Context-aware Handover Using Active Network Technology

Qing Wei, Károly Farkas*, Christian Prehofer, Paulo Mendes

Bernhard Plattner*

DoCoMo Communications Laboratories Europe
{wei, prehofer, mendes}@docomolab-euro.com
Landsberger Str. 308-312, 80687 Munich, Germany
*Computer Engineering and Networks Laboratory, ETH Zürich
*{farkas, plattner}@tik.ee.ethz.ch
*Gloriastr. 35 8092 Zürich, Switzerland

Abstract. Context-aware computing can play a major role to improve the services of mobile networking systems. In this paper, we focus on optimizing handover decisions in heterogeneous environments, where the user has a choice among different mobile networks and access points. In our approach, the decision is not only based on the signal quality, but also on the knowledge about the context of mobile devices and networks. Since context information and context processing evolves fast, we propose a flexible, integrated approach for context management, which can adapt in several ways. Our architecture encompasses programmable platforms and distributed context management components in network nodes and mobile devices, as well as a service deployment scheme for network services. This flexible architecture is able to actively deploy different handover services. It manages dynamic context information and allows mobile devices to be always connected to the most suitable access network. Our architecture is validated in a prototype implementation and performance results are discussed.

Keywords: Context-aware handover, programmable platform, service deployment

1 Introduction

Mobile networks are more and more widespread in our daily life, so offering better support for wireless services becomes an important issue. Context-awareness is essential for the optimization of services in this heterogeneous environment in order to fulfil various user needs. In this paper, we focus on optimizing handover decisions in heterogeneous networks, where the user has a choice among different mobile networks with different capabilities.

The concept of context-aware handover can be defined as follows: a handover procedure that selects a target *Access Point* (AP), based not only on the signal quality or explicit advertisements sent by the access point, but also on the knowledge of the context information of the *Mobile Node* (MN) and the networks, in order to take intelligent and better decisions. The level of *Quality of Service* (QoS) delivered to the mobile device depends on different kinds of context information such as the bandwidth available in the access point, modulation scheme, speed and direction of the MN, etc. Meanwhile, different applications require different QoS level. It is the combination of all these facts that makes context-aware handover necessary.

Moreover, in future networks, scanning for access points of different radio technologies can be expensive (concerning computation power and battery) or even impossible for devices which only support one mode at a time. In this case, context-aware assistance of the network to avoid unnecessary search for access points can be very valuable.

However, implementing context-aware handover poses some challenges. First, the context information used to decide an optimal decision point will be much more diverse in future mobile networks:

- There will be more diverse radio access networks, e.g. wireless LAN, 2nd and 3rd generations of cellular networks, their variations and other upcoming technologies like ad-hoc networks.

- There will be many more options of network services in regard to quality of service, security, charging, roaming etc.
- Applications and user preferences evolve fast and will require optimal support from network services.
- Advanced location information and group mobility information (e.g., users travelling in a car) will be available to support the handover.

Second, the specific context information, which is relevant to handover is distributed on several nodes of the network (e.g., location server, user profile database, access points) and the mobile node, including the user's applications. However, they must be collected and provided to the right nodes for handover decision at the right time. For this, the context information should be provided proactively to the mobile node, i.e., before the handover takes place and when the mobile node has good radio connectivity.

Third, the type of context information may change over time, and new services are emerging continuously. This requires algorithms that collect and process the context information to evolve accordingly. And finally, fragile and/or low-bandwidth wireless links constrain the exchange of context information.

This paper will examine how the specific context information, which is relevant to handover, can be collected, how to cope with the complexity and profusion of information available to the mobile node and how to cope with the changing nature of context information.

We propose a framework to manage diverse, dynamic and distributed context information for supporting context-aware services. This framework is built based on flexible software technology, i.e., active networking [1], in order to cope with the changing nature of context information. It consists of two parts: a programmable platform installed on network and mobile nodes, which allows the flexible installation and use of software modules; and a service deployment framework capable of deploying different modules, each of which implements different context-aware handover services.

The main contribution of this paper is a novel integrated architecture for context-aware handover and its evaluation. It uses two main concepts to enhance flexibility and performance. First, customized modules for different situations are used for efficient context exchange and context-aware handover decisions. Second, a service deployment framework is used to support the flexible update of the customized modules on the active nodes when needed. Furthermore, we show practical evaluation results by using a prototype that includes all the components of our architecture. The evaluation of the proposed architecture is based on a prototype scenario that considers as context information the mobile node's location and the traffic load of access networks as well as the signal strength. Both the functionality and efficiency of context-aware handover are evaluated in the prototype. We show that our architecture can help the mobile node to make an optimal handover decision in different situations. However, both the signalling overhead and handover latency raised by dynamic service deployment and context processing are not significant for handover services.

The remainder of the paper is organized as follows. Section 2 provides an analysis of the requirements of flexible and active architecture for context-services. In Section 3, we describe the proposed context-aware handover architecture based on active networking technology. The functionality of the architecture is illustrated by a prototype for a selected scenario in Section 4 and evaluated in Section 5. In Section 6, we take a look at prior work related to context-services and service deployment, and Section 7 concludes the paper.

2 Requirements of Context-aware Handover Service

Our work aims to allow mobile nodes to execute handover decisions in an optimal way depending on the context. This requires a mechanism to efficiently collect and manage context information and an appropriate platform to use that information for an optimal handover decision.

Context information may be classified as static or dynamic, depending on the frequency and cause of changes. It can be classified also based on where such information is maintained. Table 1 explains our classification of context information. It is clearly just a snapshot, and mainly focuses on layer 3 and above. Novel kinds of context information may also appear. For instance, in future networks, new kinds of context information like the groups a user belongs to

may be relevant for handover decisions. Some pieces of information, such as the user's profile, appear twice, as the information is often spread over the user's device, the operator's network and possibly over several service providers. As an example, the user's profile may include subscribed services and service preferences, e.g., which services have to be downgraded or dropped if available resources are not sufficient. The potential next AP, nearby APs, user history, and user mobility can be used for location prediction and limit the options for selecting the next AP. It will help simplifying the selection of the best AP. The user's settings for applications and the types of the ongoing applications indicate the preferable QoS level, etc.

	Context Information on Mobile Devices	**Context Information in The Network**
Static	User's settings and profile Settings of applications	User's profile and history Network location (location of APs) Capacities and services of the network Charging models Network policies
Static within a Cell	Reachable APs	Potential next AP, nearby APs
Dynamic	Type of ongoing application Requirements of applications Status of devices (battery, interface status, etc.)	Location information and location prediction Status and load of the network

Table 1 Classification of Context Information

However, context information is not available immediately to the entities involved in a handover for several reasons. First, context information is distributed. For instance, some context information may be available in the user's home network, some may be available in the network to be visited and some resides on the terminal. Second, dynamic context information may change frequently or lose accuracy over time. For instance, the MN is tempting to convey information about the load of the current AP, yet its relevance decreases quickly over time. Third, the type of relevant context information and the methods to interpret it may evolve over time. Hence, algorithms for interpreting context data need to be adapted to the new requirements.

Therefore, we need a context management framework, which assures that the context information needed for handover decisions is available in time (before the handover decision needs to be made). Moreover, exchange of information between the network and the MNs should be minimized to save wireless resources. Furthermore, a context-aware handover requires an appropriate execution platform, which is flexible enough to adapt to the changing requirements of this service. It should be able to cope with dynamic context and automatically alter the handover decision policy or the algorithm in use. It should support continuous exchange of context information between the nodes involved in the handover service in an efficient way. It should also enable the mobility manager of the mobile node to take the right handover decision.

3 An Integrated Approach for Context Management

Our approach integrates a context management framework, a programmable platform and a service deployment scheme to provide the functionality needed for context-aware handover. The context management framework is in charge of collecting the relevant context information for different services and managing the context information. The programmable platform is used to download and install the suitable modules for context exchange and processing. The service deployment scheme is used to synchronize and manage the work of the involved active nodes.

In this section, first we show the overall architecture, then introduce the context management framework for this architecture as well as the properties of active programmable nodes and service deployment mechanisms. And finally, we explain the different phases of the context-aware handover service as it is realized in our design.

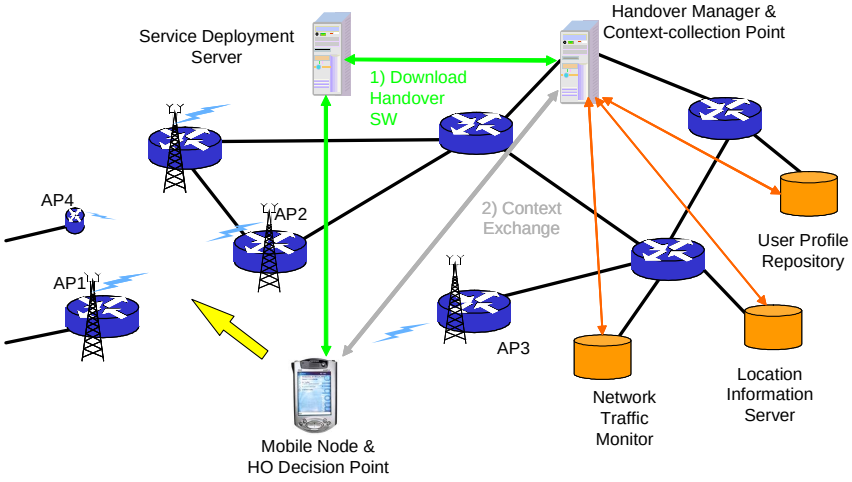

Fig. 1 Architecture for Context-aware Handover using Active Networking Technology

In our architecture, as shown in Fig. 1, context information is stored in context information repositories, such as the *Location Information Server* (LIS), *Network Traffic Monitor* (NTM) and the user's profile repository. The LIS is responsible to track the position of each mobile device in the provider's network and has the knowledge of nearby APs, while the user's profile repository stores user's profiles as seen by the network service providers. The NTM is used to monitor the available bandwidth of different APs. These are just examples of context information repositories, other types of these repositories can exist, as well. Moreover, we introduce a *Handover Manager* (HM), which controls handovers carried out in some part of an access network. The HM is responsible for filtering and processing handover-related context information. At the same time, the HM acts as a Context-collection Point which collects the needed context information from various context repositories. Finally, a *Service Deployment Server* (SDS) is used to manage and install the service modules needed on the network nodes and mobile nodes.

3.1 Context Management Framework

Our context management framework is in charge of context collection, compilation, exchange and evaluation. A detailed discussion on this framework can be found in [10]. It defines the following main entities:

- *Context-collection Point* on the network side, which collects and compiles the relevant context information from different sources. In our scenario, the context collection point is placed on the Handover Manager as shown in Fig. 1;
- *Handover Decision Point* that decides which AP is to be selected for the handover. In our design, the Handover Decision Point is the MN, as shown in Fig. 1. Of course, it is also possible to place it in the network. We use the MN because:

    1. It is more scalable to have each MN maintain one Handover Decision Point by itself.
    2. In the current mobile systems, the signal strength based handover decision mechanism is implemented in the MN. It is beneficial to be compatible with the existing handover mechanism.
    3. We can easily change the existing handover decision procedure to our context-aware decision mechanism in the MN by extending it.

The Handover Decision Point uses specific algorithms (e.g., rule-based logic) for interpreting the data delivered by the context management framework. It is the task of the service deployment to assure that both, the Handover Decision Point, and the Context-collection Point are proactively supplied with the appropriate algorithms.

It is difficult to have a generic algorithm for making handover decisions based on any network topology, configuration, architecture, or type of mobile device. Even if we can build such an algorithm, it cannot be optimized for all the cases and might be quite complex. Therefore, we

use flexible, programmable platform to install different algorithms for different cases without interrupting the proper working of the node. These algorithms need to be changed if there are structural changes in the context information, e.g., when the context format, user's profile or context processing algorithm change. Notice that software updates can be applied in different ways. For instance, update the software each time when the user's profile changes (e.g., when entering a car, new software modules that consider the changed context regarding higher speed and different applications are installed). The context information will be exchanged when needed using the context exchange protocol. This way, up-to-date context information is used for handover decision. The detailed steps for context-aware handover are the following:

1. Prepare the Handover Decision Point and the Context-collection Point proactively with the suitable context processing and exchange protocol. Besides this, the Handover Decision Point is updated proactively with the suitable handover decision algorithm.
2. The Handover Decision Point requests network context relevant to the Handover Decision Point from the Context-collection Point with some filtering parameters (e.g., ID, types of needed context information, etc.). The collection of related context information can be realized by some simple procedure such as a server-client based communication. However, such procedure is not elaborated here because of space limitation. The required software may be taken from a library of modules for specific situations.
3. The Context-collection Point gets the network context relevant to the Handover Decision Point and compiles it as far as possible (i.e., integrates some static context).
4. The Context-collection Point sends the compiled information to the Handover Decision Point. The dynamic context from the network can be sent separately and be updated using the context exchange protocol when needed.
5. The handover decision algorithm is parameterized with the information from the Context-collection Point and with the static information from the mobile node.
6. The handover decision algorithm is invoked at handover time and completed with the dynamic context of the terminal (e.g., reachable access points, application requests and sessions) before the decision is made.

From these steps, we can see that some static context information is compiled before being sent out. In this way, the context information to be exchanged is minimized. Furthermore, this procedure ensures that the most up-to-date context information is used for handover decision.

3.2 Active Node Platform and Service Deployment

In this subsection, we describe the node platform for our architecture. We use active networking technology [1] to meet the requirements as outlined in Section 2. Since the algorithms executed for determining the best AP for handover are context-dependent, the network elements and the mobile nodes involved in this process need to be programmable. Active networking technology is a good candidate to fulfil this requirement. Our active node consists of the basic processing hardware, a node operating system and several execution environments, in which active applications will execute handover algorithms. The node has to support the dynamic installation of the handover modules at run-time, without interrupting the proper working of the node. The architecture and implementation that we chose for the active nodes in our prototype system is PromethOS [8]. PromethOS is a generic platform for running active applications in a Linux environment, allowing for on-demand installation of user space or kernel space modules.

While PromethOS provides a basic active node platform, in our architecture design we also need a framework capable of handling the selection, installation, configuration, and management of the service components. The Chameleon service deployment framework [7] provides node level service deployment functions suitable for our prototype. We complemented it with a simple, centralized network level service deployment scheme, which we called Octopus, to provide functions to identify service participants, explore network resources and synchronize among Chameleon instances. In the rest of this subsection, we give a brief overview about our use of Chameleon and the new Octopus scheme.

3.2.1 Chameleon, a Node Level Service Deployment Framework

Node level service deployment comprises selecting, downloading, installing and configuring implementations of service components on an active node, such that these components jointly provide the specified service. Chameleon uses an XML service specification – to be generated by the network-level service deployment scheme – and a description of the intrinsic properties of the active node to determine which implementations of service components need to be installed on the active node.

The service specification contains general information about the service; details about connections (ports) to enable modular service decomposition; the additional sub-services required together with the connections among them; information about the service code together with the resource demands for code execution; as well as other service specific things. Fig. 2 depicts the structure of the XML document type definition used for service specifications in Chameleon. In this figure, XML elements are graphically represented as ellipses and their attributes are encircled by a hexagon. Elements surrounded by double ellipses may appear more than once. Dashed arrows point to optional XML elements, while solid arrows mark elements as mandatory. Note that here we disregard the detailed discussion of all the elements due to the lack of space and its reduced relevance to understanding the main idea.

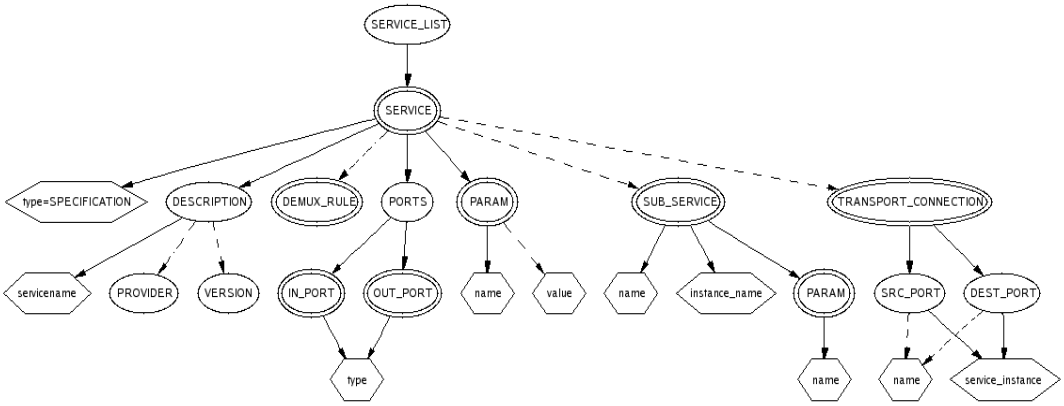

Fig. 2 XML Service Specification Graph

Chameleon resolves the service specification (Fig. 3 shows an XML code snippet of our context-aware handover service) against the description of the node properties, thereby creating a tree-like structure representing all possible implementations of a service. Then it selects an implementation to install, configure and run the service based on predefined criteria. In our application of Chameleon, such service specifications are generated by Octopus, our network-level service deployment scheme.

```
<?xml version="1.0" encoding="UTF-8"?>
<SERVICE_LIST xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Chameleon\XML\chameleon.xsd">
<SERVICE xsi:type="IMPLEMENTATION">
    <DESCRIPTION xsi:servicename="HDM" xsi:option="normal">
    <PROVIDER>DoCoMo</PROVIDER>
          <VERSION>0.1</VERSION>
    </DESCRIPTION>
    <ENVIRONMENT>
          <OS>
                <NAME>Linux</NAME>
                <VERSION>2.4.20</VERSION>
          </OS>
          <EE>
                <NAME>PROMETHOS</NAME>
                <VERSION>1</VERSION>
          </EE>
```

```
    </ENVIRONMENT>
    <PORTS>
          <IN_PORT xsi:type="push_in"/>
          <OUT_PORT xsi:type="push_out"/>
    </PORTS>
    <CODE_LOCATION>handover.o</CODE_LOCATION>
</SERVICE>
</SERVICE_LIST>
```

Fig. 3 XML Code Snippet of the Context-aware Handover Service

3.2.2 Octopus, a Network Level Service Deployment Scheme

The core of Octopus is a central management entity, called *Service Deployment Server* (SDS), as illustrated in Fig 4. The SDS contains a *Service Deployment Manager* (SDM) module, which controls the network-wide signalling and all related synchronization functions needed during service deployment. Moreover, it contains a *Service Server* (SS), which stores the descriptors of the services available in the network and a *Code Server* (CS), which stores the implementations (code modules) of the service components. These servers are managed by the SDM, and they can be located anywhere in the network.

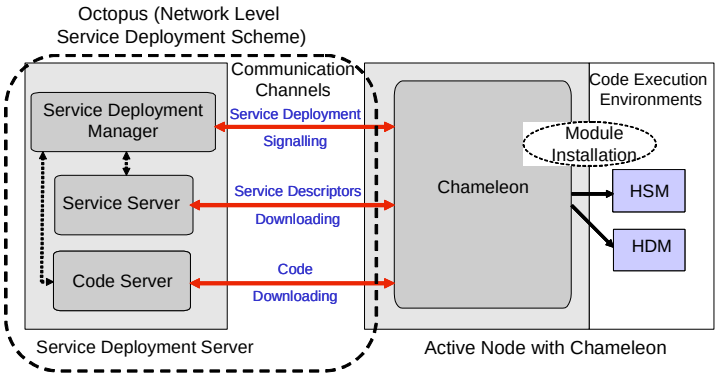

Fig. 4 Communications Between the Service Deployment Server and a Chameleon Instance

Fig. 4 shows how Octopus and Chameleon communicate. Three signalling channels are used: (1) Service deployment signalling for requesting and/or sending the proper service specification to the Chameleon instance on the active node; (2) Service descriptor download signalling for retrieving refined service descriptions, to which the service specification (1) refers. The download is initiated by Chameleon after the service specification is resolved; and (3) Code download signalling, which is used by Chameleon to retrieve the implementation modules from the CS needed for service execution. This retrieval, again, is initiated by Chameleon.
The specific operation of Octopus varies depending on where Chameleon is executed:

- If the node is a network node such as the HM in our context-aware handover service, then the SDM module of the SDS establishes a connection to the Chameleon instance running on the node (which is just waiting for incoming connections in this case) on the signalling channel and requests the installation of the given service. Subsequently, the Chameleon instance on the HM downloads and autonomously resolves the service specification, then downloads and installs the component implementations.
- If the node is a user's node as the MN in our context-aware handover service, then this node's Chameleon initiates the service deployment by requesting a list of available services and selecting a service from this list. Then the MN's Chameleon downloads the requested service specification and proceeds as in the previous case.

Fundamentally, there can be two kinds of service deployment: provider-initiated or user-initiated. In the former case, the service is deployed in the network in advance, before the arrival

of any service user. On the contrary, in the latter case the deployment is on demand, the arrival of the first service user initiates the installation of the service even in the network. Chameleon extended with Octopus can cope with both of the policies and the active nodes can act either as network node or end user node by setting the right parameters of Chameleon.

3.3 Phases of Context-aware Handover Service

The realization of our context-aware handover service can be divided into five phases, which are explained in the message sequence diagrams shown in Fig. 5 and Fig. 6. Phases A-C are referring to the service deployment mechanism. During Phase D relevant context information is collected. In Phase E, the context information is evaluated and handover decisions are made.

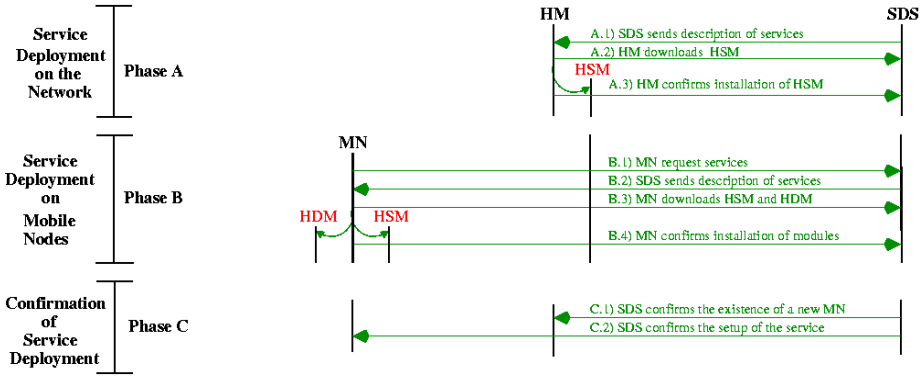

Fig. 5 Sequence of Signalling for Service Deployment

The service deployment phases include: fetching the right service components; installing them on the appropriate network node; and confirming the successful installation of all the components for that service. For example, to use a context-aware handover service, we need to install a context exchange protocol, the *Handover Support Module* (HSM), and a handover decision mechanism, the *Handover Decision Module* (HDM), on the related nodes. The HDM is installed on the MN, and the HSM is installed on both the MN and the HM. More specifically, here we need to install appropriate versions of both the HSM and HDM on the programmable platforms of the MN and HM. This is realized by the service deployment mechanism in Phase A, B and C (see Fig. 5).

We assume that the SDS will trigger the module installation in the HM using the concept of service broadcast, since the location and role of the HM are “fixed” comparing to the mobile node. On the other hand, the modules in the MN are very dependent on the terminal and the user. The terminal can move frequently from one network to another, or change its service. Therefore, it is better to have the MN initiating the module download with its requested service. Whenever the conditions change in the MN, new requests will be sent to the SDS for new services.

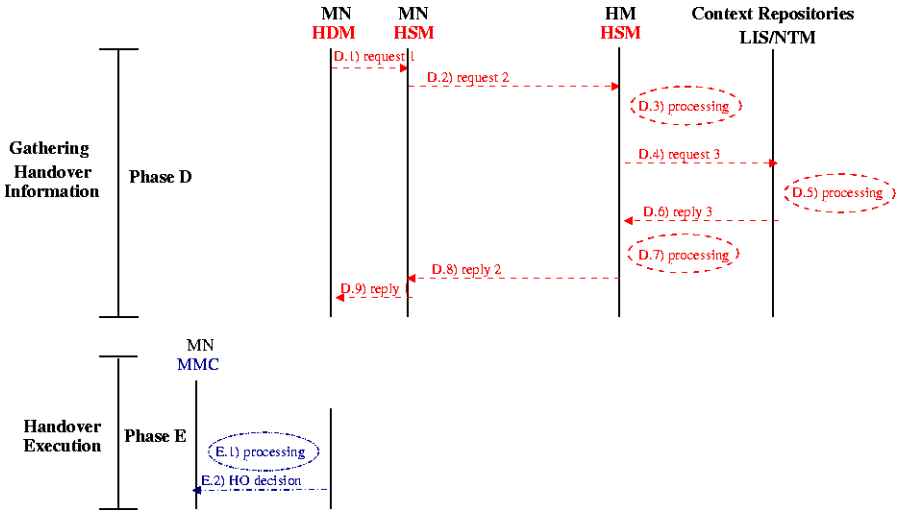

Fig. 6 Sequence of Signalling and Processing for Context-aware Handover

The signalling used by the HDM to request the needed context information is detailed in Phase D. Context information needed for handover decision is requested using server-client based mechanism. The HDM is the client of the HM, since the HM is the client of the Context repositories. After the HDM makes the decision on the target AP based on the collected context information, in Phase E the decision is sent to the *Mobility Management Component* (MMC) of the mobile node to execute the handover (see Fig. 6).

The details of the message sequence diagrams are described in Section 5.

4 Application Scenario and the Prototype Implementation

In this section, we introduce our application scenario and the prototype implementation that reproduces the environment illustrated in Fig. 7. This scenario demonstrates a situation in which context-aware handover, based on active networking technology, improves the efficiency of a mobile system by always trying to choose the best AP. This figure shows a provider's network with three *Universal Mobile Telecommunications System* (UMTS) and one *Wireless LAN* (WLAN) access networks. The user is assumed to walk or travel along a trajectory leading through an area where all three cells intersect. Without context-awareness capability, the handover of the user in network UMTS1, using Mobile IP version 6 (MIPv6) [16] would be driven by route advertisements and is therefore difficult to control: the mobile node could register with any of the three networks. However, if active networking is used, the user's MN can request the utilization of a handover service based on context information.

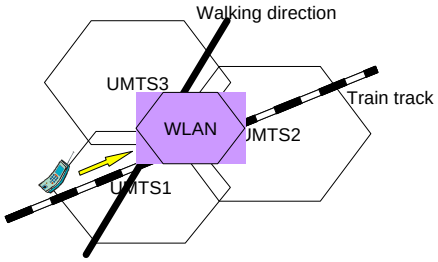

Fig. 7 Context-aware Handover in a Real Situation

To illustrate the behaviour of the proposed architecture, two types of context information are considered: the user's location, speed and trajectory, which depend on how the user moves (walk/travel by train); and the *Quality of Service* (QoS) required by the application. Considering

the user's context information, the handover may be made to a network that has a better coverage along a train track or to a network with small range, but high QoS if the user is walking. Another criterion could be the network traffic load, which normally would have an effect on the QoS perceived by the user.

The prototype implementation of context-aware handover service is depicted in Fig. 8. In our prototype, one HM controls the three access networks. The prototype encompasses ten elements: one MN, one *Home Agent* (HA), two *Foreign APs* (FAP1 and FAP2), one HM that also emulates the functionality of one LIS and one NTM, one machine providing the SDS and *Correspondent Node*[1] (CN), and one *Traffic Generator*[2] (TG). All these elements, except the MN, have an Ethernet 100 Mb/s interface. The HA, FAP1 and FAP2 use a WLAN 802.11b interface with 11 Mb/s to communicate with the MN. The SDS and HM are equipped with Linux kernel version 2.4.20 and they use 2.4 GHz Pentium 4 processors with 256 MB DDR-SDRAM. The MN uses Linux kernel version 2.4.20 as well, but it is equipped with a 1.2 GHz Celeron processor and 128 MB SDRAM. The service deployment part is implemented in Java using Java Virtual Machine and Software Development Kit version 1.4.1, and the context management framework and the decision mechanism are implemented in C.

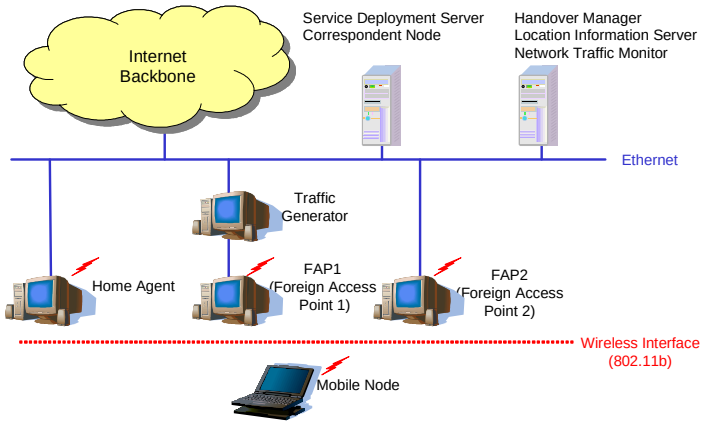

Fig. 8 Prototype for Context-aware Handover

To reduce the complexity of the prototype, the behaviours of the LIS and the NTM are emulated with static information located on the HM, and only one wireless interface type is used. In our experiments, we emulate the use of UMTS to reproduce the handover situation shown in Fig. 7. This is done by setting different ranges to different WLAN networks, i.e., a hypothetic UMTS network is set up as a WLAN network with a wider range. We also emulated overloaded networks, in order to incorporate the QoS requirements, by generating more traffic to specific networks, using the MGEN tool-set [17] installed on the TG.

Fig. 9 shows the execution platform of the nodes used in our prototype implementation, which gives the execution environments of active modules. These active modules are in our case the HSM located on the mobile node and the handover manager, and the HDM installed on the mobile node, respectively. As described in section 3.2, the PromethOS platform running on Linux was used in our prototype.

[1] The CN sends video streams in some of our experimental scenarios, see later.
[2] The TG generates traffic to introduce different load levels of the network.

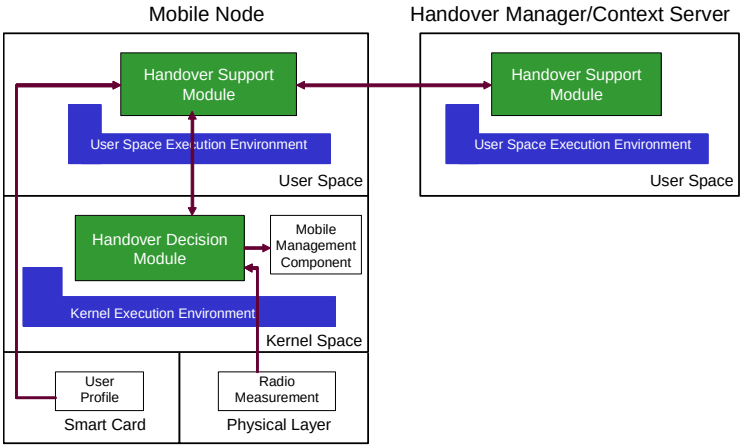

Fig. 9 Execution Platform and Active Modules for Context-aware Handover

In our prototype implementation, mobility management is carried out by mobile IP [9]. The traditional implementation of mobile IP is in charge of handover decision based on the received signal strength. We implemented the HDM as an extension to the traditional mobile IP, and replaced the old handover decision function with our advanced one. Since the traditional mobile IP module is implemented in the kernel space in the Linux environment, we placed the HDM in the kernel space as well to make the communication between the original mobile IP module and our module easier and more efficient. On the other hand, we placed the HSM in the user space to make its implementation easier.

5 Evaluation

In this section, we evaluate the proposed architecture. The goal of this evaluation is twofold: first, to exemplify the behaviour of the proposed architecture, as a proof of concept, by showing the feasibility of doing handovers with the support of context information. The second goal is to evaluate the impact that the proposed architecture has on the network, namely, the signalling overhead, and to understand how much time is required to deploy context-aware services and to collect customized handover data. The dynamic deployment of different type of handover services will introduce additional overhead (i.e., code transmission, signalling for service deployment, etc.) and latency (i.e., code retrieving, code installation, signal exchange, etc.). We need to prove that it does not significantly affect the usage of context-aware handover services. Moreover, the context exchange and processing bring additional traffic and handover latency in the network. These side-effects should be also minimized.

5.1 Functionality of the Implementation

Four test scenarios are used in the prototype to show the benefits of context-aware handover as depicted in Table 2. Scenario 1 is based on standard Mobile IP version 6 (MIPv6) [16] handover and it is used as a benchmark to evaluate the efficiency of our context-aware handover scenarios. The remaining three scenarios aim at showing the advantages of using context information and active networking to choose the right AP, while the hypothetical user travels on a train. Scenario 2 uses a service based on the user's location-context (LoA handover). Scenario 3 also considers the user's location-context, but in this case the user is receiving a streamed video sent by the CN. In the fourth scenario, the user is also receiving a streamed video, but in addition to the user's location-context we also consider the QoS requirements of this application (LoA+QoS handover). In each scenario, a context-specific HSM and HDM, able to process information about the user's location-context and/or QoS-context, are downloaded, installed and used by the HM and MN, respectively.

	Scenario 1	Scenario 2	Scenario 3	Scenario 4
Types of Handover	Standard mobile IP	Location aware (LoA) handover	Location aware (LoA) handover	Location + QoS aware (LoA+QoS) handover
Demo Service			Video stream	Video stream

Table 2 The Four Test Scenarios Used in the Prototype

In the experiments, the MN starts from the home network, which is placed out of range of FAP1 and FAP2. This allows the execution of handover to be triggered by a low signal to noise value, when the MN moves away from the HA. To map the context-aware service scenarios to the real situation illustrated in Fig. 7 the home network in our prototype corresponds to UMTS1, FAP1 to UMTS3, and FAP2 to UMTS2. For scenario three and four, FAP1 is first unloaded and then is heavily loaded. In any scenario, the home network and FAP2 are always unloaded.

Concerning the behaviour of the proposed architecture, scenarios 1 and 2 show that MNs have higher control over their handovers with context information, by avoiding the random handover behaviour of standard MIPv6[3]. In scenario 1, the MN registers with the FAP from which it gets the first route advertisement. Therefore, the experiment result is not repeatable under the same experiment environment. The MN sometimes registers to FAP1 and sometimes registers to FAP2. However, in scenario 2, the MN always registers with FAP1, independently of the order of router advertisements received from any FAP. This corresponds to the assumption that the MN is on a train, which means that the MN is moving fast, and FAP1 (UMTS3) has a better coverage along the train track than FAP2 (UMTS2).

The comparison between scenario 1 and 2 proves that handovers are more efficient when context information is also considered. The results of the experiments done with scenario 3 and 4 demonstrate the importance of using the right active modules in the presence of different context information. In these scenarios, the MN first decides to handover[4] to FAP1 according to the context. After FAP1 becoming heavily loaded, MN stays with FAP1 in scenario 3, while it handovers to FAP2 in scenario 4. In scenario 3, the MN registers with FAP1 at second 70, since this is the network with wider coverage near the train track. This ends up in the deterioration of the perceived video quality, since FAP1 is loaded afterwards and the used context-aware service modules do not consider the load of networks, as shown in Fig. 10. In scenario 3, the video remains with low quality, since the MN is only aware of the user's location, which does not change. In scenario 4, the MN also registers with FAP1 as in scenario 3, because FAP1 and FAP2 are both unloaded in the beginning. The handover decision only depends on the location context. After FAP1 is loaded at second 90, the MN with location-context plus QoS-context modules can react properly to significant changes in the load of networks. So we can see the resume of video throughput after a short decrease, because MN handovers to FAP2 after it receives the updated network load[5].

[3] By random, we mean that with MIPv6 an MN handovers to the network that corresponds to the first route advertisement received after the MN detects that the current AP is no longer bi-directionally reachable.

[4] Handover happens because the signal strength in the home network decreases below a pre-defined threshold.

[5] The duration of decreased video throughput depends on the frequency the HM is configured to check the load of the networks.

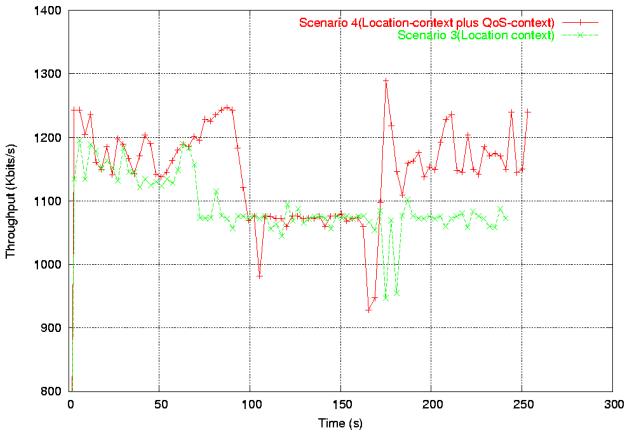

Fig. 10 Throughput of the Video Stream on Scenarios 3 and 4

5.2 Evaluation of Efficiency

We also analyze the efficiency of the proposed architecture to evaluate the impact that it has on the network. The evaluation focuses on analyzing the trade-offs between flexibility and additional processing. The processing here consists of two parts: pre-processing (service deployment – Phase A, B, C) and runtime processing (context collection and usage – Phase D, E). The evaluation consists of the communication overhead (bandwidth consumption) and latency (time introduced during service deployment). All the time measurement values are averaged over more than 50 repeated runs.

	Module Code Size (bytes)			**Signalling Overhead (bytes)**	
	HM	MN		HM	MN
	HSM	HSM	HDM		
LoA Handover	18650	19297	11652	799	697
LoA+QoS Handover	17924	20286	14459		

Table 3 Service Deployment Overhead – Size Measurement Results

Comparing the size of data exchanged during the signalling to deploy the service and the size of customized modules downloaded to run the service, it can be observed, as Table 3 also shows, that most of the bandwidth consumption comes from the module downloads. In our prototype, the downloaded code size is around 18 Kbytes for HM and 32 Kbytes for MN, respectively. These values are similar for both LoA and LoA+QoS handovers. The module code sizes seem big for the small functionality, but the modules integrate additional functions. Specifically, they are executable files implemented in C, which include the library to retrieve signal strength from the wireless LAN card, the Linux device driver library to exchange the information between kernel and user space, as well as some libraries needed by standard MIPv6[6]. Using other execution environments and programming languages, e.g. Java, may result in smaller code size. Meanwhile, data exchange for service deployment in the network can be omitted. This is because:

1. Service module installation in the network is rare, e.g., only once during the initiation of the network and once per year when the service is updated.

[6] Our code is implemented as an extension to the standardized MIPv6.

2. The network can use wired connections to deploy the services, which is very cheap compared to using wireless links.

In regard to service deployment on the mobile node, the amount of data exchange and frequency of service deployment should be minimized. However, we assume the frequency of service deployment on the MN is much less (e.g., every half a year for service update from the service provider, once every two months when the user registers for a new service) than the frequency of handover (e.g, once per hour).

The time to deploy a new service can be very different based on the service, the handling of the service descriptors and service modules, and the role of the node. To deploy a service the node's Chameleon instance has to complete the following tasks: (1) connect to the SDM to select a service; (2) connect to the SS to download the service descriptor; (3) connect to the CS to download the required code modules to the selected service (using local repositories or caching can accelerate the latter two procedures). The installation of the downloaded code modules requires some time depending on the used execution environments, as well. Furthermore, the role of the node can also influence the required installation time, for instance, a mobile node with poor network connection requiring manual intervention to select the service can substantially increase the deployment time. In our evaluation, we measured: (1) the time to fetch the module codes from the code server for both LoA handover and LoA+QoS handover services; (2) the time to install the modules on the MN and HM; (3) the time of the additional signalling steps during the deployment of these services.

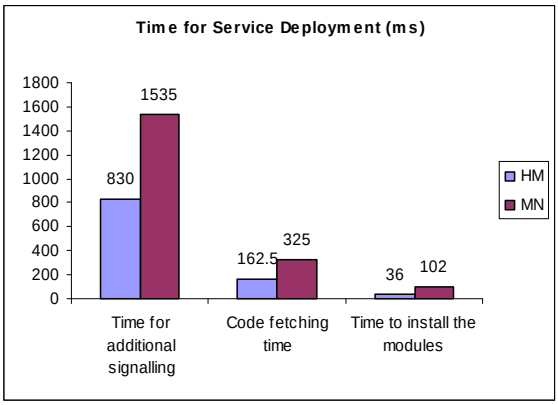

Fig. 11 Average Service Deployment Overhead of Both LoA Handover and LoA+QoS Handover Services on the HM and the MN (ms)

The measurements of the time to deploy LoA handover and LoA + QoS handover gave similar results which are depicted on Fig. 11. We can see that the biggest part of the service deployment overhead is the additional signalling time, and this value is almost doubled in case of MN. The difference can be derived from the divergent behaviours of the wired and wireless connections and the various amount of signalling exchange required to download different number of modules. The code fetching time for both MN and HM is around 15 % of the whole overhead. This value is closely related to the size of the code and the available bandwidth. Fig. 11 also indicates that the module installation time is a very small portion of the service deployment overhead. On the other hand, this installation time is smaller in case of HM than in case of MN. The reason is that the HM has to install only one module, the HSM, in the user space of Linux, while the MN has to additionally install one more module, the HDM, into the Linux kernel space. Our measurements show that the time to install the module in the kernel space is around 50 % longer than installing the module in the user space. The total time for service deployment is around 1 s for HM and 2 s for MN. This relatively long latency can be explained by the complex signalling exchange between the SDS and Chameleon, as shown in Fig. 4, and the selected programming language to implement the service deployment framework, which was Java in our prototype. However, this service deployment delay is not so critical, because services can either be deployed proactively or the older version of the service

can be kept on using before the new one is deployed. The realization of such a “soft switching” from one service module to another one will be one topic of our further investigations.

	Time (µsec)		
	Time to Collect Context from the Network	**Time to Send the Network Update**	**Processing Time for Handover Decision**
Standard Mobile IPv6			46
LoA Handover	36816		74
LoA+QoS Handover	39693	1358	78

Table 4 Processing Overhead

In the phase of context collection and context usage (Phase D and E as indicated in Fig. 6), the following steps are involved:

1. MN inquires the context info from the HM/Context-collection Point by supplying the HM with the user context, such as “Video, in Train” and “Audio, not in Train”. The inquiry comes from the HDM on the MN (D.1) and is passed to the HSM on the HM by the HSM on the MN (D.2).
2. The HM/Context-collection Point processes the incoming query and decides which type of information needs to be collected (D.3). Afterwards, the context collection requests are sent out to the context repositories (LIS and NTM in our prototype) (D.4). In case of LoA handover, only the IP address and location preference of different APs need to be collected. While in case of LoA+QoS handover, the available bandwidth on different APs needs to be collected, as well.
3. The LIS or the NTM prepares the needed context information in step D.5, and answers back to the HM/Context-collection Point in step D.6.
4. The HM/Context-collection Point compiles the received context information into a context table (D.7) and sends it back to the HSM running on the MN (D.8).
5. The HDM on the MN receives the context table in step D.9 and makes an optimal handover decision accordingly in step E.1.
6. The optimal handover decision is sent to the Mobility Management Component in step E.2.
7. The MN handovers to the selected AP, if it is different than the current AP.

Especially, in case of LoA+QoS handover, the updated context table is sent to the MN when changes of available bandwidth on different APs are detected.

Concerning these two phases, we measured the time to fetch context from the network (Phase D) as well as the time to use, process the context and make handover decision accordingly (Phase E). The time to fetch the context refers to the round-trip time from the MN inquires the context from the network by sending the context of the MN (e.g., “Audio, in Train”, “Video, not in Train”), until the moment when the MN gets the context table from the network. Our measurements show that it takes around 37 to 40 ms from the moment the HDM on the MN requests handover information until the moment it gets it. This time includes round-trip time between the MN and HM and the time to collect relevant context information from various context sources. However, the context exchange over the wireless link only needs about 1.3 ms in one way in our tests. Therefore, the process of relevant context collection from the network is the bottleneck of latency caused by context-awareness in our evaluation. There are some trade-offs between the efficiency of context collection and context transport. If we want to minimize the context exchange over the wireless link, we have to collect only the relevant context and do some pre-processing. This type of pre-processing is time consuming. On the other hand, pre-processed context will simplify the handover process. Vice-versa, less context pre-processing induces higher efforts for context exchange and handover process. However, these measurements also depend on different context-aware handover algorithms. Considering that we pro-actively collect context information and that the time required retrieving it is very small, the probability that handovers need to be done without context information, i.e., using only the MIPv6 functionality, is very low. Meanwhile, our context exchange protocol described in Section 3.1 ensures that the context information to be exchanged is minimized. Sometimes only the context updates need to be retrieved from the network. In case, when there is no context

change, context exchange can be even skipped. Moreover, the pro-active characteristics of the proposed architecture ensures that the time required to perform a MIPv6 handover is not increased.

The processing time for handover decision is 46 µs for standard MIPv6, 74 µs for LoA and 78 µs for LoA+QoS, as shown in Table 4. The increase of the handover processing time is mainly caused by the context exchange from the user space to kernel space. The reason is that, in our prototype, context information is collected in the user space of the MN, and used in the kernel space. However, this overhead can be leveraged by better module structure in the future.

We also conducted an analysis about the communication overhead involved in collecting context information. Here we discuss the worst case overhead for one AP when network load update occurs frequently. The length of the packet sent by the MN to request context information is 64 bytes: 60 bytes of TCP/IPv6 headers and four bytes to indicate the current user's context. As a reply, the HM sends the MN information about each access network. With LoA modules, such information occupies 20 bytes per access network: 16 bytes for the IPv6 address of the AP and four bytes indicating the priority of the network. With LoA+QoS modules, 4 bytes corresponding to the load of the network are added to the information sent to the MN. While in case of network load update, the location preference does not need to be sent again, this decreases 4 bytes from the updating context. Considering a scenario with 10 nearby APs, where the user is requesting LoA+QoS modules and the *Maximum Transfer Unit* (MTU) is 1500 bytes, then the context information is of 0.29 KB. Considering a worst-case situation where the load of any AP changes every 1 s, and the HM is configured to check the load of the networks also every 1 s, the control information overhead is of 2.4 Kb/s for one MN. Suppose 50 MNs[7] are using LoA+QoS handover service simultaneously in the area of one AP, and all of them get the network update in the same time. The bandwidth overhead for context exchange is around 1.1 % of a WLAN link (11 Mb/s) and 6 % of an UMTS link (2 Mb/s). As we can see from above, most of the overhead is not introduced by context information itself but by the TCP/IP headers. There can be possibilities to save the communication overhead, e.g., by header compression. Furthermore, security mechanisms can introduce additional overhead on the bandwidth. For instance, the message from the network should be authenticated before being used by the MN. In the same time, the integration check should be performed in both sides for the received information, as well. To simplify the implementation, we did not consider security issues in our prototype.

From the evaluation above, we can see that the deployment of context-aware handover can be done proactively with little overhead. The update of the modules is easy and fast. Context-aware handover only introduces very minor bandwidth overhead and latency. We have however seen that the communication needed with the service deployment infrastructure is considerable and must be considered for system design.

6 Related Work

In the following, we survey prior work in related areas. First, we consider other work on handover optimization and context management. Then we discuss other approaches regarding service deployment.

In [12][13], sophisticated handover procedures have been considered. However, the parameters, which have been used for the handover decision, are confined to the type of the radio access technology plus the signal strength. In [14], different handover policies for heterogeneous networks are used, considering as handover parameters mainly the air-interface type and the available bandwidth at the access router. However, the management of dynamic and diverse context information and programmable decision mechanisms are not discussed. There are several approaches aiming at a more programmable handover in heterogeneous access network (e.g., [15]). Yet none of them implements a fully programmable and context aware handover system. In the same time, many studies have analysed context-aware applications. Examples of such studies are the Xerox Parc's project that distributes information about the user's location to different applications based on the user's profile [18], [19]; the HP's cooltown project, which adds Web context to the environment by allowing mobile users to

[7] This is the maximum number of users which one AP can support in WLAN, UMTS, GSM, supposing all these users are mostly idle and using the minimum data rate.

receive *Universal Resource Locators* (URLs) sent by ubiquitous beacons [20]; and the Microsoft's Easyliving focusing on a smart space that is aware of the user's presence and adjusts settings to suit its needs [21]. However, none of these studies analyses how context-awareness can improve the performance of network services.

In the area of active service deployment several proposals have been made, but often these are restricted to the needs of a specific platform (e.g., the Active Networks Daemon [2] of ABone [3]). Other proposals have the potential to be used as a generic service deployment scheme such as ASCP [4], ASDP [5], pattern based service deployment [6], or Chameleon [7]. In our work, we selected Chameleon as the node level service deployment tool because it fits best with our design. Chameleon uses XML for describing services. The concept of capturing the blueprint of a network service in a formal description stems from the work on the Genesis Kernel at Columbia University [22], and XML was proposed first as a scripting language for describing network service composition information in [23]. Besides using XML descriptions Chameleon provides an automatic way to parse, check the XML documents and build a dependency tree for the service modules. However, Chameleon doesn't deal with the network level deployment functions. Thus, we complemented Chameleon with a simple, centralized network level service deployment scheme called Octopus. Octopus provides functions to identify service participants, explore and allocate network resources and synchronize among Chameleon instances.

7 Conclusions

This paper describes an architecture that aims at optimizing mobile network services based on context information. We have shown that intelligent handover decisions are important in future, heterogeneous mobile networks with different capabilities. The proposed architecture is able to gather information from different network elements and to use this information on mobile nodes for local context-awareness and better handover decisions. This architecture allows for a flexible use of different protocols to exchange different types of context information, as well as a flexible use of different context-aware decision algorithms on mobile nodes. Our solution ensures that the correct context information is available at the right place at the right time, and handles diverse, dynamic and distributed context information.

Our solution integrates three main parts: first, we use a programmable platform installed on network nodes and mobile nodes. Second, we combine this platform with a framework capable of performing network-wide and node-local deployment of various context-aware services. Third, we use an efficient and flexible context management to handle diverse, dynamic and distributed context information.

The programmable platform and the service deployment framework are not specific to the context-aware handover service rather they are more generic and can be used in case of other network services. The deployment framework is scalable in two dimensions, concerning the number of services and the size of network (number of involved nodes in a service).

The evaluation of the proposed architecture, based on a prototype, has shown the feasibility and utility of this approach. Specifically, we showed that context-awareness increased the efficiency of handovers. In our experiments, we gathered valuable experience to deploy services over wireless links. The evaluation results showed that the service deployment was fast in the network and relatively slow over the wireless link depending on the link status. But this phase is needed only when the context processing or exchange formats change and then the modules have to be updated. Furthermore, services can be deployed proactively in the networks. Hence, the deployment is less time critical with respect to handover. Compared to service deployment, context exchange and processing occur much more often. However, only the information of nearby access points needs to be exchanged in order to support context-aware handover. This introduces only a minor bandwidth overhead. The procedures to exchange, process context information and make an optimal decision are very fast. The bottleneck exists in the context collection. Therefore, in order to realize efficient context-aware handover, it is very important to collect the related context information proactively.

References

[1] Konstantinos Psounis, *Active Networks: Applications, Security, Safety, and Architectures*, IEEE Communication surveys, http://www.comsoc.org/livepubs/surveys/index.html

[2] *Anetd: Active Networks Daemon.* ACTIVE project, ISI & SRI, http://www.sdl.sri.com/projects/activate/anetd/

[3] Branden, L. Ricciulli: *PA Plan for a Scalable ABone – A modest proposal.* Technical Report, USC – Information Science Institute, January 1999.

[4] E. Amir, S. McCanne, R. Katz: *An active service frame-work and its application to real-time multimedia transcoding.* ACM SIGCOMM'98, Canada, 1998.

[5] M. Sifalakis, S. Schmid, T. Chart, D. Hutchison: *A Generic Active Service Deployment Protocol.* Second International Workshop on Active Network Technologies and Applications, ANTA, Osaka, Japan, May 2003.

[6] M. Bossardt, A. Mühlemann, R. Zürcher, B. Plattner: *Pattern Based Service Deployment for Active Networks.* Second International Workshop on Active Network Technologies and Applications, ANTA, Osaka, Japan, May 2003.

[7] M. Bossardt, L. Ruf, R. Stadler, B. Plattner: *A Service Deployment Architecture for Heterogeneous Active Network Nodes.* 7th Conference on Intelligence in Networks (IFIP SmartNet 2002), Saariselkä, Finland, April 2002.

[8] R. Keller, L. Ruf, A. Guindehi, B. Plattner, PromethOS: *A Dynamically Extensible Router Architecture Supporting Explicit Routing*, IWAN 2002, Dec. 2002

[9] Mobile IP Resources : http://mosquitonet.stanford.edu/mip/resource.html

[10] C. Prehofer, N. Nafisi, and Q. Wei. A *framework for context-aware handover decisions*. IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, Beijing, China, September 2003

[11] C. Prehofer and Q. Wei. *Active networks for 4G mobile communication: Motivation, architecture and application scenarios.* International Working Conference on Active Networks, Zurich, Switzerland, 2002.

[12] M. Stemm and R. Katz, *Vertical handoffs in wireless overlay networks*, ACM Journal on Mobile Networks and Applications, Vol. 3, No. 4, 1998.

[13] K. Pahlavan, et al, *Handoff in hybrid mobile data networks*, IEEE Communication Magazine, April 2000.

[14] Helen J. Wang, Randy H. Katz, and Jochen Giese. *Policy-Enabled Handoffs across Heterogeneous Wireless Networks*, WMCSA 99, IEEE, Feb.1999

[15] Michael E. Kounavis, et al, *Design, Implementation and Evaluation of Programmable Handoff in Mobile Networks*, ACM Journal on Mobile Networks and Applications (MONET), September 2001

[16] Johnson, C. Perkins, and J. Arkko, *Mobility Support in IPv6*, IETF Internet-Draft, June 2003, Work in Progress.

[17] http://manimac.itd.nrl.navy.mil/MGEN/

[18] W. Schilit, *A System Architecture for Context-Aware Mobile Computing*. PhD thesis, Columbia University, 1995

[19] M. Spreitzer and M. Theimer, *Providing location information in a ubiquitous environment*, Proc. of ACM Symposium on Operating Systems Principles, Aug. 1993

[20] T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. Serra, and M. Spasojevic, *People, places, things: Web presence for the real world,* Proc. of IEEE Workshop on Mobile Computing Systems and Applications, Dec. 2000.

[21] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer, *Easyliving: Technologies for intelligent environments*, Proc. of Handheld and Ubiquitous Computing Symposium, Sept. 2000.

[22] A. T. Campbell, M. E. Kounavis, D. A. Villela, H. G. De Meer, K. Miki, and J. Vicente, *The Genesis Kernel: A Virtual Network Operating System for Spawning Network Architectures*, Proc. of IEEE Openarch 1999, New York, NY, USA, March 1999.

[23] M. E. Kounavis, A. T. Campbell, S. Chou, F. Modoux, J. Vicente, and Hao Zhuang, *The Genesis Kernel: A Programming System for Spawning Network Architectures*, IEEE Journal on Selected Areas in Communications (JSAC), Vol. 19, No. 3, March 2001.