, using distributed proxy stream servers and heterogeneous multicast-capable wireless infrastructures. This TWAIO project focuses on the issues of smooth streaming of the broadcast in a selected quality in accordance with the user preferences while user roams within this environment. This research was carried out in Telematica Instituut, where from the first day of my work I felt a very warm and friendly atmosphere and a spirit of research that inspired me a lot.

I feel very lucky to meet so kind and nice people that I would like to thank for their help during my TWAIO course (a two-years designer course).

First of all, I am grateful to my first teachers at the University of Twente *Hans Daemen, Piet Kommers* and *Henk Alblas* for selecting me to participate in the EU project TRATMICT in 2000, which opened me this opportunity to be a TWAIO student at the University of Twente in 2002-2004.

During this project I enjoyed a teamwork with my supervisors *Ing Widya*, *Cristian Hesselman* and *Erik Huizer,* and learned a lot from them. I would like to express them my sincere gratitude for their kind support and generosity in teaching and encouraging me during this research. I am thankful to them for being so much devoted to their work, for the long hours of discussions analyzing the concepts, designing and developing the system, for their patience and kindness in teaching me writing a good scientific paper. I learned from them how to work, which will guide me in my future research. Due to this TWAIO project I have become rich for one more *policy* in my life: *if (you have such a great supervisors like I do) then (there is no doubt you will have a success)!*

I would like also to thank the *Design and Analysis of Communication Systems* group, the *Architecture and Services of Network Applications* group and *Telematica Instituut* for giving me the opportunity to carry out this research.

I would like to thank my colleagues from Telematica Instituut *Hans Zandbelt, Remco Poortinga* and *Arjan Peddemors* for their help during the development of the prototype.

I am thankful to my friends *Muzaffar Igamberdiev* and *Hans Daemen* for their sincere friendship, for encouragement on those days when I lost my hope and inspiration.

I would like to express my gratitude to my dear parents *Oydin Hojieva* and *Ibrohim Gafurov,* who live in O'zbekiston, for their endless love and support that I feel every step of my way even being so far away from them, for encouraging me and staying by my side in difficult times of my life. My special gratitude and love is for my sweet daughter *Muhabbat Komilova*, who with her courage, patience and love made it possible for me to accomplish this TWAIO course.

Malohat Ibrohimovna Kamilova
October 2004, Enschede.

Table of Contents

List of Figures

1. INTRODUCTION

In the near future, the fringes of the Internet will consist of different types of wired and wireless networks that are operated by different administrative authorities [1]. Additionally, the growing number of Internet services enable a mobile host to receive these Internet services via multiple networks simultaneously, for instance when they roam into a hotspot. This thesis addresses the challenges of having alternatives of multimedia content providers and wireless networks at hotspots.

This chapter is structured as follows:
Section 1.1 describes the problem statement, Section 1.2 presents the approach taken in this thesis to solve the stated problem, gives a short discussion on the work related to the techniques used in the selected approach and discusses the novelty of our work. Section 1.3 outlines the structure of this thesis by presenting an overview of chapters.

1.1 Problem Statement

In this report, we consider the distribution of real-time multimedia content (e.g. radio or TV broadcasts) through multiple aggregators. An aggregator is an intermediary content provider that aggregates content from a source and forwards it to a mobile host [2,3]. The aggregator packages content into channels (e.g. CNN radio or BBC news) and offers them in various versions (e.g. using different encodings) that differ in quality or price. The mobile host may receive these channels via the wireless Internet, which consists of multiple types of wireless networks (e.g. 802.11 and UMTS). At specific locations, the mobile host can connect to multiple networks simultaneously (e.g. in a hotspot). As a result, it can potentially receive different alternative versions of a channel from different aggregators through different interfaces.

An important feature of this environment is that mobile users can switch from one aggregator to another *while* they are receiving a channel. The reason for that can be roaming of the user within this environment as well as changing user preferences. For example, the price of the channel version might be expensive for the user, so he might switch to the aggregator that provides the same channel for a cheaper price or the quality of the channel version the user receives might be not satisfactory, so the user switches to the version of this channel with a better quality level; or an aggregator disappears as a result of roaming etc. Switching between aggregators are similar to handoffs between base stations, except that they occur at the application-level rather than at the network or IP-level.

A generic research question in such an environment is *how to maintain or adapt the service continuity*?

One of the systems developed for the above mentioned environment, is the CORD system [2-4], which enables mobile hosts to discover available aggregators in the vicinity of the mobile host, to connect to aggregator and to switch from one aggregator to another. This system puts the responsibility of switching to another aggregator with the mobile host and uses standard IETF protocols, notably SIP [5] and SDP[6].

An open issue in the CORD system is that it does not provide a (flexible) mechanism that enables mobile hosts to automatically invoke discovery of available aggregators, to select the best aggregator, and then to switch to that aggregator. Such a mechanism is important, because the complexity of the infrastructure (in terms of aggregators and networks operators) should be shielded off from the user to improve user-friendliness [7].

Therefore the detailed research question is:

When to discover alternatives of a media channel and network connectivity;
how to select amongst these alternatives the best version of a multimedia channel, which matches service facilities, network capabilities and user preferences;
and how to switch to the selected alternative transparently to the user?

The aim of this work is to develop a flexible mechanism to ensure service continuity and adaptation, in particular, to enable the mobile host to automatically select the best one among alternative aggregators, offering the same service with different quality or price through different wireless networks.

1.2 Approach

In this research, we follow a policy based approach and propose *an Application Level Policy-based Handoff Control System* (the APPLE system) as a controlling entity for the CORD system, which enables mobile hosts to automatically invoke discovery of available aggregators, to select the best aggregator, and then to switch to that aggregator.

Policies are "if-condition-then-action" rules that can be used by a controlling entity to constrain (i.e. adapt) the behavior of a controlled entity in a way that the behavior of the controlled entity becomes aligned with the goal of the policy [10, 11]. Policies provide flexibility [12-16], i.e. the conditions and actions of the policies can be flexibly changed without modifying the controlled entity. Policies are considered useful when the system has alternatives to choose from [17]. A set of alternatives can be constrained by policies in accordance with the user defined guidelines (i.e. price, quality etc.). Policies can be downloaded into the mobile host at run time, because they can be maintained in a repository. This enables reconfiguration of control with new policies in a flexible manner, see also [18].

Policies are defined in the literature in many different ways. Therefore we follow the following approach: First we investigate the literature and analyze those policy-based systems. Then we propose a policy model and afterwards we use this model in the design of the APPLE system. The design of the APPLE system may therefore be viewed as a kind of validation of the proposed model.

In this report, we largely follow the IETF's policy model [11, 18]. The IETF's policies are typically used at the network-level for network management purposes, but we use policies at application level for control purposes, particularly to control the mobile host in automatically switching between aggregators.

Known systems for Internet service control can be considered as policy-based and non-policy-based systems, according to the approach they have applied. Policy-based systems typically use network level policies rather than application level policies. For example, Murray et al. [19] and Wang et al. [17] use network level policies to determine which network (operator) provides the best service. Wang et al. [17] select the best network based on the user preferences. Murray et al. [19] select the best network for a mobile host based on the current load in overlay networks. In both of them [17, 19] handoff is triggered by policies.

Non policy-based system (e.g. Clark et al. [20] and Lee et al. [21]) take a different approach to determine the best service, which instead of policies use algorithms. Lee at al. use an input from a

prediction function, network level information, application level information (i.e. identities of running applications) and user preferences to automatically select the best network/service for the mobile host. In their work, handoff from one network to another is triggered by this selection algorithm.

The novelty of *our* policy-based system is that it uses application level policies to automatically trigger discovery of alternative aggregators, to select the best one and to switch to selected aggregator, thus providing service continuity from the user perspective.

1.4 Structure of the thesis

This thesis consists of 7 chapters, where we describe our work and the issues have been dealt with throughout the research process. The structure of this thesis is as follows:

Chapter 2 describes the CORD system and the mobile multimedia streaming environment for which an Application level Policy-based Handoff Control System is designed.

Chapter 3 discusses the concepts around policies based on the literature and their application in communication networks. We also discuss the properties of policies and policy specification approaches. Furthermore, we discuss a generic policy model that is commonly used in policy research community.

Chapter 4 describes the process of designing and introducing policies into the APPLE system. The section also provides examples of policies and a description of our approach in specifying those policies.

Chapter 5 presents the architecture of the APPLE system. We also discuss the related work in policy based control, handoffs in heterogeneous networks and automatic service selection.

Chapter 6 explains the implementation phase of the research and the results obtained on the test bed.

Chapter 7 gives the conclusions of this thesis and recommendations for future investigation in this research area.

2. MOBILE STREAMING ENVIRONMENT

This chapter presents briefly the CORD system that enables live-multimedia broadcast [2-4]. It's purpose is to provide a better insight in the requirements of the controlling entity, i.e. the APPLE system introduced in the previous chapter.

This chapter is structured as follows: Section 2.1 gives and overview of the CORD system and its environment. Section 2.2 explains a Lightweight Application Level Protocol of the CORD system and its interactions. Section 2.3 discusses how a channel configuration is specified in this research. Section 2.4 discusses an open issue in the CORD system, the issue that is a challenge of the current research.

2.1 Streaming via multiple content aggregators

Hesselman et al. [2-4] consider an environment that consists of application-level service providers that deliver real-time multimedia content (e.g. radio or TV broadcasts) to mobile hosts in the form of channels (e.g. CNN News). Two types of providers are distinguished in their work: content sources and content aggregators. A *content source* is the origin of one or more channels and distributes them to mobile hosts via one or more *content aggregators*. A content aggregator operates a pool of proxy streaming servers (Figure 1) and offers channels in a way suitable for wireless links and the limited capabilities of mobile devices, possibly by processing (e.g. transcoding [22,23]) the streams it receives from sources. As a result the users will be able to make use of a wider range of competitive offers from different aggregators via network providers, which have special agreements with these aggregators.

The proxy-like distribution scheme via aggregators increases scalability in the absence of IP multicast [24], which is important when channels need to be distributed to a potentially large number of receivers.

Figure 1 shows an example in which source cnn.com[1] distributes video channel CNN News via aggregators: stream-it.com and multimedia-forward.nl. User Bob receives CNN News either from media-forward.nl through the UMTS network of network operator connect-you.nl, or from stream-it.com through the 802.11 network of Twente.railway.nl. The solid line between stream-it.com and Twente.railway.nl indicates that stream-it.com is only available through the 802.11 network. Similarly, media-forward.nl is only available through the UMTS network.

An aggregator can deliver its channels in different *versions* (see [22,23]). This enables it to deal with different user requirements (e.g. pertaining to cost or quality) and to serve different types of hosts that connect to the Internet through different types of wireless links. We refer to the description of a version of a channel as a *configuration of a channel*. Each aggregator supports its own set of configurations of a channel. For example, in Figure 1 stream-it.com could support various high-quality configurations of CNN News channel (e.g. in 'TV' quality), while media-forward.nl could only support medium-quality configurations of the same channel (e.g. in 'videophone' quality). Mobile hosts can thus receive the same channel from different aggregators at different configurations, possibly through different interfaces (e.g. between points B and C in Figure 1).

[1] The domain names in this paper are for illustrative purposes only.

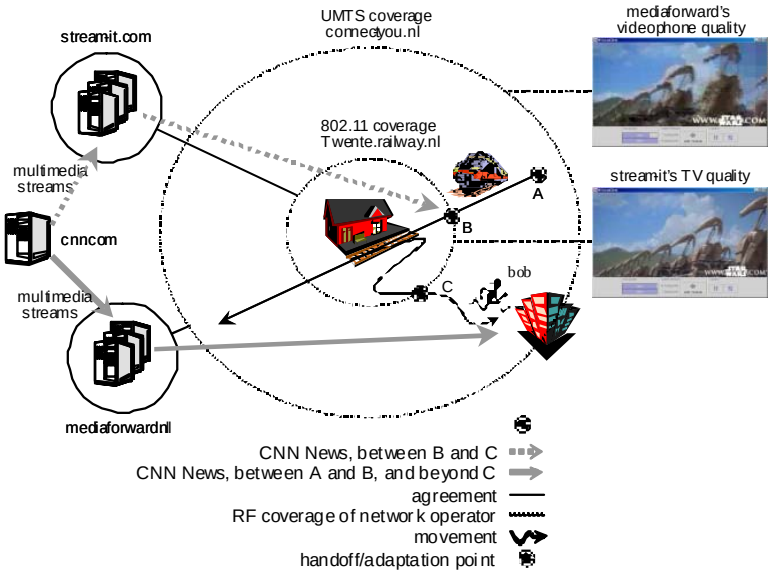

Figure 1. Streaming via multiple content aggregators

Hesselman et al. have developed the CORD system for the above mentioned environment. The CORD system enables mobile hosts to discover alternative aggregators that mobile host can reach, to connect to an aggregator and to handoff from one aggregator to another. The CORD system puts the responsibility of switching to another aggregator with the mobile host. The challenge in this environment is to maintain multimedia streams despite events that force mobile hosts to switch to another aggregator. When the user roams, the set of available configurations of a channel at the aggregator as well as the set of available aggregators and network operators might change. Mobile host must deal with these changes automatically and preferably transparently to the user. For example, consider the scenario shown in Figure 2:

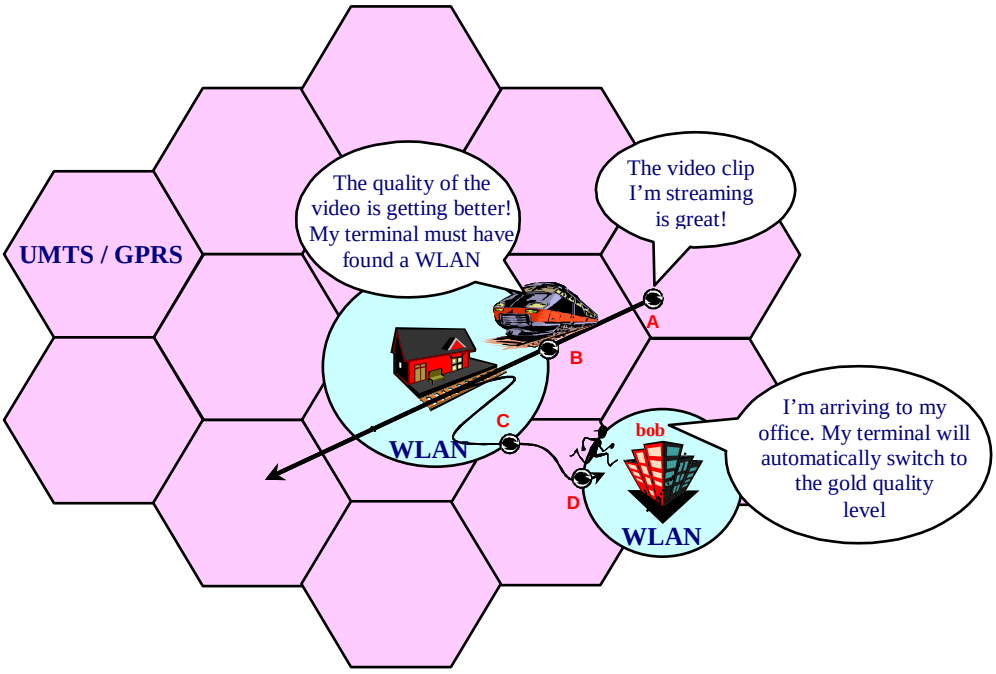

Figure 2. Scenario with a roaming user.

In this scenario, assume that Bob is coming from Utrecht to his office. He watches a video clip in the train (point A). His terminal is connected to the UMTS network. When the train enters the

train station area, the mobile terminal detects WLAN and switches smoothly to the WLAN interface without interrupting the video clip. Bob notices that the quality of the video clip became better (point B).

When Bob moves to the direction of his office, the WLAN at the train station becomes out of reach and his terminal switches back to the UMTS interface without disrupting the video clip. When Bob comes close to his office building, office WLAN becomes available to his terminal that will be automatically switched to the WLAN interface again, because it offers a better version of the same channel that Bob is watching.

During the entire trajectory, Bob's terminal adapts to Bob's movements and tries to provide uninterrupted video streaming. Bob only needs to provide his preferences (e.g. quality, price), and the rest is taken care by the mobile terminal itself. Bob's terminal selects the service according to the Bob's preferences in price and quality level, that of course should match with the available resources, such as available networks, available configurations of the selected channel, mobile terminal's local resources. To create such a self-controlling flexible system, we need a controlling mechanism for the CORD system. Because, the CORD system enables a mobile host to discover available alternative aggregators in the vicinity of the mobile host and to handoff to one of the alternative aggregators, but not yet flexibly, neither does it take user preferences into account. The controlling mechanism (the APPLE mechanism) , which will be developed in this work, provides this flexibility and additionally includes control based on user preferences.

The CORD system realizes discovery and handoff by means of a lightweight application-level protocol that builds on the Session Initiation Protocol (SIP) [5], which is an Internet standard. Section 2.2 discusses this lightweight application-level protocol in more detail.

2.2 Application level Protocol of the CORD system

The application-level protocol makes use of SIP transactions and SIP headers. The protocol has discovery and handoff service elements.

2.2.1 Discovery

Discovery service element enables a mobile host to request for configurations of a channel available at the aggregators it can reach. Figure 3 shows the protocol interactions when Bob's mobile host is querying media-forward.nl and stream-it.com at point B of Figure 1 to check which configurations of CNN News channel they support.

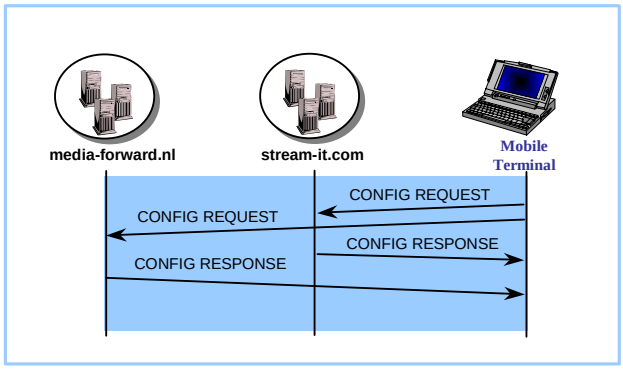

Figure 3. Protocol interactions for discovery of channel configurations at aggregators.

Bob's host sends a configurations request CONFIG REQUEST to stream-it.com via its 802.11 interface (twente.railway.nl), and a request to media-forward.nl through its UMTS interface

(connect-you). The aggregators respond to the mobile host with CONFIG RESPONSE, which contains the list of available configurations of the requested channel.

Triggers for discovery of alternatives

A mobile host may invoke the protocol for discovery when it is looking for a 'better' configuration of the channel it is receiving, for example when it *moves into a subnet*, where an aggregator with a better configuration may appear. The mobile host may also invoke the protocol for discovery when the user *moves out of a subnet*, which means that current aggregator may disappear, consequently the mobile host needs to discover an alternative aggregator to continue streaming. Other triggers that may invoke the protocol to make discovery of alternatives are the assignment of a (new) IP address to one of the host's network interfaces (e.g. to Bob's 802.11 interface at point B), increasing packet losses or decreasing of the signal strength on the network interface (e.g. at point C) and a change in user preferences.

2.2.2 Handoff

Handoff service element of the protocol comes into play, when the mobile host performs handoff from one aggregator to another. Handoff is performed by connecting to the corresponding proxy streaming server of the aggregator to which handoff is being executed. The Figure 4 shows the protocol interactions when the mobile host hands off to media-forward.nl by sending a DISCONNECT REQUEST to stream-it.com and a CONNECT REQUEST to media-forward.nl.

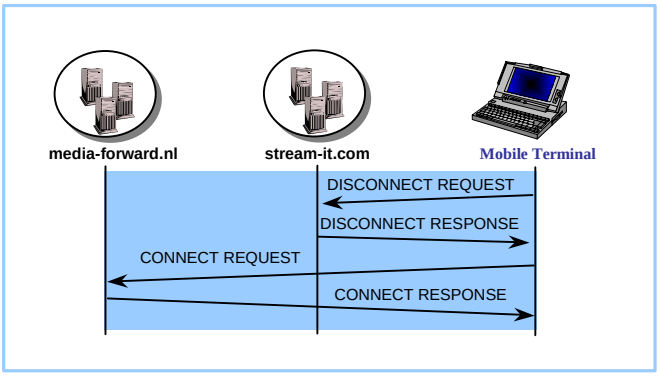

Figure 4. Typical protocol interactions for handoff.

The aggregator media-forward.nl sends CONNECT RESPONSE and begins to stream using the configuration of the channel selected by the mobile host. From this moment the mobile host continues receiving the same channel, being handed of to media-forward.nl (e.g. Figure 1, point C). The handoff between the aggregators: media-forward.nl and stream-it.com is similar to a handoff between base stations or access routers, except that it occurs at the application level, rather than at the network level or IP-level. We call it an application level handoff, because aggregators are application level entities and the handoff is executed from one configuration of the channel to another configuration of this channel, so it is a handoff between different configurations of the same channel. In the next Section we describe how we specified channel configurations.

2.3 Channel configurations at Aggregators

To provide flexibility to the users, aggregators need to be able to serve different types of mobile hosts that connect to the Internet through different types of wireless networks. Besides,

aggregators need to be able to deal with different user requirements regarding price and perceptual quality. For this reason, each aggregator supports a number of configurations of a channel that differs from the set of channel configurations offered by another aggregator in price and quality. This enables a mobile host to receive the same channel in different versions (supported by different configurations of a channel) through different aggregators and therefore the users can choose from a range of competitive offers. Therefore, the users might want to switch from one aggregator to another while they are receiving a channel. For example, the price of the channel configuration might be expensive for the user, so he might want to switch to another aggregator that provides the same channel configuration for a cheaper price. Another example is the quality of the channel configuration the user is receiving might be not satisfactory, therefore the user might want to switch to another channel configuration with better quality level etc.

In this research, we used the Session Description Protocol (SDP) [6] as a language to specify the configurations of a channel. These configurations are conveyed in the payload of SIP messages, sent by an aggregator to a mobile host, i.e. CONFIG RESPONSE in Figure 3.

Figure 5 illustrates an example of configurations of a CNN News channel, supported by stream-it.com and media-forward.nl specified in SDP.

CNN_News@stream-it.com.sdp

```
. . . . .

m=video 10000 RTP/AVP 31
a=rtpmap:31 H261/90000
a=fmtp:31 price=45; quality=90; framerate=30; bitrate=1024
a=fmtp:31 price=35; quality=75; framerate=25; bitrate=512
a=fmtp:31 price=25; quality=50; framerate=20; bitrate=256
a=fmtp:31 price=15; quality=25; framerate=15; bitrate=128
```

CNN_News@media-forward.nl.sdp

```
. . . . .

m=video 10000 RTP/AVP 32
a=rtpmap:32 H263/90000
a=fmtp:32 price=50; quality=70; framerate=20; bitrate=512
a=fmtp:32 price=40; quality=40; framerate=15; bitrate=256
a=fmtp:32 price=30; quality=20; framerate=10; bitrate=128
a=fmtp:32 price=20; quality=10; framerate=5; bitrate=64
```

Figure 5. Channel configurations at aggregators, specified in SDP.

Each line starting with "`a=fmtp`" together with the line starting with "`a=rtpmap`" in Figure 5, specifies a configuration of a channel. Therefore, in the example of Figure 5, there are eight configurations of the CNN News channel are specified, four of them are supported by stream-it.com and four of them by media-forward.nl. *A configuration of a channel* specifies QoS parameters, i.e. framerate (a value representing the number of frames one second of media stream contains), quality (a relative value in % representing e.g. the number of pixels in one frame), bitrate (a value in kbps representing required bitrate for this media stream). It also specifies its price (in cents per minute). As you can see from Figure 1, in the hotspot between points B and C the user has a choice of eight different configurations of the same channel, because in the hotspot the mobile host can reach both aggregators.

2.4 An open issue in the CORD system

An open issue in the CORD system, is that it does not include a (flexible) mechanism that enables mobile hosts to automatically invoke discovery of available aggregators, to select the best aggregator, and then to switch to that aggregator. Such a mechanism is important, because the

complexity of the infrastructure (in terms of aggregators and networks operators) should be shielded off from the user to improve user-friendliness [7].

In this research we designed an *Application Level Policy-based Handoff Control System* (the APPLE system) for the CORD system. The APPLE system is designed as a controlling entity for the CORD system. The APPLE system uses policies to flexibly control the CORD system when to discover and how to switch between aggregators in order to achieve service continuity and service adaptation. In addition, the APPLE system selects among discovered alternatives of the channel configurations *the best*, which matches user preferences, network capabilities and service facilities.

Design of the APPLE system will be described in Chapter 4 and Chapter 5. First, in the next chapter we address the concepts of policies and a policy model that we used for design of the APPLE system.

3. POLICIES AND A POLICY MODEL

As discussed in Chapter 2 the CORD system requires a flexible mechanism to enable a mobile host to adapt to environment changes (e.g. due to roaming). We follow a policy-based approach to design and develop such a mechanism (i.e. the APPLE system). This chapter reports our literature study on policies and policy models. It is presented in a way containing our interpretation and useful in the design of the APPLE system. The readers who are familiar with policy systems may skip reading this chapter and directly read Chapter 4.

This chapter is structured as follows: Section 3.1 gives a brief introduction on efforts in the policy research community on defining the concepts for policy-based systems. Section 3.2 elaborates on the concept of a policy. Section 3.3 describes a generic policy model commonly used in communication networks.

3.1 Introduction

Our research is towards designing a mechanism for the CORD system that would control the behavior of the mobile host, so that it maintains the multimedia streaming while user roams across overlay networks. Our approach is to use policies to achieve this control over the mobile host.

Policies are commonly used in network management area to control or to manage a system. Policy-based control and management is considered more flexible, because the systems can adapt to the rapid changes in the environment, in management area for instance by run-time reconfiguration, without re-engineering [12,25,26].

The literature however shows that policy-based control and management is still a research topic. The meaning of the term “policy” is not clearly defined yet, nor consistently used by different authors and communities (i.e. ITU, IETF, DMTF and OMG) [25]. Although, there are strong similarities in the concepts and techniques used by the different communities, there is no commonly accepted terminology or notation for specifying policies [26].

In this research, we focus on the policy based control of the behavior of the mobile host. Similar to policy-based management systems, a policy based control system typically contains a controlling entity and a controlled entity (Figure 6), see also [25].

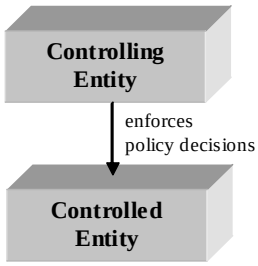

Figure 6. Controlling Entity and Controlled Entity.

The controlling entity controls the behavior of the controlled entity using policies. Polices under certain predefined conditions enforce policy decisions on the controlled entity (Figure 7). In the following Section we discuss the definition of policies, their properties and specification.

3.2 Policies

3.2.1 Definition

The literature shows that different authors give different definitions for the term "policy". For example:

- "Policies are administratively prescribed rules that specify actions that have to be performed in response to defined criteria" [27];
- "Policy rule is the binding of a set of actions to a set of conditions – where conditions are evaluated to determine whether the actions are performed" [11].
- "Policies are rules governing the choices in the behavior of the system" [12];
- "A Policy is formally defined as an aggregation of policy rules each of which has a set of conditions and a set of actions" [25];
- "Management policies describe the rules that must or may be applied to active nodes in order to force a desirable service quality" [28];

From these examples we can conclude that a policy is a rule. We adopt the concepts of IETF [11] and follow IETF's policy model [18], which however focuses on admission control policies at network level. Therefore we also generalize the concepts towards their use in the control area at application level. Our description of a policy is therefore as follows:

Policy is a rule that describes Actions to be taken when certain Conditions happen.

The *action part* of the policy determines a desired behavior of a system, typically represented by the controlled entity. The condition part of the policy defines a condition under which this desired behavior should be enforced.

In our work, *policy conditions* contain changes in packet loss, signal strength values on the network interfaces of the mobile host, dynamical assignment of a new IP address to a network interface of the mobile host during roaming and availability of a better alternative configuration of a channel among discovered ones etc.

The *action part* of the policy may determine a decision to accept or reject a service request. In general, the action part of the policy may also contain the decision to fetch other policies from a policy server [13,27] or to evaluate some other policies. Because actions of a policy may result in some post conditions which match with the conditions of other policies, therefore causing a chaining effect of policy actions [25].

Similar to the literature in the management area [18,25,26], we can refine the description of a policy by introducing a new element, i.e. a policy goal. *A policy goal* describes the intention strived by the policy actions. The goal of a policy in our system will reflect *what* we want to control in the behavior of the mobile host. Therefore, we can define a policy as

an if (condition) then(action) rule with a goal .

In the literature, especially in the management area, other elements of policies besides goals, conditions and actions, have been introduced, for instance *policy targets* (entities where the actions are enforced), *policy subjects* (entities where conditions are evaluated) and *triggers* (entities that initiate the evaluation of the policy conditions) [16,25,28,29]. Since our controlling and controlled entities both reside on the mobile host and the mobile host is the only entity of our design concern, we do not include subjects and targets in our policy definition.

In our policy based system, policy decisions, i.e. the actions are enforced on the controlled entity by controlling entity. The controlling entity is the APPLE system and the controlled entity is the CORD system. Therefore, Figure 6 applied in our case can be seen as shown in Figure 7:

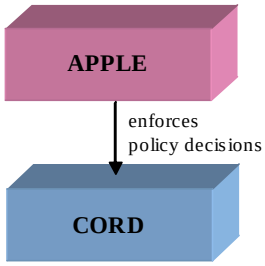

Figure 7. The APPLE system controls the CORD system using policies.

3.2.2 Properties of policies

Policies are widely used because of their advantageous properties, however they also have some disadvantages. In this Section we summarize these properties, i.e. persistency, flexibility, hierarchies, types, classes and conflicts.

Persistency

Policies are persistent and meant for persistent monitoring of system's behavior [25,26]. Therefore, policy based control differs from the one-shot *if-then* or *event-condition-action (ECA)* commands, used in state machine specification or agent based systems [30]. ECA commands will result in an immediate action being taken by a system, but once the action is taken the command has been completed and often stops to exist, whereas policies control the system's behavior during its lifetime persistently. This means that policies continuously affect the behavior of the system, until they are revoked or replaced by other policies. This also implies that policies continuously monitor the system's or its environment's behavior [e.g. 12].

Flexibility

Flexibility of policies is the most important reason why many researches tend to use the policy based systems. Because policies help to flexibly control the system in dynamically changing environment. By applying policies the system's behavior can be adapted to the environment changes. Introducing new policies into the system may result in a system's new observable behavior.

We distinguish three forms of flexibility that may exist in the policy based systems:

- *adaptive behavior of the system:* because a set of actions (i.e. policy decisions) is determined by a set of conditions, which reflect different situations;

- *flexible control of the system's behavior*: because the conditions and the actions (i.e. policy decisions) can be tuned at runtime. Therefore the behavior of the system can be flexibly altered by dynamically updating the policy rules [see also 14].

- *flexible evolution of the system:* because new policies can be introduced (e.g. downloaded) while the policy based controlled system is in operation. Therefore, a policy based system is flexible as it does not require stopping or reengineering after new policies have been downloaded into the controlling entity.

Policy hierarchy and refinement

Depending on how policies are presented, what level of details they provide, they can be placed in different levels of abstraction [15,25]. In this way, we can observe a policy hierarchy. Benefits of this hierarchy is that administrator can easily control the system, defining high level policies, which are interpreted by the policy-based system into low-level machine executable policies.

An illustration of policy refinement from abstract policies into concrete policies is presented by Cox and Davidson in [25]. They give an example of a possible policy hierarchy by the following picture:

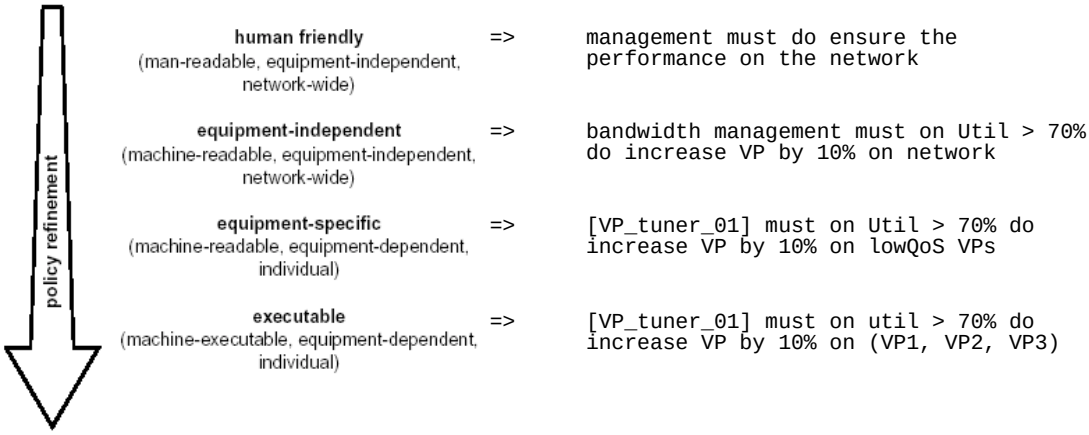

Figure 8. Policy hierarchy and refinement [25].

Human-friendly policies represent the highest level of the policy hierarchy shown in Figure 8. On this level policies are expressed in a form easy to read for humans, abstracting from network equipment details, the lower level and equipment specific information. The example on the right side is a human friendly high level abstract policy.

Equipment-independent policies are network-wide applicable and abstract from the equipment, but not easy to read for humans. The example on the right side illustrates the refined policy that contains lower level details, such as bandwidth management, Util (i.e. utilization) and VP (i.e. virtual path).

The lowest level in the hierarchy shown in Figure 8 accommodates *executable policies* –policies with the actions to execute a code in a target system (i.e. the controlled entity, Figure 6).

In conclusion, policy hierarchy can be considered as a means to hide the system complexity by bridging the gap between the high level (administrative, business or other) goals and network level configuration [26]. Having high-level user friendly policies a policy-based system provides the network administrator a simple high-level view of the complex low-level policies.

Even in a non-hierarchical policy-based system, the policy goal which reflects higher level aim, can be used to select the relevant policies for specific situations.

Policy classes

Policies can be classified according to their goals. Policies of different classes therefore differ from each other by their use (usage area) or intent. In policy based systems, the control can be performed by several interleaved and simple (i.e. not complex) policies, each of them classified in respect to a concern.

Calo and Sloman [12] describe two classes of policies found in the adaptive systems: management policies and security policies. Management policies are related to the network management, for example resource allocation strategy for a particular user. Security policies contain authorization policies that are used to define what services or resources can be accessed by user etc. PCIM [31] defines policy classes that can be used for different purposes: Motivational, Configuration, Installation, Error and Event, Usage, Security and Service policies. OMG permits only two classes of policies – Initialization policies and Validation policies [25].

In our system, we define two policy classes: Discovery and Handoff. More details can be found in Chapter 4.

Policy types

ODP's enterprise viewpoint introduces the following types of policies [25,32,33]: Permission, Obligation and Prohibition. According to the ODP Reference Model [33]:

- Permission policy contains actions, prescribing that a particular behaviour is allowed to occur, but there is no obligation for this behavior to occur.

- Obligation policy contains actions, prescribing that a particular behaviour is required, so this behavior must occur.

- Prohibition policy contains actions, prescribing that a particular behaviour must not occur. This is equivalent to the obligation for this behaviour not to occur.

In our policies, whenever the condition of the policy becomes true, the CORD system must execute the action part of the policy. Therefore, our policies are Obligation policies.

Policy conflicts

Policies need a careful design, because the actions of policies enforced on the system may conflict each other [25]. It may happen, for example when the condition or the action parts of different policies overlap. Conflicting policies running in the system may result in inconsistency of system's behavior, because one policy may negate the effect of the other policy, so that the controlled system might not behave as it is expected to behave and even might crash.

The policy-based system itself should be designed in a way, that it would know how to behave in the case of conflicting policies. For example, using means of detecting and resolving conflicts, the system may go to the well-behaved error state where some conflict resolution function corrects the policies, or some higher level decision entity resolves the conflict [see also 13,14].

In our work, the policy evaluation may execute a chain of policies, e.g. when the outcome of one policy action serves as condition for another policy. But the APPLE policies are simple and not conflicting. However, policy conflict detection, analysis and resolution are beyond the scope of this research.

3.3 Policy specification approaches

"… there are no success stories around policy specification languages…" [34].

Several specification languages have been developed. However, currently there is no standard language for specifying policies [34], there are only some draft proposals, each concerning a different application field. N. Damianou et al. [14] provide a nice overview of different approaches in the specification of management and security policies.

The abstraction level of a policy may influence the suitability of a language. Cox and Davison [25] provide an example of a high level abstract policy and derived from it a low level policy, which can be enforced by the controlling entity, i.e. Virtual Path tuner in ATM network. The abstract policy is on ensuring network performance in ATM networks and is as follows:

The management must do ensure the performance on the network.

The associated policy at a lower level is specified as follows:

(VP_tuner_01)	*must on*	*Util > 70%*
	do increase	*VP by 10%*
	on	*(VP1, VP2, VP3).*

This policy reads: the Virtual Path tuner 1 must increase the bandwidth allocation to 10% on Virtual Paths 1 - 3, if the utilization of the available bandwidth becomes more than 70 %. We can observe that the "*must on Util >70%"* corresponds to the "*if (condition)***"** part and "*do increase VP by 10%*" corresponds to the "*then (action)*" part of our policy description. In this specification *(VP_tuner_01)* is the subject part and *(VP1, VP2, VP3)* are the target parts of the policy.

We discuss some of the IETF approaches ([11,35]), Ponder [36] and the XML approach [28] to specify policies.

3.3.1 IETF approach

As in most policy specification approaches, IETF describes a policy as an ***if<condition(s)>*** ***then<action(s)>*** rule [11]. The condition part of the policy can be simple or compound expression, the action part of the rule can be a set of actions that must be executed when the conditions are true. This type of IETF policies have semantics similar to an obligation type of policies but does not explicitly specify triggers, subjects and targets. An example of a simple IETF policy that can be specified by an administrator is given in [14]:

```
if((sourceIPSubnet = 240.0.0.0/240.0.0.0) AND
   (timeOfDay = 1800-2300) AND
    (dayOfWeek = Monday))
 then set Priority:= 5
```

This rule can be interpreted, as follows: traffic for the corporate management sub-network gets priority level 5 on Monday nights from 6:00 pm to 23:00 pm (e.g. for important sports broadcasts).

IETF has also proposed a Security Policy Specification Language (SPSL). It is a limited language designed for communications security policies. The policy actions specified in SPSL can be to permit, deny and forward the communication [62], which are Permission type of policies.
However, in this report we focus only on Obligation type of policies.

3.3.2 Ponder

Ponder is a language defined at Imperial College for specifying policies [36] and is popular in policy research community [16,37-39]. It is a declarative, object-oriented language that can be used to specify management and security policies.

Lytfiyya et al. [16] use Ponder for specifying policies on QoS requirements. An example of a QoS requirement for a multimedia application that receives a video stream is the following: "The number of video frames per second displayed to the user must be at least 25 plus or minus 2 frames". The application QoS policy realizing this requirement can be specified in Ponder in the following way:

```
1   oblig NotifyQoSViolation {
2     subject VideoApplication/qosl_coordinator
3     target fps_sensor, jitter_sensor, buffer_sensor, QoSHostManager

4     on  not (frame_rate = 25(+2)(-2) AND jitter_rate < 1.25)

5     do  fps_sensor->read(out frame_rate);
6         jitter_sensor->read(out jitter_rate);
7         buffer_sensor->read(out buffer_size);
8         QoSHostManager->notify(frame_rate, jitter_rate, buffer_size);
9   }
```

This is an Obligation policy on notifying the QoSHostManager by the violation of QoS. The **subject** is the real actual application that the policy applies to and it will have the responsibility for the policy. `VideoApplication` refers to the name of an application, `qosl_coordinator` referes to the component that evaluates the conditions stated in policies at run-time. The **targets** of the policy are sensors that monitor the `frame_rate`, `jitter_rate` and communication `buffer_sizes` and the `QoSHostManager`.

The line 4 starting with the term "**on**" is the condition of the policy. In this policy, if the value of the `frame_rate` attribute is less than 23 or greater than 27 or the `jitter_rate` is less than 1.25, then the policy is considered to be violated.

The line 5 starting with the term "**do**" specifies the actions of the policy, that are to be carried out when the policy has been violated. The actions of this policy are specified in the lines 5-8. The actions to be taken are: to read and notify QoSHostManager about the frame rate, jitter rate and buffer size.

Ponder is also used by Lymberopoulos et al [37] for specifying Obligation policies for network services managers. Montanari et al. [38] propose Policy-Enabled Mobile Applications (Poema) framework, for which they specify reconfiguration strategies in terms of Ponder Obligation policies.

3.3.3 IRML

The Intermediary Rule Markup Language (IRML) is an XML-based language to specify policies [35,55]. It was proposed in IETF Internet drafts from the Working Group Open Pluggable Edge Services. IRML can be used to describe service-specific execution policy rules for network edge intermediaries. It allows clients, access providers and content providers to specify when and how to execute intermediary services.

The following is an example of a policy rule, which is specified in IRML for HTML page adaptation to the allocated bandwidth on the client host.

```
<rule processing-point="4">

    <!-- checks the allocated bandwidth, adapts accordingly -->
```

```
    <property     sub-system=”QoS”
                  name="allocated-bandwidth"
                  matches="<9600"  >

       <!-- Bandwidth is low, no image  -->
       <action>proxylet://localhost/html2wml?image=no</action>
    </property>

    <property  sub-system=”QoS”
                  name="allocated-bandwidth"
                  matches=">=9600" >

       <!-- Bandwidth is high, can embed image  -->
       <action>proxylet://localhost/html2wml?image=yes</action>
    </property>
 </rule>
```

In this specification approach the conditions of the polices are specified within <*property*> tags, and actions of the policies are specified with <*action*> tags. Policies do not explicitly specify subjects, targets and triggers.

In this example, the allocated bandwidth on client host is used to determine how the HTML page is translated to WML page. This example illustrates two policies. The first policy reads:

If the allocated bandwidth is smaller than 9.6 kbps, the translated WML page will not contain bitmaps, i.e. bitmaps will be replaced by alternate text.

The second policy reads:

If the allocated bandwidth is large than 9.6 kbps, the translated WML page will contain some bitmaps.

3.3.4 XML

Liabotis et al. [28] express the policies of their system in XML. Their example policy for enforcing the resource allocation according to the user requirements is the following:

```
<policy>

  <creator>
    <authority>
      <domain>USER</domain>
      <role>ApplicationProvider</role>
    </authority>
    <identity>/VideoApplication/ADMIN</identity>
    <reply_address>128.40.40.18</reply_address>
  </creator>

  <info>
    <name>Resource Allocation for Image Transcoder</name>
    <modality>Obligation</modality>
    <servicelevel>
      <delivery>Guaranteed</delivery>
      <execution>Optional</execution>
    </servicelevel>
    <priority>1</priority>
  </info>

  <subject var = "RAC">
    <subjectlist>ResourceManager</subjectlist>
  </subject>

  <trigger>
```

```
    <event>
      <source>USER</source>
      <category>ResourceManagement</category>
      <eventtype>ProxyletLoad</eventtype>
    </event>
  </trigger>

  <actions>
    <action>
      <target>ResourceAllocationController</target>
      <method>AllocateCPU(b2jlet,25%)</method>
    </action>
  </actions>

</policy>
```

The policy information, including a policy name, modality and a service level is given after the creator tags describing the creator details. This policy specification approach explicitly defines tags for a policy <*subject*> (e.g. Resource Manager), a policy <*target*> (e.g. Resource Allocation Controller), a <*trigger*>, an <*event*> and the <*actions*>. However, there is no <condition> tag in this policy, the condition of the policy is placed into <*eventtype*> tag. In this specification, the receipt of an event is considered as the condition to trigger the policy.

The Resource Manager (i.e. subject) on the receipt of the event with the type "ProxyletLoad", invokes the “AllocateCPU” method on the Resource Allocation Controller (i.e. target). As an enforcement of this policy, the Resource Allocation Controller allocates 25 % of CPU on “b2jlet” proxylet.

3.3.5 Summary

We have discussed some approaches, i.e. IETF approach, Ponder language and XML for specifying policies. In these examples we observe that all specification approaches specify conditions and actions, although *conditions* in some of them (i.e. IRML) are named differently.
The following table shows what elements of a policy are specified in these approaches:

	IETF	Ponder	IRML	XML
Condition	+	+	+	+
Action	+	+	+	+
Subject	-	+	-	+
Target	-	+	-	+
Event	-	+	-	+

Table 1. Comparison of policy specifications.

In the discussed examples, policies are written to achieve a certain control, namely:
1. The IETF’s policy example given in this Section *controls* the priority settings to certain traffic at certain periods of a week.
2. The Ponder example is a policy that *controls* the violation of QoS requirements of a certain media application.
3. In the IRML example [35], the policy *controls* the HTML page adaptation to the allocated bandwidth on the client host at translating this page to a WML page to display it on the client host.
4. The XML policy [28] *controls* the CPU allocation by Resource Manager to running processes in the system.

A policy specification approach is important to facilitate the integration of policies into the system and their implementation. We discuss our approach to specify the APPLE policies in Chapter 4.

3.4 A Policy Model

3.4.1 A generic policy model

Our literature study gives us an overview of a generic policy model used by the policy research community, most of them from the management area, Figure 9.

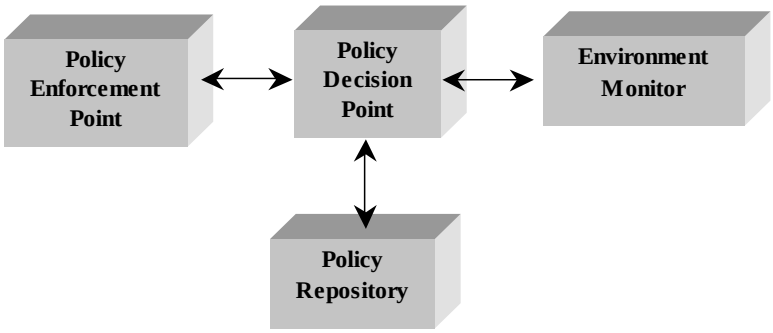

Figure 9. A generic Policy Model

In our research, we identify the following components of the policy models typically used in most (Obligation type) policy based control and management systems:

- a functionality responsible for making policy decisions - *Policy Decision Point* (PDP);
- a functionality responsible for executing policy decisions - *Policy Enforcement Point* (PEP);
- a facility for storing policies (rules) - *Policy Repository* (PR);
- a functionality responsible for monitoring the environment - *Environment Monitor* (EM).

In this research, we follow the IETF's policy model [18] which uses the concepts of a Policy Decision Point (PDP) and a Policy Enforcement Point (PEP).

A *PDP* represents a controlling entity that applies policies to control the behavior of a controlled entity (the PEP). The PDP evaluates policies. If the circumstances are such that the "if" condition of a policy becomes true, then the PDP decides to enforce the actions defined in the "then" part of the policy.

A *PEP* represents the controlled entity upon which policy decisions are being enforced (by the PDP), yielding a constrained behavior of the PEP. A PEP therefore receives directives from a PDP.

A *PR* contains (inactive) policies written in a policy specification language such as IRML [35]. A policy repository allows policies to be flexibly downloaded into a PDP, possibly at run-time. Therefore, while inactive policies reside in the PR, the active (i.e. running) policies reside in the PDP. Policies running in the PDP can be replaced by downloading new policies from the PR. Policies can be downloaded according to a certain criteria (e.g. a goal of the policy). The PDP and the PEP together realize the goal of a policy the PDP retrieves.

A PDP can generally use external information sources to come to its decisions [18]. The external information source in the policy model is *Environment Monitor* (EM). It is responsible for monitoring available resources, such as availability of networks, available bandwidth, signal strength on the network interfaces etc. The PDP accesses this information by requesting it or by listening to events from the EM.

Location/distribution of the components.

The components PEP, PDP, PR and EM can be co-located (local) in one network node or distributed (remote) across the domain(s). A PEP resides in the local node, because it enforces the policy decisions on that node.

There are two types of PDP defined in the IETF's policy model: Local PDP (LDP) and Remote PDP. A Local PDP is located in the node and is responsible for making local policy decisions, which depend on information and conditions local to that particular node. If a PDP is located in another node, it is called a Remote PDP. For instance, Remote PDP can reside at a centralized policy server. If so, it could be responsible for policy decisions for multiple network nodes of an administrative domain and evaluates policies that are common across the administrative domain. In this case, its policy decision might be enforced in several PEPs in this domain. This is convenient, for example for network administrators to reconfigure several nodes in one shot with a single policy. Remote PDP can convey the policy decision to a PEP at the controlled node using COPS (Common Open Policy Service) protocol [40-42] or some other proprietary protocol.

A PR can be local or remote, similar to a PDP. A remote PR can store policies for only one single node or for the entire administrative domain. If the PR is remote, e.g. resides in a centralized node of the administrative domain, then it is possible to consistently apply the policies to all nodes in this domain. Policies can be downloaded from the PR to the PDP any time when required. To download policies a PDP uses a protocol, for example LDAP (Light-weight Directory Access Protocol) or some other proprietary protocol.

An EM can also be local or remote. A remote EM can be exemplified by a Bandwidth Broker in the UMTS domain that monitors the load in the domain. A local EM can be a component that resides on the mobile host and monitors events local to the mobile host (e.g. events coming from the network interfaces, battery power etc.).

Interaction of components

Different choices in interaction patterns between components and in distribution of components of the policy model leads to different implementation scenarios of policy-based systems built according to the generic policy model illustrated in Figure 9.

There are two interaction patterns, that can be modelled between any of two components of the policy model shown in Figure 9: Polling and Notification. We discus the interaction patterns in the example of PEP and PDP.

In the policy model presented earlier, the basic interaction between the components (Figure 9) begins when PDP receives an event from any of other components and this event triggers a policy decision. PDP by the receipt of the event, evaluates the conditions of the policies, and if a condition of some policy becomes true – this "policy fires". PDP produces a policy decision containing the actions to be executed and pushes the decision to PEP. PEP then enforces this policy decision, i.e. executes the action of the fired policy. This interaction model corresponds to the Notification model, Figure 10.

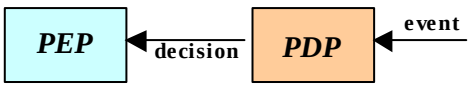

Figure 10. Notification model of interactions between PDP and PEP.

In the Polling model, a PEP receives an event that requires a policy decision. The PEP then polls the PDP for such a policy decision. The PDP in its turn evaluates the policies associated to the request (or event) and produces the decision, which is then is enforced on the PEP (Figure 11).

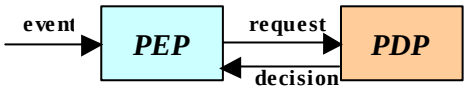

Figure 11. Polling model of interactions between PDP and PEP.

In this thesis we use the terms “Polling” and “Notification”, although in the literature these interaction patterns are called “Outsourcing” and “Provisioning” interaction types [43].

In the next section we discuss how the generic policy model is applied in a distributed environment.

3.4.2 Application of the generic policy model in a distributed environment

A policy model for admission control

The generic policy model (Figure 9) has a broad applicability in the literature. Figure 12 shows the distribution of the components of a policy model, applied in the network nodes for providing a policy control over the admission control in the RSVP routers [18].

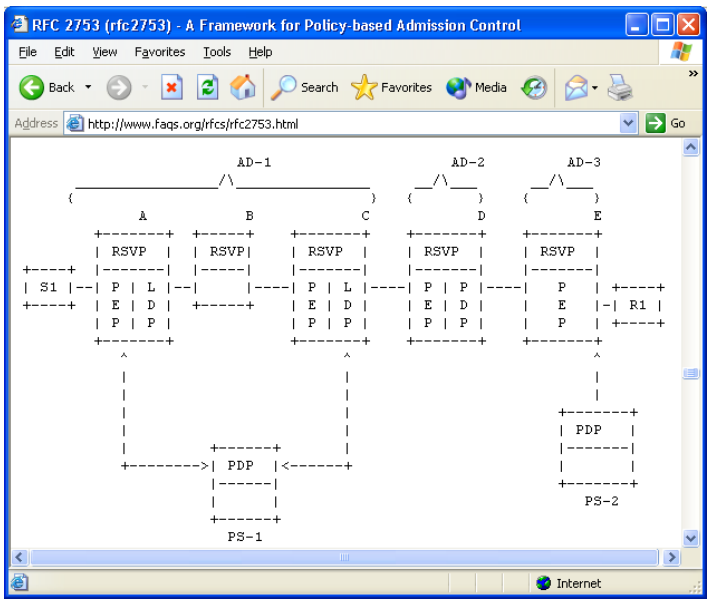

Figure 12. Placement of the policy model components in the network nodes [RFC 2753].

Figure 12 shows the path between the sender (S1) and the receiver (R1). All nodes (routers) in the picture support RSVP and reside in Administrative Domains identified with abbreviation “AD”.

In Figure 12 there are different nodes are exemplified in respect to policy awareness. Consider AD-1 with the nodes: A, B and C and AD-2 with a node D. The nodes A and C have co-located PEP and LDP (Local PDP). This LDP can only produce a partial policy decision (see [18]), therefore the PEPs of these nodes in AD-1 will query a centralized PDP (at PS-1, i.e. Policy Server 1) for a final policy decision. The Node B of AD-1 does not have any PEP, it is not aware of policies therefore does not support a policy control. The node D at AD-2 has co-located PEP and PDP. Because this PDP produces a final policy decision, the PEP does not query an external PDP for a policy decision.

Implementation of a policy model for a network node

IETF's policy model discussed in [18] is a generic policy model, without defining an implementation details. Figure 13 illustrates a Policy Implementation Model proposed in [27].

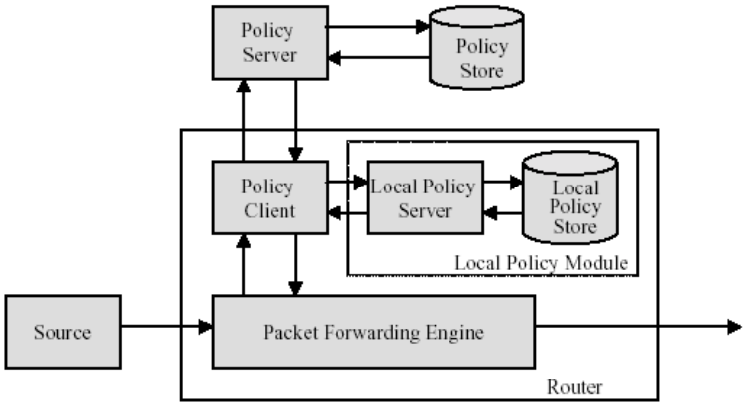

Figure 13. Policy Implementation Model [27].

Figure 13 illustrates the implementation scenario of the policy model that resides in the router and controls the packet forwarding through this router. In this model, the Packet Forwarding Engine acts as a *PEP*, Policy Server acts as a (remote) *PDP*, and Local Policy Server acts as a *Local PDP*. The Policy Client is an intermediate component that forwards requests for a policy decision to the Policy Servers and forwards policy decisions to the Packet Forwarding Engine.

Although, the proposed model conforms to the policy model defined in [18], it also defines new architectural elements and concepts. IETF's policy model [18] does not explicitly define other components besides PEP and PDP (see also Figure 12). The Policy Implementation Model illustrated in Figure 13 introduces two new functional elements: a Local Policy Store and a remote Policy Store. The policies in the Local Policy Store are retrieved by the Local Policy Server, and the policies in the distributed Policy Store[2] are retrieved by the (remote) Policy Server.

Implementation of a policy model for a multi domain scenario

Zhuang et al. [13] applied the generic policy model for the multi domain, multi technology scenario, Figure 14. Their work is on the cooperation (integration) of UMTS and WLAN environments using policies for QoS management. The authors use the concepts defined in the policy model of [18], namely PDP and PEP. Their Policy Decision Function (PDF) is a PDP and Policy Enforcement Function (PEF) is a PEP (Figure 14). Their policy decision is on enabling handoff of a mobile host from one network interface (e.g. WLAN) in one policy domain to

[2] In our research, we use the term "Policy Repository", instead of a "Policy Store".

another interface (e.g. UMTS) in another policy domain. One of their examples illustrates the integration of two policy domains under one PDP, called Master PDF (MPDF, Figure 14). The MPDF is a final policy decision making point for the policy decisions involving the integration of two domains. It is situated on the higher level of hierarchy. The MPDF is also responsible to resolve policy conflicts if any. If there is insufficient information for a policy decision or if there is a policy conflict to be resolved, the PDF can retrieve additional policies from the Policy Repository.

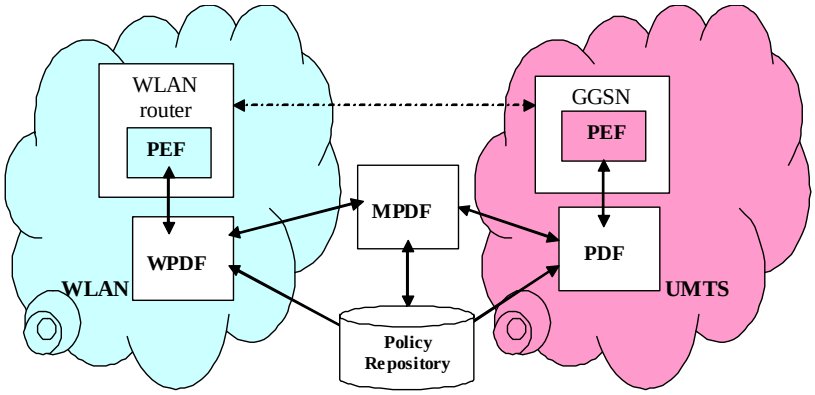

Figure 14. WLAN and UMTS policy domains integrated under the same operator

3.5 Summary

In this chapter based on literature study we have discussed the concepts and models of policy-based systems used in the policy research community. Having investigated the structure of a policy model and how the policy model can be applied, we have proposed a generic policy model (Section 3.4.). In the following chapters, based on this policy model we design the APPLE system and policies for the APPLE system to validate the model that we have proposed.

4. THE APPLE SYSTEM POLICIES

The APPLE system is designed as a controlling entity for the CORD system. The APPLE system uses policies to control the behavior of a mobile host in the environment discussed in Chapter 2. In this chapter we discuss the requirements to the APPLE system, the structure of the APPLE policies and their specifications.

This chapter is organized as follows: Section 4.1 discussed the requirements to the APPLE system. Section 4.2 describes the high level behavior of the mobile host. Section 4.3 introduces the policies of the APPLE system. Section 4.4 presents the approach that we used in this research to specify these policies. Section 4.5 presents conclusions of the chapter.

4.1 Requirements

As it is discussed in Section 2.1, an open issue in the CORD system, is that it does not include a (flexible) mechanism that enables mobile hosts to automatically invoke discovery of available aggregators, to select the best aggregator, and then to switch to that aggregator. Such a mechanism is important, because the complexity of the infrastructure (in terms of aggregators and networks operators) should be shielded off from the user to improve user-friendliness. The CORD system puts the responsibility to discover alternatives and to switch between aggregators with the mobile host; thereby putting control close to the user but in a user friendly manner. Therefore in designing a flexible control mechanism for the CORD system, our main design choice is to be inline with this choice and centralize our control at the mobile host.

The overall goal of the APPLE system is to enable seamless handoffs between aggregators, as it is shown in the scenario with the roaming user in Figure 2. To be able to control the CORD system flexibly, the APPLE system must meet the following requirements:

1. Respond adequately to resource monitoring events that might require a handoff.
2. Initiate the moment of discovery.
3. Automatically select the best aggregator or network, based on the user preferences.
4. Initiate the moment for handoff in a way conforming to the user preferences.

Based on these requirements we design the policies and the architectural components of the APPLE system. The design of the APPLE system architectural components will be discussed in Chapter 5.

This chapter describes the design of the policies, but first we discuss a high level behavior of the mobile host controlled by the APPLE policies.

4.2 Policy-controlled Host Behavior

Figure 15 shows the high-level behaviour of a mobile host in the form of an FSM. Figure 15 also shows the embedding of the APPLE policies in the states of the FSM.

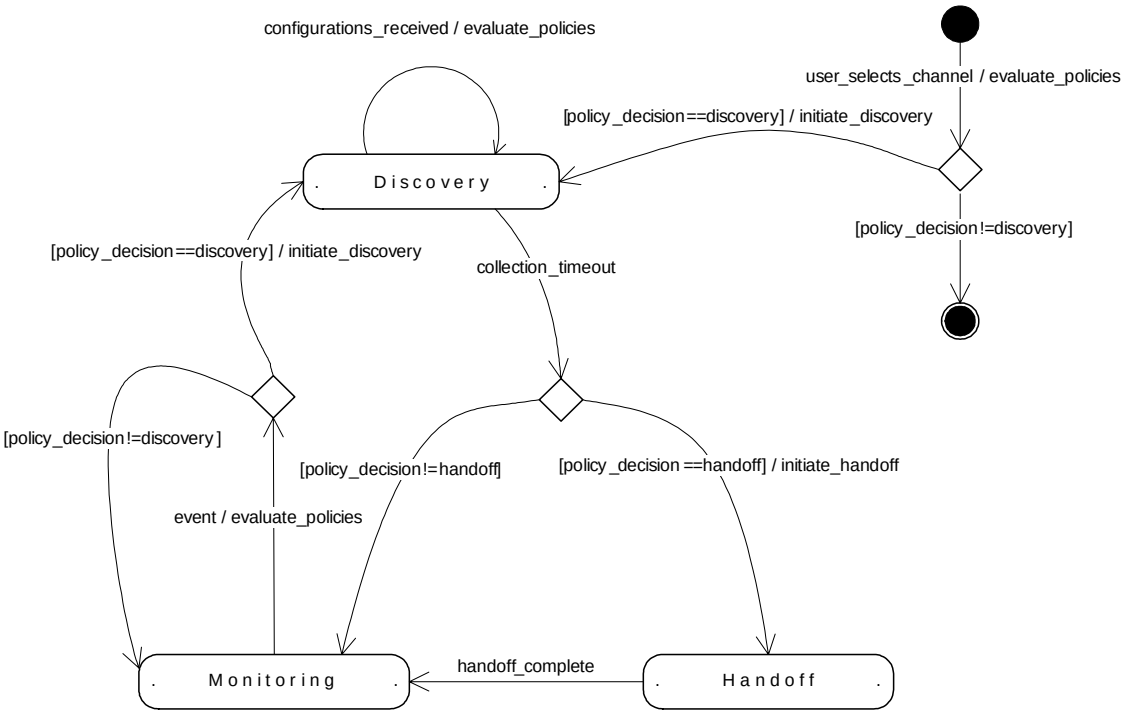

Figure 15. High level behavior of the mobile host

The FSM has 5 states: an initial state, a monitoring state, a discovery state, a handoff state and a final state.

Initial state

In the Initial state, the user selects a channel and requests to connect to this channel. By receipt of this event, the APPLE system evaluates its policies, to determine if the mobile host should discover the aggregators offering this channel. We call these policies *discovery policies.* If the policy decision is *discovery* then the mobile host initiates discovery: by running the discovery protocol to discover what configurations of the channel are available at the aggregators, i.e. sends queries to available aggregators (see Figure 3). By sending queries the mobile host sets the timer to collect responses from the aggregators and moves from initial state to discovery state. We call the time set to collecting responses *a collection time.*
It the policy decision is not *discovery*, then the mobile host, i.e. the FSM moves to the final state.

Discovery

In this state the mobile host collects the responses from the aggregators that contain the configuration descriptions of the selected channel (see Figure 3). By receipt of a configuration list from an aggregator, the APPLE system evaluates its policies, to determine if the mobile host should handoff to another aggregator (i.e. move the FSM to the handoff state). We call these policies *handoff policies.* During the policy evaluation process, each configuration in the list is processed and the best configuration that matches the user preferences (i.e. price, quality) and available resources (e.g. interface status) will be selected. the APPLE system produces a policy decision indicating to which configuration of the channel the mobile host should handoff.

When collection time expires, the mobile host moves to the next state. Depending on the policy decision the next state can be handoff or monitoring. the APPLE system produces a policy

decision to handoff, if there is a better configuration available than the current one (in terms of price or quality). But if the current configuration is still the best available one, the APPLE system produces a policy decision not to handoff. As a result, the mobile host moves to the monitoring state, without executing any handoff.

Handoff

In this state the mobile host initiates handoff (see Figure 4) to switch from the current configuration to a newly selected configuration. The behavior of the mobile host in handoff state is controlled by the handoff policies. We discuss this in Section 4.2.3 with examples of the handoff policies.

When handoff is complete (at application level) the mobile host moves to the monitoring state.

Monitoring

In this state the mobile host waits for events from its environment. This environment is dynamic, for instance due to the user's movements and changing set of available aggregators and networks. When an event (for example, packet loss increases on the network interface etc.) occurs, the APPLE system evaluates its policies, to determine if the mobile host should initiate discovery (i.e. move the FSM to the discovery state). We call these policies *discovery policies.* If the policy decision is *discovery*, the mobile host runs the discovery protocol and sets the collection time to the value indicated in the policy decision. By doing so, the mobile host moves to the discovery state. If the policy decision is not *discovery*, then the mobile host stays in the monitoring state.

Final state

In this state the mobile host does not serve the user.

We can observe that the behaviour of the mobile host is controlled by discovery and handoff policies. These policies govern the state transitions of the FSM. In the next section we discuss discovery and handoff policies.

4.3 Policy classes of the APPLE system

In general, Policy classes can be designed and tailored according to the system's tasks and goals of the system's designer. In our research we want to control the behaviour of the mobile host, to control the invocation of discovery and handoff protocols by the CORD system. We define the following policy classes for the APPLE system: *discovery* policies (rules describing when to invoke the discovery protocol) and *handoff* policies (rules describing when and how to invoke the handoff protocol). Due to the introducing policies to control the behaviour of the mobile host, the discovery of alternatives and handoff to the best aggregator become policy driven.

4.3.1 Discovery policy

Changes in the environment where the mobile host operates, may create the necessity to discover alternative configurations to handoff, for example, when the current configuration is no longer available. The behavior of the mobile host in such a situation is controlled by discovery policies. An example of a high level discovery policy can be as follows:

```
If
        new network appears in the vicinity of the user
        while user is watching a channel,

then
        discover alternative configurations of this channel.
```

Discovery policy is a rule that controls the behavior of the mobile host when it encounters a change in remote or local resources, in user preferences; enters a hotspot, roams within the hotspot and leaves the hotspot. It comprises the rules when to scan for an alternative aggregator/configuration. Discovery policy can be reactive, starts to react when the current configuration starts to degrade, or proactive, reacts when the system anticipates that current configuration might begin to degrade in the near future. Discovery policy is a kind of "hunting" policy that defines when to start hunting for an alternative aggregator/configuration.

Another example of discovery policy can be defined for controlling the behavior of the mobile host when user preferences are changed.

```
If
        user preferences are changed
        while user is watching a channel,

then
        load new set of policies and
        discover alternative configurations of this channel
        that match new user preferences.
```

4.3.2 Handoff policy

Handoff policy defines when the mobile host should execute handoff from one configuration to another. Handoff policy also controls the behavior of the mobile host during handoff. This control is needed because handoff execution may affect the quality of the received stream during handoff. An example of a high level handoff policy can be written as follows:

```
If
        a better alternative of the current channel is discovered,

then
        handoff to this alternative in a user transparent manner.
```

Thus, the handoff policy is a rule that defines when (i.e. if better alternative is available) and how (i.e. transparently to the user) to execute handoff. The way of handoff execution can be defined by user preferences. For example, if the user prefers "*high viewing smoothness*" during handoffs the

handoff can be executed, by means of connecting to a new aggregator first and then disconnecting from the old aggregator. This way the mobile host will be handed off more smoothly.

Another example condition for Handoff policy might be the situation, when the mobile user moves with high speed, e.g. faster than 300 m/s, in this case WLAN networks with small coverage area, can be excluded from the selection, because it does not have sense to handoff to the network that will be passed in several seconds.

Besides discovery and handoff policies, we can write policy examples for other policy classes, for instance *refreshing policies*. A *refreshing policy* is a rule that defines setting a refresh frequency value for querying aggregators about the status of their configuration lists. The refresh frequency determines how quickly the mobile host detects that a configuration of an aggregator is no longer available, or when new configurations are available.

In this research we focus only on discovery and handoff policies.

4.3.3 Goals and examples of the APPLE system policies

As discussed in Section 3.2, policies have goals and they constrain the behavior of the system so that it becomes aligned with the goal of the policies. The APPLE system policies also have goals. The example goals for the APPLE system policies can be *smoothness, price*, *quality level* or *power consumption* etc.

In this research we consider "viewing smoothness" as a goal of all APPLE policies. This goal serves for provision of a desired smoothness, while the user is watching the channel. This goal has *high*, *moderate* and *low* values. If the discovery policy has a goal *high viewing smoothness*, this goal is achieved by timely invocation of the discovery protocol (i.e. when packet loss has already reached a certain threshold), see the following example:

Example *discovery policy* (1):

```
/* policy_type=discovery, leaving hotspot
 * policy_goal=high_viewing_smoothness
 */
  if (packet_loss >= 20 && receiving_interface == "802.11")

       /* discover alternatives to handoff */
          invoke_discovery_protocol();
```

The goal of the discovery policy (1) is to provide high smoothness of the video or audio, so the user will not notice glitches while he roams. This goal is achieved, for example, by monitoring the packet loss ratio and the signal strength on the network interface. When the user moves towards the edge of the network coverage, the signal strength drops continuously and the packet loss increases due to the increasing distance between the mobile host and the base station. To provide high viewing smoothness, the system should make early preparations to timely handoff. This is indicated in the *if* statement of the example discovery policy: if the packet loss is equal or exceeds 20 %, then the policy forces the mobile host to discover new configurations for a possible handoff.

The discovery policy with a goal *moderate viewing smoothness* is written as follows:

Example *discovery policy* (2):

```
/* policy_type=discovery, leaving hotspot
 * policy_goal=moderate_viewing_smoothness
 */
  if (packet_loss >= 80 && receiving_interface == "802.11")
```

```
/* discover alternatives to handoff */
   invoke_discovery_protocol();
```

Being controlled by the discovery policy (2) the mobile host behaves in a reactive manner, i.e. the mobile host starts to prepare to discovery and handoff when the user is very close to the edge of the network coverage, which might be indicated by the packet loss ratio over 80 %. The user might experience glitches and jitter when he roams out from one network to another.

Handoff policies control the behavior of the mobile host in the handoff state. We have two strategies to execute handoff: *break-before-make* and *make-before-break*. We consider these two strategies, because in some cases when the available bandwidth in the network allows, then during the handoff state the mobile host can receive the channel from two aggregators at the same time, and smoothly handoff from one to another. This provides more chances for higher viewing smoothness during handoff. We used these two strategies to compose handoff policies for the APPLE system. If the handoff policy has a goal *high viewing smoothness*, this goal is achieved by first connecting to the selected configuration and then disconnecting from the current configuration, see the following example:

Example *handoff policy* (1):

```
/* policy_type=handoff
 * policy_goal=high_viewing_smoothness
 */
  if ( configuration_list_received &&
       find_ better_alternative ( configuration_list,
                                  current_configuration))

       /* First connect, then disconnect */
       {
              connect(new_ configuration);
              disconnect(old_configuration);
       }
```

To provide high smoothness, the handoff policy (1) forces the mobile host first to connect to the selected configuration and then to disconnect from the old configuration, so that the viewing smoothness during the handoff will be higher.

Figure 16 illustrates how the handoff policy controls the behavior (i.e. the handoff execution) of the mobile host in the handoff state.

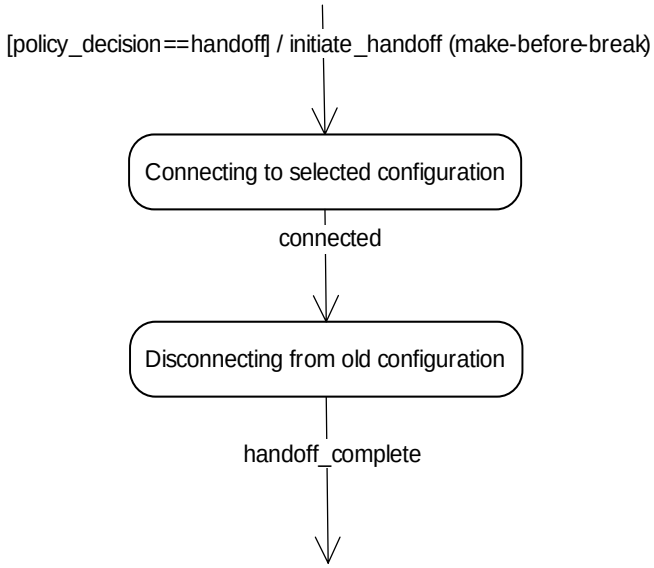

Figure 16. Make-before-break strategy during the handoff state.

The following example is the handoff policy with a goal *moderate viewing smoothness:*

Example *handoff policy* (2):

```
/* policy_type=handoff
 * policy_goal= moderate_viewing_smoothness
 */
  if ( configuration_list_received &&
       find_ better_alternative ( configuration_list,
                                  current_configuration))

       /* First disconnect, then connect */
       {
              disconnect(current_configuration);
              connect(new_ configuration);
       }
```

Being controlled by the handoff policy (2), the mobile host will take a *break-before-make* strategy by first disconnecting from the current configuration and then connecting to the selected configuration. This strategy will provide fewer chances for smooth handoff, which is inline with the goal of the policy – *moderate_viewing_smoothness.*.

We have composed different examples of policies that might be used by the APPLE system. Appendix B illustrates more examples of discovery and handoff policies.

4.3.4 Conclusion

Introducing policies makes the mobile host flexible. The discovery and handoff policies force the mobile host to automatically find a better alternative (if available) and to switch to this alternative. For example, at point B in Figure 1 mobile host discovers a better alternative of the channel version at *stream-it.com* aggregator and switches to this aggregator through WLAN interface. At point C, when the current aggregator becomes out of reach the mobile host switches back to *media-forward.nl* aggregator through UMTS interface. So, the discovery and handoff policies of the APPLE system provide a flexible control of the mobile host's behavior in the environment illustrated in Figure 1 by facilitating the switching between aggregators.

4.4 Specifying the APPLE system policies

In the previous section we introduced discovery and handoff policies for the APPLE system. We used the programming language (C) to write these policies. The power of the policies is in their downloadability and modifiability at run time [25]. Therefore, if policies are specified in a programming language, they will be *hard coded policies*, so that it will not be possible to modify them at run time. We decided to specify the APPLE policies in a specification language. Our contribution is not to develop a policy specification language, but to use existing means to specify simple obligation policies of the mobile host. In Chapter 3 we have discussed a number of policy specification approaches from in the literature: Ponder, IRML and XML. In our work we did not choose the Ponder policy specification language, because it is an advanced language. It has many features, which we do not need for the APPLE system (i.e. transfer of control, conflict detection and resolution, extensibility etc.). Besides, our implementation is in C programming language, while Ponder parser creates Java policy objects [44,38]. Ponder provides a complete deployment model with a Ponder compiler that compiles policies into Java classes. These classes are represented at runtime by Java objects. We also did not choose IRML language to specify the APPLE policies, because the tags that are defined in this language are not suitable for our work (see Section 3.3). We chose eXtensible Markup Language (XML) to specify the APPLE policies.

XML is a language for flexibly describing data. As its name suggests, it is extensible according to user needs, having an unlimited vocabulary to define new tags. XML is also attractive for its independence of operating systems and programming languages, it can operate in totally heterogeneous platforms. XML is a good solution for structured data description, storage, retrieval, transformation into other formats and exchanging data between heterogeneous distributed systems. XML based approach has become popular for policy specification [45-48]. This is because a structured specification such as XML is convenient for policy analysis and policy distribution across domains or networks.

Besides above mentioned advantages of using XML, our motivation to choose XML to specify policies for the APPLE system is also derived from the followings:

1. XML syntax is easy to understand;
2. We need a text based declarative language, to specify simple obligation policies for the mobile host;
3. We need to define variables tailored for our policies – XML gives freedom for defining tags;
4. We cannot run heavy processes; because the APPLE system runs on the mobile device, hence the runtime parsing of the policies should be light-weight;
5. XML parsers are widely available;

According to these statements XML seems to be a proper solution for us. In the next section we discuss how we specify our policies.

4.4.1 XML syntax for policy specification

Based on the observations on policy specification approaches, hints from the structure of PCIM classes defined in [31] and requirements what do the APPLE system policies need to express, we have defined the following tags for specifying the APPLE system policies. Table 2 illustrates the names of the tags and their meaning in our specification:

Tags	Specifies
policies	a root element of the XML file, that contains all policies defined for the system

goal	a goal of a policy
smoothness	a viewing smoothness of the channel while roaming and handoff. Values: high, moderate or low
condition	a condition part of a policy
action	an action of a policy
receiving_interface	a name of the receiving interface
packet_loss	a packet loss value reported by the EM
signal_strength	the value of a signal strength on the network interface of the mobile host
operator	an operator, such as “>”, “=” etc. that belongs to the condition statement
new_network_available	a roaming into a new network
interface_down	roaming out of the network
collection_time	the time interval assigned for collecting the responses from aggregators
discover_alternatives	the invocation of discovery protocol
configuration_list	the receipt of a configuration list from PEP
better alternative	the availability of a better alternative to a current configuration
connect_order	a variable indicating the handoff order. Values: FIRST - connect first, disconnect second SECOND - connect second, disconnect first
user_preferences	a change in user preferences
user_selects_channel	a selection of a channel by the user and a request to connect to this channel
velocity	the velocity of the mobile user

Table 2. Tags defined to specify the APPLE policies in XML format.

Examples of policies specified in XML are included in the Appendix D.

4.4.2 Examples of policies in XML

Using the tags defined in Section 4.4.1 we designed the structure of the APPLE policies in XML SPY. We wrote an XML Schema document that illustrates the structure of the policies of the APPLE system. Each policy contains an action, a condition and a goal. The APPLE policies are grouped according to their goals. In this research we used *viewing_smoothness* as a goal, which is specified by <smoothness> tag.

The XML Schema for the policy specification in XML SPY tool is shown in Figures 17.

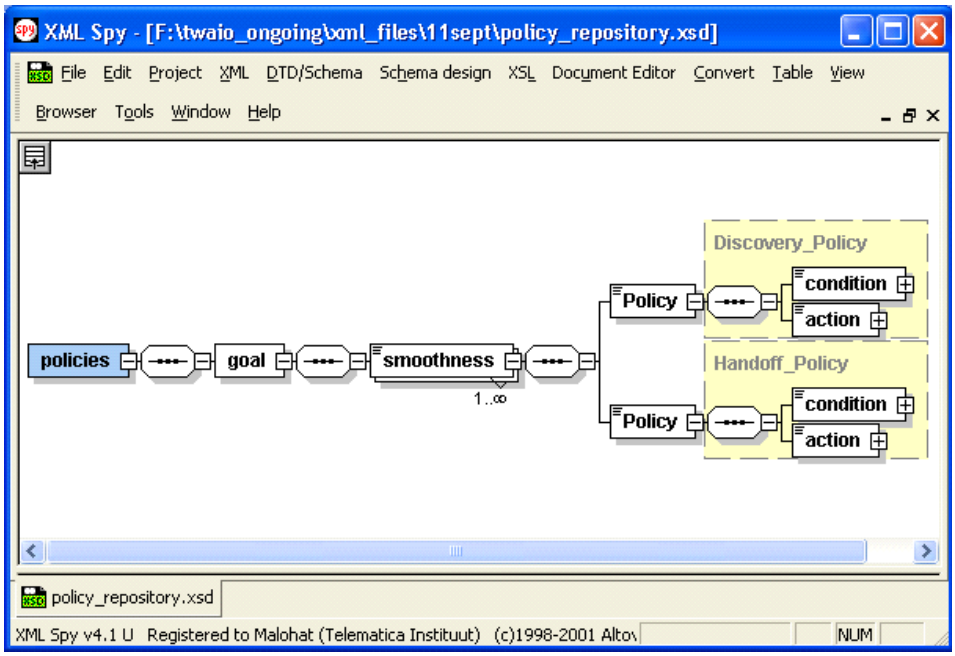

Figure 17. Graphical representation of the XML schema for the APPLE policies.

Figure 17 shows illustrates the structure of discovery and handoff policies and how there are organized within a root element *policies.* Figures 18 and 19 illustrate the structure of the discovery policy and the handoff policy.

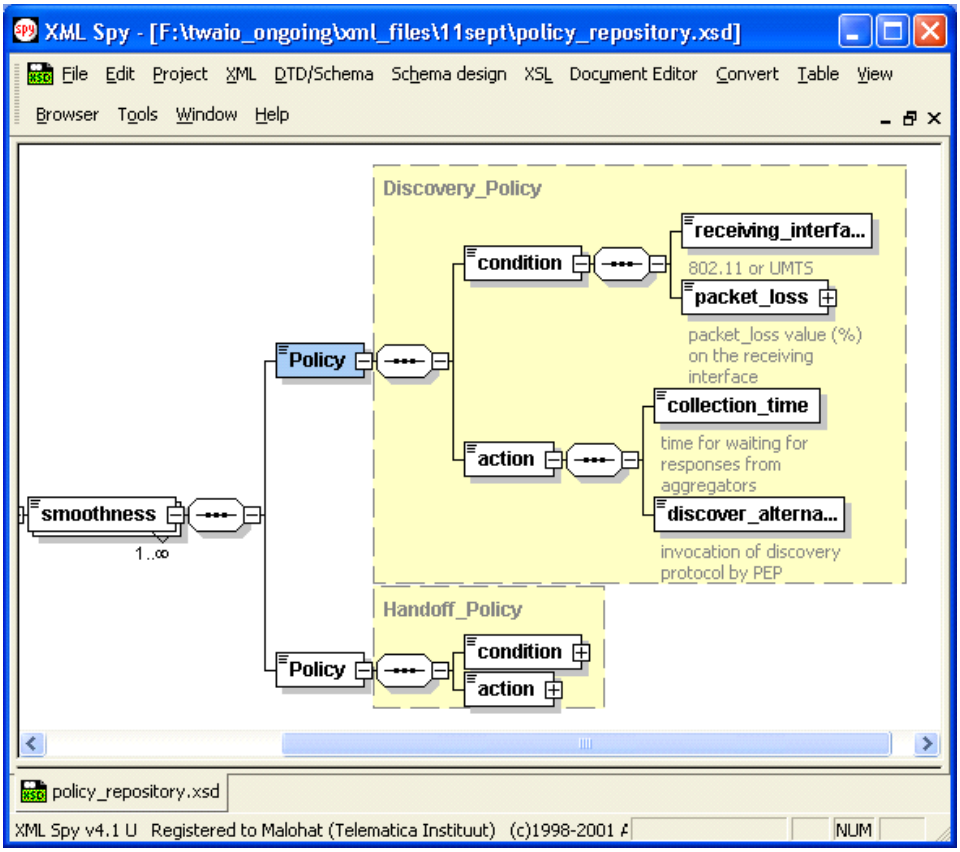

Figure 18. The structure of the example discovery policy.

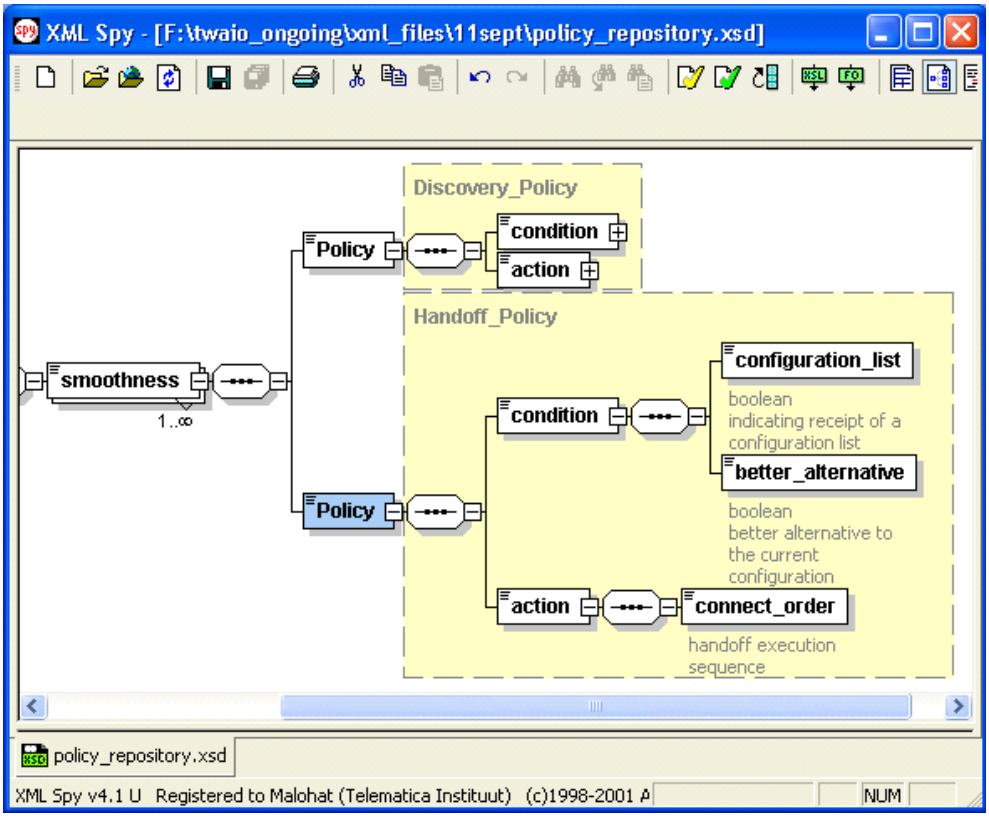

Figure 19. The structure of the example handoff policy

Figure 20 shows an example of the discovery policy of the APPLE system in XML format, that conforms the XML Schema shown in Figure 18.

```
<?xml version="1.0" encoding="UTF-8"?>
<policies xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="policy_repository.xsd">
   <goal>
      <smoothness>HIGH
         <Policy>DISCOVERY
            <condition>
               <receiving_interface>WLAN</receiving_interface>
               <packet_loss>20
                  <operator>GRATER_THAN</operator>
               </packet_loss>
            </condition>
            <action>
               <collection_time>SHORT</collection_time>
               <discover_alternatives/>
            </action>
         </Policy>
      </smoothness>
      <smoothness>MODERATE
         <Policy>DISCOVERY
            <condition>
               <receiving_interface>WLAN</receiving_interface>
               <packet_loss>50
                  <operator>GRATER_THAN</operator>
               </packet_loss>
            </condition>
            <action>
               <collection_time>DEFAULT</collection_time>
               <discover_alternatives/>
            </action>
         </Policy>
      </smoothness>
   </goal>
</policies>
```

Figure 20. Discovery policy markup with XML.

Figure 21 shows an example of the handoff policy of the APPLE system in XML format, that conforms the XML Schema shown in Figure 19.

```
<?xml version="1.0" encoding="UTF-8"?>
<policies xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="policy_repository.xsd">
   <goal>
      <smoothness>HIGH
         <Policy>HANDOFF
            <condition>
               <configuration_list/>
               <better_alternative/>
            </condition>
            <action>
               <connect_order>CONNECT_FIRST</connect_order>
            </action>
         </Policy>
      </smoothness>
      <smoothness>MODERATE
         <Policy>HANDOFF
            <condition>
               <configuration_list/>
               <better_alternative/>
            </condition>
            <action>
               <connect_order>DISCONNECT_FIRST</connect_order>
            </action>
         </Policy>
      </smoothness>
   </goal>
</policies>
```

Figure 21. Handoff policy markup with XML

4.5 Conclusion

In this section we discussed policies designed for the APPLE system. We explained our specification approach.
The APPLE policies have the following features:

1. The APPLE system policies are application level policies. The actions of the policies are enforced at the application level, because the actions of the APPLE system policies are invocation of the application level CORD protocol (see Figure 3, 4).

2. The APPLE system policies explicitly specify a goal, i.e. *viewing smoothness*. The policies with this goal control the behavior of the mobile host to achieve desired smoothness while user is watching the multimedia channel. Other example goals might be price of the service, quality level of the multimedia stream or power consumption of the mobile host etc.

3. The APPLE system policies are downloadable, because we specify our policies in a text-based markup language, i.e. XML, rather than a programming language. Therefore, it is possible to flexibly change the policies in runtime, which will result in flexible changing of control over the behavior of the mobile host. However, parsing XML policies remains as a future work.

4. Conditions and actions of one policy can be reused by another policy, e.g. policies can have the same goal, condition or action [see also 14, 16], although the observable behavior of the system defined by these policies are different. For example, the discovery policies 1 and 2 exemplified in Section 4.2.3 have different conditions but the same action. The handoff policies 1 and 2 in Section 4.2.3 have the same conditions but different actions. See more examples in Appendix B.

5. The APPLE system policies control the behavior of a mobile host. In our environment not only a mobile host but also an aggregator and a network operator can have their own policies. For example, an aggregator policy might be as following:

If
user roams into a foreign network,
then
allow only silver and bronze quality level configurations.

The concrete policy derived from the above policy is as follows:

```
/*  policy_class = restriction
 *  permitting a certain configuration at foreign network
 *  policy_goal= restriction
 */
  if(foreign_network_flag)

      /* Disable certain configuration */
      disable(gold_configuration);
```

This policy is an example of an aggregator's Obligation policy. The goal of this policy is to restrict the set of configurations of a channel when the user roams into a foreign network. This policy can also be derived from service level agreements between aggregators. However, the policies of aggregators and service level agreements between them are beyond the scope of this thesis.

5. THE APPLE SYSTEM ARCHITECTURE

In this chapter we describe the architectural design of the APPLE system, which applies the concepts and model discussed in Chapter 3 and the requirements analyzed from the environment discussed in Chapter 2.

This chapter is structured as follows:
Section 5.1 discusses the architectural components of the APPLE system and the CORD system.
Section 5.2 discusses the behavior of the mobile host in several scenarios.
Section 5.3 discusses the related work. Section 5.4 concludes the Chapter.

5.1 Components

Based on the generic policy model (Figure 9) and the requirements stated in Section 4.1, we designed an architecture of the Policy based System for the environment described in Chapter 2. The system consists of the CORD system and the APPLE system, Figure 22.

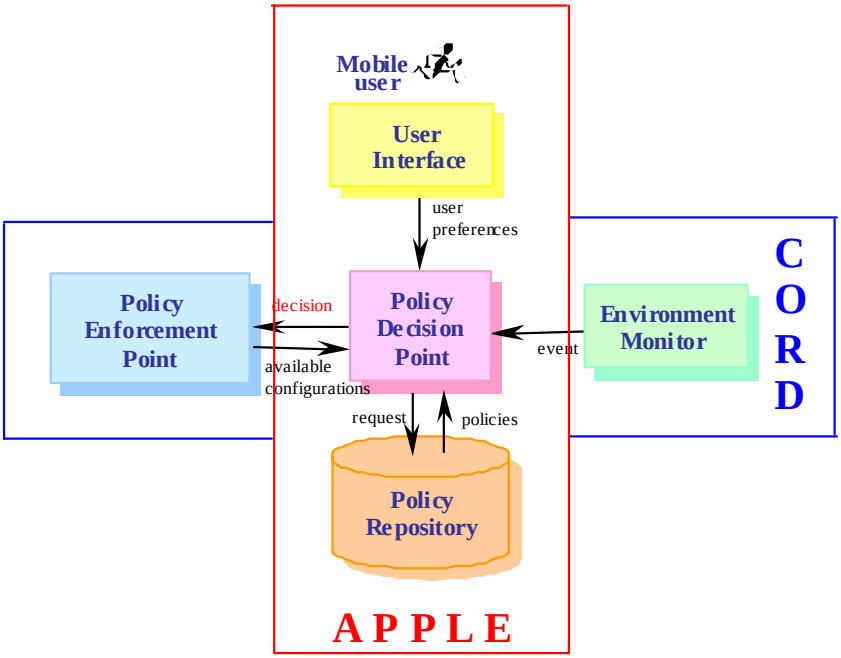

Figure 22. Architecture of the Policy-based System

The system components are:

1. *User Interface* (UI), belongs to the APPLE system.
2. *Policy Decision Point* (PDP), belongs partly to the APPLE system.
3. *Policy Repository* (PR), belongs to the APPLE system.
4. *Policy Enforcement Point* (PEP), belongs to the CORD system.
5. *Environment Monitor* (EM), belongs to the CORD system (although one may also view the monitoring component as a part of a controlling entity).

In the following sections we describe each component in detail.

5.1.1 Policy Repository

The Policy Repository (PR) is a storage for the APPLE policies. It contains two classes of policies, i.e. *discovery* and *handoff* policies.

Policy retrieval

Policies are retrieved (*polled*) from the PR at runtime by the PDP (Figure 22, *policies*). Retrieval of proper policies from the PR can be complex, e.g. based on several criteria and circumstances. In this research we simplify the retrieval of the policies, using for a query only one type (i.e. representing the *policy goal*) and one parameter (i.e. representing *viewing smoothness*). The APPLE policies have a goal *viewing_smoothness* (see Section 4.3.3) with three different values: *high, moderate, low.* We also introduced a *viewing smoothness* as one of the user preferences, to make the matching straightforward. A query structure is the following:

```
Query : <type, parameter, value>.
```

For example, if the user has selected *high viewing smoothness* as his preference, then all discovery and handoff policies in the PR with this goal will be retrieved from the PR using the query:

```
Query : <goal, viewing_smoothness, high>.
```

Location

In our research, we locate the PR locally in the mobile host. As an alternative, we may also distribute the PR or place the whole PR remotely. In this case, the remotely located PR can be shared by different mobile hosts of a user or by different mobile hosts of different users, but administrated by one manager. A drawback of this approach is that we would need a protocol to download policies from the PR into the PDP. For simplicity, our PR is located in the mobile host.

In the case that the policies in the PR have been modified, the PR may notify (i.e. *notification model*) this to the PDP and the "policy download" policy may decide to retrieve updated policies. In this work, we only discuss this option conceptually but we don't implement this interaction.

5.1.2 User Interface

The User Interface (UI) is a part of the APPLE system. We designed the UI only for illustration purposes to demonstrate the effect of user preferences on the behavior of the mobile host. User preferences provided through the UI (Figure 22, *user preferences*) determine the mobile host's behavior, in particular the PDP. For instance the user preferences are used in the selection of the best configuration of a channel.

Figure 23 shows an example of the UI. We introduces several parameters, that user can provide to the APPLE system as his preferences: (maximum target) *price*, (maximum target) *quality level*, *adaptation* and *viewing smoothness*.

Figure 23. Example User Interface.

Maximum Target Quality Level denotes the quality levels of a channel version, which can be chosen by the user. One may view this parameter as a subscription level. *Adaptation* in the UI denotes a permission of the user to the mobile host to switch to a lower quality level version of a channel. For example in case of limited network capabilities or unavailability of a desired quality level of versions of a channel, a "gold" user might receive a channel version which belongs to the silver quality level, if he allows adaptation (Figure 23). *Viewing smoothness* denotes perceptual viewing smoothness of the channel version, e.g. indicated by the number of glitches or still images on the screen while user is watching a channel. As it is discussed earlier, this parameter is used to match proper policies from the PR. *Maximum Target Price* denotes the maximum price, that a user is willing to pay for the service.

We modeled the interactions between the PDP and the UI as a *notification model*. The UI notifies the PDP when the user has changed his preferences.

5.1.3 Environment Monitor

In our policy-based system, the Environment Monitor (EM) is responsible for monitoring available resources in the dynamic environment in which the mobile host operates. The EM is a part of the CORD system, although one may also consider this component as part of the APPLE system. In our architecture, the EM is a component that is responsible to detect the environment changes (*notification model*). The EM sends an event to the PDP (Figure 22, *event*) when for example it detects that the network has become unreachable (i.e. interface is down) or when a new network has become available.

Components

The EM might contain the components as illustrated in Figure 24:

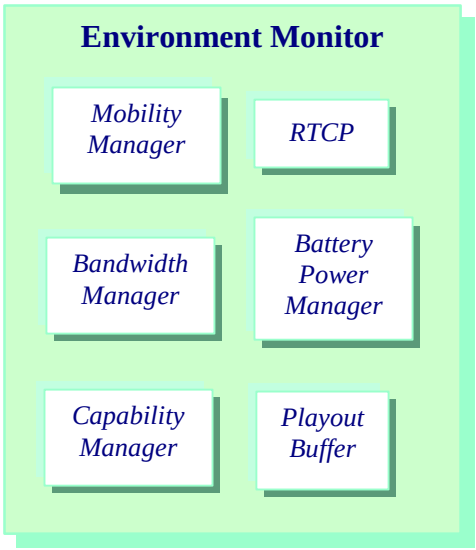

Figure 24. Components of the Environment Monitor

The components of the EM constantly monitor the environment, for example:

1. *Mobility Manager* (MM), keeps track of the status of the interfaces of the mobile host. For example, it can report to the PDP on events such as the assignment of a new IP address or the active interface is down.

2. *Real Time Control Protocol* (RTCP) entity reports on packet losses of an established connection;

3. *Bandwidth Manager* reports on the available bandwidth (e.g. kbps) on the network interface.

4. *Battery Power Manager* reports on the remaining battery power of the mobile host.

5. *Capability Manager* stores the information on capabilities of the mobile host, such as available codecs in the mobile host, computational intensity and battery power needed for a certain codec, screen size of the mobile host etc.

6. *Playout Buffer* provides the dynamic information on the actual buffer size of the mobile host for playout of a stream.

In this thesis, we use a *Mobility Manager* (MM) [49] and an *RTCP* entity as monitoring and eventing components in the EM. Other components of the EM remain as a possible future work.

The structure of an event from the EM is as follows:

```
<event_type, parameter, value, interface_name>.
```

Location

In our work, the EM is a local component, i.e. the MM and the RTCP entity are located in the mobile host, because they monitor the local resource on the mobile host, such as packet losses and signal strength on the network interface. As an alternative design choice, the EM can also be distributed. It can be for example a centralized Bandwidth Broker (see also [42]) that monitors the overall load on the network, which can be queried by the mobile hosts (*polling*) on the available bandwidth of the network at a certain moment.

5.1.4 Policy Decision Point

In our policy-based system the PDP belongs to the APPLE system.

The PDP is a place where the APPLE policies, which are relevant for control reside and evaluated. The PDP "downloads" from the PR all APPLE policies with the goal matching the user preferences (i.e. *viewing smoothness*), see also the query structure in Section 5.1.1. Therefore, the PDP has several active *discovery* and *handoff* policies.

In addition to the APPLE policies, the PDP also uses inputs (Figure 22) from the UI (i.e. *user preferences*), the EM (i.e. *events*) and the PEP (i.e. *available configurations*) to come to policy decision.

The events of the APPLE system are:

1. Signal strength on the receiving interface (trigger: the MM of the EM).
2. New network is detected, i.e. roaming into a new network (trigger: the MM of the EM).
3. The current network is not reachable, i.e. interface is down (trigger: the MM of the EM).
4. Packet losses in the received stream (trigger: the RTCP entity of the EM).
5. User has changed his preferences (trigger: the UI).
6. The configuration list is received from an aggregator (trigger: the PEP).

Example 1:

Consider the first event in the above list: the MM of the EM sends an event containing the signal strength. This event has the following values (cf. event structure in Section 5.1.3):

```
<roaming_out, signal_strength, poor, wlan>.
```

This event triggers the evaluation of the discovery policy:

```
if (signal_strength <= poor && receiving_interface == wlan)
      initiate_discovery();
```

If fired, the PDP enforces to the PEP to the discovery of aggregators via available networks.

Example 2:

Consider the second event in the above list. When a user roams into a new network, the MM of the EM sends an event with the following values:

```
<roaming_in, interface_status, up, wlan>.
```

This event will trigger the policy evaluation and the following discovery policy will fire:

```
if (interface_status == up && receiving_interface == wlan)
      initiate_discovery();
```

The decision of the PDP (i.e. invoke the discovery protocol) will be executed at the PEP.

Location

In our research, we locate the PDP locally in the mobile host. As an alternative, we may also place the PDP remotely in a central node, which can be responsible for policy decisions for several mobile hosts of a user or a company. A drawback of putting the PDP remotely is that we would need a protocol to retrieve a policy decision, while it is not efficient to rely on the wireless environment. Yet another alternative is to put the PDP at an aggregator side. But as far as a policy decision involves processing of the configurations of a channel offered by different aggregators,

the privacy rules of an aggregator might not allow exposing its offers to other aggregators. Therefore, we consider the best place to locate the PDP is the mobile host, which conforms our general aim to put control close to the user.

5.1.5 Policy Enforcement Point

In our policy-based system PEP is a part of the CORD system. As an enforcement of the policy decisions made by the PDP (Figure 22, *decision*), the PEP runs discovery and handoff protocols of the CORD system that is discussed in Section 2.2 (Figures 2 and 3).

Types of handoff

PEP executes the following types of handoff using the protocol explained in Section 2.2:

1. The best configuration of a channel might belong to the same aggregator; in this case the so-called handoff will be from one configuration of a channel to another configuration of this channel at the *same aggregator* and using the *same network interface*.

2. The best configuration of a channel might belong to *another aggregator* that is available through the *same network interface*, in this case the handoff will be from one configuration of a channel at one aggregator to another configuration of this channel at another aggregator.

3. The best configuration of a channel might belong to *another aggregator* that is available through *another network interface*, in this case the handoff will be from one configuration of a channel at one aggregator through one network interface to another configuration of a channel at another aggregator through another network interface. In this case, the application level handoff results in a network level handoff, because the mobile host needs to be handed off from one network interface to another network interface.

4. The best configuration of a channel might belong to the *same aggregator* that is however available through *another network interface*, in this case the handoff will be from one configuration of a channel at the aggregator to another configuration of this channel at the same aggregator through another network interface. In this case, the application level handoff results in a network level handoff, because the mobile host needs to be handed off from one network interface to another network interface.

For the interactions between the PDP and the PEP, we designed and implemented both a *polling model* and a *notification model*. For example, when user has changed his preferences, this event is sent to the PDP. Based on this event, the PDP evaluates the policies and makes a policy decision to invoke the discovery protocol. The PDP then *pushes* this decision to the PEP. This case exemplifies a *notification model.*

Consider another example; the PEP receives a configuration list from an aggregator. The PEP sends this configuration list to the PDP. The receipt of the configuration list triggers a policy evaluation in the PDP, which selects the best configuration and produces a policy decision. This policy decision is returned to the PEP for enforcement. This case exemplifies a *polling model.*

Location

The PEP sits in the mobile host, because it is a component that executes handoff and discovery.

5.2 Behavior

In this section we discuss the behavior of the components discussed in the previous section using scenarios. We describe the following scenarios:

Scenario 1. User provides his preferences,
Scenario 2. User selects a channel to watch,
Scenario 3. User roams into a new network.
Scenario 4. User roams out of the network.

These scenarios are derived from Figures 1 and 2 of Chapter 2.

5.2.1 Scenario 1: User provides his preferences

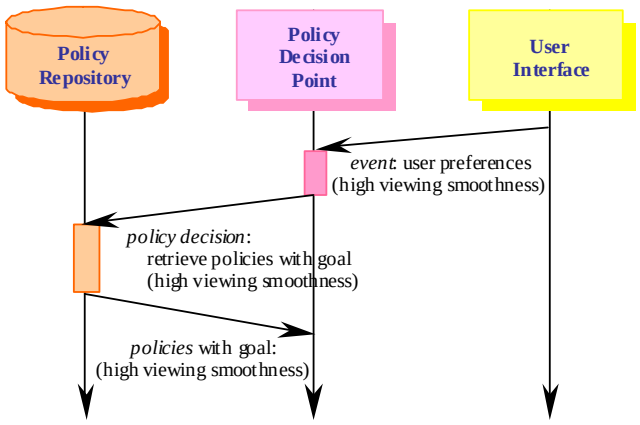

Figure 25. Behavior of the mobile host when user preferences are provided

Figure 25 shows the behavior of the mobile host when user Bob changes or provides his preferences regarding *viewing smoothness* (Figure 23). In this scenario, all policies with a goal matching the value of the viewing smoothness are retrieved from the PR by the PDP. From that moment the PDP uses these newly retrieved policies for future policy evaluations. The newly retrieved policies will be active until they are replaced by other policies.

5.2.2 Scenario 2: User selects a channel to watch

Figure 26 shows the behavior of the system after the above-mentioned policies are downloaded and the user Bob selects a channel to watch (Figures 1 and 2, point A). Once Bob has selected a channel, the PDP makes a policy decision to discover available aggregators. The PEP executes the policy decision by invoking the discovery protocol. At point A only media-forward.nl is available and therefore the PEP collects only one list of configurations. The PEP sends to the PDP the received list of configurations. The PDP again evaluates the APPLE policies. During the evaluation it processes the list of configurations to determine the best one. When the best configuration is determined, a handoff policy fires and the PDP produces a policy decision to handoff to the best configuration in the list. This policy decision again enforced by the PEP and the mobile host gets connected to the media-forward.nl aggregator to receive the selected version of the channel. The media-forward.nl aggregator is available through the UMTS interface (Figures 1 and 2).

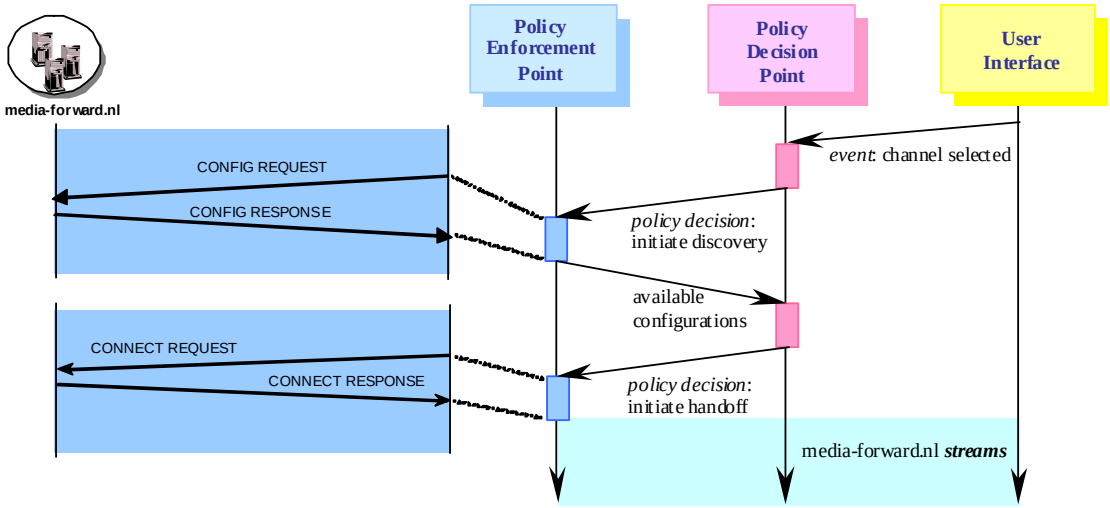

Figure 26. Behavior of the mobile host when user selects a channel.

5.2.3 Scenario 3: User roams into a new network

Figure 27 shows the behavior of the mobile host, when Bob is at point B of Figures 1 and 2.

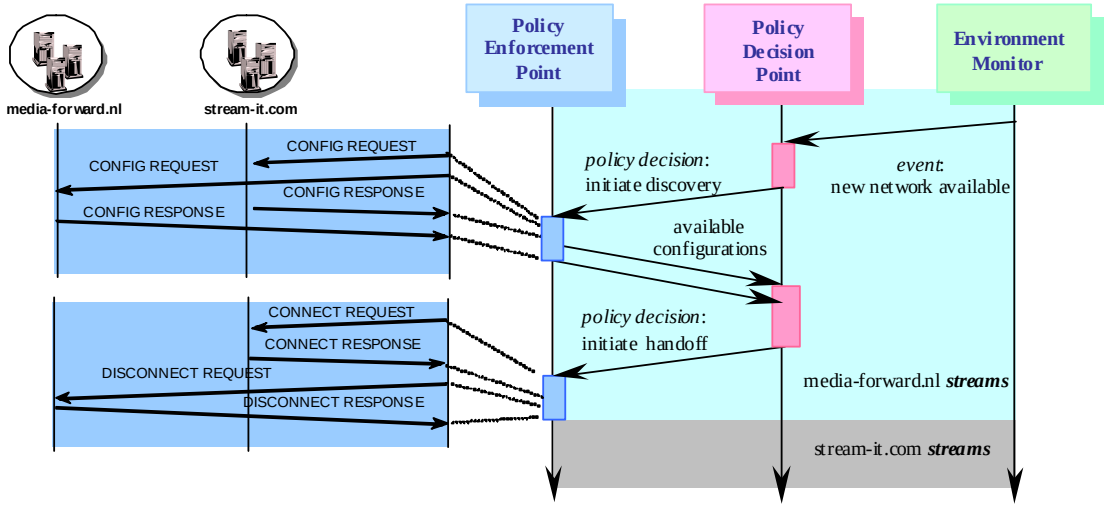

Figure 27. Behavior of the mobile host at entering the hotspot

At point B the mobile host enters the hotspot, where it can reach two aggregators. The mobility manager of the EM informs the PDP that new network has become available. The PDP evaluates the APPLE policies. The discovery policy with the condition *if(new_network_available)* fires and the PDP produces a policy decision to initiate discovery. The policy decision is pushed to the PEP, which invoke the discovery protocol. This time the PEP collects two list of configurations from tow aggregators. Each time it receives the configuration list from the aggregator it sends the list to the PDP. The PDP evaluate the APPLE policies, processes the list of the configuration and determines the best configuration. When the PDP receives the second configuration list, it determines the best configuration of the channel *among two lists of configurations* offered by two

aggregators. In this scenario at point B, the best configuration of the channel is offered by stream-it.com aggregator. Therefore the policy decision is to handoff to this aggregator. Once the collection time expires, the policy decision is sent to the PEP, which executes handoff to stream-it.com that is available through the WLAN interface of the mobile host.

Note that, the handoff is carried out using a *make-before-break* strategy (see Figure 16). Because in this scenario, the PDP uses a handoff policy with the goal *high viewing smoothness* (see Section 4.3.3).

5.2.4 Scenario 4: User roams out of the network

Figure 28 shows the behavior of the mobile host, when Bob is moving towards the point C of Figures 1 and 2.

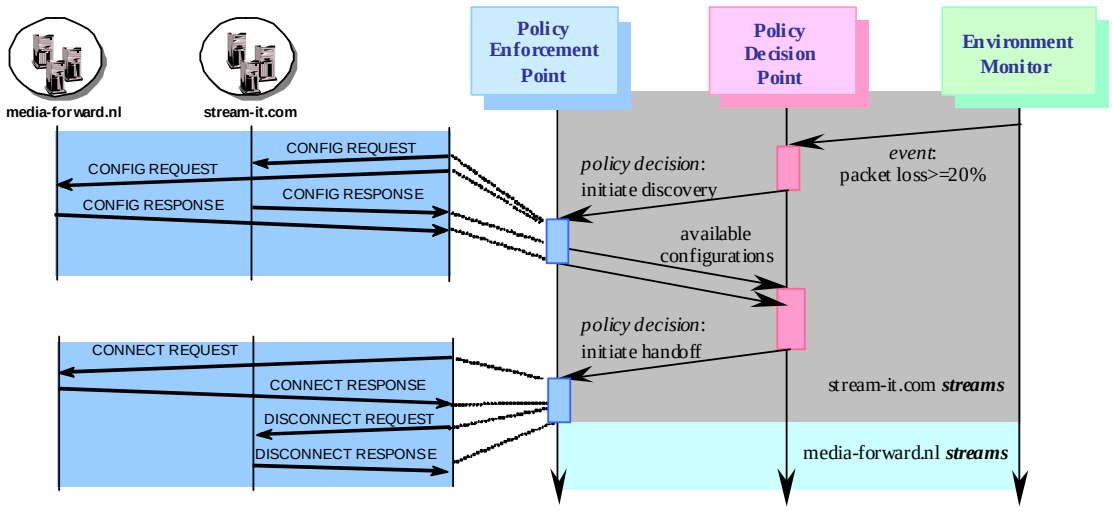

Figure 28. Behavior of the mobile host at leaving the hotspot

Due to the roaming of Bob towards the edge of the WLAN network (e.g. point C), the packet losses on the WLAN interface are continuously increasing. The mobility manager of the EM periodically informs the PDP about the packet losses of the connection by sending an event to the PDP. The PDP at receiving of the event evaluates the APPLE policies. At some moment, the packet loss exceeds 20 %, which results in firing of a discovery policy in the PDP. The PDP produces a policy decision to discover alternative configurations of the channel version. The rest of the behavior of the mobile host in this scenario is the same with its behavior in the scenario 3, except that the PEP executes handoff from stream-it.com to media-forward.nl and switches back to its UMTS interface. Due to the early activation of the discovery policy with the goal *high viewing smoothness* (e.g. packet loss>= 20%) the mobile host is handed off to another configuration more smoothly and in time (i.e. before loosing the connection to stream-it.com).

Because the user allowed the adaptation to lower quality level of the channel version, the PDP might select *silver* version of the channel instead of *gold*, if the latter is not available among discovered alternatives.

The behavior of the mobile host at point D in Figure 2 is similar the behavior at point B.
In this section we covered the behavior of the mobile host in the environment illustrated in Figures 1 and 2.

5.3 Related work

The APPLE system is an application level policy-based *handoff control* system designed for the CORD system. It controls switching between aggregators in a user transparent manner and selects the best configuration of a channel according to the user preferences and network capabilities. In this section we discuss several papers, related to our work, in particular:
policy-based and non policy-based handoffs, policy-based QoS provisioning, automatic selection of a network/service in heterogeneous environment.

Stemm et al. [50] discuss vertical handoffs in wireless overlay networks, where *handoffs are realized only according to the strength of beacon signals*. The handoff in our work is *policy-based* and is triggered at the application layer: the mobile host executes handoff from one configuration of a channel to another configuration of the same channel. *In our work*, the policy decision to handoff is made not only according to the capability parameters of resources (e.g. signal strength on the network interface), but also taking into account user preferences, available networks and aggregators.

Wang et al. [17] discuss *policy-based handoffs* in heterogeneous wireless networks *at network level*. Their policies are based on the user preferences (i.e. price, quality or power consumption) which determine which network is the best for the user. Because they focus on choosing the *best* network, we consider their policies as *network level* policies.

The work of G. Lee et al. and D. Clark et al. [20,21] is on the *automatic service selection* while the user roams between different networks. The service selection is not policy based but is carried out by means of prediction functions using multilayer neural networks, user preferences, available network resources and identities of running applications in the mobile host.

Policies are also used for controlling QoS. For example Lutfiyya et al. [16] addresses the problem of QoS requirements for multimedia applications like telemedicine, telelearning, e-commerce etc. to be able to co-exist with the traditional applications for data processing and transactions. They use *policies to meet QoS requirements by reallocating memory resources* between simultaneously running applications.

Murray et al. [19] use policies to *guarantee QoS by controlling the access to a network*. Using policies, they select the best (i.e. less loaded) network for a mobile host. Their policies provide a kind of intelligent access to a network. In addition, their policies force a mobile host to handoff to another network, when current network reaches a congested state. Therefore the control in their work is placed in the infrastructure, rather than close to the mobile host.

What makes our work different?

The following aspects that we consider in our research make our work different among the discussed related work.

1. We use well-defined, well-structured policies that contain goals, conditions and actions. Our policies are application level policies, because our policies invoke an application level discovery protocol and initiate application level handoffs.

2. Our policy-based system handle the environment with multiple content and connectivity providers (which is not the case in [46,48]), supportability in non mobile-IP environments (which is not the case in [17]), user-centric approach (which is not the case in [19]) taking care of the user preferences in price and quality level of a channel version, and smooth handoffs during roaming.

3. Input information for making a decision, in our work is: a) mobile host's status, b) interface status, c) user preferences as well as the d) configurations of a channel offered by the aggregators. This also makes our work different and original among discussed related work.

4. Our overall goal of using policies, is to enable the user to seamlessly roam within the overlay networks, so that his mobile host can maintain service continuity in a way conforming user preferences in price, perceptual quality level, viewing smoothness etc.

5.4 Conclusion

In this Chapter we presented a high level design of our policy-based system that consists of the APPLE system and the CORD system. The APPLE system is a policy-based *handoff control* system designed for the CORD system (see Section 2.4). The APPLE is based on the generic policy model (see Section 3.4). We have shown how the APPLE and the CORD systems relate by means of the scenarios in Section 5.2. In the next chapter we explain the implementation details of the APPLE system based on the high level design.

6. IMPLEMENTATION

In this chapter we explain the implementation of the APPLE system. The goal of this chapter is to demonstrate the concepts of policies and the policy model we have proposed in Chapter 3.

This chapter is structured as follows: Section 6.1 discusses the software organization of the APPLE system. Section 6.2 explains the testbed environment in which we tested the APPLE system integrated with the CORD system. This section also gives the results and screenshots obtained from the testbed.

6.1 Software organization

The prototype of the APPLE system was developed in C programming language. The code runs on a Linux Redhat laptop. The laptop runs the client side of the CORD system and the APPLE system as one process.

Figure 29 provides an overview of the implementation components of the APPLE system.

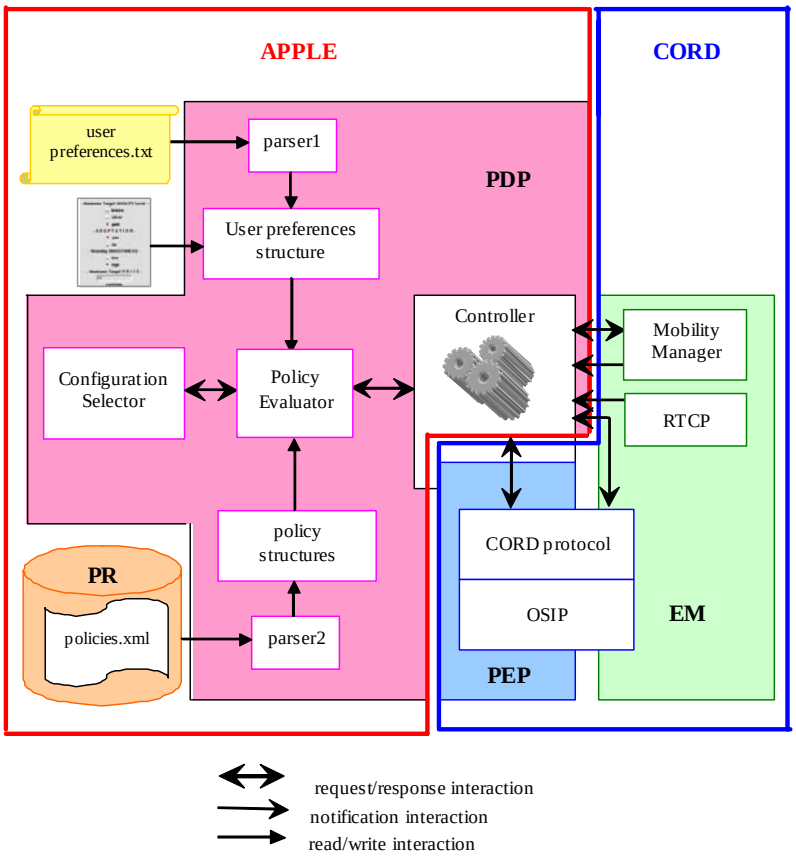

Figure 29. Implementation organization

6.1.1 The APPLE system components

The APPLE system has the following building blocks, which conceptually correspond to the high level view of the system architecture in Chapter 5:

1. User Interface (UI) components are:

- *user_preferences.txt*, a text file that stores user's default preferences (see Section 6.2.1).
- a simple *user interface*, which is implemented in Tcl/Tk (see Figure 23) to modify the user preferences while user is watching the channel.

2. Policy Repository (PR) component:

- *policies.xml*,a file that stores the APPLE system policies in XML format (see Figures 19, 21).

Our policy implementation approach is to specify policies in XML format, then to parse these policies into policy structures using the parser2 (see Figure 29).

3. Policy Decision Point (PDP) components are:

- *parser1*, to parse user_preferences.txt file into user preferences structure;
- *parser2*, to parse policies.xml file into the list of policy structures.
- *Policy Evaluator*, to evaluate policies and produce a policy decision;
- *Configuration Selector*, to assist policy evaluation, to select the best configuration;

Parser1

User_preferences.txt file is parsed into a memory structure shown in Figure 30.

```
struct user_preferences{
       int quality;          /* Quality level of the configuration,
                                possible values: BRONZE, SILVER, GOLD.       */

       int adaptation;       /* Adaptation to other quality level when
                                the target quality level is not available,
                                possible values: no, yes.                    */

       int smoothness;      /* Viewing smoothness during roaming and switching
                                from one configuration to another,
                                possible values: HIGH, MODERATE, LOW.        */

       int price;           /* Maximum value for the price the user is
                                willing to pay for the service,
                               for example, 10 cents per minute.             */
};
```

Figure 30. User preferences structure stored in the memory

The *user preferences* structure is used for selection of the best configuration of a channel version and for retrieving policies (see Section 5.1.1).

Parser2

The *policies.xml* file is parsed into the representation of policies in memory. The requirements for a parser are:

- it has to run on Linux, because the APPLE system runs on Linux Redhat laptop;
- it has to be written in C.

We used Expat parser because it fulfils the above mentioned requirements. In addition, it uses SAX [51] libraries, which mean that it is fast and uses less memory resources.

The parsed *policies.xml* file will have a memory representation in the following structure (Figure 31):

```
// policy

struct policy{
        char *goal;           /* goal of the policy */
        int goal_value;
        char *policy_class;
        struct condition *condition;
        struct action *action;
};
// condition of a policy
struct condition{
        int configuration_list;
        int better_alternative;
        int signal_strength;
        int new_network_flag;
        int receiving_interface;
        int user_preferences;
};
// action of a policy
struct action{
        int collection_time;
        int discover_alternatives;
        char *connect_order;
        char *decision_type;
};
```

Figure 31. A policy structure

When the parser reads each policy from the XML file, it parses into the `policy struct`. Each `policy struct` contains the goal, the value of the goal, `condition struct` and `action struct` (see Figure 31). After parsing, the `policy struct` is added to the `list of policies`. The length of the list depends on the number of policies specified in the XML document.

Due to the time limitations, a run time parsing part of the implementation is not integrated into the testbed. This work remains as a future work. The APPLE system integrated into the testbed uses policies precompiled (preparsed) as *policy structs*.

Policy evaluator

In our implementation we have two policy lists:

- the *list1* contains *all policies* preparsed from the *polices.xml* file, this list represents an in-memory policy repository;
- the *list2* contains only active policies, i.e. the policies selected according to the user preferences (i.e. *viewing smoothness*, see Figure 25) from the in-memory policy repository.

In the policy evaluation process, the *policy evaluator* uses only the active policies in the *list2*. The *list2* can be updated at run time for example when the user changes his preferences.

Our approach is to evaluate the policies only when the policy evaluator receives an event from the controller [38,52,57]. The following events are reported by the mobility manager of the EM or the user interface to the controller:

- a network interface is up,
- a network interface is down,
- the list of configurations of a channel at aggregators is received

- change in user preferences,

The controller requests a policy decision from the policy evaluator, when it receives such events. Figure 32 illustrates the event structure, which is sent by the controller in the request to the policy evaluator (Figure 29, see also Section 5.1.3):

```
struct event{
  int type;   /* event type, possible values are: */
              /* 1. RAOM_IN - roaming into a new network */
              /* 2. ROAM_OUT - packet loss increases, signal strength decreases */
              /* 3. CONFIGURATIONS - configuration list is received from an aggregator */
              /* 4. USER_PREFERENCES – user preferences have been changed */
  int packet_loss;            /* packet losses of an established connection */
  int signal_strength;        /* signal strength on the network interface */
  int new_network_available;  /* detecting a new network, a new IP address assignment */
  int receiving_interface;    /* the active network interface */
  struct sdp_t *sdp;          /* configurations of a channel in sdp format */
  char *aggregator_id;        /* the name of an aggregator that offers configurations */
  char *aggregator_intf;      /* interface through which this aggregator is reachable */

};
```

Figure 32. The event structure.

When the policy evaluator receives a request for a policy decision from the controller, it uses the values from the event (Figure 32) received in this request and evaluates the policies, i.e. evaluates the conditions of the policies in the active list of policies.

When the condition of a certain policy is evaluated true then the policy fires and the policy evaluator produces a policy decision. Figure 33 illustrates the policy decision structure produced by the policy evaluator:

```
struct policy_decision{
  char *decision;          /* policy decision type */
                           /* possible values: discovery, handoff */
  int collection_time;     /* time to collect responses from aggregators */
  int connect;             /* handoff execution: make-before-break */
  int disconnect;          /* handoff execution: break-before-make */
  list_t *configurations; /* list of configurations of a channel */
};
```

Figure 33. The decision structure.

If for example, a discovery policy fires, the decision will contain the policy decision type (i.e. discovery) and the value for the collection time. If a handoff policy fires, then the decision will contain the policy decision type (i.e. handoff) and the list of configurations of a channel that are selected according to the user preferences and network capabilities. The policy decision struct is returned to the controller, which enforces the decision on the CORD protocol entity (Figure 29).

Configuration selector

The configuration selector is responsible for selecting the best configuration that matches user preferences, and checks whether the bitrate required by a configuration can be supported by the potential capacity of the available network (e.g. 802.11 potential capacity is 11Mbs). The configuration selector processes each configuration and produces a list of configurations, where the best configuration comes first in this list. The policy evaluator includes this list of configurations into the policy decision.

Controller

The controller component is an integration place of the APPLE system components into the CORD system components. The controller is implemented as an eventing finite state machine by C. Hesselman. The controller schedules all activities in the mobile host, it also forwards events to the PDP and enforces policy decisions on the PEP.

6.1.2 The CORD system components

In accordance with the high level view of the system architecture discussed in Chapter 5 and Figure 22, we identified the following building blocks of the CORD system in our software organization:

1. Policy Enforcement Point (PEP) components

PEP contains the CORD protocol entity that runs on top of SIP. This PEP also contains a part of the controller that runs the CORD protocol.

2. Environment Monitor (EM)

The EM contains the mobility manager and the RTCP entity, the other entities of Figure 30 are not relevant in our demo scenario and are not implemented yet.

For more details on the CORD system of Hesselman et al. refer to [2-4].

6.2 Testbed

We integrated the APPLE system into the CORD testbed. Figure 34 shows its main hardware and software components.

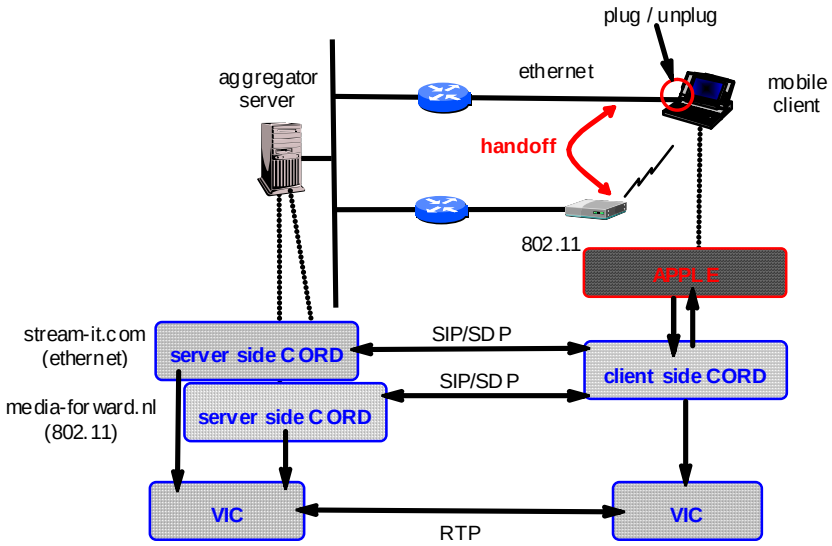

Figure 34. Testbed

Hardware

The main hardware components of the testbed are a Toshiba laptop, an 802.11 base station and a server machine. The laptop is equipped with an Orinoco 802.11b Gold card and an Ethernet card. The laptop runs Redhat Linux.

The 802.11 base station and the Ethernet LAN form two separate subnets representing two separate network operators. The 802.11 base station is located in an indoor office environment. The network environment is thus similar to that of the example of Figure 1, except that it consists of a WLAN and a wired LAN. The WLAN simulates the UMTS network and the LAN simulates the 802.11 network of Figure 1.

Software

The CORD protocol software is implemented in C on top of open SIP [5], which also includes an SDP parser.

The server runs two processes that execute the server-side software of the CORD protocol (see Figure 36). One process represents an aggregator *media-forward.nl* another process represents an aggregator *stream-it.com* (Figure 1). The aggregator processes authenticate a user with a Free Radius [53] server (not shown in Figure 34) through a simple request-response interaction.

The server also runs the server part of VIC to simulate streaming servers at the aggregators (see Figure 36). The server part of VIC has an extension that enables the aggregator processes to configure the framerate, quality and bitrate (kbps) of VIC according to the selected configuration details. Aggregator processes also inform a target IP address to which VIC should stream the content.

The laptop runs the client side of the CORD protocol (see Figure 37). The protocol software is configured such that it communicates with aggregator *stream-it.com* through the laptop's Ethernet interface and with aggregator *media-forward.nl* through the 802.11 interface.

The laptop hosts a second process that runs the mobility manager [49] that is developed in the 4G+ project [57]. The mobility manager keeps track of the state of the laptop's network interfaces (e.g. which interfaces currently have link-layer connectivity, the signal strength of the 802.11 networks and so on). The mobility manager serves requests (e.g. for the current signal strength) from the client-side protocol software and can also sends events to it (e.g. signaling that a new network has become available or that the current network has become unavailable). The protocol software uses some of these events as a trigger to run the protocol (Figures 2 and 3).

The third process on the laptop is the client part of VIC (see Figure 37). We extended the VIC software so that it can receive a stream through a specific network interface. This enables the mobile host to perform a handoff from one to another network to receive the stream. The server part of VIC transfers multimedia content to the client part of VIC using RTP packets [54], see Figure 34.

6.2.1 Roaming scenario

We tested the prototype for the scenario discussed in Figure 2 of Chapter 2, Figure 35:

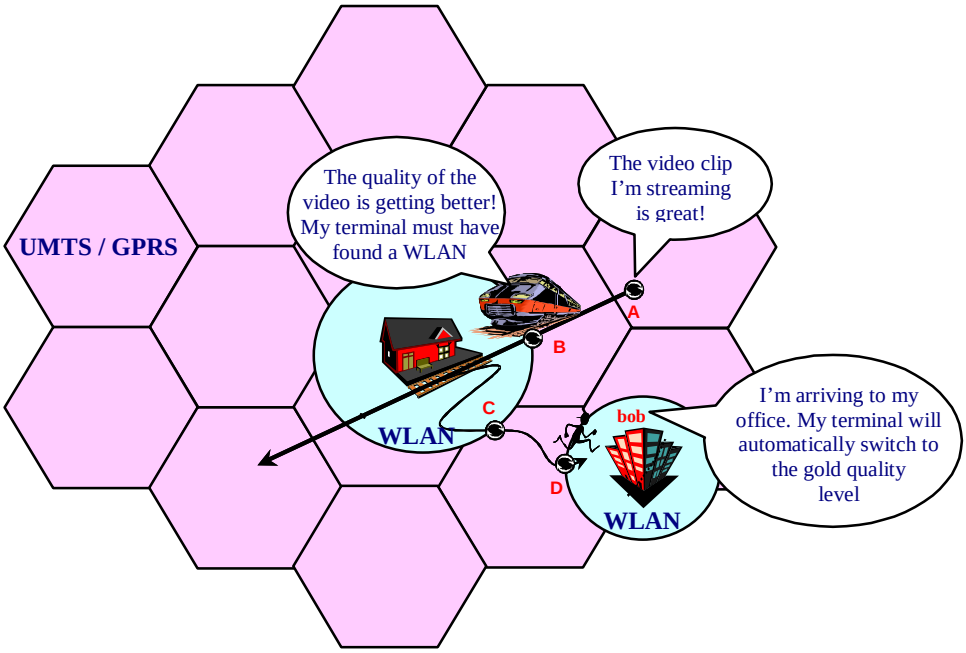

Figure 35. Roaming user scenario on testbed.

We refer to the Section 2.1 for a detailed discussion.
First we start the server side software as it is shown in Figure 36:

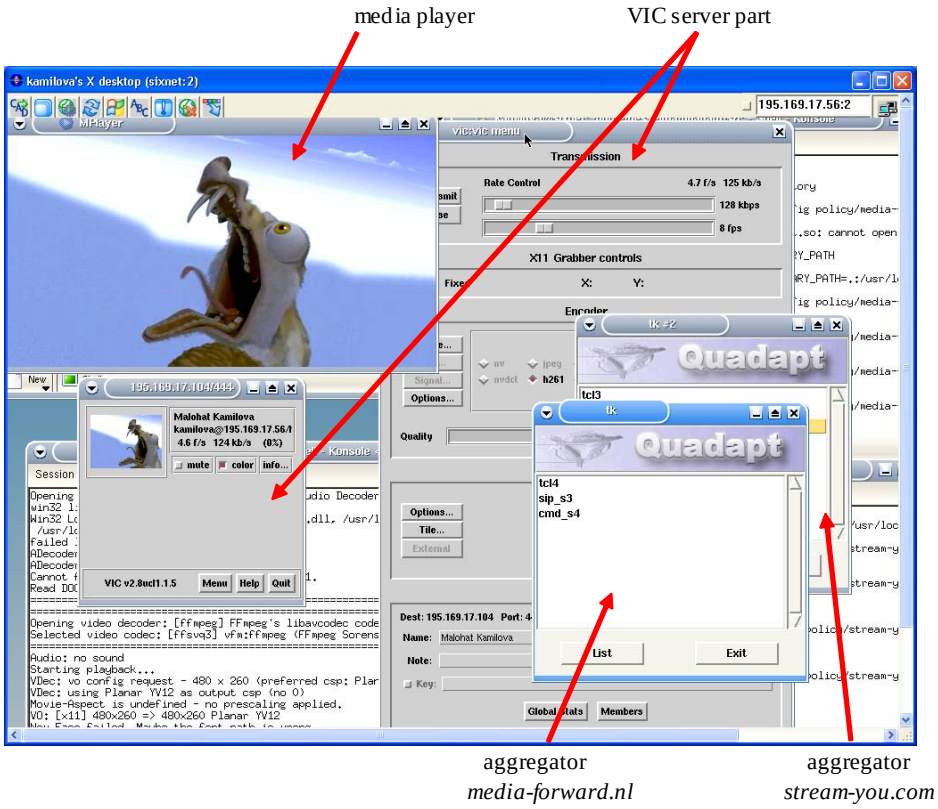

Figure 36. Server side software running at the aggregator server.

Figure 36 shows running processes at the server side: VIC server part, media player and two aggregator processes.

After that we run the client. Figure 37 shows (left side) the user interface through which the user requests to connect to a selected channel (e.g. CNN video) and a VIC client part (right side) where the user is going to watch the selected channel.

We start our scenario at point A of the Figure 35. In our implementation, the default user preferences are:

maximum target quality level	*GOLD*
adaptation	*yes*
viewing smoothness	*HIGH*
maximum target price	*30 cents per minute*

Point A.

Bob selects a CNN video channel though the user interface and waits for the connection, Figure 37.

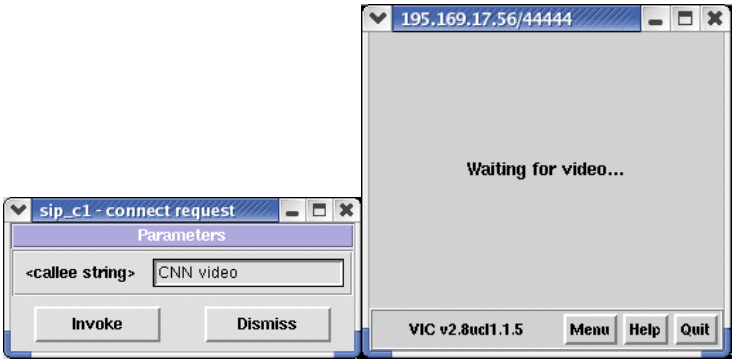

Figure 37. User selects a channel to watch at point A.

At this point there is only one aggregator: *media-forward.nl* available through the WLAN network. Bob gets the *bronze* configuration of the channel. (*Gold* subscriber Bob accepts a lower quality configuration- *bronze), Figure 38.* The left side of Figure 38 shows the status messages: messages about the status of the connection, as if we are peaking into the inside of the APPLE system to know what is going on there, what policy decisions are made and what are the current configuration details. The right side of the Figure 37 shows the video that is being streamed.

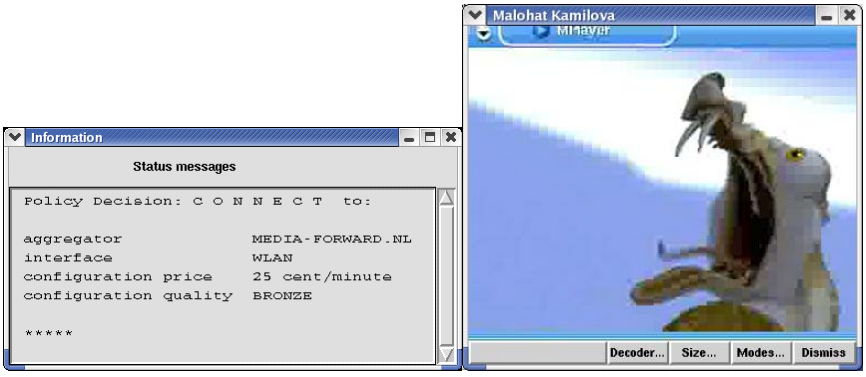

Figure 38. User is connected at point A.

Point B.

Bob roams into a hotspot at point B. We simulate this scenario by plugging the Ethernet cable into the laptop. At point B, two aggregators are available: *media-forward.nl* through the WLAN network and *stream-you.com* aggregator through the Ethernet.

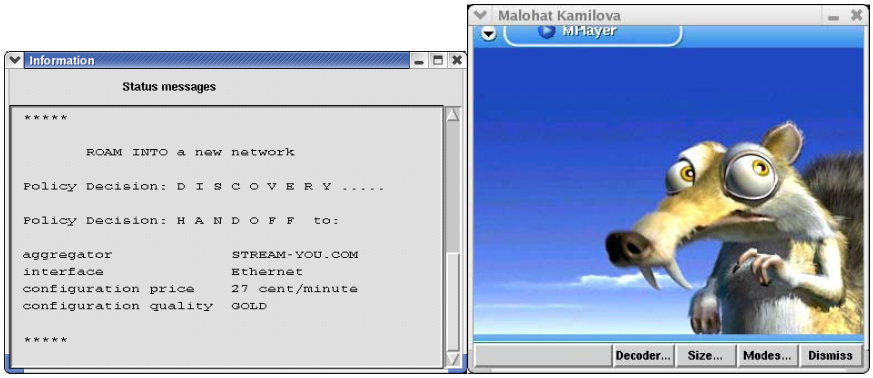

Figure 39. Handoff to a better alternative at point B.

The mobile host of Bob automatically discovers and switches to the *stream-you.com* aggregator, because it offers a better configuration of the same channel, in terms of the quality. Now Bob gets the *gold* configuration of this channel, which provides higher quality, in terms of the framerates and bitrates, Figure 39.

Point C.

Bob roams out of the hotspot. We simulate this scenario by unplugging the Ethernet cable from the laptop. At this point aggregator *stream-you.com* becomes unreachable.

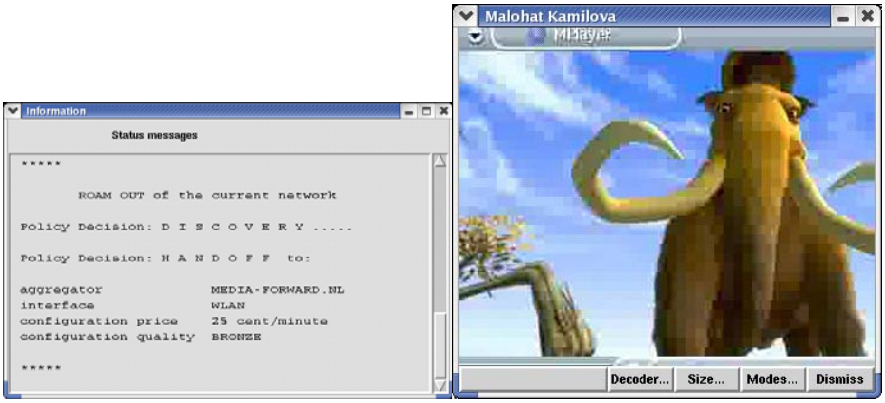

Figure 40. Handoff to available alternative at point C.

Bob's mobile host executes a handoff to aggregator *media-forward.nl* and gets *bronze* configuration of the channel (lower quality), Figure 40.

Point C-D.

Bob changes his preferences at while he is receiving a channel through the user interface. For example, he prefers to receive the channel in better quality and changes the price to 60 cents/m to receive a gold quality configuration of the channel, Figure 41.

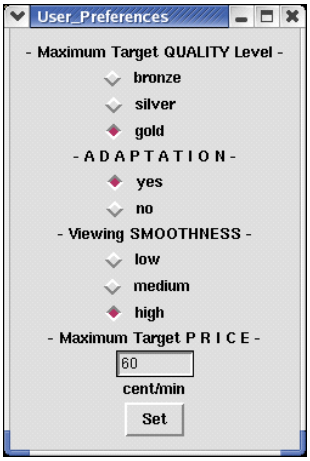

Figure 41. User changes his preferences.

The mobile host discovers and performs a handoff to the *gold* configuration of the current channel, which is selected according to Bob's new preferences, Figure 42.

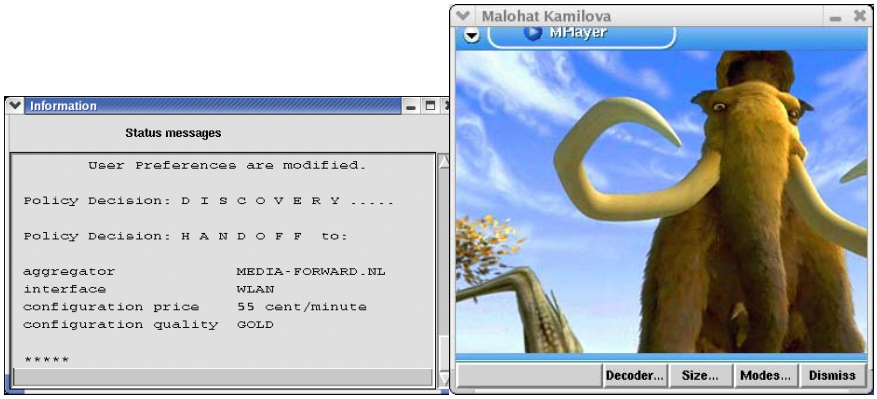

Figure 42. Handoff to the gold quality level..

The quality of the channel is improved from *bronze* to *gold*, but the price of the channel version is higher. This is pure application level handoff because the mobile host is handed off between different configurations of the same channel at the same aggregator, without handing of to another network.

Point D.

Bob arrives to his office and roams into another hotspot. We simulate this scenario by plugging the Ethernet cable back into its slot on the laptop. At this point again there are two aggregators available: *media-forward.nl* through the WLAN network and *stream-you.com* aggregator through the Ethernet network.

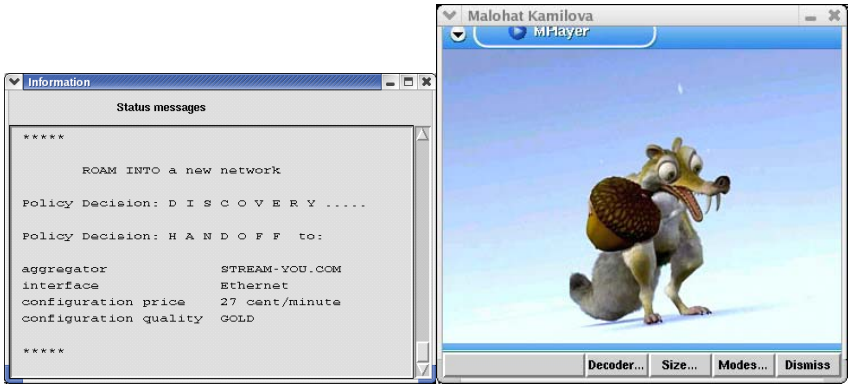

Figure 43. Handoff to a better alternative at point D.

Bob's mobile host again discovers better alternative of the channel (in terms of price and quality), which is available now through the office network. It performs handoff to *stresm-you.com* aggregator that provides the *gold* quality level of the same channel for a *cheaper price*, Figure 43.

7. CONCLUSIONS

In this Chapter we review the contribution of this thesis, our conclusions and discuss the points for further investigation. This chapter is structured as follows: Section 7.1 presents our conclusions. It also summarizes the main contribution of this thesis. Section 7.2 identifies some future work.

7.1 Conclusions and observations

This research was motivated by the necessity for a flexible control mechanism for the CORD system to control discovery and handoff initiation.

The literature "promotes" policy based systems to be flexible. Therefore, in designing a flexible system we followed a policy-based approach. Our approach to design the flexible control mechanism for the CORD system included:

1. A literature survey into policies and policy-based systems, automatic service selection, policy based handoffs in heterogeneous environments and policy-based QoS provisioning.

2. The design of a set of simple policies to control the CORD system

3. The design of the architecture of the policy based system, which builds on the existing CORD system.

4. The implementation of a prototype of the control system.

5. The integration of the control mechanism with the CORD system and testing it on the testbed.

In this research, we propose and application level handoff control system, the APPLE system, to control the existing CORD system. The APPLE system, using policies, ensures service continuity and adaptation. In particular, it enables a mobile host to automatically select the best one among alternative aggregators, offering the same service with different quality or price through different wireless networks.

In the design of the APPLE system, we have applied a policy-based control model, which is inspired by the IETF policy framework. The APPLE system is a controlling entity for the CORD protocol entity.

The main contribution of this research is the design and implementation the APPLE system for the CORD system that tackles the following challenges:

1. It automatically triggers the CORD protocol based on the policy evaluation,
2. It selects the best aggregator among discovered ones in accordance with the user preferences and network capabilities.

Another contribution of this thesis is a poster paper [56], which was published in the proceedings of 13th Mobile and Wireless Summit, held in Lyon, France, on 26-29th of June 2004, see Appendix E.

Conclustions*:*

- This prototype implementation demonstrates the flexibility of policy based control, which means that our approach on policy-based control is a promising solution.
- The APPLE system fulfils the requirements as a control mechanism for the CORD system.
- Using policies we can control the behavior of the mobile host in a flexible way.
- This is a pilot research that suggests future investigation in many directions (see Section 7.2).
- Although the IETF's policy model is mainly used for management purposes at network level, our work suggests that the IETF model can be applied at the application level for control purposes.

7.2 Future work

This research was our pilot research in the area of application level handoffs using policies. The scope of this research was limited to the policies that control the discovery and handoff behavior of mobile hosts. We can however extend our initial investigations in many directions.

1. *New policies for the mobile host.*

There are several new policies we have in mind for the APPLE system to extend the behavior of the mobile host, but due to time limitations we did not implement them. These policies can be considered as a future work:

- policies based on the packet losses reported by RTCP entity
- policies to download new policies when the policy repository is updated
- policies based on the time and date events, such as rush hour, weekend etc.
- policies based on the event on the remaining battery power on the mobile host.
- policies based on the velocity of the mobile user.

2. *Distribution of the components*

In this research we placed all components of the policy-based system on the mobile host.
To reduce the complexity of the mobile hosts, the components of the policy-based system can also be distributed. We could for instance use:

- a remote PDP, to provide policy decision from a centralized PDP, which would facilitate the conflict resolution and consistent policy decisions to be enforced in different nodes.

- a remote policy repository, for instance to maintain a consistency of the policies in several user devices.

- a remote environment monitor, for instance to check the load and stability inside a network.

The challenges in a more distributed setting includes the design of the protocols to communicate with the remote components of the policy-based system. The wireless link might introduce problems, such as higher delays, higher loss rate and a lower degree of security.

3. *A notification protocol*

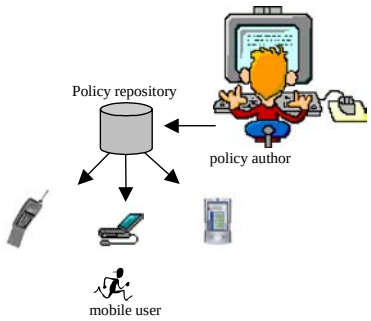

Figure 44. Remote policy repository.

If we use a centralized policy repository, it is possible to modify the policies by an administrator or the user, so that all devices that belong to this administrative domain (e.g. company) or the user can be configured with policies in a consistent manner, Figure 44. An interesting issue in this case is to design a notification model of interactions between the PDP (e.g. notification protocol) and the remote policy repository, so that when the policy repository is updated a notification should be sent to the PDP. This notification can be a simple notification message to the PDP telling that the policy repository is updated or by sending (pushing) new policies right away to the PDP.

4. *Policy specification language and a parser*

In this research the parser implementation is not linked to the policy evaluator. The integration of the parser to the APPLE system remains a future work. It is recommended to investigate more in the policy specification approaches and to find out maybe a better way to specify policies rather than in XML.

5. *Context – based policies*

It is also an interesting issue to download the policies according to the context, for example current network conditions (e.g. available bandwidth), location of the user, time of the day, type of the channel (e.g. news, entertainment etc.). The context-based policies might have in their conditions a context information such as the type of a channel the user has selected. The policy goal might be high viewing smoothness if the channel for example, is an important for the user news channel or a football game. The policy goal might be moderate, if the channel for example is a music channel.

6. *Policies of aggregators (and network operators).*

In this research was only considered policies that control the behavior of mobile hosts. The system can be extended also with the aggregator and operator policies.

7. *Conflict detection and resolution*

Downloading new policies into the mobile host might create policy conflicts. Policies of aggregators, operators and mobile hosts might conflict as well, when they run concurrently. It is a challenge to design and develop a mechanism to detect these conflicts before the enforcement of the policy action, or even at parsing the downloaded policies.

8. *Policy prioritization*

When the mobile hosts and the aggregators have their own policies, it would be interesting to explore the behavior of the mobile host that is now controlled not only by its own policies but also by the policies of the aggregators. An interesting issue is how to prioritize the enforcement of the policies in the mobile host to facilitate the control of the mobile host.

9. *The impact of policies on network performance and stability.*

The network performance and stability might be affected by the policies used by the mobile hosts, for example the policies with the goal *high viewing smoothness* will force the mobile hosts to behave proactively. This behavior results in increasing the load in the network by triggered discovery and handoff in an early stages of the packet losses etc. It would be interesting to test the network performance for different policies, e.g. with different goals.

References

[1] M. Haardt and W. Mohr, “The Complete Solution for Third-Generation Wireless Communications: Two Modes on Air, One Winning Strategy”, IEEE Personal Communications, December 2000

[2] C. Hesselman, H. Eertink, I. Widya and E. Huizer, “A Mobility-aware Broadcasting Infrastructure for a Wireless Internet with Hotspots”, Proceedings of the First ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (WMASH’03), San Diego, USA, September 2003

[3] C. Hesselman, H. Eertink, I. Widya and E. Huizer, ”Delivering Live Multimedia Streams to Mobile Hosts in a Wireless Internet with Multiple Content Aggregators”, to appear in Mobile Networks and Applications Journal (MONET), special issue on Wireless Mobile Applications and Services on WLAN Hotspots, Summer 2005

[4] C. Hesselman, I. Widya, H. Eertink and E. Huizer, “A Comprehensive Framework for Broadcasting Multimedia Content in the Future Mobile Internet”, Proceedings of the 2nd IEEE Workshop on Applications and Services in Wireless Networks (ASWN’02), Paris, France, July 2002

[5] H. Schulzrinne, “Dynamic Host Configuration Protocol (DHCP-for-IPv4) Option for Session Initiation Protocol (SIP) Servers”, RFC 3361, August 2002

[6] M. Handley, V. Jacobson, “SDP: Session Description Protocol”, RFC 2327, April 1998

[7] L. Kleinrock, “An Internet Vision: the Invisible Global Instrastructure”, AdHoc Networks Journal, Vol. 1, No. 1, pp. 3-11, July 2003

[10] M. Cox and R. Davidson, “Concepts, Activities and Issues of Policy-based Communications Management”, BT Technology Journal, Volume 17, Issue 3, July 1999, Pages 155 - 169

[11] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry and S. Waldbusser, “Terminology for Policy-Based Management”, RFC 3198. November 2001

[12] S. Calo and M. Sloman, “Policy-Based Management of Networks and Services”, Journal of Network and Systems Management, Vol.11, No.3, September 2003, pp. 249-252.

[13] W. Zhuang, Y.S. Gan, K.J. Loh and K.C.Chua, Policy-based QoS management architecture in an integrated UMTS and WLAN environment. IEEE Communications Magazine, November 2003.

[14] N. Damianou, A. Bandara, M. Sloman and E. Lupu, “A Survey of Policy Specification Approaches”, Department of Computing, Imperial College of Science Technology and Medicine, London, April 2002.

[15] J. Moffet and M. Sloman. Policy Hierarchies for Distributed Systems Management. IEEE JSAC 11(9 Special Issue on Network Management): 1404-14, December 1993.

[16] H. Lutfiyya, G. Molenkamp, M. Katchabaw and M. Bauer, “Issues on managing soft QoS requirements in distributed systems using a policy-based framework”. http://www-2.cs.cmu.edu/~bumba/filing_cabinet/papers/lutfiyya-policy.pdf

[17] H.Wang, R. Katz, J.Giese, “Policy-Enabled Handoffs Across Heterogeneous Wireless Networks”, 2nd IEEE Workshop on Mobile Computing and Applications (WMCSA 1999), New Orleans, USA, Feb. 1999.

[18] R.Yavatkar, D. Pendarakis and R. Guerin, “A Framework for Policy-based Admission Control”, RFC 2753, January 2000

[19] K. Murray, R. Mathur, D. Pesch, Intelligent Access and Mobility Management in Heterogeneous Wireless Networks using Policy, Adaptive Wireless Systems Group, Department of Electronic Engineering, Cork Institute of Technology, Ireland.

[20] D.D. Clark, J. Wroslawski, “The Personal Router whitepaper”. MIT Technical Report, 2000.

[21] G. Lee, P. Faratin, S, Bauer and J. Wroslawski, “Automatic Service Selection in Dynamic Wireless Network Environments”, a poster presentation at MobiCom (co-located with First ACM International Workshop WMASH’03), San Diego, USA, September 2003

[22] D. Xu, K. Nahrstedt, “Supporting Multimedia Service Polymorphism in Dynamic and Heterogeneous Environments”, Technical Report UIUCDCS-R-2000-2159, University of Illinois at Urbana-Champaign, USA, October 2000

[23] T. Plagemann, V. Goebel, L. Mathy, N. Race, and M. Zink, “Towards Scalable and Affordable Content Distribution Services”, Proc. 7th International Conference on Telecommunications (ConTEL 2003), Zagreb, Croatia, June 2003

[24] J. Chennikara, W. Chen, A. Dutta, O. Altintas, “Application-Layer Multicast for Mobile Users in Diverse Networks”, IEEE Globecom 2002, Taipei, Taiwan, November 2002

[25] M. Cox and R. Davison, “Concepts, Activities and Issues of Policy-based Communications Management”, BT Technology Journal, Volume 17, Issue 3, July 1999, Pages 155 - 169

[26] H. Lytfiyya, F. Garcia and J. Moffett, , “Policy 2003: Workshop on Policies for Distributed Systems and Networks”, Journal of Network and Systems Management, Vol.11, No.3, September 2003, pp. 373-376.

[27] INTAP, Survey on Policy-Based Networking, Final Report.

[28] I. Liabotis, O. Prnjat, L. Sacks, Policy-based Resource Management for Application Level Active Networks, University College London, England.

[29] N. Dulay, E. Lupu, M. Sloman and N. Damianou, “A Policy Deployment Model for the Ponder Language”. Proceedings IM 2001: 2001 IEEE/IFIP International Symposium on Integrated Network Management, Seattle, USA, May 2001, pp. 529-544.

[30] J.M. Bradshaw, “Making Agents Acceptable To People”, CEEMAS 2003, Lecture Notes in Computer Science, vol. 2691/2003.

[31] B. Moore, E. Ellesson et al, Policy Core Information Model – Version 1 Specification, Network Working Group - RFC3060, 2001.

[32] P. Linington, Z. Milosevic, K. Raymond “Policies in Communities: Extending the ODP Enterprise Viewpoint”. http://www.dstc.edu.au/Research/Projects/Elemental/resources/edoc98-policypaper.pdf

[33] ISO/IEC IS 10746-2. International Standard 10746- 2, ITU-T Recommendation X.902: Open Distributed Processing - Reference Model - Part 2: Foundations, January 1995.

[34] N. Dulay, “Policy-based Management: the Holy Grail”, presentation at Policy 2004 workshop, New York, June 2004.

[35] A. Beck and M. Hofmann, “IRML: A Rule Specification Language for Intermediary Services”, Lucent Technologies, Internet-draft, February 2001.

[36] N. Damianou, N. Dalay, E. Lupu and M. Sloman. Ponder: A language for specifying security and management policies for distributed systems: The language specification (version 2.1). Technical Report Imperial College Research Report DOC 2000/01, Imperial College of Science, Technology and Medicine, London, England, April 2000.

[37] L. Lymberopoulos, E. Lupu and M. Sloman, “An adaptive Policy Based Framework for Network Services Management”, Journal of networks and Systems management, Special Issue on Policy Based Management of Networks and Services.

[38] R. Montanari, E. Lupu and C. Stefanelli, “Policy-Based Dynamic Reconfiguration of Mobile-Code Applications”, Computer, magazine published by IEEE Computer Society, 2004, pp.73-80.

[39] A. Meissner, S. B. Musunoori, L. Wolf, “MGMS/GML- Towards a new Policy Specification Framework for Multicast Group Integrity”, 2004 IEEE Proc. 2004 Symposium on Applications and the Internet (SAINT 2004), Tokyo, Japan, 26-30 January 2004, pp. 233-239.

[40] D.Durham, J. Boyle , R. Cohen , S. Herzog , R. Rajan and A. Sastry, RFC 2748, COPS, January 2000.

[41] O. Akoz, M. Zeren, “Security Considerations in Mobile IP Networks using Stateful Packet Filtering Firewalls”, Proceedings of the 13th IST Mobile & Wireless Communications Summit, Lyon, France, June 2004, pp 218-222.

[42] N. Nafisi, L. Wang, H. Aghvami, R. Ferrus, A. Gelonch, J. Perez, O. Sallent, R. Agusti, “Extending QoS Policy-based mechanisms to B3G Mobile Access Networks, Proceedings of the 13th IST Mobile & Wireless Communications Summit, Lyon, France, June 2004, pp 518-522.

[43] W. Boehm and P. Braun, “Policy based Architecture for the UMTS Multimedia Domain”, Proceedings of the Second IEEE international Symposium on Network Computing and Applications (NCA’03), 2003.

[44] G. Tonti, J.M. Bradshaw, R. Jeffers, R. Montanari, N. Suri and A. Uszok, Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KaoS, Rei and Ponder. 2nd International Semantic Web Conference (*ISWC2003*), October 20-23, 2003, Sanibel Island, Florida, USA.

[45] DMTF Specification for the Representation of Common Information Model (CIM) in XML, version 2.0, 1999.

[46] N. Blefari Melazzi, G. Ceneri, G. Cortese, N. Davies, N. Dellas, A. Friday, J. Hamard, E. Koutsoloukas, C. Niedermeier, C. Noda, J. Papanis, C. Petrioli, E. Rukzio, O. Storz, J. Urban “The Simpilcity Project: easing the burden of using complex and heterogeneous ICT devices and services” Proceedings of the 13th IST Mobile & Wireless Communications Summit, Lyon, France, June 2004, pp. 746-753.

[47] N. Alonistioti, Z. Boufidis, A. Kaloxylos, M. Dillinger “Integrated Mangement Plane for Policy-based End-to-End reconfiguration Services”, Proceedings of the 13th IST Mobile & Wireless Communications Summit, Lyon, France, June 2004, pp. 132-136.

[48] France Telecom, Demonstration at The 13th IST Mobile & Wireless Communications Summit, Lyon, France, June 2004.

[49] A. Peddemors, H. Zandbelt, and M. Bargh, "A Mechanism for Host Mobility Management supporting Application Awareness " , *In Proceedings of the Second International Conference on Mobile Systems, Applications, and Services (MobiSys'04)*, June 2004.

[50] M. Stemm and R. H.Katz, Vertical Handoff in Wireless Overlay Networks.

[51] http://sax.sourceforge.net/ About SAX.

[52] P. Martinez, M. Brunner, J. Quittek, F. Strauss, J. Schoenwaelder, S. Mertens and T. Klie, "Using the Script MIB for Policy-based Configuration Management".

[53] C. Rigney, A. Rubens, W. Simpson and S. Willens, RFC 2138 Remote Authentication Dial In User Service (RADIUS), April 1997

[54] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996

[55] C.W. Ng, P. Y. Tan and H. Cheng, "Quality of Service Extension to IRML", Panasonic Singapore Labs, Internet-draft, July 2001.

[56] M.I. Kamilova, C. Hesselman, I. Widya and E. Huizer, "A Policy-based System for Handoffs between Intermediary Content Providers in the Wireless Internet" (poster paper), Proceedings of the 13th IST Mobile & Wireless Communications Summit, Lyon, France, June 2004, pp. 813-817.

[57] http://www.freeband.nl/kennisimpuls/projecten/4gplus/ENindex.html

[58] P. Vidales, R. Chakravorty, C. Policroniades , "PROTON: A Policy-based Solution for Future 4G devices".

[59] F. Caldeira and E. Monteiro, "A policy-based approach to firewall management".

[60] M. Kangasluoma, "Policy Specification Languages", draft, Helsinki University of Technology, 1999.

Appendix A Abbreviations

APPLE	APPlication Level Policy-based Handoff Control System
CORD	Continuous Channel Reception System for Mobile Devices
DiffServ	Differentiated Service
DMTF	Desktop Management Task Force
GGSN	Gateway GPRS support node
GPRS	General Packet Radio Service
IETF	Internet Engineering Task Force
IntServ	Integrated Service
IP	Internet Protocol
ISP	Internet Service Provider
OMG	Object Management Group
QoS	Quality of Service
RSVP	Resource ReSerVation Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SLA	Service Level Agreement
TV	Television
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System
WLAN	Wireless LAN

Appendix B Examples of policies

Policy Class Discovery

```
POLICY 1

        /* policy_class = discovery, leaving hotspot
         * policy_goal = high_viewing_smoothness
         */

        if (packet_loss >= 20% && receiving_interface == “802.11”) {

        /* Invoke discovery */
          pep.set_collection_time(short);
          pep.invoke_discovery_protocol();
        }


POLICY 2

        /* policy_class = discovery, leaving hotspot
         * policy_goal = moderate_viewing_smoothness
         */

        if (packet_loss >= 50% && receiving_interface == “802.11”) {

        /* Invoke discovery */
           pep.set_collection_time(default);
           pep.invoke_discovery_protocol();
        }


POLICY 3

        /* policy_class = discovery, leaving hotspot
         * policy_goal = low_viewing_smoothness
         */

        if (packet_loss >= 80% && receiving_interface == “802.11”) {

        /* Invoke discovery */
          pep.set_collection_time(default);
          pep.invoke_discovery_protocol();
        }
```

This is a discovery policy telling the system how to behave if the user roams out of the network coverage through which he is receiving the channel.

In order to provide seamless roaming, the mobile host should handoff to another configuration that is available through another network. The policies 1-3 are discovery rules, defining when to initiate discovery of alternatives to possible handoff in order to achieve the goal of the policy. Depending on the goal the condition part of the policies 1, 2 and 3 slightly differ.

These policies are based on the packet loss percentage on the receiving network interface. These policies force the mobile host to query available configurations from the aggregators when the packet loss value exceeds 20% (high viewing smoothness), 50 % (moderate viewing smoothness) and 80 % (low viewing smoothness).

The conditions of these policies define proactive (POLICY 1) or reactive (POLICY 2 and POLICY 3) behaviour of the system based on the packet loss value.

POLICY 4

```
/* policy_class =discovery, leaving hotspot
 * policy_goal= high_viewing_smoothness
 */

if (signal_strength== MEDIUM && receiving_interface == “802.11”) {

/* Invoke discovery */
   pep.set_collection_time(short);
   pep.invoke_discovery_protocol();
}
```

POLICY 6

```
/* policy_class =discovery, leaving hotspot
 * policy_goal= high_viewing_smoothness
 */

if (signal_strength== MEDIUM && receiving_interface == “UMTS”) {

/* Invoke discovery */
   pep.set_collection_time(short);
   pep.invoke_discovery_protocol();
}
```

POLICY 5

```
/* policy_class =discovery, leaving hotspot
 * policy_goal= moderate/low_viewing_smoothness
 */

if (signal_strength== LOW && receiving_interface == “802.11”) {

/* Invoke discovery */
   pep.set_collection_time(default);
   pep.invoke_discovery_protocol();
}
```

POLICY 7

```
/* policy_class =discovery, leaving hotspot
 * policy_goal= moderate/low_viewing_smoothness
 */
```

```
if (signal_strength== LOW && receiving_interface == “UMTS”) {

/* Invoke discovery */
   pep.set_collection_time(default);
   pep.invoke_discovery_protocol();
 }
```

POLICIES 4 - 7 have the same semantics as POLICIES 1 - 3, but the conditions of these policies are based on the value of the signal strength on the receiving network interface.

These policies tell the system when to discover alternative configurations based on the signal strength on the receiving interface.

POLICY 4 and POLICY 5 are used when the receiving interface is 802.11.
POLICY 6 and POLICY 7 are used when the receiving interface is UMTS.

Policy Class Discovery

POLICY

```
/* policy_class =discovery
 * policy_goal= high_viewing_smoothness
 */

if (user_preference_changed) {

/* Invoke discovery */
   pep.set_collection_time(short);
   pep.invoke_discovery_protocol();
}
```

Policy Class Handoff

POLICY 8

```
/* policy_class =handoff, handoff execution
 * policy_goal= high_viewing_smoothness
 */

if (configuration_list &&
    better_alternative(configuration_list, current_configuration)) {

/* First connect, then disconnect */
   pep.connect_to(new_configuration);
   pep.disconnect_from(old_configuration);
}
```

POLICY 9

```
/* policy_class =handoff, handoff execution
 * policy_goal= moderate/low_viewing_smoothness
 */

if (configuration_list &&
    better_alternative (configuration_list, current_configuration)) {
```

```
   /* First disconnect, then connect */
      pep.disconnect_from(old_configuration);
      pep.connect_to(new_ configuration);
   }
```

These policies define the rules for an execution of handoff, if newly selected configuration is better than the current configuration..

POLICY 8 has a goal “high viewing smoothness”. To achieve this goal, the policy action defines the rules to execute handoff: first connect to a selected aggregator, then disconnect from the old aggregator, providing smooth handoff from one stream to another stream.

POLICY 9 defines the rule for executing handoff to provide moderate viewing smoothness. To accomplish this the policy action tells first to disconnect from the current aggregator and then to connect to the new aggregator. This might result in some data loss and service degradation during the handoff, but still inline with the goal of the policy, which is “moderate / low viewing smoothness”.

POLICY 10

```
   /* policy_class =handoff, handoff execution
    * policy_goal= cheapest service  as much as possible
    */

   if (configuration_list &&
       better_alternative (configuration_list, current_configuration)) {


   /* First disconnect, then connect */
      pep.disconnect_from(old_configuration);
      pep.connect_to(new_ configuration);
   }
```

This policy defines the rule for an execution of handoff, if the user wants always to be connected to the cheapest service.

Obviously, that first disconnecting and then connecting to the service provider will cost less amount of money for the user, rather then other way around. Because in the latter case, the mobile host will receive streams of the same channel on two interfaces simultaneously for some time, which might be costly for the user.

Policy Class Handoff

POLICY 11

```
   /* policy_class =handoff, high speed roaming
    * policy_goal= high_viewing_smoothness
    */

   if (user_speed>=100 && handoff_flag && receiving_interface == “UMTS”)

   /* Don’t handoff to WLAN interface */
     pep.disable_handoff(wlan);
```

POLICY 12

```
/* policy_class =handoff, high speed roaming
 * policy_goal= high_viewing_smoothness
 */

if (user_speed>=100 && handoff_flag && receiving_interface == 802.11”)
{

/* Don’t handoff to WLAN interface */
   pep.handoff_to(umts);
   pep.disable_handoff(wlan);
}
```

These policies define the rules to avoid continuous frequent handoffs from one to another network (i.e. WLAN with a small coverage) in the situation when the user moves with high speed within overlay networks.

POLICY 11 and POLICY 12 disable handoff to WLAN networks, so that the system stays connected to the global available (e.g. UMTS) network at that time. This policy is designed to prevent a “ping-pong” effect during the roaming. Disabling unnecessary handoffs will keep the viewing smoothness stable during user movements.

POLICY 13

```
/* policy_class =handoff, entering hotspot
 * policy_goal= cheapest service as much as possible
 */

if (foreign_network_flag)

/* Disable handoff to foreign network */
   pep.disable_handoff(foreign_network);
```

This policy is used when the user does not want to use his mobile device in the foreign network, because the foreign network might be expensive for the user. Therefore this policy has a goal “cheapest service”. This policy disables handoff to a foreign network.

Policy Class Discovery

POLICY 14

```
/* policy_class = discovery, entering hotspot
 * policy_goal = cheapest service as much as possible
 */

if (new_IP_address_assignment) {
 //assignment of IP address to WLAN interface

/* Invoke discovery */
  pep.set_collection_time(short);
  pep.invoke_discovery_protocol();
```

```
        }
```

POLICY 15

```
        /* policy_class = discovery, entering hotspot
         * policy_goal = highest quality level
         */

        if (new_IP_address_assignment) {
        //assignment of IP address to WLAN interace

        /* Invoke discovery */
          pep.set_collection_time(default);
          pep.invoke_discovery_protocol();
        }
```

These policies define the behavior of the system, when a new IP address is assigned to one of mobile host's interfaces.

Assigning a new IP address indicates that the mobile host has roamed into a new network that might offer better configuration of the same channel that user receives at that moment.

As a result, discovery protocol is invoked to inquire available configurations from all aggregators in the mobile host vicinity through all interfaces, including the interface with newly assigned IP address. Thus, the mobile host takes advantage of the user mobility to discover "better" versions of the channel for the user.

Appendix C Fragments from the implementation code

Some functions of the *configuration selector*

```
/*
 * validates the configuration against user preferences and the bitrate
 * that can be supported on the interface
 */
void
validate_config(int price, int quality, int framerate, int bitrate, sdp_attribute_t
*att_rtpmap, sdp_attribute_t *att_fmtp){

        // check whether the interface can support the configuration
        if(intrfeis->bandwidth>bitrate){

                // if the user allows the switching to another quality level
                if (user_pref->adaptation==1){

                        // check the quality in user preferences
                        if (price<=user_pref->price && define_qty(quality,
                                framerate)<=user_pref->quality){

                                // add to the valid list
                                add_to_validlist(sdp_one, valid_media(), uac_one);

                        }
                // if the user does not allow the switching to another quality level
                }else {
                        if (price<=user_pref->price && define_qty(quality,
                                framerate)==user_pref->quality){
                                count_c++;

                                // add to the valid list
                                add_to_validlist(sdp_one, valid_media(), uac_one);

                        }
                }
        }
}

/*
 * defines the quality level according to the quality and framerate from the sdp file.
 *
 */
int
define_qty(int quality, int framerate){
        int q_level;

        if((framerate<=30 && framerate>=21) && (quality>=71 && quality<=100))
                q_level = GOLD;

        if((framerate<=20 && framerate>=11) && (quality>=31 && quality<=70))
                q_level = SILVER;

        if((framerate<=10 && framerate>=0) && (quality>=1 && quality<=30))
                q_level = BRONZE;

        return q_level;
}



/**
 * final ordering of selected configurations
```

```
 */
list_t
*order_configurations(){
	// compare configurations and order them according quality
	struct tuples* t;
	best_tuple=(struct tuples*)malloc(sizeof(struct tuples));

	for(i=0; i<list_size(ordered_configuration_list); i++){
		for(j=0; j<list_size(ordered_configuration_list)-1; j++){
			first=(struct tuples*)list_get(ordered_configuration_list,j);
			second=(struct tuples*)list_get(ordered_configuration_list,j+1);

			aggr=first->uac;
			aggr2=second->uac;
			config=first->sdp;
			config2=second->sdp;

			// get media lists from configrations
			m=config->m_medias;
			m2=config2->m_medias;

			// get the media structs from the media lists
			media_struct=(struct sdp_media_t*)list_get(m,0);
			media_struct2=(struct sdp_media_t*)list_get(m2,0);

			// get the attribute lists from the media structs
			attr=media_struct->a_attributes;
			attr2=media_struct2->a_attributes;

			retrieve_data(attr, attr2);   // retreives media attributes data

			// get the quality level
			p1=define_qty(compare_qty1, compare_fr1);
			p2=define_qty(compare_qty2, compare_fr2);

			// comparison part
			if(p1<p2){// compare the quality
				swap=*first;
				*first=*second;
				*second=swap;
			}else if (p1==p2){// if quality is the same
				if(compare_br1<compare_br2){// compare the bitrate
					swap=*first;
					*first=*second;
					*second=swap;
				}else if (compare_br1==compare_br2){
					// if bitrate is the equal
					if(compare_pr1>compare_pr2){ // compare the price
						swap=*first;
						*first=*second;
						*second=swap;
					}
				}
			}
		}
	}

	return ordered_configuration_list;
}

/*
 * compares current configuration with the new configuration
 */
int
find_better_alternative(list_t *all_configurations, struct tuples
*current_configuration){
	// compare configurations and order to find better one this function sets the
	// variable better_alternative that is used for the handoff policy evaluation

	int found=0;
	second=(struct tuples*)list_get(all_configurations,0);
	first=current_configuration;
	aggr=first->uac;
	config=first->sdp;
	aggr2=second->uac;
	config2=second->sdp;
```

```
        // get media lists from configurations
        m=config->m_medias;
        m2=config2->m_medias;

        // get the media structs from the media lists
        media_struct=(struct sdp_media_t*)list_get(m,0);
        media_struct2=(struct sdp_media_t*)list_get(m2,0);

        // get the attribute lists from the media structs
        attr=media_struct->a_attributes;
        attr2=media_struct2->a_attributes;
        retrieve_data(attr, attr2);   // retrieves media attribute

        // get the quality level of the configuration
        p1=define_qty(compare_qty1, compare_fr1);
        p2=define_qty(compare_qty2, compare_fr2);

        if(p1<p2){// compare quality
                found=1;
        }else if (p1==p2){
                if(compare_br1<compare_br2){// compare bitrate
                        found=1;
                }else if (compare_br1==compare_br2){
                        if(compare_pr1>compare_pr2){ // compare price
                                found=1;
                        }
                }
        }
        if(found==0) printf("\n\n  better alternative is NOT FOUND\n");
        else printf("\n\n  better alternative is FOUND\n");
        return found;
}
```

Some functions of the *policy evaluator*

```
/*
 * initializes the POLICY EVALUATOR
 * creates the policy list
 * loads proper policies
 * reads user preferences
 */
void
policy_evaluator_init(){

        ordered_configuration_list=(list_t *)malloc(sizeof(list_t));
        list_init(ordered_configuration_list);

        // call the user unterface to read preferences
        read_user_preferences();

        // call the policy repository to create the list of policies
        create_policies();

        // create a list for storing active policies
        active_policies=(list_t *)malloc(sizeof(list_t));
        list_init(active_policies);

        // load policies to active policies list
        load_policies(user_pref->smoothness);

        // allocate memory for a decision structure
        decision=(struct decision *)malloc(sizeof(struct decision));

        printf("\nPOLICY EVALUATOR is initialized successful.");
}
```

```
/*
 * loads the policies with the goal matching to the user preferences
 *
 */
void
load_policies(int smoothness){

        int z;
        struct policy *temp_policy;

        // clean the list before loading the policies
        while(list_size(active_policies)!=0){
                list_remove(active_policies, 0);
        }
        for (z=0; z<list_size(policies); z++){
                temp_policy= (struct policy *)list_get(policies, z);
                if(strcmp(temp_policy->goal, "SMOOTHNESS")==0){
                        if(temp_policy->goal_value == smoothness){
                                list_add(active_policies, temp_policy, -1);
                        }
                }
        }
        printf("\n Policy Repository contains %d policies.\n", list_size(policies));
        printf("\n Policy List contains %d active policies\n",list_size(active_policies));
        print_policies(active_policies);
}


/*
 * this is the interface between the CONTROLLER and Policy evaluator
 * to be called by CONTORLLER
 * returns the policy decision
 */

struct
decision *evaluate_policies(struct event *ev){
        int y;
        struct policy *temp;
        struct condition *temp_c;
        struct action *temp_a;

        // read the event structure
        event_incoming=ev->type;

        if(event_incoming==ROAM_IN){

                real_network_interface=ev->receiving_interface;

                // traverse the list for discovery policies
                for(y=0; y<list_size(active_policies); y++){
                        temp= (struct policy *)list_get(active_policies, y);
                        temp_c=temp->condition;
                        temp_a=temp->action;
                        if(strcmp(temp->policy_class, "DISCOVERY")==0){
                          if(temp_c->new_network_flag==1){
                            if(temp_c->receiving_interface == real_network_interface){
                            // set the collection time and the policy decision type
                                decision->d_collection_time=temp_a->collection_time;
                                decision->policy_decision=temp_a->decision_type;
                            }
                          }
                        }
                }
        }
        if (event_incoming==ROAM_OUT){

                real_network_interface=ev->receiving_interface;
                real_signal_strength=ev->signal_strength;

                // traverse the list for discovery policies
                for(y=0; y<list_size(active_policies); y++){

                        temp= (struct policy *)list_get(active_policies, y);
                        temp_c=temp->condition;
                        temp_a=temp->action;
```

```
                        if(strcmp(temp->policy_class, "DISCOVERY")==0){
                                if(temp_c->receiving_interface == real_network_interface){
                                  if(temp_c->signal_strength == real_signal_strength){
                                    // set the collection time and the policy decision type
                                    decision->d_collection_time=temp_a->collection_time;
                                    decision->policy_decision=temp_a->decision_type;
                                  }
                                }
                        }
                }
        }

        else if (event_incoming==CONFIGURATIONS){

                // allocate memory for an aggregator structure
                aggreg = (struct aggregator *)malloc(sizeof(struct aggregator));

                aggreg->aggregator_name=ev->aggregator_id;
                aggreg->aggregator_interface=ev->aggregator_intf;
                config_received=1;

                // call get_configurations function and get the processed list

                ordered_configuration_list=get_configuration(aggreg, ev->sdp);

                if(list_size(ordered_configuration_list)!=0){
                        display_list(ordered_configuration_list);

                        if(start_up==0)
                                better_alternative=1;
                        else
                                better_alternative= find_better_alternative
                                (ordered_configuration_list, current_configuration);

                        // traverse the list for handoff policies
                        for(y=0; y<list_size(active_policies); y++){

                                temp= (struct policy *)list_get(active_policies, y);
                                temp_c=temp->condition;
                                temp_a=temp->action;

                                if(strcmp(temp->policy_class, "HANDOFF")==0){
                                  if(config_received && better_alternative){
                                    if(strcmp(temp_a->connect_order, "CONNECT_FIRST")==0){
                                      // set the decision type,
                                      // configuration list and handoff strategy
                                      decision->policy_decision=temp_a->decision_type;
                                      decision->d_configuration_list=
                                               ordered_configuration_list;
                                      decision->connect=FIRST;
                                      decision->disconnect=SECOND;
                                    }else if(strcmp(temp_a->connect_order,
                                               "CONNECT_SECOND")==0){
                                      decision->policy_decision=temp_a->decision_type;
                                      decision->d_configuration_list=
                                               ordered_configuration_list;
                                      decision->connect=SECOND;
                                      decision->disconnect=FIRST;
                                    }else{}
                                  }
                                }
                        }
                }
                else if(list_size(ordered_configuration_list)==0){
                        // if the list is empty
                        decision->policy_decision="DISCONNECT";
                }else { }
        }else { }
        return decision;
}
```

Some hard coded policies of the *policy evaluator* used in the prototype

```
/*
 * initialize a policy structure
 */
struct policy *init_policy(){

        struct policy *new_policy;

        // allocate memory for policy, condition and action structures
        new_policy=(struct policy *)malloc(sizeof(struct policy));
        a_condition=(struct condition *)malloc(sizeof(struct condition));
        an_action=(struct action *)malloc(sizeof(struct action));

        // assign initial values to the policy struct
        new_policy->goal=NULL;
        new_policy->goal_value=-1;
        new_policy->policy_class=NULL;

        a_condition->configuration_list=-1;
        a_condition->better_alternative=-1;
        a_condition->new_network_flag=-1;
        a_condition->receiving_interface=-1;
        a_condition->signal_strength=-1;

        an_action->collection_time=-1;
        an_action->connect_order=NULL;
        an_action->discover_alternatives=-1;
        an_action->decision_type=NULL;

        //add condition and action structs to the new policy struct
        new_policy->condition=a_condition;
        new_policy->action=an_action;

        return new_policy;
}

/*
 * creates a policy list containing of the policies of the APPLE system
 */
void create_policies(){

        policies=(list_t *)malloc(sizeof(list_t));
        list_init(policies);


        // POLICIES ON ROAMING INTO A NEW NETWORK

        // ********************************************
        // 1   D I S C O V E R Y    policy |   ROAM IN   |
        // ********************************************

        // initialize a memory block for a new policy
        a_policy=init_policy();

        a_policy->goal="SMOOTHNESS";
        a_policy->goal_value=HIGH;
        a_policy->policy_class="DISCOVERY";

        a_condition->new_network_flag=1;
        a_condition->receiving_interface=ETH;

        an_action->collection_time=SHORT;
        an_action->discover_alternatives=1;
        an_action->decision_type="DISCOVERY";

        list_add(policies, a_policy, -1);


        // ********************************************
        // 2   D I S C O V E R Y    policy |   ROAM IN   |
        // ********************************************

        // initialize a memory block for a new policy
        a_policy=init_policy();
```

```
a_policy->goal="SMOOTHNESS";
a_policy->goal_value=MODERATE;
a_policy->policy_class="DISCOVERY";

a_condition->new_network_flag=1;
a_condition->receiving_interface=ETH;

an_action->collection_time=DEFAULT;
an_action->discover_alternatives=1;
an_action->decision_type="DISCOVERY";

list_add(policies, a_policy, -1);


// ********************************************
// 3   D I S C O V E R Y    policy |   ROAM IN   |
// ********************************************

// initialize a memory block for a new policy
a_policy=init_policy();

a_policy->goal="SMOOTHNESS";
a_policy->goal_value=LOW;
a_policy->policy_class="DISCOVERY";

a_condition->new_network_flag=1;
a_condition->receiving_interface=ETH;

an_action->collection_time=LONG;
an_action->discover_alternatives=1;
an_action->decision_type="DISCOVERY";

list_add(policies, a_policy, -1);


// ********************************************
// 4   D I S C O V E R Y    policy |   ROAM IN   |
// ********************************************

// initialize a memory block for a new policy
a_policy=init_policy();

a_policy->goal="SMOOTHNESS";
a_policy->goal_value=HIGH;
a_policy->policy_class="DISCOVERY";

a_condition->new_network_flag=1;
a_condition->receiving_interface=WLAN;

an_action->collection_time=SHORT;
an_action->discover_alternatives=1;
an_action->decision_type="DISCOVERY";

list_add(policies, a_policy, -1);


// ********************************************
// 5   D I S C O V E R Y    policy |   ROAM IN   |
// ********************************************

// initialize a memory block for a new policy
a_policy=init_policy();

a_policy->goal="SMOOTHNESS";
a_policy->goal_value=MODERATE;
a_policy->policy_class="DISCOVERY";

a_condition->new_network_flag=1;
a_condition->receiving_interface=WLAN;

an_action->collection_time=DEFAULT;
an_action->discover_alternatives=1;
an_action->decision_type="DISCOVERY";

list_add(policies, a_policy, -1);
```

```
// ********************************************
// 6   D I S C O V E R Y    policy |   ROAM IN   |
// ********************************************

// initialize a memory block for a new policy
a_policy=init_policy();

a_policy->goal="SMOOTHNESS";
a_policy->goal_value=LOW;
a_policy->policy_class="DISCOVERY";

a_condition->new_network_flag=1;
a_condition->receiving_interface=WLAN;

an_action->collection_time=LONG;
an_action->discover_alternatives=1;
an_action->decision_type="DISCOVERY";

list_add(policies, a_policy, -1);


// POLICIES ON SIGNAL STRENGTH


// ********************************************
// 7   D I S C O V E R Y    policy |   ROAM OUT  |
// ********************************************

// initialize a memory block for a new policy
a_policy=init_policy();

a_policy->goal="SMOOTHNESS";
a_policy->goal_value=HIGH;
a_policy->policy_class="DISCOVERY";

a_condition->signal_strength=MODERATE;
a_condition->receiving_interface=WLAN;

an_action->collection_time=DEFAULT;
an_action->discover_alternatives=1;
an_action->decision_type="DISCOVERY";

list_add(policies, a_policy, -1);


// ********************************************
// 8   D I S C O V E R Y    policy |   ROAM OUT  |
// ********************************************

// initialize a memory block for a new policy
a_policy=init_policy();

a_policy->goal="SMOOTHNESS";
a_policy->goal_value=HIGH;
a_policy->policy_class="DISCOVERY";

a_condition->signal_strength=LOW;
a_condition->receiving_interface=WLAN;

an_action->collection_time=SHORT;
an_action->discover_alternatives=1;
an_action->decision_type="DISCOVERY";

list_add(policies, a_policy, -1);


// ********************************************
// 9   D I S C O V E R Y    policy |   ROAM OUT  |
// ********************************************

// initialize a memory block for a new policy
a_policy=init_policy();

a_policy->goal="SMOOTHNESS";
a_policy->goal_value=MODERATE;
```

```
a_policy->policy_class="DISCOVERY";

a_condition->signal_strength=MODERATE;
a_condition->receiving_interface=WLAN;

an_action->collection_time=DEFAULT;
an_action->discover_alternatives=1;
an_action->decision_type="DISCOVERY";

list_add(policies, a_policy, -1);


// *******************************************
// 10   D I S C O V E R Y    policy |   ROAM OUT  |
// *******************************************

// initialize a memory block for a new policy
a_policy=init_policy();

a_policy->goal="SMOOTHNESS";
a_policy->goal_value=MODERATE;
a_policy->policy_class="DISCOVERY";

a_condition->signal_strength=LOW;
a_condition->receiving_interface=WLAN;

an_action->collection_time=DEFAULT;
an_action->discover_alternatives=1;
an_action->decision_type="DISCOVERY";

list_add(policies, a_policy, -1);


// *******************************************
// 11   D I S C O V E R Y    policy |   ROAM OUT  |
// *******************************************

// initialize a memory block for a new policy
a_policy=init_policy();

a_policy->goal="SMOOTHNESS";
a_policy->goal_value=LOW;
a_policy->policy_class="DISCOVERY";

a_condition->signal_strength=MODERATE;
a_condition->receiving_interface=WLAN;

an_action->collection_time=LONG;
an_action->discover_alternatives=1;
an_action->decision_type="DISCOVERY";

list_add(policies, a_policy, -1);


// *******************************************
// 12   D I S C O V E R Y    policy |   ROAM OUT  |
// *******************************************

// initialize a memory block for a new policy
a_policy=init_policy();

a_policy->goal="SMOOTHNESS";
a_policy->goal_value=LOW;
a_policy->policy_class="DISCOVERY";

a_condition->signal_strength=LOW;
a_condition->receiving_interface=WLAN;

an_action->collection_time=LONG;
an_action->discover_alternatives=1;
an_action->decision_type="DISCOVERY";

list_add(policies, a_policy, -1);


// *******************************************
```

```
// 13   D I S C O V E R Y    policy |   ROAM OUT  |
// *******************************************

// initialize a memory block for a new policy
a_policy=init_policy();

a_policy->goal="SMOOTHNESS";
a_policy->goal_value=HIGH;
a_policy->policy_class="DISCOVERY";

a_condition->signal_strength=MODERATE;
a_condition->receiving_interface=ETH;

an_action->collection_time=SHORT;
an_action->discover_alternatives=1;
an_action->decision_type="DISCOVERY";

list_add(policies, a_policy, -1);


// *******************************************
// 14   D I S C O V E R Y    policy |   ROAM OUT  |
// *******************************************

// initialize a memory block for a new policy
a_policy=init_policy();

a_policy->goal="SMOOTHNESS";
a_policy->goal_value=HIGH;
a_policy->policy_class="DISCOVERY";

a_condition->signal_strength=LOW;
a_condition->receiving_interface=ETH;

an_action->collection_time=SHORT;
an_action->discover_alternatives=1;
an_action->decision_type="DISCOVERY";

list_add(policies, a_policy, -1);


// *******************************************
// 15   D I S C O V E R Y    policy |   ROAM OUT  |
// *******************************************

// initialize a memory block for a new policy
a_policy=init_policy();

a_policy->goal="SMOOTHNESS";
a_policy->goal_value=MODERATE;
a_policy->policy_class="DISCOVERY";

a_condition->signal_strength=MODERATE;
a_condition->receiving_interface=ETH;

an_action->collection_time=DEFAULT;
an_action->discover_alternatives=1;
an_action->decision_type="DISCOVERY";

list_add(policies, a_policy, -1);


// *******************************************
// 16   D I S C O V E R Y    policy |   ROAM OUT  |
// *******************************************

// initialize a memory block for a new policy
a_policy=init_policy();

a_policy->goal="SMOOTHNESS";
a_policy->goal_value=MODERATE;
a_policy->policy_class="DISCOVERY";

a_condition->signal_strength=LOW;
a_condition->receiving_interface=ETH;
```

```
an_action->collection_time=DEFAULT;
an_action->discover_alternatives=1;
an_action->decision_type="DISCOVERY";

list_add(policies, a_policy, -1);


// ********************************************
// 17   D I S C O V E R Y    policy |   ROAM OUT  |
// ********************************************

// initialize a memory block for a new policy
a_policy=init_policy();

a_policy->goal="SMOOTHNESS";
a_policy->goal_value=LOW;
a_policy->policy_class="DISCOVERY";

a_condition->signal_strength=MODERATE;
a_condition->receiving_interface=ETH;

an_action->collection_time=LONG;
an_action->discover_alternatives=1;
an_action->decision_type="DISCOVERY";

list_add(policies, a_policy, -1);


// ********************************************
// 18   D I S C O V E R Y    policy |   ROAM OUT  |
// ********************************************

// initialize a memory block for a new policy
a_policy=init_policy();

a_policy->goal="SMOOTHNESS";
a_policy->goal_value=LOW;
a_policy->policy_class="DISCOVERY";

a_condition->signal_strength=LOW;
a_condition->receiving_interface=ETH;

an_action->collection_time=LONG;
an_action->discover_alternatives=1;
an_action->decision_type="DISCOVERY";

list_add(policies, a_policy, -1);



// POLICIES ON CONFIGURATION LIST AND BETTER ALTERNATIVE

// ************************************************
// 19    H A N D O F F    policy |   CONFIGURATIONS  |
// ************************************************

// initialize a memory block for a new policy
a_policy=init_policy();

a_policy->goal="SMOOTHNESS";
a_policy->goal_value=HIGH;
a_policy->policy_class="HANDOFF";

a_condition->configuration_list=1;
a_condition->better_alternative=1;

an_action->connect_order="CONNECT_FIRST";
an_action->decision_type="HANDOFF";

list_add(policies, a_policy, -1);


// ************************************************
// 20    H A N D O F F    policy |   CONFIGURATIONS  |
// ************************************************
```

```
        // initialize a memory block for a new policy
        a_policy=init_policy();

        a_policy->goal="SMOOTHNESS";
        a_policy->goal_value=MODERATE;
        a_policy->policy_class="HANDOFF";

        a_condition->configuration_list=1;
        a_condition->better_alternative=1;

        an_action->connect_order="CONNECT_SECOND";
        an_action->decision_type="HANDOFF";

        list_add(policies, a_policy, -1);


        // ************************************************
        // 21    H A N D O F F    policy |   CONFIGURATIONS  |
        // ************************************************

        // initialize a memory block for a new policy
        a_policy=init_policy();

        a_policy->goal="SMOOTHNESS";
        a_policy->goal_value=LOW;
        a_policy->policy_class="HANDOFF";

        a_condition->configuration_list=1;
        a_condition->better_alternative=1;

        an_action->connect_order="CONNECT_SECOND";
        an_action->decision_type="HANDOFF";

        list_add(policies, a_policy, -1);
}
```

Appendix D XML Schema for policy repository

policy_repository.xsd to specify the structure of the discovery and handoff policies

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.1 U (http://www.xmlspy.com) by Malohat (Telematica Instituut)
-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
   <xs:element name="policies">
      <xs:annotation>
         <xs:documentation/>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="goal">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="smoothness" maxOccurs="unbounded">
                        <xs:complexType mixed="true">
                           <xs:sequence>
                              <xs:element name="Policy">
                                 <xs:complexType mixed="true">
                                    <xs:complexContent mixed="true">
                                       <xs:extension base="Discovery_Policy"/>
                                    </xs:complexContent>
                                 </xs:complexType>
                              </xs:element>
                              <xs:element name="Policy">
                                 <xs:complexType mixed="true">
                                    <xs:complexContent mixed="true">
                                       <xs:extension base="Handoff_Policy"/>
                                    </xs:complexContent>
                                 </xs:complexType>
                              </xs:element>
                           </xs:sequence>
                        </xs:complexType>
                     </xs:element>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:complexType name="Handoff_Policy" mixed="true">
      <xs:sequence>
         <xs:element name="condition">
            <xs:complexType mixed="true">
               <xs:sequence>
                  <xs:element name="configuration_list">
                     <xs:annotation>
                        <xs:documentation>
                           boolean indicating receipt of a configuration list
                        </xs:documentation>
                     </xs:annotation>
                     <xs:complexType mixed="true"/>
                  </xs:element>
                  <xs:element name="better_alternative">
                     <xs:annotation>
                        <xs:documentation>
                           boolean better alternative to the current configuration
                        </xs:documentation>
                     </xs:annotation>
                     <xs:complexType mixed="true"/>
                  </xs:element>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="action">
            <xs:complexType mixed="true">
               <xs:sequence>
                  <xs:element name="connect_order">
```

```
                    <xs:annotation>
                        <xs:documentation>handoff execution sequence</xs:documentation>
                    </xs:annotation>
                    <xs:complexType mixed="true"/>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="Discovery_Policy" mixed="true">
  <xs:sequence>
    <xs:element name="condition">
      <xs:complexType mixed="true">
        <xs:sequence>
          <xs:element name="receiving_interface">
            <xs:annotation>
              <xs:documentation>802.11 or UMTS</xs:documentation>
            </xs:annotation>
            <xs:complexType mixed="true"/>
          </xs:element>
          <xs:element name="packet_loss">
            <xs:annotation>
              <xs:documentation>
                packet_loss value (%) on the receiving interface
              </xs:documentation>
            </xs:annotation>
            <xs:complexType mixed="true">
              <xs:sequence>
                <xs:element name="operator">
                  <xs:complexType mixed="true"/>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="action">
      <xs:complexType mixed="true">
        <xs:sequence>
          <xs:element name="collection_time">
            <xs:annotation>
              <xs:documentation>
                time for waiting for responses from aggregators
              </xs:documentation>
            </xs:annotation>
            <xs:complexType mixed="true"/>
          </xs:element>
          <xs:element name="discover_alternatives">
            <xs:annotation>
              <xs:documentation>
                invocation of discovery protocol by PEP
              </xs:documentation>
            </xs:annotation>
            <xs:complexType mixed="true"/>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

Appendix E Publication

During this research we have published a poster paper at the IST 13-Mobile and Wireless Summit, that was held in Lyon, France, 26-30 June 2004.

A Policy-based System for Handoffs between Intermediary Content Providers in the Wireless Internet

Malohat Ibrohimovna Kamilova[1,2], Cristian Hesselman[1,2], Ing Widya[2], and Erik Huizer[2]

[1]Telematica Instituut, P.O. Box 589, 7500 AN, Enschede, The Netherlands

E-mail: malohat.kamilova@telin.nl, cristian.hesselman@telin.nl

[2] University of Twente, P.O. Box 217, 7500 AE, Enschede, The Netherlands

E-mail: i.widya@utwente.nl, e.huizer@utwente.nl

ABSTRACT

We consider the distribution of real-time multimedia content (e.g., radio or TV broadcasts) through multiple aggregators. An aggregator is an intermediary content provider that operates a pool of proxy servers to aggregate content from sources and forward it to mobile hosts. Aggregators package content into channels (e.g., CNN or ABC) and offer them in various versions (e.g., using different encodings) that differ in quality or price. Mobile hosts receive channels via the wireless Internet, which consists of multiple types of wireless networks (e.g. 802.11 and UMTS). At specific locations, mobile hosts can connect to multiple networks simultaneously (e.g., in a hotspot) and can thus potentially receive different alternative versions of a channel from different aggregators through different interfaces. In this paper, we propose a control system that enables mobile hosts to automatically deal with these (changing) alternatives in a manner transparent to the mobile user. The system's novelty lies in the use of application-level policies. They for instance define when to look for a 'better' version of a channel (e.g., if packet loss increases to a certain threshold) and what constitutes 'best' based on the user's preferences. The policies thus define when and how to adapt the reception of a channel to changes in available resources or user's preferences.

I. INTRODUCTION

In the near future, the fringes of the Internet will consist of different types of wired and wireless networks that are operated by different administrative authorities [1]. As a result, mobile hosts will generally be able to receive service from multiple networks of different operators, for instance when they roam into a hotspot [2, 3].

At the application-level, the same real-time multimedia content (e.g., radio or TV broadcasts) can be streamed through multiple proxy servers, with mobile hosts handing off from one server to another as a result of mobility (e.g., because different proxy servers serve different networks) [4-7]. This idea can be extended to the distribution of channels through multiple *aggregators* [8, 2, 3]. An aggregator is an intermediary content provider that operates a pool of proxy servers to aggregate content from sources and forward it to mobile hosts [2, 3]. Aggregators package content into channels (e.g., CNN or ABC) and offer them in various versions (e.g., using different encodings) that differ in quality or price. As a result, mobile hosts can potentially receive different versions of a channel from different aggregators, possibly through different network interfaces (e.g., in a hotspot).

A research challenge is to develop a control system, which enables mobile hosts to automatically deal with such a (changing) set of alternatives in a manner invisible to the user [9]. We are designing such a control system based on policies (i.e. "if-condition-then-action" rules). Policies are rules that can be used by a controlling entity to constrain the behavior of a controlled entity in a way that the behavior of the controlled entity becomes aligned to the goal of the policy [10]. Policies are commonly used in network management, for instance to configure an RSVP router [11]. The advantage of policies is that they can be maintained in a central repository and then rolled out, which enables policy-controlled entities (e.g., routers) to be reconfigured with new policies (i.e., behavior) in a flexible manner.

The novelty of our control system is that it uses well-defined application-level policies. This means that the actions of the policy are enforced at the application-level. An application-level policy could for instance read: if the number of lost packets of a channel increases to a certain threshold (the condition), then invoke an application-level protocol (the action) to look for another aggregator that can offer the channel, possibly on another interface. Other application-level policies define when to handoff to another aggregator, and what constitutes the 'best' version of a channel based on the predefined user preferences. Using these policies, the system can adapt the reception of a channel to the capabilities of the Internet environment in the vicinity of the mobile host (e.g., in terms of available bandwidth), to the available resources of the mobile host (e.g., available battery power), and so on.

Known policy-based systems for Internet service control typically use network level policies rather than application-level policies and focus on determining which network (operator) provides the best service [12-15]. Wang et al. [12] however do not use well-defined policies like we do (i.e. rules with goals). Murray et al. [15] discuss the selection of a best network for a mobile host according to the current load on the networks. The selection in their system is controlled by policy decision logic that sits in the infrastructure, while ours only sits on mobile hosts. Clark et al. [13] and Lee et al. [14] take a different approach to determine the best service, which uses algorithms rather than policies.

The rest of this paper is organized into four sections. In Section II, we describe the environment for which our policy-based control system is designed. In Section III, we present the system's architecture. Thereafter, we discuss some of the policies that our system uses in Section IV. Finally, Section V summarizes the state of our research and explains our future work.

II. ENVIRONMENT

We consider an environment that consists of application-level service providers that deliver real-time multimedia content (e.g., radio or TV broadcasts) to mobile hosts in the form of channels (e.g., CNN Radio or BBC Television). We distinguish two types of providers: content sources and content aggregators [8, 2-3]. A *content source* is the origin of one or more channels and transmits them in a mobile agnostic manner (e.g., unaware of the changing IP addresses of mobile hosts). A *content aggregator*, on the other hand, is specifically designed to serve mobile hosts. It receives channels from sources and forwards them to mobile hosts in a mobile and wireless aware manner (e.g., it forwards channels in a way suitable for the limited capabilities of mobile hosts). The proxy-like distribution scheme via aggregators increases scalability in the absence of IP multicast [16], which is important when channels need to be distributed to a potentially large number of receivers. Sources and aggregators primarily process and forward application-level data units, typically in the form of RTP packets [17].

Figure 1 shows an example in which source cnn.com[3] distributes audio channel CNN Radio via aggregators stream-it.com and multimedia-forward.nl. User Bob receives CNN Radio either from multimedia-forward.nl through the UMTS network of network operator connect-you.nl, or from stream-it.com through the 802.11 network of hotspot.nl. The solid line between stream-it.com and hotspot.nl indicates that stream-it.com is only available through the 802.11 network. Similarly, media-forward.nl is only available through the UMTS network.
An aggregator can deliver its channels in different *versions* (cf., [4, 18]). This enables it to deal with different user requirements (e.g., pertaining to cost or quality) and to serve different types of hosts that connect to the Internet through different types of wireless links. We refer to the description of a channel version as a *configuration* (e.g., using SDP [19]). Each aggregator supports its own set of configurations of a channel. For example, stream-it.com could support various high-quality configurations of CNN Radio (e.g., in 'studio' quality), while media-forward.nl could only support medium-quality configurations of the same channel (e.g., in 'FM radio' quality). Mobile hosts can thus receive the same channel from different aggregators at different configurations, possibly through different interfaces (e.g., at point A in Figure 1).

[3] The domain names in this paper are for illustrative purposes only.

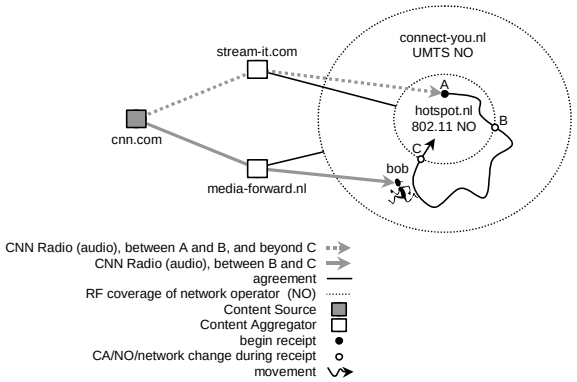

Figure 1. Streaming via multiple alternative aggregators

An application-level protocol [2, 3] enables mobile hosts to request which versions of a channel are available from the aggregators it can reach. A mobile host invokes the protocol when it is looking for a 'better' configuration of the channel it is receiving, for example when it moves into a subnet (an aggregator with a better configuration may appear) or moves out of one (aggregators may disappear, which may result in a new best aggregator). The assignment of a (new) IP address to one of the host's network interfaces (e.g., to Bob's 802.11 interface at point C), and the loss of packets or a decreasing signal strength (e.g., of the 802.11 network at point B) could signal these two events, respectively.

Figure 2 shows Bob's mobile host querying media-forward.nl and stream-it.com at point C of Figure 1 to check which configurations of CNN Radio they support.

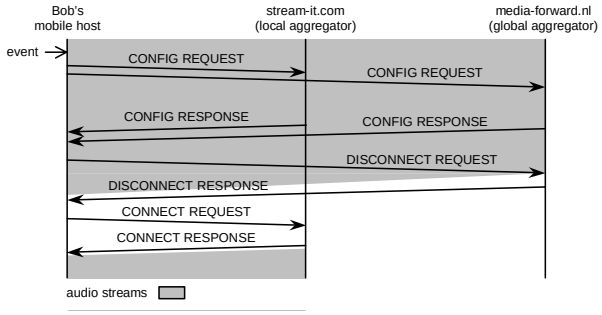

Figure 2. Typical protocol interactions for discovery and handoff.

Bob's host sends a configurations request to stream-it.com via its 802.11 interface (hotspot.nl), and a request to media-forward.nl through its UMTS interface (connect-you). Analyzing the responses of the aggregators, the host decides that stream-it.com provides a better version of CNN Radio than media-forward.nl. It therefore hands off to stream-it.com by sending a disconnect request to media-forward.nl and a connect request to stream-it.com (or the other way around). As a result, Bob's mobile host now receives the 'better' version of CNN Radio from stream-it.com via hotspot.nl's 802.11 network. The protocol's behavior is similar at points A and B, except that stream-it.com becomes unavailable around point B. We have implemented the protocol of Figure 2 using SDP [19] and SIP [20].

As we will see in Section III, the selection of the best aggregator and the trigger for querying aggregators is policy-driven. Examples of other occasions at which the mobile host could consult aggregators are when the host's battery power drops, when the available bandwidth on one of the host's network interfaces drops, when the user changes his preferences and so forth.

III. ARCHITECTURE

We use policies (i.e., "if-condition-then-action" rules) to flexibly define the behavior of mobile hosts roaming in the environment of Section II. We adopt the policy framework of the IETF [11, 21], which uses the concepts of a *Policy Decision Point* (PDP) and a *Policy Enforcement Point* (PEP).

A. Components

Figure 3 shows the high-level architecture of our control system. It consists of a PDP, a PEP, a policy repository, a resource manager, and a set of user preferences. In our current design, the PEP, the PDP and the resource manager are located on the mobile host.

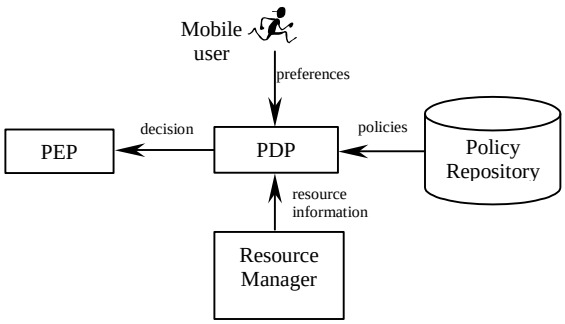

Figure 3. Architecture of the policy-based system.

A *PDP* represents a controlling entity that applies policies to control the behavior of a controlled entity (the PEP). In our control system, the PDP persistently monitors the state of the available resources and of the user's preferences and uses this information to evaluate the policies' conditions. If the circumstances are such that the "if" condition of a policy becomes true, then the PDP decides to enforce the actions defined in the "then" part of the policy. For example, if the if condition of a policy says "packet loss >= 20%", and the action reads "invoke protocol" (to discover new configurations, see Figure 2), then the PDP will enforce the discovery action if the number of lost packets of a channel exceeds 20% in a certain time interval. Our PDP is also responsible for selecting the best configuration according to the user preferences and the current available resources.

A *PEP* represents the controlled entity upon which policy decisions are being enforced (by the PDP), yielding a constrained behavior of the PEP. A PEP therefore receives directives from a PDP. In our system, the application-level protocol of Figure 2 embodies the PEP because it executes policy decisions such as "invoke protocol" or "handoff smoothly" (also see Section IV).

A *Policy Repository* contains (inactive) policies written in a policy specification language such as IRML [22]. A policy repository allows policies to be flexibly downloaded into a PDP, possibly at run-time. Another advantage is that policies become platform independent. In our system, the repository for instance contains discovery policies (they define when to invoke the protocol of Figure 2) and handoff policies (they determine how to execute a handoff). We will elaborate on these and additional policy classes in Section IV.
In our system, each policy has a goal (e.g., "high viewing smoothness"), which is part of the specification of a policy. To retrieve the appropriate policies, the PDP matches the preferences (i.e., goals) of the user with the goals of the policies in the repository. The PDP and the PEP together realize the goal of a policy the PDP retrieves.
We expect that the policy repository will typically reside in the fixed Internet, thus enabling a user to consistently apply the same policies to all of his devices.

A PDP can generally use external information sources to come to its decisions [11]. The external information source in our architecture is the *Resource Manager*. It is responsible for monitoring available resources, such as availability of networks, available bandwidth, signal strengths of networks, packet loss of a channel, and available aggregators and configurations. The PDP accesses this information by requesting it or by listening to events from the Resource Manager (e.g., appearance of an IP address of an interface).

B. Behavior

Figure 3 also shows the interaction between the components of the policy-based system. PDP receives user preferences from the user (arrow labeled "preferences"). On analysis of the new user preferences PDP may decide to retrieve new or additional policies from the Policy Repository (arrow "policies"), that match with the new goals of the user. PDP may consult the Resource Manager (arrow "resource information") for the available resources. Having all necessary information PDP for example makes a selection of the best

configuration, which is inline with the preferences of the user in price and quality level. Finally, PDP sends it's decision to PEP (arrow "decision").

Figure 4 shows the system's behavior when the user moves towards and comes close to the point B, the figure 1. The Resource Manager informs the PDP by sending an *event* that, for example a packet loss is continuously increasing.

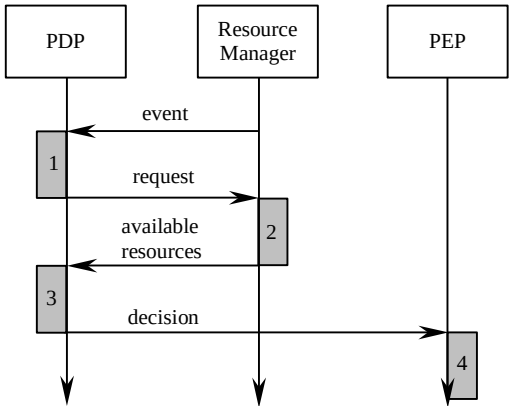

Figure 4. Example of system's behavior by receiving an event from Resource Manager

By receipt of the event the PDP evaluates the condition of the discovery policy (1), if the condition is true, PDP *requests* new information from the Resource Manager on *available resources* at that moment (2) and makes a new selection of the best configuration (3). Once the selection is made, the *decision* is sent to PEP, which executes a handoff (4) connecting the mobile host to the selected aggregator (in this case media-forward.nl, see figure 1) using the selected configuration.

IV. EXAMPLES OF POLICIES

The entire system goes through three phases (see Figure 2): a *discovery phase* to send out *config* requests and collect the responses, a *selection phase* to determine which aggregator provides the 'best' configuration, and a *handoff phase* to handoff to a 'better' aggregator (if any).

We distinguish policies for each of the above phases. *Discovery policies* define when to invoke the protocol of Figure 2; *selection policies* define which aggregator provides the 'best' configuration of a channel based on the user's preferences; and *handoff policies* determine how to execute a handoff.

In this section, we discuss a few examples of discovery and handoff policies. We use "viewing smoothness" as a goal. In our system, when the user chooses high viewing smoothness as his preference, the system provides seamless roaming by means of early handoffs. If the user chooses moderate/low smoothness, then system allows some data loss and glitches during the handoffs.

To explain the effects of the policies, we consider the situation in which user Bob (see Figure 1) is at point B while receiving CNN Radio from stream-it.com through its 802.11 interface. We assume that the user has expressed high smoothness of viewing video. According to this, the policies with the corresponding goal have been downloaded into (i.e. activated on) the PDP. These policies could for instance look like this:

```
/* policy_type=discovery, exiting hotspot
 * policy_goal=high_viewing_smoothness
 */
if (packet_loss >= 20% &&
    receiving_interface == "802.11") {
  /* Invoke discovery */
  run_protocol(); }

/* policy_type=handoff
 * policy_goal=high_viewing_smoothness
 */
if (handoff_flag &&
```

```
  receiving_interface == “802.11”)) {
/* First connect, then disconnect */
connect_to(new_aggregator);
disconnect_from(old_aggregator); }
```

The discovery policy uses the degradation of the streams that the mobile host receives as an indication that the mobile host is moving out of the hotspot [23]. It provides high viewing smoothness because it causes the PDP to react proactively on packet loss: if the PDP detects that it has lost 20% of the packets it received on the host’s 802.11 interface during a certain period, then it will decide to enforce the discovery policy by ordering the PEP to run the protocol (cf. Figure 2). If the user would have selected low viewing smoothness, then the PDP would have downloaded another discovery policy, for instance one that behaves in a more reactive manner (e.g., using a packet loss threshold of 80%). The discovery policy could also have used the monotonic decrease of signal strength instead of increasing packet loss.

The handoff policy realizes high viewing smoothness by first connecting to a new aggregator on the overlay network (e.g., media-forward.nl on the UMTS network), and then disconnecting from the old aggregator (stream-it.com on the 802.11 network). A handoff policy that provides low smoothness could for instance do this the other way around.

The policy examples also show that policies with common goals can be combined to a more complex one (thus also be decomposed in more elementary ones). Policies may furthermore depend on another, in the sense that they are not commutative during processing. Independent policies may be processed in any order without influencing the result. These research issues are, however, beyond the focus of this paper [10].

V. SUMMARY AND FUTURE WORK

We have presented the design of a control system, which uses policies to automatically deal with different networks, aggregators, and channel configurations. The system takes the preferences of the user into account, thus allowing for automatic adaptation without user involvement.
We are currently implementing a prototype of the system in which the PDP, the PEP, the policy repository, and the user preferences are co-located on the mobile host. Next step is to design and implement the policy-based system for a distributed scenario, where the PDP and the policy repository are located on remote machines and the PEP is located on the mobile host. The motivation to put the PDP remotely is to reduce the complexity at the mobile host, since some mobile devices are very small and have limited capabilities. Furthermore, we plan to describe policies in a policy specification language (e.g., in XML [24, 25]) and to test our policy-based system in stationary and roaming scenarios.

REFERENCES

[1] M. Haardt and W. Mohr, “The Complete Solution for Third-Generation Wireless Communications: Two Modes on Air, One Winning Strategy”, IEEE Personal Communications, December 2000.

[2] C. Hesselman, H. Eertink, I. Widya, E. Huizer, “A Mobility-aware Broadcasting Infrastructure for a Wireless Internet with Hotspots”, Proceedings of the First ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (WMASH’03), San Diego, USA, Sept. 2003

[3] C. Hesselman, H. Eertink, I. Widya, and E. Huizer, ”Delivering Live Multimedia Streams to Mobile Hosts in a Wireless Internet with Multiple Content Aggregators”, to appear in Mobile Networks and Applications Journal (MONET), special issue on Wireless Mobile Applications and Services on WLAN Hotspots, Summer 2005.

[4] D. Xu, K. Nahrstedt, “Supporting Multimedia Service Polymorphism in Dynamic and Heterogeneous Environments”, Technical Report UIUCDCS-R-2000-2159, University of Illinois at Urbana-Champaign, USA, October 2000.

[5] H-Y. Hsieh, K-H. Kim, Y. Zhu, R. Sivakumar, “A Receiver-Centric Transport Protocol for Mobile Hosts with Heterogeneous Wireless Interfaces”, Proc. MobiCom 2003, San Diego, USA, September 2003.

[6] A. Dutta, H. Schulzrinne, S. Das, A. McAuley, W. Chen, and O. Altintas, “MarconiNet supporting Streaming Media over Localized Wireless Multicast”, M-Commerce 2002 Workshop, Atlanta, USA, 2002.

[7] S. Roy, B. Shen, V. Sundaram and R. Kumar, “Application Level Hand-off Support for Mobile Media Transcoding Sessions”, NOSSDAV’02, Miami Beach, Florida, May 2002.

[8] C. Hesselman, I. Widya, H. Eertink, and E. Huizer, “A Comprehensive Framework for Broadcasting Multimedia Content in the Future Mobile Internet”, Proceedings of the 2nd IEEE Workshop on Applications and Services in Wireless Networks (ASWN’02), Paris, France, July 2002.
[9] L. Kleinrock, “An Internet Vision: the Invisible Global Instrastructure”, AdHoc Networks Journal, Vol. 1, No. 1, July 2003, pp. 3-11.
[10] M. Cox and R. Davison, “Concepts, Activities and Issues of Policy-based Communications Management”, BT Technology Journal, Volume 17, Issue 3, July 1999, pp. 155-169.
[11] R.Yavatkar, D. Pendarakis and R. Guerin, “A Framework for Policy-based Admission Control”, RFC 2753, January 2000.
[12] H.Wang, R. Katz, J.Giese, “Policy-Enabled Handoffs Across Heterogeneous Wireless Networks”, 2nd IEEE Workshop on Mobile Computing and Applications (WMCSA 1999), New Orleans, USA, February 1999.
[13] D.D. Clark, J. Wroslawski, “The Personal Router whitepaper”, MIT Technical Report, March 2001.
[14] G. Lee, P. Faratin, S. Bauer, J. Wroslawski, “Automatic Service Selection in Dynamic Wireless Network Environments”, a poster presentation at MobiCom (co-located with First ACM International Workshop WMASH’03), San Diego, USA, 2003.
[15] K. Murray, R. Mathur, D. Pesch, “Intelligent Access and Mobility Management in Heterogeneous Wireless Networks using Policy”, Adaptive Wireless Systems Group, Department of Electronic Engineering, Cork Institute of Technology, Ireland.
[16] J. Chennikara, W. Chen, A. Dutta, O. Altintas, “Application-Layer Multicast for Mobile Users in Diverse Networks”, IEEE Globecom 2002, Taipei, Taiwan, November 2002.
[17] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications”, RFC 1889, January 1996.
[18] T. Plagemann, V. Goebel, L. Mathy, N. Race, and M. Zink, “Towards Scalable and Affordable Content Distribution Services”, Proc. 7th International Conference on Telecommunications (ConTEL 2003), Zagreb, Croatia, June 2003.
[19] M. Handley, V. Jacobson, “SDP: Session Description Protocol”, RFC 2327, April 1998.
[20] H. Schulzrinne, “Dynamic Host Configuration Protocol (DHCP-for-IPv4) Option for Session Initiation Protocol (SIP) Servers”, RFC 3361, 2002.
[21] A. Westerinen et al, “Terminology for Policy-Based Management”, RFC 3198. November 2001.
[22] W. Ng et al, “Quality of Service Extension to IRML”, Panasonic Singapore Labs, Internet-draft, July 2001.
[23] C. Hesselman, H. Eertink, and A. Peddemors, “Multimedia QoS Adaptation for Inter-tech Roaming”, Proceedings of the 6th IEEE Symposium on Computers and Communications (ISCC’01), Hammamet, Tunisia, July 2001.
[24] I. Liabotis, O. Prnjat, L. Sacks, “Policy-based Resource Management for Application Level Active Networks”, University College London, England, UK
[25] N. Damianou, A. Bandara, M. Sloman, E. Lupu, “A Survey of Policy Specification Approaches”, Department of Computing, Imperial College of Science Technology and Medicine, London, 2002.