Real-time device capabilities management

Warnier, J.C.J.M.

Award date:
2003

TU/e technische universiteit eindhoven

Real-Time Device Capabilities Management

By J.C.J.M. Warnier

/faculteit elektrotechniek

Eindhoven University of Technology
Faculty of Electrical Engineering
Division of Telecommunication Technology and Electromagnetics
Radiocommunications Group

Real-Time Device Capabilities Management

door J.C.J.M. Warnier

Master of Science Thesis
completed 1 August 2003

Supervisors:
Hans Nelissen, Vodafone
Peter Smulders, TU/e

Graduation professor:
prof. dr. ir. E.R. Fledderus, TU/e

TU/e

Real-Time Device Capabilities Management

J.C.J.M.Warnier

vodafone

Document History

Issue	Date	Reviewed by	Remarks
0.7		Peter Smulders	Review TU/e
0.8		Maurice Martin	Review Vodafone
0.9		Erik Fledderus	Review TU/e
1.0		Jos. Warnier	Final Version

Document Control

Title	Real-Time Device Capabilities Management
Abstract	A functional service architecture is designed for real-time device capabilities management
Document Number Author ID (TU/e) Reviewers Project Project Manager Date	RC_NL_2003_039 Jos. Warnier (J.C.J.M.) 0406059 Maurice Martin (Vodafone R&D) Erik Fledderus (Eindhoven University of Technology) Peter Smulders (Eindhoven University of Technology) Real-Time Device Capabilities Management Hans Nelissen August 2003
Classification	For internal use only at Vodafone and the Eindhoven University of Technology
Persons Responsible at Vodafone	Maurice Martin Hans Nelissen
Key Words	Service Architecture, Device Management, Real-Time Device Capabilities Management
Distribution List	Bos, Bert van den Burden, Jasper Fledderus, Erik (TU/e) Lahaije, Paul Martin, Maurice Nelissen, Hans Smulders, Peter (TU/e) Warnier, Jos.

Executive Summary

The scope of this report is to develop a service architecture for real-time device capabilities management. In order to allow providers and third parties to deliver content specifically tuned for the device of the mobile user, the configuration of the device in terms of hardware and software capabilities has to be known by these parties. It goes without saying that this information about the capabilities of the mobile device has to be as accurate as possible, thus a mechanism has to be designed that updates the information about the capabilities of the device as soon as changes take place, i.e. on a real-time basis. As a result, the conceptual and functional architectures for the solution of the above problem are presented in this report.

First of all, the design method of the Service Architecture competence team of Vodafone Group Research and Development is explained in order to provide a clear understanding of the modelling process. Subsequently, the problem of real-time device capabilities management is split up in three stages, that are ascending in complexity. First a simple device swap is considered, then the case where software download is effectuated and finally the situation where the capabilities of a personal area network have to be managed.

For all three stages of the solution of the general problem, the design part of the modelling process has been executed. This resulted for all three stages in a conceptual architecture, the functional requirements (use case descriptions and analysis) and the functional architecture (sequence diagrams and interface descriptions). For the device swap stage, the implementation deployment has been effectuated as well, i.e. the non-functional requirements and the technical architecture have been presented. Some workarounds for this stage have been developed as well.

The work presented in this report can be used as a guideline for further design of the technical architecture, thus leading to the implementation of real-time device capabilities mangement. This implementation of this technical architecture in the network will lead eventually to enhancement of the user experience, because the user will receive content that is always specifically formatted for the mobile device that he is using.

Preface

This Real-Time Device Capabilities Management study has been done for my thesis of my study Electrical Engineering of the Eindhoven University of Technology. I am pleased to acknowledge the unconditional support and friendship of all the people at Vodafone Group Research and Development -The Netherlands. They assisted me on the subject with encouraging enthusiasm.

But foremost I would like to thank my direct supervisors Maurice Martin at Vodafone and Erik Fledderus and Peter Smulders at the Eindhoven University of Technology. Without their direct support on the subject and their accurate reviews it simply would not have been possible to present this report.

Finally I would like to express my gratitude to my teammanager Hans Nelissen and R&D-NL manager Bert van den Bos for giving me the opportunity to gain this working experience at Vodafone Group Research and Development.

Jos. Warnier
Maastricht, August 2003

Table of Contents

Chapter 1
Introduction

1.1 Problem context

As the business-traveller leaves his European home, he takes his personal mobile phone with him. This mobile phone just has the usual features like GSM 900/1800, SMS text editing and a WAP 1.0 browser. He just bought it so that his wife Barbara can reach him in case of emergency and to access some stock-information when he's on his way to work.

When the business-traveller has to go to America, he changes his personal mobile phone at the office for a more advanced triple-band GPRS device with an integrated camera, MMS and Java capabilities, WAP 2.0 browser, Bluetooth and an advanced OS. He places his SIM in the new device, switches his advanced phone on, but does not initiate a new WAP session. This way, the capabilities of his advanced phone are not communicated to the network, so it assumes he is still using his personal mobile phone with less advanced features. On his way to the airport his colleague Bart sends him a MMS with a picture of their latest prototype, so he can show it to his American customers. However, because the network is not aware of the fact that at the moment the business-traveller is using his MMS capable phone, he just receives a link to a webpage where the photo of the prototype is stored, so he can't access it immediately on his mobile phone.

Despite the fact that he could not show the photograph to his American customers in Hillbilly County, a state that is infamous for it's absolute lack of computers, his business trip was quite successful and he has a day left before he flies back to Europe. When he notices that the latest version of the OS of his advanced mobile phone is already available in Hillbilly County, he decides to download it to his phone, so he can show off with it to Bart when back in Europe. Just before catching his flight, the business-traveller buys at the airport a state-of-the-art digital camera with Bluetooth capabilities.

Inside the plane, he unpacks the camera and starts to read the manual, while in the meantime a stunning stewardess serves him his whiskey. Reading all the preferences he has to set manually, the connections he has to initiate and the capabilities he has to communicate to the network just before he can send a picture from his camera via Bluetooth and his advanced mobile phone to another mobile device, he orders his third whiskey and thinks "It would be really nice when all these preferences and capabilities were communicated dynamically to the network. At the very moment that the two devices shake hands, the capabilities and preferences of my mobile device should be dynamically updated to the network, so that I receive all my messages in the correct advanced format." When she serves him his fifth whiskey, the business-traveller takes a picture with his new camera of the beautiful behind of the stunning stewardess and sends it to his colleague Bart. After that he takes a picture of himself and sends it to his wife Barbara.

During the taxi ride home, the business-traveller gets bored, so he decides do download a new Java game to his mobile phone. After downloading 95% of the game, he gets the message that his phone is out of memory, hence the already downloaded part of his game is useless and he stays bored.
When arriving at his home, the business-traveller wonders why Barbara is waiting with the rolling pin in her hands. In the meantime Bart is wondering why he received a picture of the business-traveller. Well, dear business-traveller, real-time device capabilities management most certainly is very handy, because it dynamically updates the capabilities and preferences of your mobile device, but, painful as it may be, it will not solve all your problems...

1.2 General problem description

The current mobile devices become more and more advanced. The improvements include hardware updates as well as software updates. To exploit advanced features, for instance a larger display or a new version of the operating system, it is desirable that the network is aware of the new Device Configuration (DC).

Currently, the update of the DC is not done real time. For example, with WAP the network becomes aware of the new DC just then when a new WAP session is started, otherwise the network assumes that the situation has not been changed. So if no new session is initiated, the network makes use of the old capabilities and preferences and the new DC is disregarded, this is a static situation.

1.3 Problem approach

To avoid the situation described in the previous section, a real-time update of the DC of the mobile device is necessary. That is, as soon as the hardware or software update is executed, the network has to be made acquainted with the new DC in order to exploit its capabilities to the maximum.
The goal of the real-time update of the DC is to improve the user experience by assuring that changes in capabilities and preferences are instantaneously reported to the network. This way the service layer is capable of tuning the services correctly to the available output devices.

In order to tackle the described problem in a structural way, the problem will be solved in three stages. The three stages are device swap, software download and external devices and are described in more detail below. The order of the stages is consciously chosen in terms of market introduction. Device swaps already occur daily, software downloads start to make their way to the market and the adoption of external devices in Personal Area Networks (PAN) will be a matter of a few years.

For the solution of each stage it is assumed that when the current DC is communicated to the network, the device configuration is stored in a Device Configuration Database (DCD). The existence of such a DCD is assumed for the rest of this report. The DCD will thus be regarded as a passive actor in which the current DC can be found and in case of the change of the DC of the mobile device, the new DC will be presented to. Now the general problem is described, it can be depicted as in Figure 1-1.

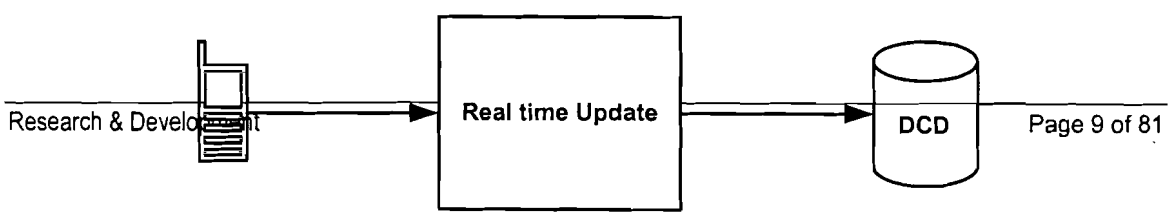


Figure 1-1: *General problem*

1.4 Structure of the report

The structure of this report is as follows. Chapter 2 describes the design method that is used for solving the general problem. After a short introduction on the conceptual, functional and technical architecture, the modelling process is explained. In Chapter 3 the first stage of solving the problem is treated, that is the device swap. The second stage that leads to the solution of the problem is how to detect whether crucial software download has taken place and if this has to be updated to the network, this is described in Chapter 4. The final stage that has to be overcome is when external devices are attached to the mobile device. The extra capabilities of these external devices may be useful to be known by the network in order to deliver content in the best way for the user. This problem is tackled in Chapter 5. Finally the results of the previous will be discussed in Chapter 6.

Chapter 2
Design method

In this chapter the design method that is used to solve the general problem described in Chapter 1 is explained. The design method is the one used at the Service Architecture competence team of Vodafone Group Research and Development.

2.1 Introduction

Architecture modelling is a technique used to structure architectures into various complementary units. Each unit addresses a specific concern, thus enabling separation of concerns. Because current telecom services are becoming more and more complex, a common service architecture is needed on top of the well known network architecture. The differences between the terms service architecture and network architecture in this report can be described as follows. The network architecture is the architecture that provides connectivity for the user, it just connects the user with the network. The service architecture can be seen to be placed on top of the network architecture. It provides services that enhance the user experience.
To achieve the above, three types of service architecture models are introduced:

The purpose of the *Conceptual Architecture* is to direct attention at an appropriate decomposition of the system without delving into details. Moreover, it provides a useful vehicle for communicating the architecture to non-technical audiences, such as management, marketing, and users. It consists of the architecture diagram (without interfaces) and an informal component specification for each component.

The *Functional or Logical Architecture* adds precision, providing a detailed "blueprint". It incorporates the detailed Architecture Diagram (with interfaces), element and Interface Specifications. This architecture is used by technical specialists to make a detailed architecture on which an implementation can be made.

The *Technical Architecture* is actually an execution architecture showing deployment information (showing which physical systems take care of which functional element) like IP addresses, hardware information (dual node clusters), SS7 information, etc. This architecture is used to implement the service architecture within the mobile network by the (deployment) engineers.

Figure 2-1 shows the architecture models used for designing the service architecture.
Since the picture of the technical architecture is rather hard to see, an enlargement for instructive purposes can be found in Figure 2-2. Note that the technical architecture depicted in this figure is not a solution for the general problem, but is just used here as an example for a technical architecture.

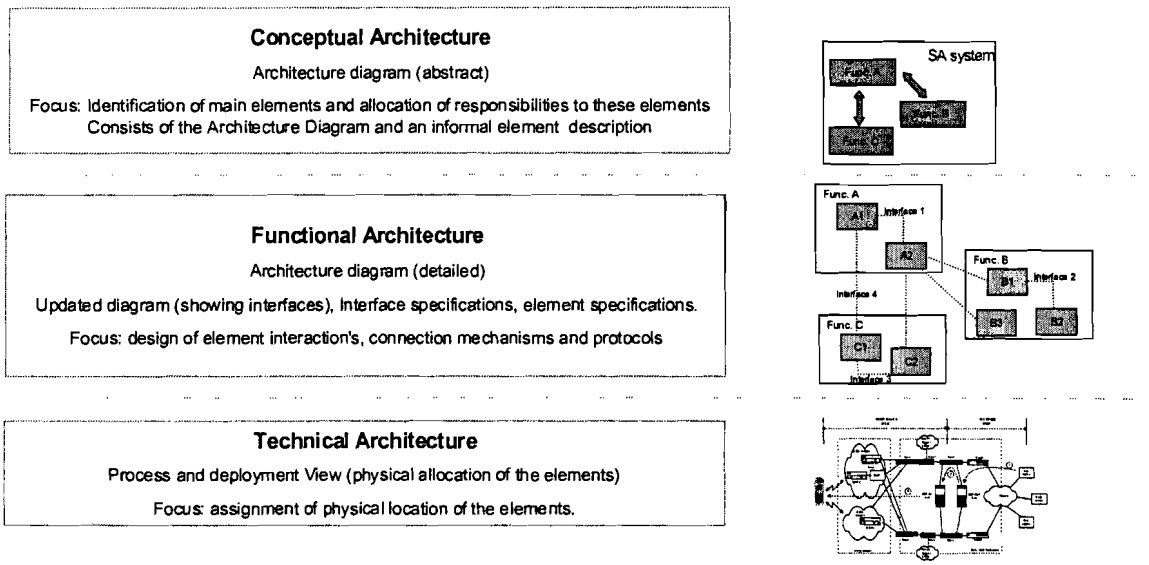


Figure 2-1: *The used Service Architectural model*

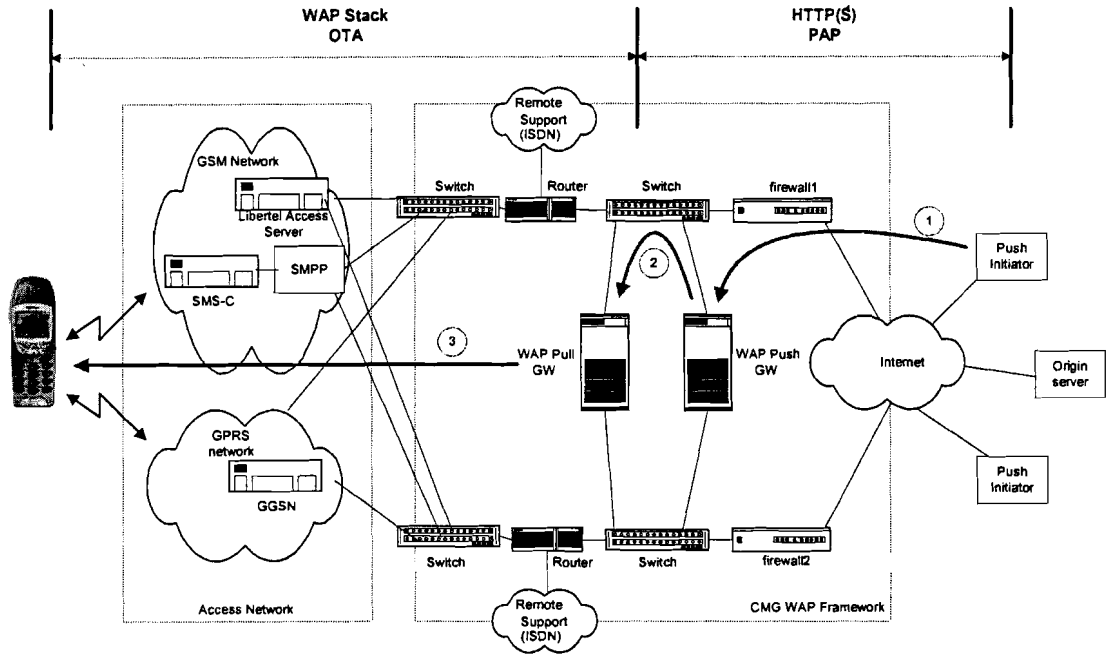
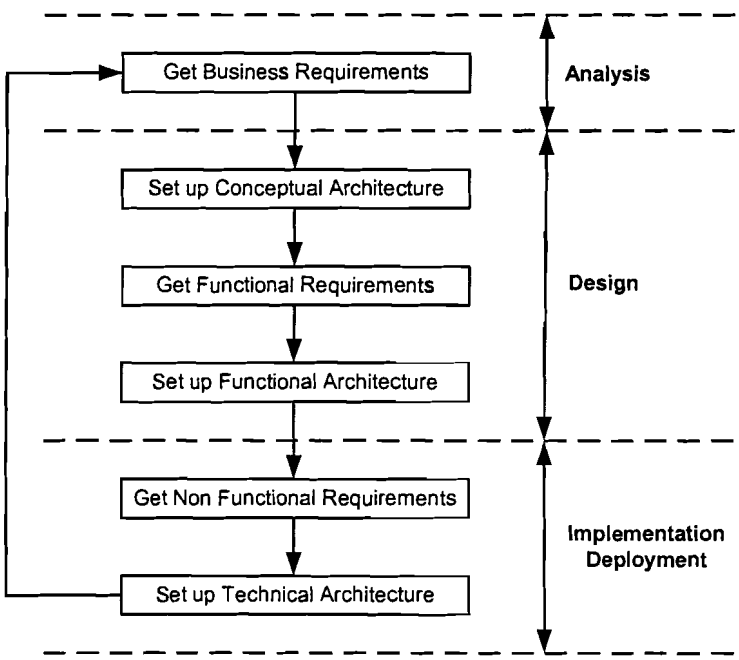


Figure 2-2: *Enlarged picture of a Technical Architecture*

2.2 The modelling process

Figure 2-3: *The Service Architecture modelling process*



The Service Architecture process as depicted in Figure 2-3 is used to get to the first increment

1. analyse the Vodafone vision and define the business requirements (e.g. consistent user experience towards end users, access for third parties to the network and intefacing)
2. set up a conceptual service architecture
3. analyse the conceptual service architecture and define the functional service requirements (e.g. update of the DC has to be effectuated in real time, the DC has to be stored in an existing database)
4. design the functional service architecture
5. collect the non-functional service requirements (e.g. costs, ease of implementation of a particular solution)
6. design the technical service architecture

It needs to be noted that the whole process is iterative, meaning that depending on results (or missing requirements) a jump can be made to a previous process step e.g. from functional architecture towards conceptual architecture.
Technology and business strategy is changing on regular bases. For this reason, the process needs to deal with this flexibility and therefore it will be an iterative process. The iteration process is based upon the fact that building a service architecture is an evolving activity which can never be completed.
After making the first increment of the service architecture, the process will be triggered by a market push (a change in business requirements), or a technology push (a change in technological aspects, e.g. new emerging technologies).

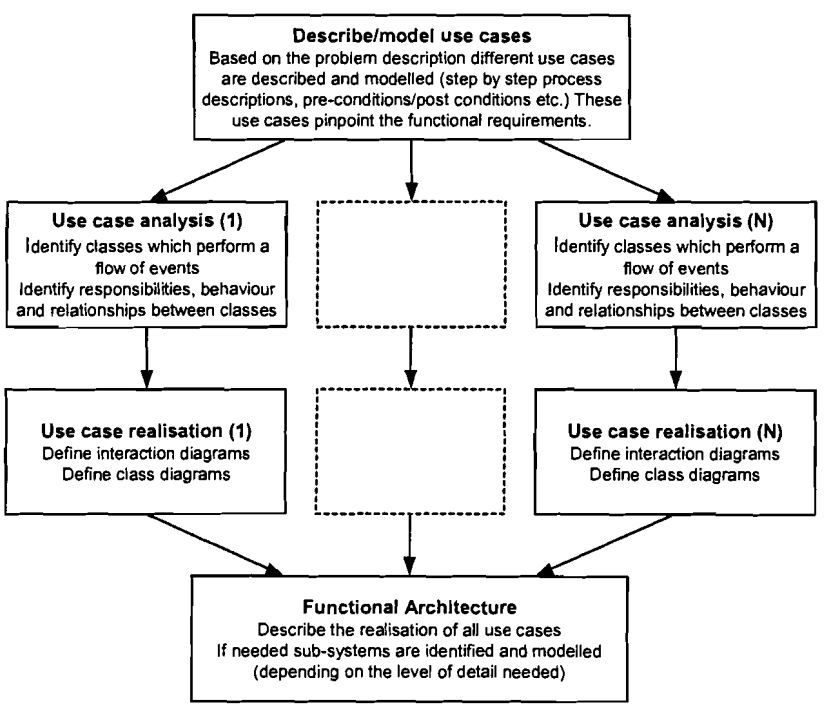


Figure 2-4: *Modelling the Functional Architecture*

The step from "Get Functional Requirements" to "Set up Functional Architecture" in Figure 2-3 is worked out in Figure 2-4. The functional requirements are distilled from the use cases during their analysis, subsequently interaction and class diagrams are designed. Finally, all use cases and their realisations are merged into the final functional architecture.

Chapter 3
Device Swap

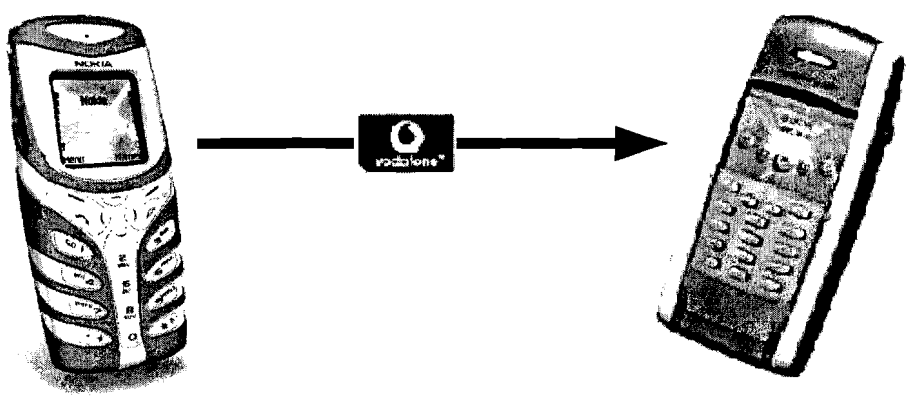


Figure 3-1: *Device swap*

3.1 Problem statement

When a device swap is executed, the SIM is placed in to a different mobile device. This new device may have a different device configuration (DC) compared with the previous mobile device and thus an update of this new configuration to the network is desirable. However, in the current situation this update of the DC is not executed unless a new WAP session is started. WAP is the worldwide standard for providing internet communications and advanced telephony services on digital mobile phones and other wireless terminals [WAP01a]. If no new WAP session is started, it

may be possible that the network delivers data to the new mobile device that is formatted for the previous device. To avoid this situation, the new DC should be communicated to the Device Configuration Database (DCD) in the network as soon as a device swap is executed.

3.2 Existing technologies

3.2.1 UAProf

The User Agent Profile [UAP02] is a schema developed to describe classes of device capabilities and preference information. These classes contain information about the hardware and software capabilities of devices, sometimes information about the network as well. With the User Agent Profile (UAProf) information, origin servers can format the content they deliver to mobile devices.

Figure 3-2 depicts the UAProf end-to-end architecture.
As can be seen, the mobile device can connect in several ways to an Origin Server (OS). WAP clients can connect to a WAP gateway by using the Wireless Session Protocol (WSP), the WAP gateway in turn connects to the origin server via the HyperText Transfer Protocol (HTTP). In theory it is possible as well to connect immediately to the origin server, given that the origin server is capable to handle WSP.
Another way to connect to the origin server is by using Wireless Profiled HTTP, as with WSP, it is possible to connect via a HTTP Proxy server to the origin server or immediately to the origin server.
The origin server can connect to the mobile device by using the WAP push mechanism.
All these ways of communication between the mobile device and origin server are discussed in the subsequent sections.

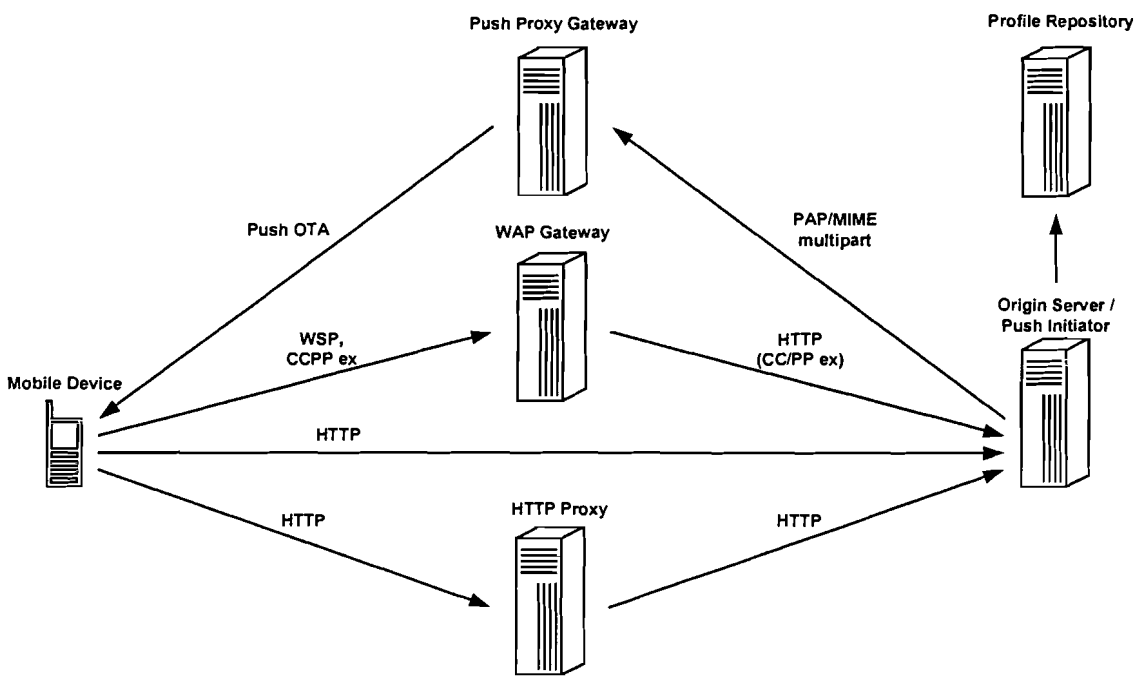


Figure 3-2: *UAProf end-to-end architecture*

3.2.1.1 WAP/WSP clients

Nowadays, WAP/WSP clients generate most mobile data traffic. The architecture that describes the request by and the delivery of formatted content to a mobile device is depicted in Figure 3-3. The user agent profile references are transmitted as a parameter of the WSP [WAP01b] session to the WAP gateway and cached. This information is then transferred over HTTP using the Composite Capabilities/Preference Profile Exchange Protocol (CC/PPex), which is an application of the HTTP Extension Framework (HTTPex).

The sequence that is executed in order to receive the requested content is described below. First of all, the mobile device sends a WSP request with its profile information or profile difference to the WAP gateway. The gateway caches the WSP header. Subsequently, the gateway sends the request with the profile information using HTTP to the origin server, which composes the current profile by resolving the profile references and retrieving the information from the profile repository. Then the origin server adapts the content according to the user agent profile and sends the adapted content back to the gateway. Finally, the WAP gateway returns the adapted information using WSP to the mobile device.

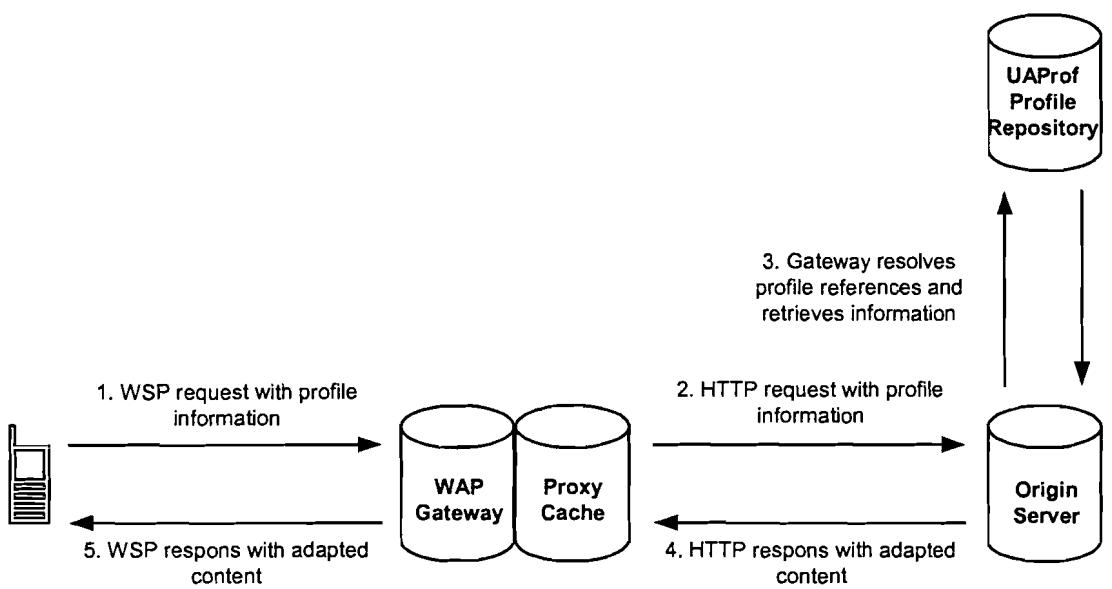


Figure 3-3: *WAP/WSP architecture*

3.2.1.2 Wireless Profiled HTTP clients

Wireless Profiled HTTP will be used in future WAP 2.0, the Capabilities and Preference Information (CPI) data is sent with each HTTP request that is made. As already described above, it will be possible to communicate directly with an origin server that is able to handle wireless profiled HTTP, or the mobile device sends its wireless profiled HTTP request to a HTTP proxy, which in turn hands the request to the origin server using HTTP.

3.2.1.3 Push environment

The push environment is depicted as well in Figure 3-2. The Push Initiator (PI) submits a push message to the Push Proxy Gateway (PPG) using the Push Access Protocol (PAP). The push message contains, besides a control entity and a content entity, an optional capabilities entity in UAProf format. The push proxy gateway (PPG) may use the capability information to validate if the message is appropriate for the client. If so, the push proxy gateway (PPG) delivers the message to the mobile device using the Push Over-The-Air (OTA) protocol.
It should be noted that the push initiator has the ability to query the push proxy gateway (PPG) for the capabilities of a specific device.

There are four different content types for push messages transferred to the mobile terminal: Service Loading, Service Indication, Cache Operations and User Defined. The User Defined content type can be for a specific application. The first three content types are described below.
The Service Loading (SL) content type is standardised by the WAP Forum. It provides the ability to make a user agent on a mobile client load and execute a service. The SL is in the form of a URI (Uniform Resource Identifier). The SL, containing a URI, is pushed to the mobile client. So far the end-user is not made aware of this. The user agent now retrieves (pulls) the service indicated by the URI from the origin server (or from its cache). When the page is displayed the user is notified. An example of a SL can be a message containing a stock quote with a graph, the kind of information that a stock trader needs to get instantly.
The Service Indication (SI) content type is standardised by the WAP Forum. It provides a way to send a notification to the end-user. The SI basically contains a short message and a URI indicating a service. The message is presented to the end-user, who can then choose to either start the service indicated by the URI, or postpone the SI till later so that it is stored and the indicated page can be downloaded later.
An example of a SI can be a notification of a newly arrived e-mail. The short message can contain the name of the sender. The subscriber can then choose to download the message directly or postpone it till later.
The Cache Operation (CO) content type provides a means to invalidate content objects in the user agent cache. This means that the next time the object is requested, it must be re-loaded from the origin server. This operation is performed without the end-user's knowledge.
An example of a use for CO can be a WAP site containing news. When the news is updated the cache should be removed so that the new news is downloaded the next time the service is accessed.

3.2.2 SyncML Device management

The goal of the Device Management Working Group of the SyncML initiative is to specify protocols and mechanisms that achieve management of mobile devices [Syn01].
Management includes:
- Setting initial configuration information in devices
- Subsequent updates of persistent information in devices
- Retrieval of management information from devices
- Processing events and alarms generated by devices

In the scope of Device Management, information includes:
- Configuration settings
- Operating parameters
- Software installation and parameters
- Application settings
- User preferences

Though SyncML DM is very handy in configuring and updating devices, no such thing is embedded as a registry that is able to provide real-time the DC of the terminal. It has to be emphasised that the main areas for SyncML include the five settings and preferences mentioned above, so the hardware capabilities are not handled.

3.3 Conceptual Architecture

The standard conceptual architecture that is used by Vodafone is depicted in Figure **3-4**. The service architecture (SA) system (the system which offers all "telecom services to its users") can be derived from the users of the system.
The following users of the service architecture (SA) system are identified:

Mobile Subscriber
3rd parties (& IT partners) thus the application developers and 3rd party networks (non-Vodafone).

Taking these users and the business requirements into account the following high level elements of a conceptual service architecture (see Figure **3-4**) can be identified:

User Interaction
Takes care of dealing with device specific information and "tuning" the content for the device (Vodafone branding). Also personalisation is covered by this component. (control elements making up the user experience/branding)

Service Broker
Enables the generic functions needed for offering the Vodafone building blocks towards 3rd parties, the 3rd party management functions. To support business to business integration, the service broker also needs to support billing and service management kind of functionality ("open up" the networks, secure simple chargeable access)

Service Layer
The Vodafone building blocks (functional offering) which will be offered to the external Vodafone world and the service creation aspects around it.

Network Layer
The building blocks need to have connectivity with their users (e.g. other elements). The network layer takes care of this responsibility. within the mobile networks (the IT infrastructure like OSS & BSS are seen as part of it). To get more focus, this component is already split up in 2 sub-components:
Switching Network
Integration of building blocks with switching networks (GSM, GPRS)
Backend
Integration of building blocks with backend systems (like billing, provisioning)

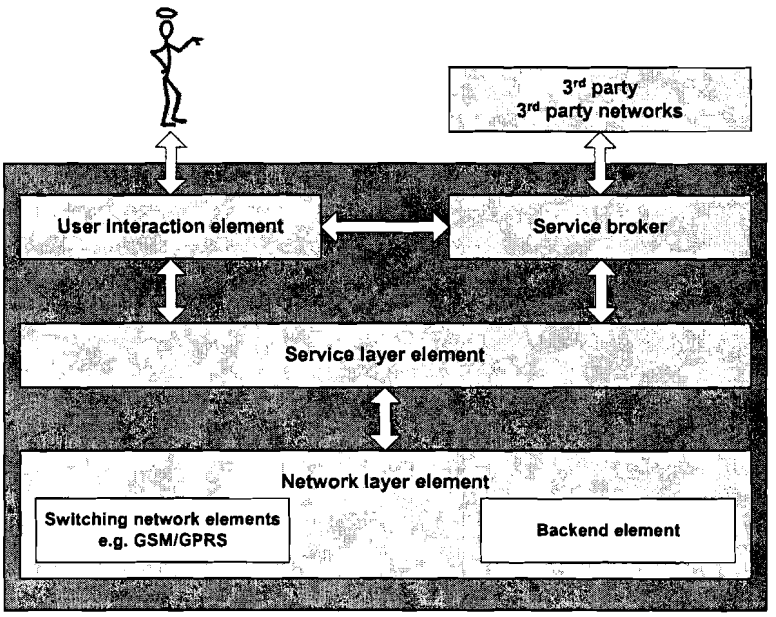


Figure 3-4: *Vodafone conceptual service architecture*

The final goal of real time communicating the device configuration (DC) of devices to the network is to allow Vodafone and 3rd parties to use this information in order to tune their content specifically for each device. However, since this report is focussed on dynamically updating the device configuration, the interaction with 3rd parties and the service broker are beyond the scope of it. Hence,
Figure **3-4** can be simplified and represented as Figure 3-5.

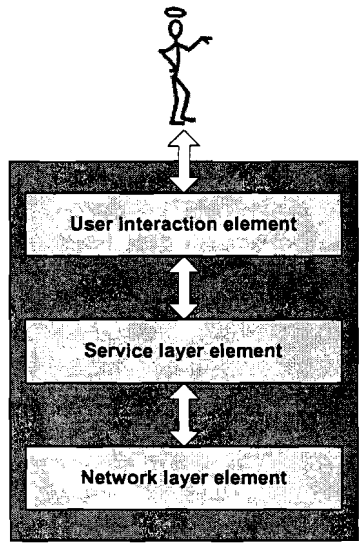


Figure 3-5: *Conceptual service architecture DDC*

3.4 Functional requirements

3.4.1 Brief use case description

This use case allows the internal and external device capabilities database (DCD) to be updated real time after a device swap has taken place.

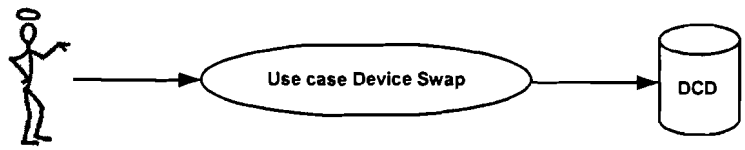


Figure 3-6: *Use case Device Swap*

3.4.2 Flow of events

- The user places the SIM in another mobile device
- The user switches the device on and enters his PIN code
- The devices executes a device scan to determine the current DC
- The device connects to the network
- The device updates the current DC to the DCD
- The network offers data to the device according to the current DC

3.4.3 Analysis functional requirements of the use case

In order to translate the functional requirements into a functional architecture, a closer look will be taken at the flow of events and different components of the functional architecture will be derived. The first "component" is of course the user that swaps the mobile devices and then switches the current mobile device on. The second component is then inevitably the mobile device itself. The

device scan, which is mentioned in section 3.4.2, has to determine the current DC of the mobile device, furthermore the mobile device has to connect to the network and update the DCD. As stated in section 1.2, it is assumed that the DCD is a given parameter, so the DCD is considered as a passive actor. Hence, the DCD is a separate component as well.
To effectuate the device scan and the update of the DCD, it might be wise to introduce a component that controls the actions that are needed to do so. This component is labelled the DC Scan. For the first stage of the solution of the general problem, the device scan suffices when it can detect whether a device swap has taken place or not. So when the ID's of the previous and the current device scan are known, a simple comparison between these two will reveal whether the SIM is placed in another device or not. This analysis leads to two more components that will have to be implemented. First of all there will have to be some sort of database that stores the outcome of a device scan. It will have to store at least the two most recent device ID's in order to be able to compare them and find out whether a device swap has taken place. This component will be called the Device ID Storage (DIDS). The final component will compare the two most recent outcomes of the device scan and is named comparator. This completes the functional analysis of the use case and has lead to the characterisation of the components.

3.5 Functional Architecture

3.5.1 Sequence Diagram

From the flow of events described in section 3.4.2, it is clear that the device scan has to be performed by a component that is capable to manage the scan itself and the accompanying communication of the device scan to the network. It is not very likely that the device scan application should be implemented in the mobile device itself, since the possibility exists that when a swap occurs between a device scan enabled mobile device and a device that has not this application, the DC cannot be updated. Implementation of the device scan application is more likely on the SIM, because it is the only piece of equipment that moves to the new mobile device in case of a device swap. This way the device scan application is always located in the device that is going to be used.
After the DC scan has been executed, the DC has to be communicated to the network. To avoid unnecessary load to the network, it might be wise to check whether the DC has changed at all. There is no use in "updating" an external database with information that it already contains. A solution can be found in introducing an internal database that stores the DC. When a new DC scan is run, a component named comparator can compare the outcome of the DC scan and the DC that is stored in the internal database. Should the comparator return any differences, both databases will have to be updated. In case the comparator returns no differences, the result of the DC scan can be discarded and no update will have to be performed and the network will not have to transport the update.

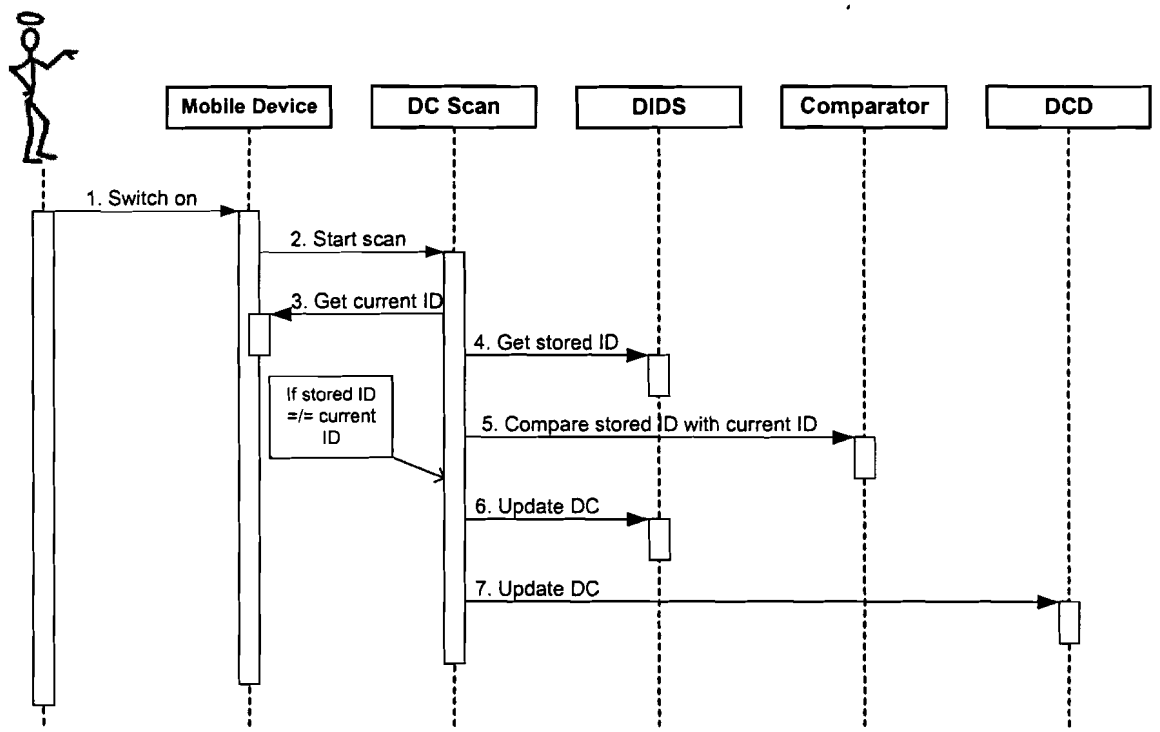


Figure 3-7: *Sequence diagram device swap*

3.5.2 Exception handling

The sequence diagram and its description in section 3.5.1 do not cover the entire range of possible occurrences. Exceptions to the sequence of section 3.5.1 are described in the subsequent paragraphs.

3.5.2.1 The DC has not changed

In the case that the comparator returns that the stored DC matches the current DC, the DC database will not have to be updated, so the DC Scan is finished after comparison, as depicted in Figure 3-8

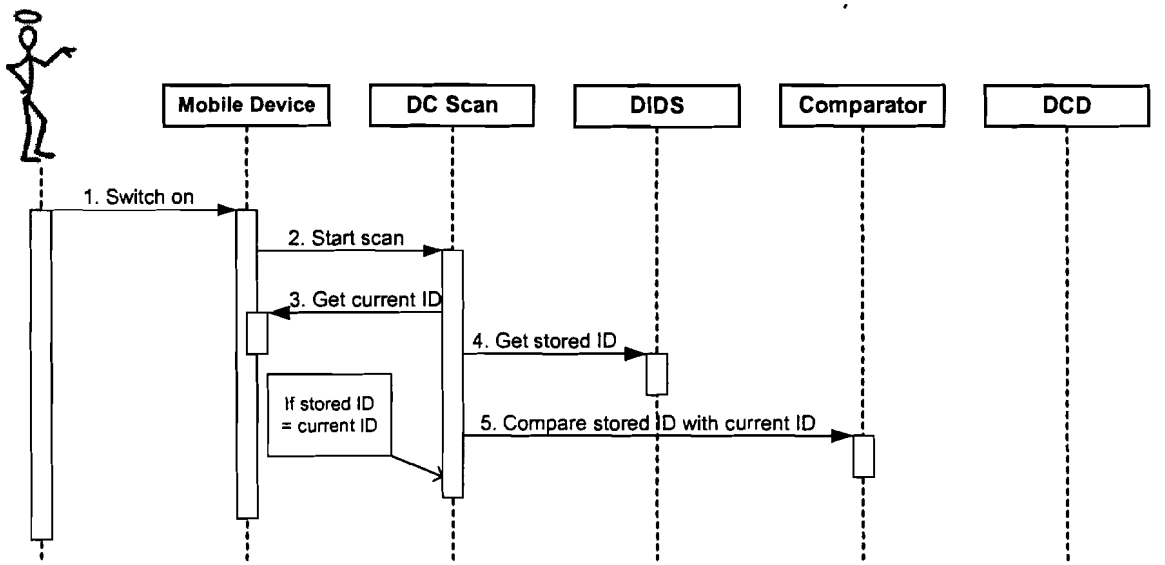


Figure 3-8: *Sequence diagram when the stored ID matches the current ID*

3.5.2.2 The network is not available

If the network is not available to update the DCD, the mobile device should trigger the DC Scan as soon as the mobile device connects to the network. Immediately after connecting to the network, the DC should be send to the DCD.

3.5.2.3 The DCD is not available

If the network is available to update the DCD, but the DCD itself is not, then the current DC cannot be stored on the DCD. We assume that the DCD is very reliable and consider the chance that the DCD is not available small. We assume as well that when the DCD is not available, some back-up scenario almost immediately will take over its tasks, so the down time will be very small as well. So when the DCD is not available, it has to be sufficient that the DC Scan retransmits the current DC up to three times.

3.5.3 Architecture

As can be seen in Figure 3-9, the high level architecture of the device swap consists of six building blocks: User, Mobile Device, DC Scan, DIDS, DC Comparator and DCD.

User
The user is the actor that uses the mobile device.

Mobile Device
The entity of which the DC has to be determined and dynamically updated.

DC Scan
As soon as the new device is switched on by the user, a DC scan is started by the mobile device. The responsibility of the DC scan is to discover the capabilities of the new device. Initially the DC scan only will determine the device type and its standard capabilities. The result of the DC scan, in the first stage the ID of the mobile device, is stored in the DIDS.

DIDS
The Device ID Storage (DIDS) contains the result (device ID's) of at least the two latest device scans.

DC Comparator
The DC comparator compares the current DC with the existing information, which is stored in the DIDS as well. If the current DC differs from the stored DC, the contents of the current DC field is placed in the stored DC field and subsequently the contents of the current DC field can be discarded. In the case that the stored DC and current DC are exactly the same, the contents of the current DC field just can be discarded.

DCD
In the case that the DC is updated in the DIDS, the DCD needs to be updated as well.

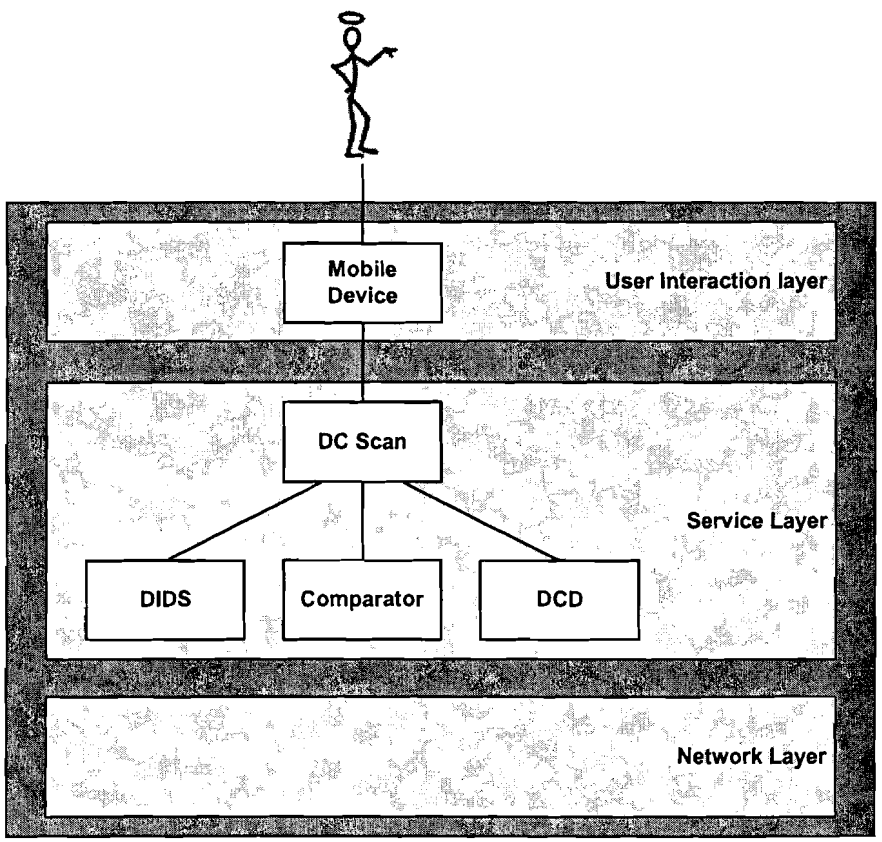


Figure 3-9: *Functional architecture device swap*

Figure 3-9 shows as well how the conceptual architecture of the handset swap fits into the overall conceptual architecture. The mobile device belongs to the user interaction layer, whereas the DC scan, DIDS, comparator and the DCD are integrated in the service layer. It might seem strange that the DCD is classified as a service layer element, because it is physically present in the network. But since the conceptual architecture just identifies the main elements this classification is valid, because the DCD in the network is used to enhance the user experience and thus belongs in the service layer.

3.5.4 Interface descriptions

Mobile Device	
SwitchOn()	The mobile device is switched on
Parameters	
Returns	
DC Scan	
StartScan()	The DC scan is activated
Parameters	
Returns	CurID: The current ID of the mobile device
DIDS	
GetStoID()	The stored ID is retrieved from the DIDS
Parameters	
Returns	StoID: The stored ID of the mobile device
UpdateDIDS()	The current ID is stored in the DIDS
Parameters	CurID: The current ID of the mobile device
Returns	OK or NotOK
Comparator	
CompStoCurID()	Compare the stored and the current ID of the mobile device
Parameters	StoID: The stored ID of the mobile device
	CurID: The current ID of the mobile device
Returns	SameID or DiffID
DCD	
UpdateDCD()	The current ID is stored in the DCD
Parameters	CurID: The current ID of the mobile device
Returns	OK or NotOK

3.6 Non-Functional Requirements

Non-functional requirements specify aspects of the system other than its capacity to do things. Examples of non-functional requirements include those relating to performance, accessibility, usability, branding and visual style.
One non-functional requirement that is important for this report is the user-experience. Updating the DCD real time leads to the correct delivery of content to the mobile device, which enhances the user experience.

3.7 Technical Architecture

3.7.1 Location of the components

Three locations can be indicated for implementation of the components described in section 3.5.3, i.e. the SIM, the mobile device and the network. Each of these locations have their pro's and cons depending on the component concerned and are described in this section. For each component the selection of its location of implementation is explained, the results are to be found in Table 3-1

Table 3-1: *Location of Device Swap components*

	SIM	*Device*	*Network*
DC Scan	**Yes**	Yes	Yes
DIDS	**Yes**	No	No
Comparator	**Yes**	Yes	Yes
DCD	No	No	**Yes**

3.7.1.1 Location of the DC Scan

SIM pro's: No need to implement in device, device independence, SIM under Vodafone supervision
SIM cons: Not all SIMs are SIM Aplication Toolkit (SAT) enabled or capable of executing Over-The-Air (OTA) commands, SIM space might be scarce

Device pro's: SIM can be spared
Device cons: Some devices might not have the DC scan implemented

Network pro's: No need to update or replace SIMs or devices
Network cons: Capacity is used when no changes have taken place, heavy signalling load over the air

The ease of over-the-air (OTA) implementation on the SIM outweighs the fact that not all SIMs are OTA enabled. It is for sure that implementation on the SIM is easier to introduce than to implement this function in the device. If the function were to be implemented on the device, only the new devices would be able to effectuate a device scan, so a satisfying penetration of this function would span many years.
Implementation in the network obviously would be even more easy, but as the network would be loaded every time a DC Scan would be effectuated, the signalling load would become disproportionate.

3.7.1.2 Location of the DIDS

SIM pro's: No need to implement in device, easy interaction if DC scan is implemented on SIM as well
SIM cons: SIM space might be scarce, when SIM is placed in new device, the DC stored on the SIM is of the old device and most certainly inaccurate

Device pro's: Plenty of memory available, when SIM is placed in new device, the DC stored in the device is of the old device and most likely more accurate than in the SIM case
Device cons: Some devices might not have an internal database allocated for DC storage

To determine the best location of the DIDS we have to bear in mind the goal of this report. Because we want to track changes in the DC, it is important that the location of the DIDS is device independent. So it can be concluded that the DIDS will have to be implemented on the SIM, since it's the only constant in case of a device swap.

3.7.1.3 Location of the comparator

SIM pro's: No need to implement in device, device independence, SIM under Vodafone supervision
SIM cons: Not all SIMs are SIM Application Toolkit (SAT) enabled or capable of executing Over-The-Air (OTA) commands, SIM space might be scarce

Device pro's: SIM can be spared
Device cons: Some devices might not have the comparator implemented

Network pro's: No need to update or replace SIMs or devices
Network cons: Capacity is used when no changes have taken place, heavy signalling load over the air

For the location of the comparator, the same conclusion as in section 3.7.1.1 can be stated. Moreover, since the comparator will have to interact with the DC Scan it would be very convenient if it were located in the same location. So the comparator will be implemented on the SIM as well.

3.7.1.4 Location of the DCD

Since the assumption is made that the DC is stored in a database in the network, it is clear that the location of the DCD is in the network.

3.7.2 SIM basics

Because most part of the components will be implemented on the SIM, the principles of SIM storage and SIM communication are briefly explained.

The file structure of the SIM is depicted in Figure 3-10.

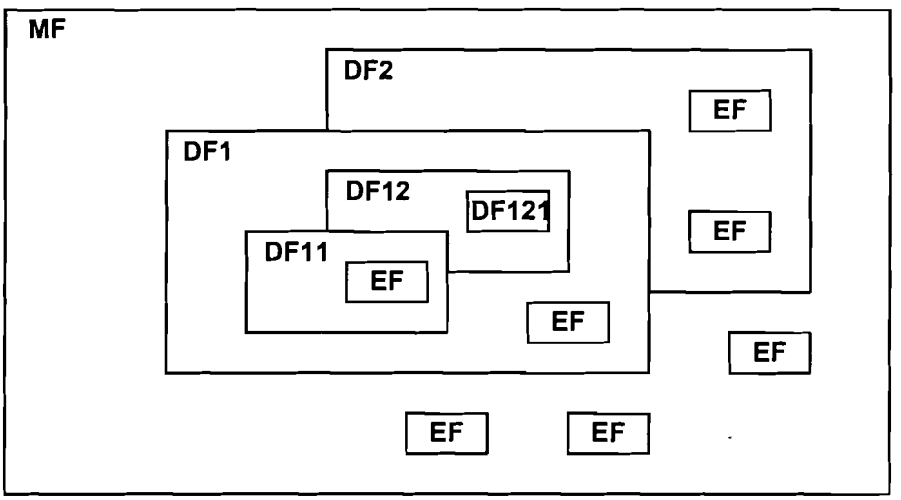


Figure 3-10: *File structure of the SIM*

The root directory is referred to as the MF (Master File). It is implicitly selected after the SIM is reset. The MF contains all other directories and all files, it represents the entire extent of the SIM memory that is available for the file region. The MF must be present in every SIM.
Dedicated Files (DF) may exist underneath the MF as needed. A DF is a directory in which other files (DFs and EFs) cab be grouped together. A DF may contain other DFs, in principle there is no limit on the number of levels of DFs. However, it is rare for there to be more than two levels of DFs under the MF, due to the very limited amount of memory on the SIM.
The useful data that are needed for an application are located in the Elementary Files (EF). EFs may be placed directly under the MF or under a DF.

Communication between the ME and the SIM always takes place within the framework of a transmission protocol, namely T=0. This relatively uncomplicated protocol meets the special requirements of the SIM applications and is optimised for this role. Deviation from this precisely specified protocol within application processes is not permitted. The transmission protocol makes it possible to send data to the SIM and receive data from the SIM, the data are embedded in a sort of container called the Application Protocol Data Unit (APDU). Two types of APDUs exist: command APDUs and Response APDUs. Command APDUs are sent by the ME to the SIM and are thus the commands to the SIM, the ME also receives the responses to its commands in response APDUs embedded in the transmission protocol.

The general format of the command and response APDUs is found in Figure 3-11.

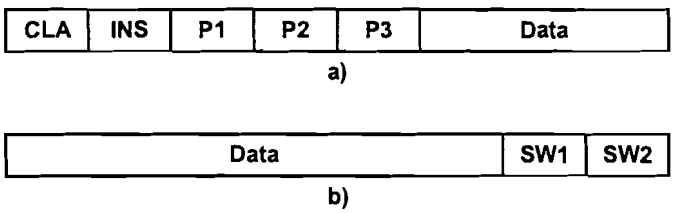


Figure 3-11: *General format of a) command APDU and b) response APDU*

The bytes have the following meaning:
CLA is the class of the instruction, 'A0' is used in the GSM application,
INS is the instruction code for each command
P1, P2 and P3 are parameters for the instruction. P3 gives the length of the data element. Since P3 gives the length of the data in bytes, the maximum length of the data element is 255 bytes.
SW1 and SW2 are the status words indicating the successful or unsuccessful outcome of the command.

On top of the SIM commands described in 3GPP TS 11.11 [3GP01], an extra set of commands is found in 3GPP TS 11.14 [3GP02], i.e. the SIM Application Toolkit (SAT) commands. The SIM Application Toolkit provides mechanisms which allow applications, existing in the SIM, to interact and operate with any mobile device which supports the specific mechanisms required by the application.

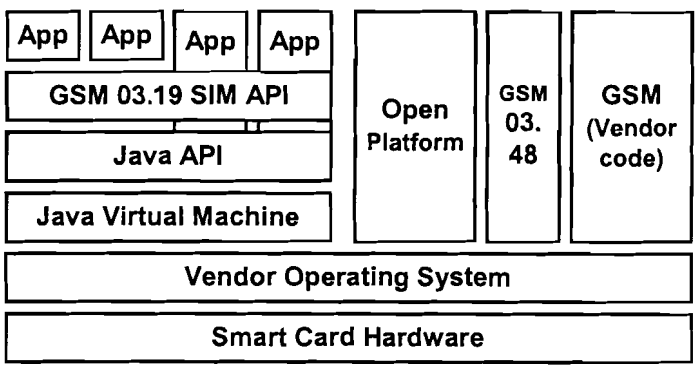


Figure 3-12: *Java Card platform*

A relatively new kind of SIM is the Java Card. The Java Card specifications enable Java technology to run on smart cards and other devices with limited memory. The Java Card API allows applications written for one smart card platform enabled with Java Card technology to run on any other such platform. As shown in the Figure 3-12, the Java Card Virtual Machine (JCVM) is built on top of the smart card hardware and native operating system implementation. The JCVM layer hides the manufacturer's proprietary technology with a common language and system interface. The Java Card framework defines a set of Application Programming Interface (API) classes for developing Java Card applications and for providing system services to those applications. A specific industry or business can supply add-on libraries to provide a service or to refine the security and system model. Java Card applications are called applets. Multiple applets can reside on one card.
As can be seen in Figure 3-12, it is possible to execute SIM Application Toolkit commands using Java, since the SAT commands can be reached by the Java platform using the common Vendor Operating System.

3.7.3 Implementation in SIM Application Toolkit

3.7.3.1 Implementation of the components

Mobile device
The mobile device is given.

DC Scan
The DC Scan should be seen as a sort controller of the events that will have to take place in order to execute the dynamic update mechanism. An application Update Device Capabilities is therefore created.

DIDS
For the implementation of the DIDS, an elementary file EF_{DIDS} is created.

Comparator
The comparator can be implemented in Java or in the vendor operating system. However, since the vendor operating system is proprietary, implementation in Java will be more generic.

DCD
For the DCD two options can be distinguished. One option is to extend Vodafone's Common User Repository [] with a section that takes care of the DC of the mobile device of the user.
One can think of a logical situation where the DCD is built up of a Common User Repository which links the mobile number of a device to the user agent string of the device and a Common Device Repository which links the user agent string to the the user agent profile of the device. This situation is logically depicted in Figure 3-13.

Another option is to develop a distinct database, like has been done by [Mar02a], [Mar02b] and is depicted in Figure 3-16 and will be discussed in section 3.8.1.

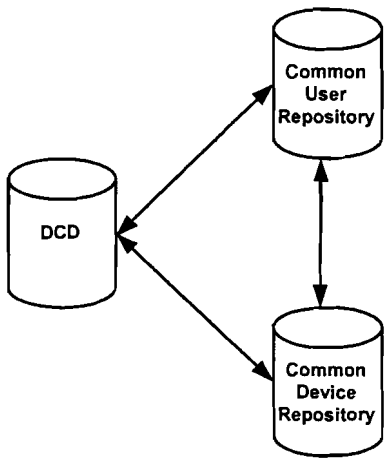


Figure 3-13: *Logical representation of the DCD*

3.7.3.2 Implementation of the interfaces

Start Scan
The function is implemented in the SIM initialisation procedure. The SIM initialisation procedure is run after SIM activation, as can be found in [3GP01]. Since Update Device Capabilities is implemented in the SIM initialisation procedure, this function is executed immediately after SIM activation and thus implicitly after every device swap as well.

Device Capabilities Scan
Provide Local Information (IMEI)
Input: Proactive SIM command tag, length, command details, device identities
Output: IMEI

Update DIDS
Select EF_{DIDS} Input: file ID
Output: file ID, file size, access conditions, invalidated / not invalidated indicator, structure of EF and length of the records in case of linear fixed structure or cyclic structure
Update Record Input: - mode, record number (absolute mode only) and the length of the record
- the data used for updating the record
Output: none

Compare stored ID with current ID
Some comparison must exist or otherwise be written.

Update DCD
To update the DCD the current DC has to be transported from the mobile device to the network. At the moment three possibilities exist: GSM, SMS and GPRS. Since GSM isn't exactly designed to send data over the network and it would mean an extra load to the network as well, the GSM option is put aside. Nevertheless, the two remaining possibilities for the implementation of the update of the DCD are valid.

The first implementation is to send the UAProf URL to the DCD using SMS. The architecture for this solution is depicted in Figure 3-14.

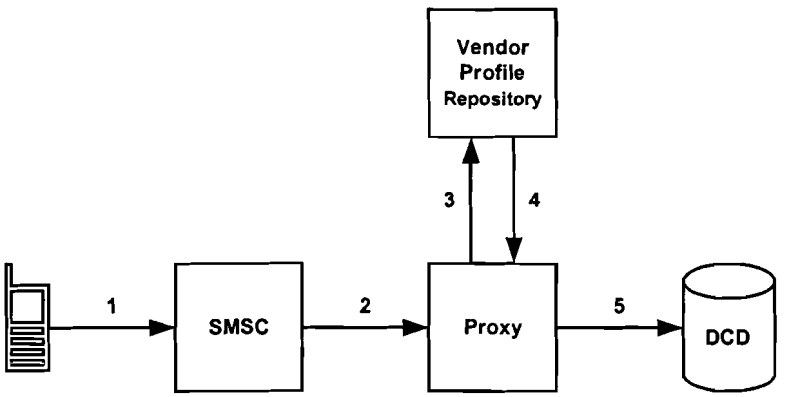


Figure 3-14: *DCD update using SMS*

The URL that is fetched from the mobile device leads to the vendor profile repository in the network. This URL is sent by the application in a SMS to the SMSC by using the SS7 protocol, see Figure 3-14 (1). The SMSC sends it to a proxy server (2), the SMSC supports standardised protocols as TCP/IP to communicate to the proxy server. The proxy server uses the URL to retrieve the profile of the mobile device from the vendor profile repository using the HTTP_GET command (3), (4). Finally, the proxy sends the profile information of the mobile device to the DCD to update it (5).

Read Record EF_{DIDS}
Send Short Message

The second implementation of the update of the DCD is using IP. In this case the URL that leads to the UAProf information of the mobile device is sent over a IP connection as depicted in Figure 3-15.

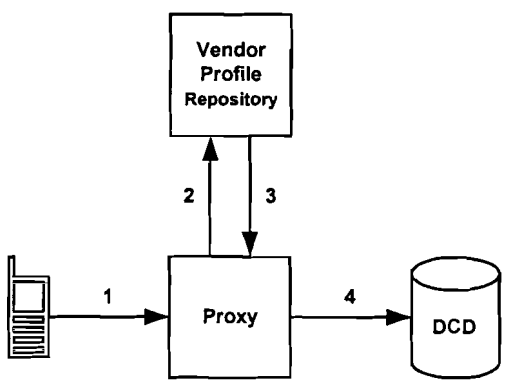


Figure 3-15: *DCD update using IP*

The architecture is then simplified compared with the SMS solution. This time, the UAProf URL can be embedded in the HTTP header and send to the proxy (1). Using the URL, the proxy retrieves the UAProf information from the vendor profile repository using HTTP_GET (2), (3) and stores it in the DCD (4).

Open Channel	Input:	Proactive SIM command tag, length, command details, device identities, address, bearer description, buffer size
Read Record EF_{DIDS}		
Send Data	Input:	Proactive SIM command tag, length, command details, device identities, Channel data length
Close Channel	Input:	Proactive SIM command tag, length, command details, device identities

Since SMS roaming is implemented on a much wider basis then GPRS roaming, at this point in time it might be more wise to choose for the SMS solution than the IP solution.

Some SAT commands in bytes of the above commands can be found in Appendix D.

3.7.4 JavaCard implementation

The solution has not been implemented using JavaCard due to time restrictions. JavaCard presents the possibility implement the solution in Java applets, which is easier to implement then using SAT commands. It is still possible to use SAT commands while using Java. However, at the moment, not all SIMs are of the JavaCard type, but is estimated that in the future most cards will be JavaCards and the decision whether to use the JavaCard can be re-evaluated.

3.8 Workarounds

Other mechanisms can be implemented that yield an update of the device capabilities, but are not solutions in the strict way as stated in section 1.2.

3.8.1 Updating the IMEI

Vodafone Spain has developed a user's handset database that stores device-specific information [MaL02a], [MaL02b]. This LDAP database is divided into two separate branches, see Figure 3-16. The first contains the user devices. In this part the MSISDN/IMEI pair for each user is stored, it links the user to his mobile device. The second part of the database contains the device features, here each IMEI is linked to all its technical properties.

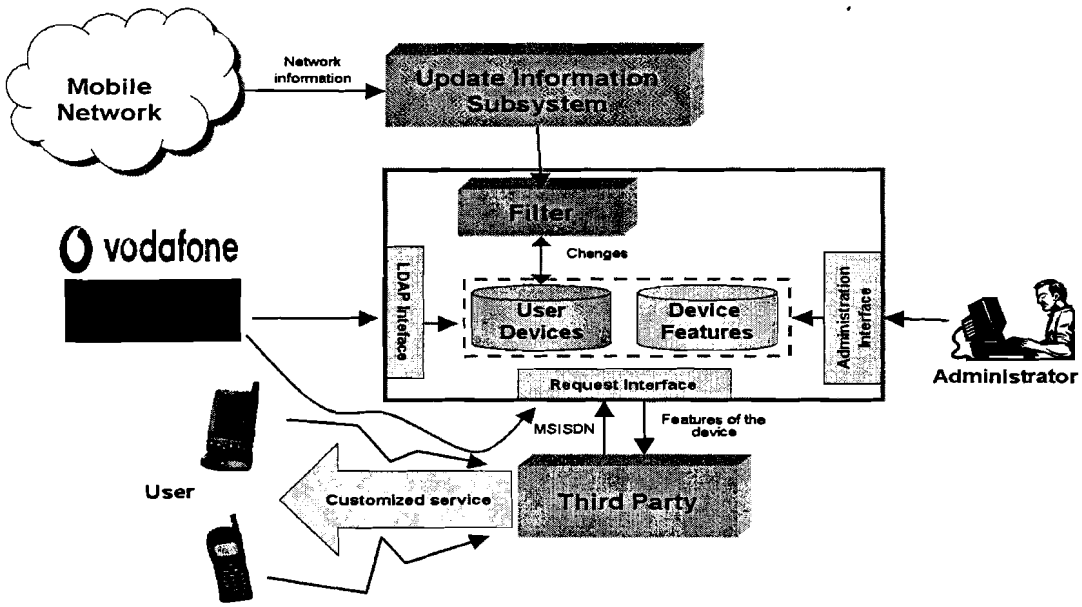


Figure 3-16: *User's handset database of Vodafone ES*

The main disadvantage of this solution is that the device features have to be updated by an administrator, this is not done automatically as proposed in our original solution by making use of the UAProf information embedded in the mobile device.

However, in this case it suffices to send the MSISDN and IMEI to the database and eliminates the need to send the UAProf information. The architecture is depicted in Figure 3-16. The difference is that now only the IMEI is send over the link. An interface will have to be created that translates the output of the SMSC to the input of the LDAP database.
In case of an IP link, the IMEI can be transported at once over the link and the SMSC can thus be excluded of the picture, which leads to figure.

3.8.2 Initiating a WAP Push session

A workaround to get hold of the user agent information can be constructed with the aid of the WAP Push mechanism. The WAP Push mechanism is described in section 3.2.1.3 and in [Mar01] and [WAP01a]...[WAP01f]. As opposed to the solution described in section 3.5, the solution is not SIM/terminal based, but rather network based. Because it has proven to be hard to retrieve the user agent information in an autonomous way (see Appendix E), the user agent information can be filtered from the WSP or HTTP headers as an WAP session is set up. Bear in mind that the trigger to the initiating of the WAP Push is still performed by the SAT application that detects the device swap. The WAP Push solution is described below.
The architecture for retrieving the user agent information after a device swap is depicted in Figure 3-17. The following steps can be distinguished. After detection of the device swap by the SAT application described in previous sections, first a SMS is sent to a dedicated number in the SMSC (step 1 in Figure 3-17). Because of the use of a device swap specific number, the SMSC is aware of the fact that it has to trigger the Push Initiator (2). The Push Initiator then submits a service loading push message [WAP01e] to the Push Proxy Gateway (PPG) making use of the Push Access Protocol (PAP) (3). The PPG delivers the service loading push message to the mobile device using the Push OTA protocol [WAP01d] (4). After the service loading message has been

received, the mobile device will contact the URL described in the service loading push message. This is executed by contacting the WAP gateway using the WSP GET command (5). Along with this WSP command, the user agent information is sent in the WSP header. Because this user agent information can be filtered out of the WSP header by the WAP gateway, it is important that the WAP gateway sends this information first to the DCD (6). In fact, this can be implemented by using the interface UpdateDCD() as described in section 3.5.4, which would have to be embedded in the WAP gateway. The trigger for sending the user agent information to the DCD can be the destination address that is specified by the URL. If this would be done after the WML return of the GET command, the user agent information might be filtered out. Naturally, if the user agent information is not filtered out by the WAP gateway, it is theoretically possible to implement the UpdateDCD() interface in the origin server instead of in the WAP gateway. The WAP gateway then translates the WSP GET request in a HTTP GET request and sends it to the origin server pointed out in the URL (7). After the information indicated in the URL of the service loading push message has been fetched from the origin server, it is returned as a WML page to the WAP gateway (8). The WAP gateway sends the fetched information to the mobile device, where it can be displayed to the user (9). The message might possibly indicate that the DCD in the network has been updated with the current DC.

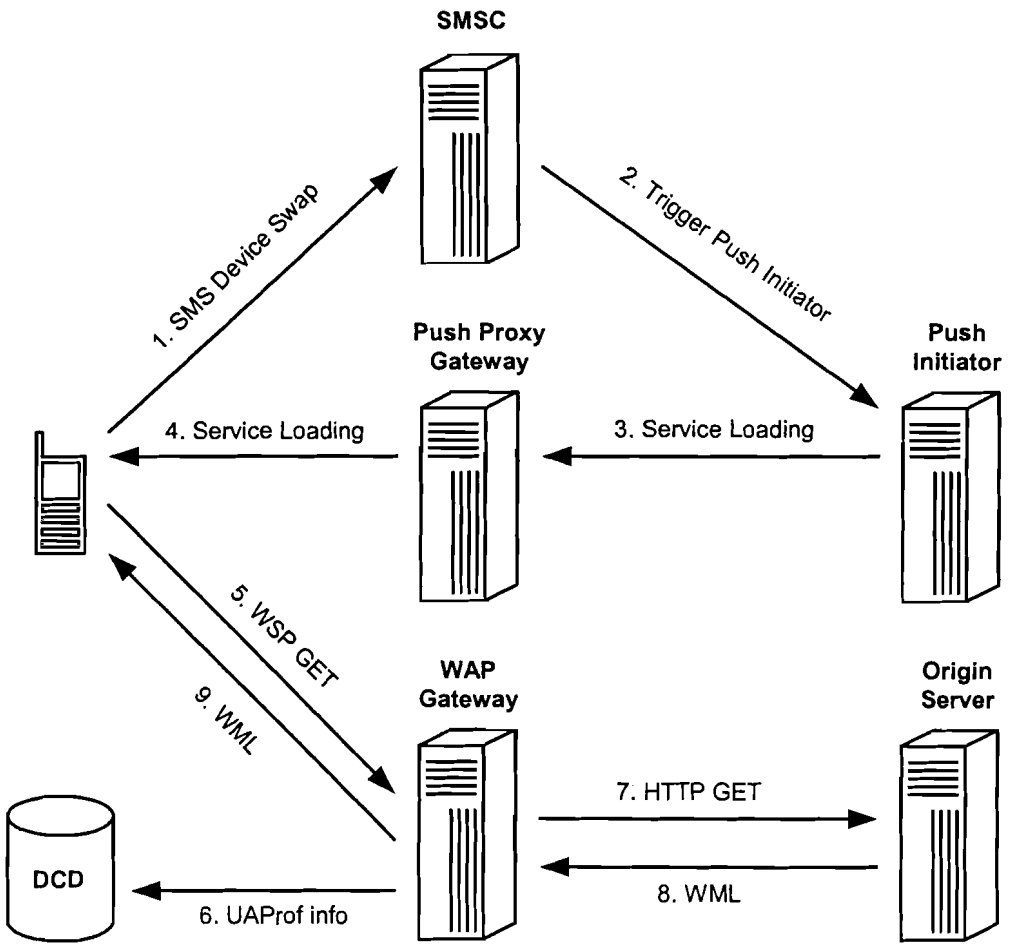


Figure 3-17: *WAP Push workaround*

Chapter 4
Software Download

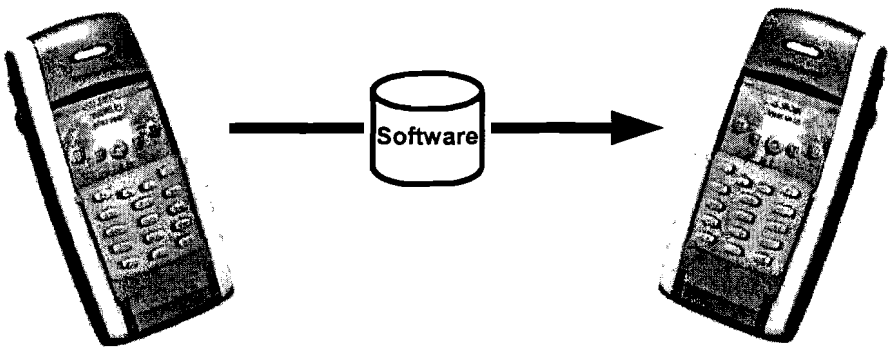


Figure 4-1: *Software download*

4.1 Problem statement

As the technology for mobile devices advances, the implemented software becomes more and more important. It is foreseen that in the future over-the-air (OTA) software download will enhance the capabilities and preferences without having to go to a store for a software update or having to replace a device. However, since it still will be possible to install software in a non-OTA manner, this will have to be taken into account as well. In either way, for making use of the software update, the network will have to be informed about it.

4.2 Existing technologies

4.2.1 OMA download

The OMA download specification [OMA02b] addresses requirements such as content discovery, authentication, delivery negotiation, content delivery and delivery confirmation. An interesting part of OMA download is that it uses a download descriptor. This is metadata about a media object and instructions to the download agent for how to download it. The object triggers the download agent in the client device. It describes the media object to be downloaded and allows the client device to decide if it has the capabilities to install and render/execute the media object.
The download descriptor itself is not a dynamic entity (it is just designed to enable accurate download of media objects), but it possibly could be used to trigger the real-time update mechanism of the DCD if relevant software is to be installed on the mobile device.

4.3 Conceptual architecture

Since the business requirements have not changed, the conceptual architecture as described as in section 3.3 is also applicable for Chapter 4 and repeated in Figure 4-2. The third parties can again be left out, because the main focus of this report is the real time provisioning of the device capabilities. Third parties will benefit from this feature, but will not play an active role in the process, they are the users of it.

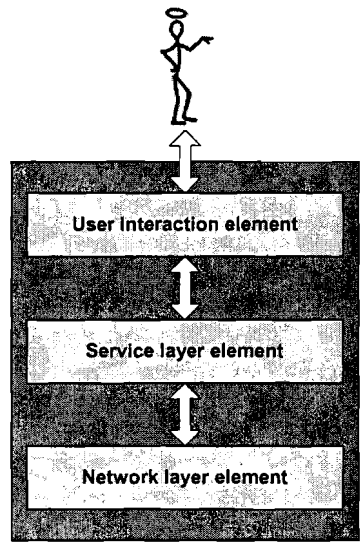


Figure 4-2: *Conceptual architecture software download*

4.4 Functional requirements

4.4.1 Use cases

This use case allows real time provisioning of device capabilities after a software download has taken place.

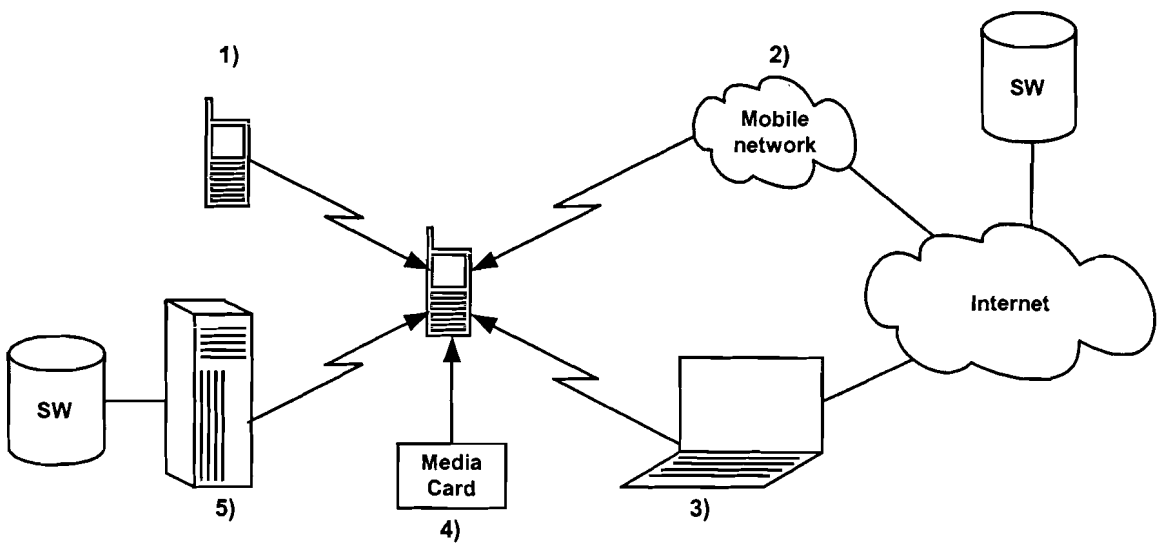


Figure 4-3: *Software download possibilities*

Figure 4-3 depicts several scenarios in which software can be downloaded to a mobile device. In scenario 1 the software is downloaded from another terminal, for instance via Bluetooth, infrared or a cable. Scenario 2 represents the case in which the software is downloaded from a commercial website to the mobile device via the mobile network. In scenario 3 the software is first downloaded from the server to a pc and subsequently a connection is established between the pc and the mobile device and the downloaded software is transferred to the mobile device. Scenario 3 is comparable to scenario 2, accept for the fact that the software is not downloaded via the mobile network, but via a pc that connected to the software server via the internet. In scenario 4 the software is provisioned to the terminal by a media card. Finally scenario 5 reflects when the software is downloaded via a pc that has direct access to the software to be downloaded, an example of this situation is the software that is delivered with the terminal on a CD-rom.
For all situations goes that the downloaded software has to possess the correct Digital Rights Management (DRM) settings [OMA02] in order to be able to use the downloaded software on the device.

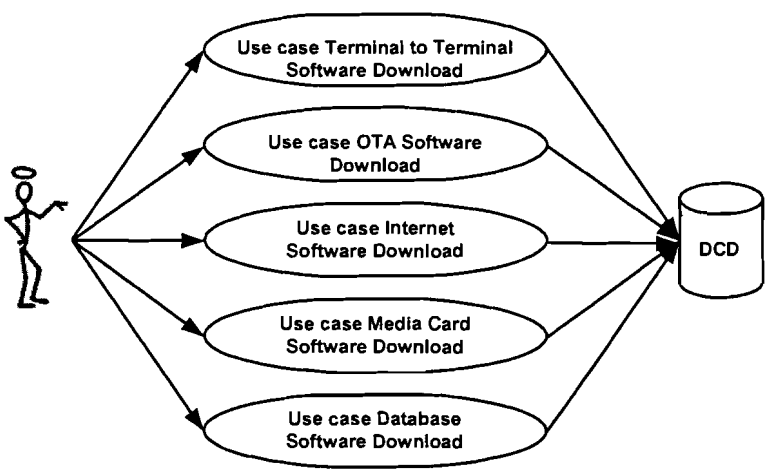


Figure 4-4: *Use cases Software Download*

All of these use cases will have to be considered to ensure a reliable update of the DCD, as depicted in Figure 4-4. Each use case will be described in the following sections.

4.4.1.1 Terminal to terminal software download

4.4.1.1.1 Use case description

This use case can be described as the situation where a friend of the user possesses software on his terminal that the user would like to have on his own terminal as well.
The software can be downloaded to the terminal of the user using for instance Bluetooth.

4.4.1.1.2 Flow of events terminal to terminal software download

1) The user discovers software for his mobile device that is located on another terminal
2) The donor device connects to the user device or the other way around
3) The software is transferred from the donor device to the user device
4) The software is stored on the user device
5) The user installs the downloaded software
6) The terminal updates the DCD
7) Data is offered to the mobile device according to the new profile in the DCD

4.4.1.2 OTA software download

4.4.1.2.1 Use case description

The use case is when the user downloads the desired software by pulling it via the mobile network. The user can download the file by clicking on an URL that leads via the mobile network to an origin server on the internet that has the desired software stored.

4.4.1.2.2 Flow of events OTA software download

1) The user browses on a mobile terminal to a WAP page that offers the possibility to download software
2) The user selects the download option
3) The download server checks the DCD for the current DC
4) The download server optimises the software for the terminal
5) The software is transferred to the mobile device
6) The software is stored on the mobile device
7) The user installs the downloaded software
8) The terminal updates the DCD
9) Data is offered to the mobile device according to the new profile in the DCD

4.4.1.3 Internet software download

4.4.1.3.1 Use case description

Scenario 3 is slightly different from scenario 2 in that not the mobile network is used to connect to the internet, but a pc or laptop. The desired software is first downloaded to the computer and then transferred to the terminal by using a short range communication system (e.g. infrared, Bluetooth or UWB).

4.4.1.3.2 Flow of events internet software download

1) The user browses on a computer to a web page that offers the possibility to download software for a mobile device
2) The user selects the download option that is best suited for his mobile device
3) The download server transfers the software to the computer of the user
4) The software is stored on the computer of the user
5) The user establishes a connection between the computer and his mobile device
6) The downloaded software is transferred from the computer of the user to his mobile device
7) The software is stored on the mobile device
8) The user installs the downloaded software
9) The terminal updates the DCD

10) Data is offered to the mobile device according to the new profile in the DCD

Alternative flow

2a) The user selects the download option and enters his mobile number
2b) The download server queries the DCD for the current DC
2c) The download server optimises the software for the terminal

4.4.1.4 Media card software download

4.4.1.4.1 Use case description

Software can also be stored or bought on a media card that can be inserted in the terminal. The terminal can read the software then directly from the media card, which is described by scenario 4. This way software can be sold in the same fashion as the games for game consoles.

4.4.1.4.2 Flow of events media card software download

1) The user inserts a media card in to his mobile device
2) The user selects the software on the media card to be installed
3) The user installs the software
4) The terminal updates the DCD
5) Data is offered to the mobile device according to the new profile in the DCD

Alternative flow

2a) The user selects software on the media card to be transferred to his mobile device
2b) The user transfers the software on the media card to the mobile device
2c) The software is stored on the mobile device

4.4.1.5 Database software download

4.4.1.5.1 Use case description

It is possible as well that software is sold on CD-rom that can be read by a computer or laptop, as in scenario 5. The computer connects wireless or via cable to the terminal and transfers the software to it. Another use case that can be thought for this situation is when the user wants his terminal to be flashed with new software at the retailer's shop. The computer of the retailer transfers the latest software to the terminal.

4.4.1.5.2 Flow of events database download

1) The user selects on a computer the software to be transferred to his mobile device
2) The user selects the download option that is best suited for his mobile device
3) The user establishes a connection between the computer and his mobile device
4) The software is transferred from the computer to the mobile terminal of the user
5) The software is stored on the mobile device
6) The user installs the downloaded software
7) The terminal updates the DCD
8) Data is offered to the mobile device according to the new profile in the DCD

Alternative flow

2a) The user selects the download option and enters his mobile number
2b) The computer queries the DCD for the current DC
2c) The computer optimises the software for the terminal

4.4.2 Analysis functional requirements of the use case

Studying the use cases of the previous section, it is apparent that the user has to be put forward as an actor. The user is the one that selects the software that has to be downloaded to the mobile device. The mobile device itself is of course a component that cannot be let out. The ultimate goal is to transfer the capabilities of the device real time to the DCD in the network. This DCD is considered as a given parameter, so it is a passive actor.

If one takes a closer look at the flow of events presented in section 4.4.1, it catches the eye that the last three events of every flow are the same: the user installs the downloaded software, subsequently the terminal updates the DCD and finally data is offered to the mobile device according to the new profile in the DCD. Each flow of event can be split up in two parts. The first part takes care of the transfer of the selected software to the mobile device and the storage of it on the terminal. This part differs for the five described use cases. The second part enables the real time update of the DCD with the current Software Profile (SP). As mentioned before, the second part is the same for each use case and will now be analysed.

In order to be able to keep track of the changes in software, a mechanism will have to be developed that can identify software changes. A mechanism similar to the device scan mechanism used in the previous chapter can be implemented. The software scan mechanism detects software that is present on the terminal that is relevant for a correct operation of the terminal.

As with the solution for the device swap in section 3.4.3, it might be wise to introduce here a component that controls the software scan and the update of the DCD. This component is labelled SW Scan.
To avoid unnecessary load to the network, a database can be introduced that stores the last SP that is relevant for content delivery to the mobile device. This component will be called the Software Profile Storage (SPS). The current SP is then compared with the stored SP. If the two software profiles match, no update will have to be sent to the network and thus network load is diminished. If the two software profiles differ, an update to the network is of course inevitable and even desired.

It is possible that this SPS will be located on the same place of the DIDS of section 3.4.3, they might even be part of the same internal database that keeps track of the capabilities of the mobile device. As with the device swap, for the software download part of the general solution, a comparator has to be able to compare the old and new software versions on the SPS. If there is any difference between them, an update of the DCD will have to be carried out.

As for the solution of the software download problem the way the software is transferred to the terminal is rather irrelevant, the main part of the functional architecture is the same for all five use cases. Figure 4-4 can then be simplified and replaced by the following figure.

Figure 4-5: *Use case software download*

4.5 Functional Architecture

4.5.1 Sequence diagrams

For every flow of events described in section 4.4.1, a sequence diagram is presented in the following sections. Only the way of downloading differs in the sequence diagrams, the process of discovering the relevant software and the possible updating of databases is the same for every use case and is depicted in Figure 4-6.
After the software is installed, the mobile device initiates a software scan that has the purpose to determine whether the installed software on the mobile device has to be known by third parties in order to deliver optimised content for the mobile device. This software is defined as relevant software, examples for this relevant software are the operating system or the browser. Along with the knowledge of which software is used, it is also important to know which version of the software is used, for different software versions of the same software have different capabilities. So as the SP Scan is initiated, first the SP Scan queries whether the installed software is relevant software and determines the versions of the relevant software. The next step in the process is to update the Software Profile Storage (SPS) and the DCD.

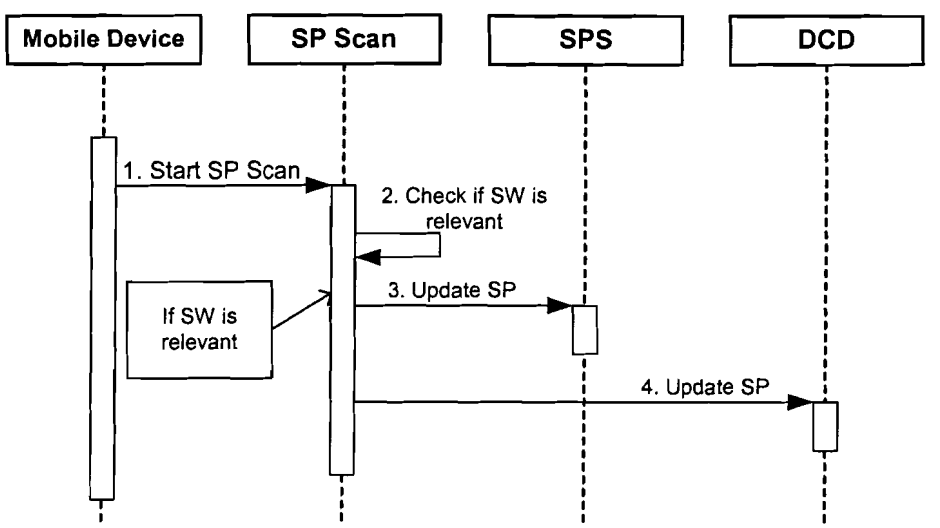


Figure 4-6: *Sequence diagram SW Scan*

4.5.1.1 Sequence diagram terminal to terminal software download

For terminal to terminal software download, the user needs to select the desired software on the donor mobile device. When a connection is established between the donor terminal and the user terminal, the desired software will be transferred to the user terminal. As soon as the software has been saved and executed on the mobile device of the user, this terminal will start the SW Scan process as described in the previous section.

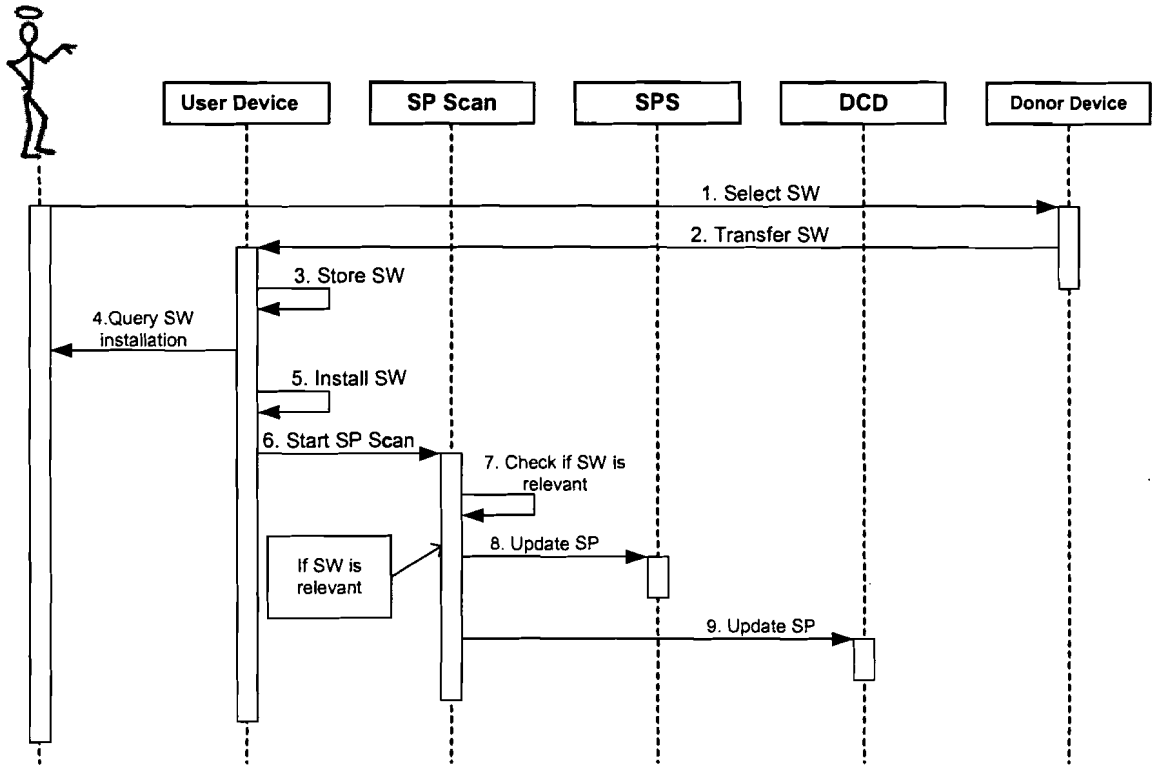


Figure 4-7: *Sequence diagram terminal to terminal software download*

4.5.1.2 Sequence diagram OTA software download

If the user wants to download software over the air (OTA), he first selects on his mobile device a download link to an origin server. The origin server in turn queries the DCD for the device capabilities of the terminal, so that it can return the software optimised for the specific terminal. As the selected software is optimised for the terminal, it is transferred to the mobile device. As soon as the software has been saved and executed on the destination mobile device, this terminal will start the SW Scan process as described in section 4.5.1.

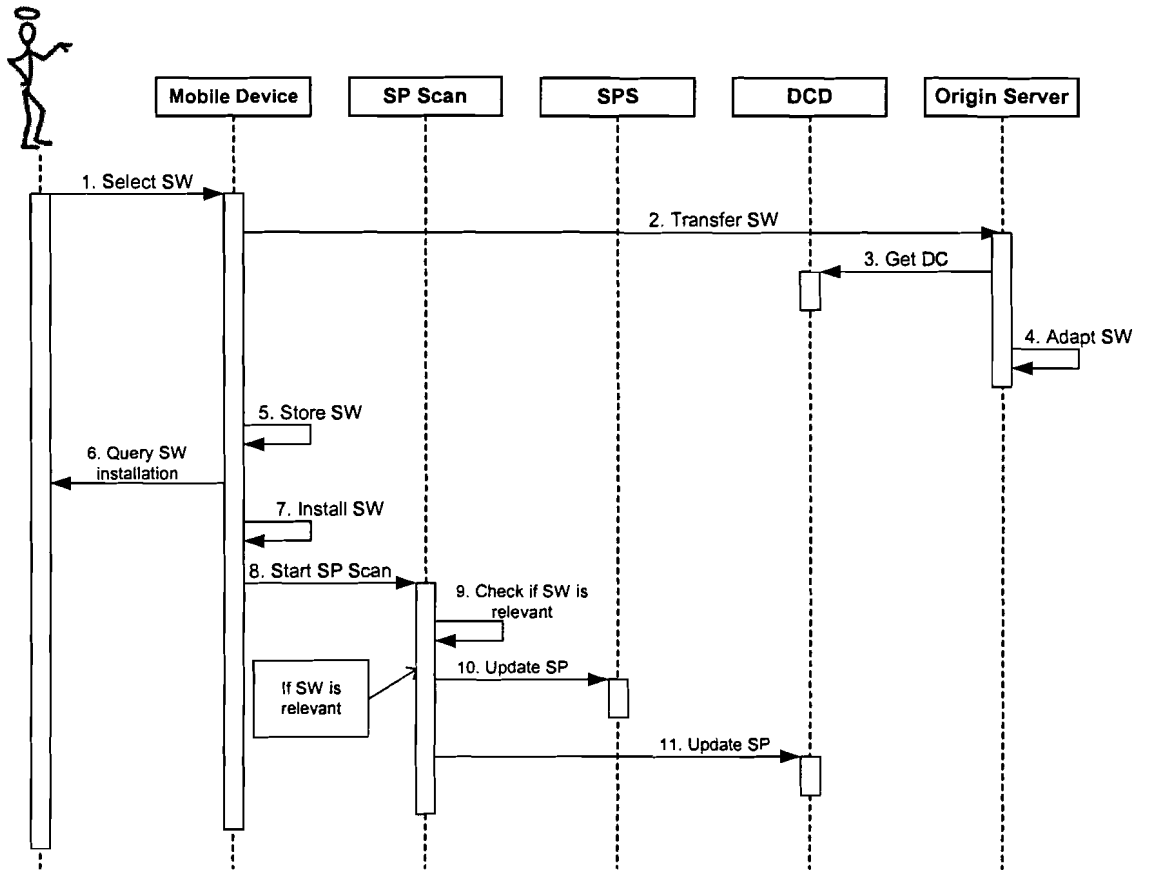


Figure 4-8: *Sequence diagram OTA software download*

4.5.1.3 Sequence diagram internet software download

Internet software download can be considered as a download in two stages, as can be seen in Figure 4-9. The first stage is when the user browses on the internet and selects with his pc out of a list of software versions for specific operating systems the correct software to be downloaded. The pc then contacts the download server where the desired software can be found and the software is downloaded to the pc.
The second stage is when the user selects the downloaded software on the pc to be transferred to the mobile device. The pc connects to the mobile device using a short range communication system or a data cable and transfers the software. The mobile device saves the software and prompts the user if it should install the software.

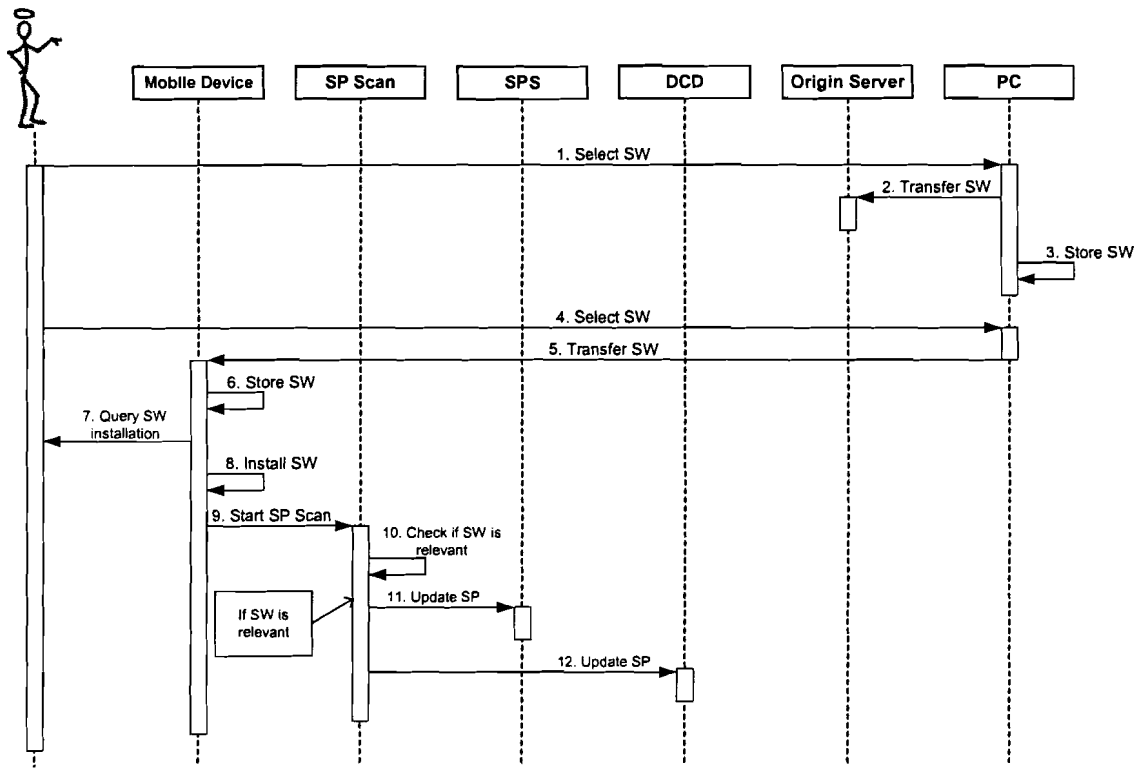


Figure 4-9: *Sequence diagram internet software download*

Instead of letting the user select the correct software version for his operating system, a slight adaptation can be made for the previous sequence diagram which leads to **Figure 4-10**. In this sequence diagram, the first stage of internet software download is changed. The download server now selects the correct software version for the capabilities for the mobile device by querying the DCD before sending the software back to the mobile device. With the knowledge of the capabilities of the mobile device, the download server can adapt the content, so the software is optimised for use on the specific mobile device. The rest of the actions are identical to the ones for normal internet software download.

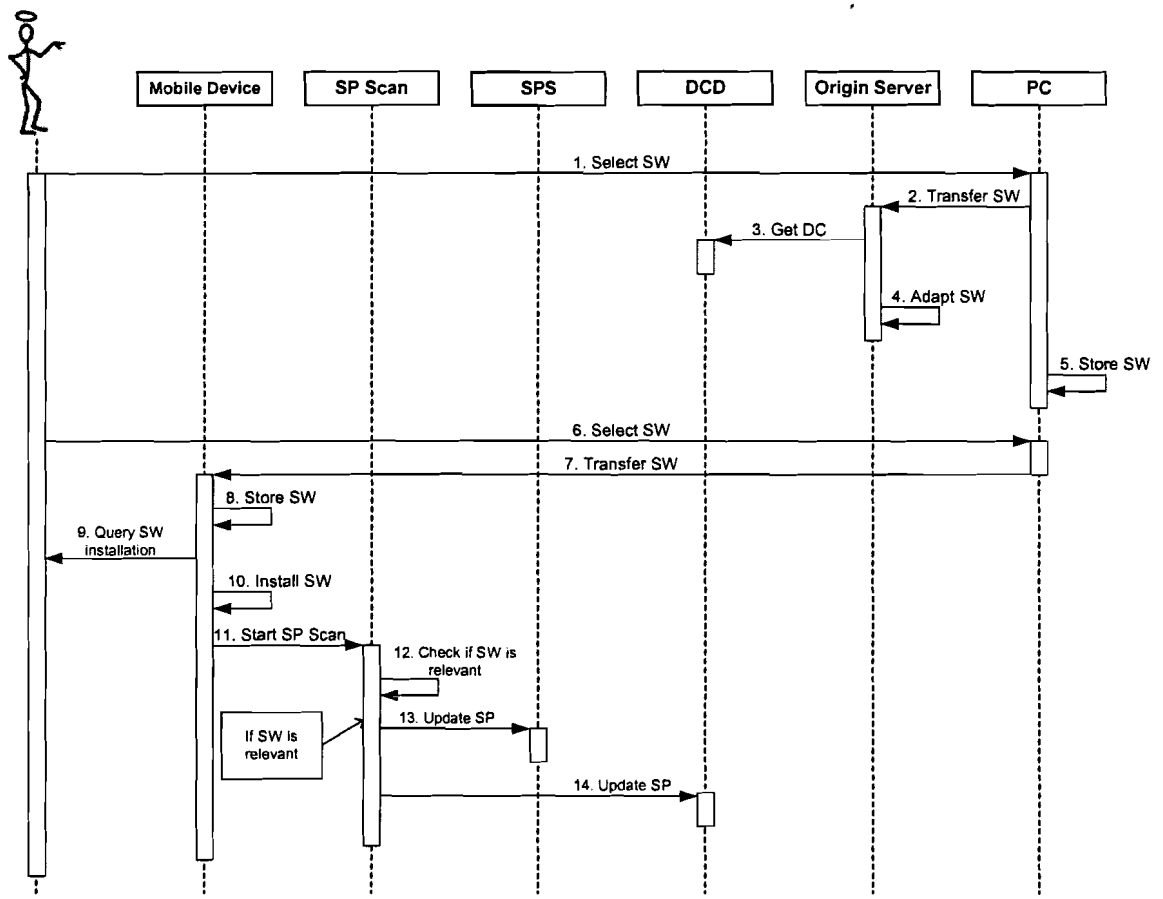


Figure 4-10: *Sequence diagram internet software download with content adaptation*

4.5.1.4 Sequence diagram media card software download

A very easy way to add new software to the mobile is to use a media card. This media card contains the software that the user wants and is inserted in to the mobile device. The mobile device can then access the software directly.

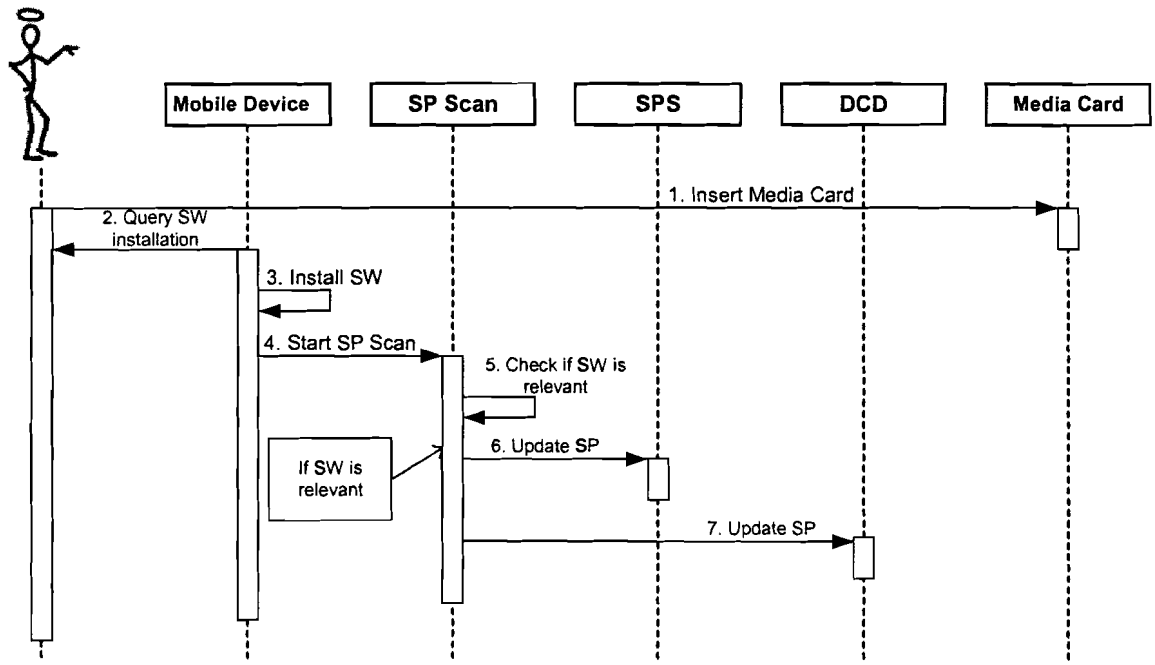


Figure 4-11: *Sequence diagram media card software download*

Another way for using the media card is to transfer the software that it contains to the mobile device, as depicted in the sequence diagram of Figure 4-12. First, the media card is inserted in to the mobile device. The user selects then via a viewer in the mobile device which software has to be transferred from the media card to the mobile device and subsequently this transfer takes place. As soon as the software is stored on the mobile device, the mobile device prompts the user whether it should install the software or not.

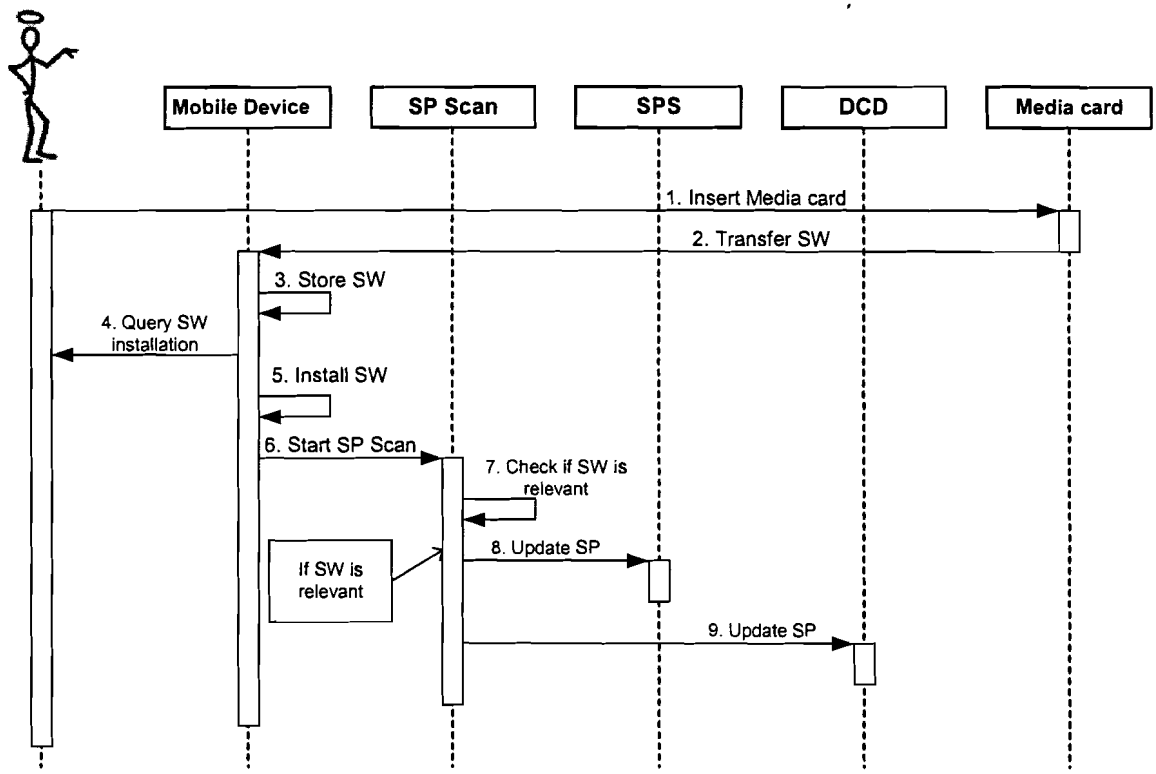


Figure 4-12: *Sequence diagram media card software download with software transfer*

4.5.1.5 Sequence diagram database software download

The database software download should be seen as a simplified version of the internet software download. As a matter of fact, only the second stage of the internet download is used for database software download. So first the user selects on the pc the database (for instance a CD-ROM) that contains the mobile software and then he selects the appropriate software for his operating system. Then the pc connects to the mobile device and transfers the software. As the software is saved, the user is prompted whether the software should be installed or not.

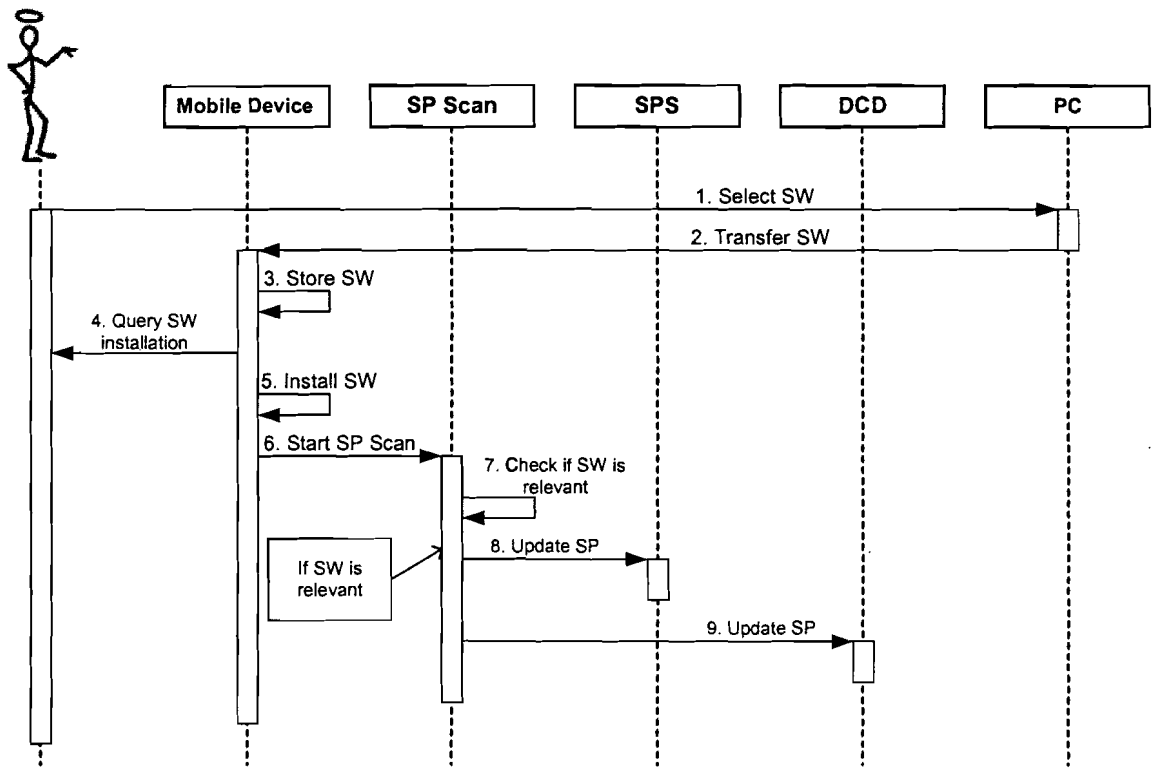


Figure 4-13: *Sequence diagram database software download*

As with the internet software download, an alternative flow exists for the database software download. Instead of selecting the appropriate operating system version, it is possible to let the pc query the DCD for the capabilities of the mobile device, in order to let the pc select the correct software versions. This way, the software is transferred from the pc to the mobile device only then when the DCD is queried and the content is optimised for the specific mobile terminal. Then the software is stored and installed if desired.

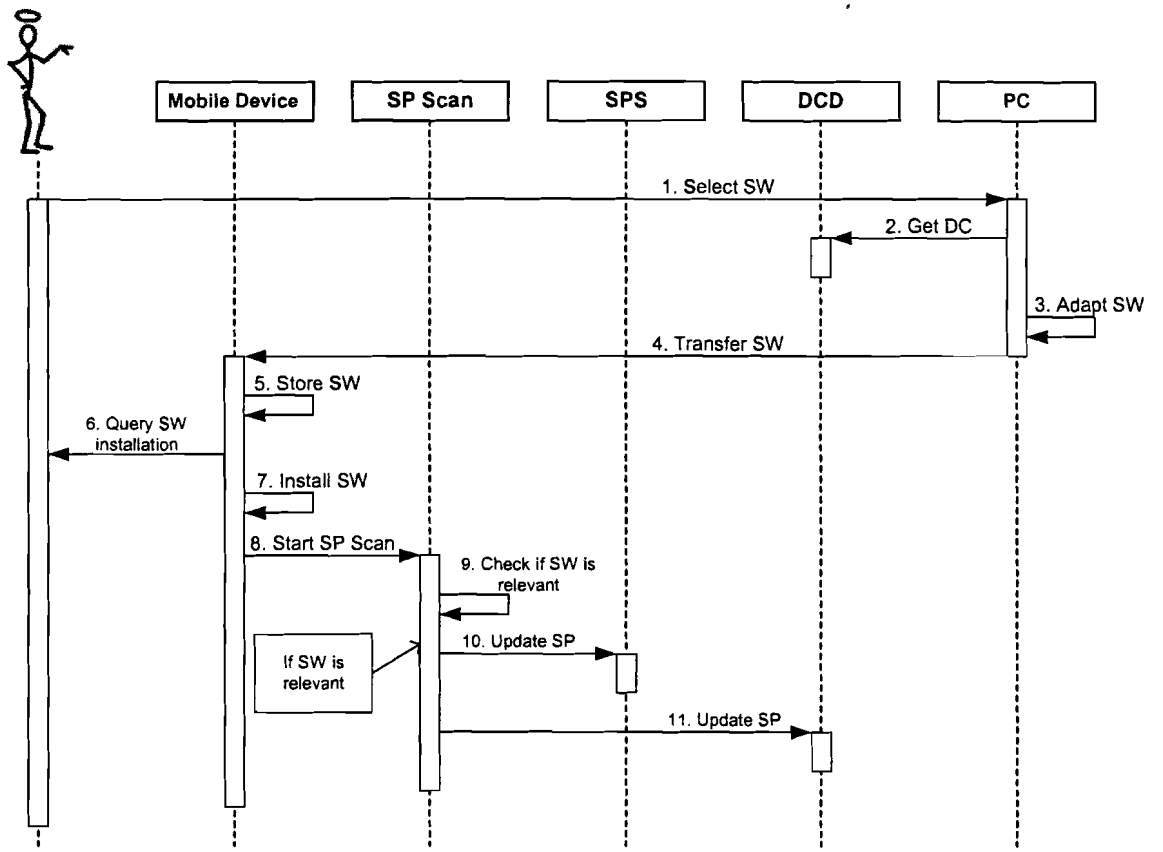


Figure 4-14: *Sequence diagram database software download with content adaptation*

4.5.2 Exception handling

4.5.2.1 Software deletion

4.5.2.1.1 Use case description

Software will not only be downloaded to the mobile device, but sometimes software will be deleted as well. The deletion of relevant software can lead to a change in capabilities of the terminal and hence this change in capabilities will have to be communicated to the DCD.

4.5.2.1.2 Flow of events software deletion

1) The user discovers software on his mobile device that he values redundant
2) The user deletes the redundant software
3) The terminal updates the DCD
4) Data is offered to the mobile device according to the new profile in the DCD

4.5.2.1.3 Sequence diagram software deletion

The user selects the software that he considers redundant on his mobile device, subsequently the selected software is deleted. After deletion, the standard procedure of the SP Scan takes place.

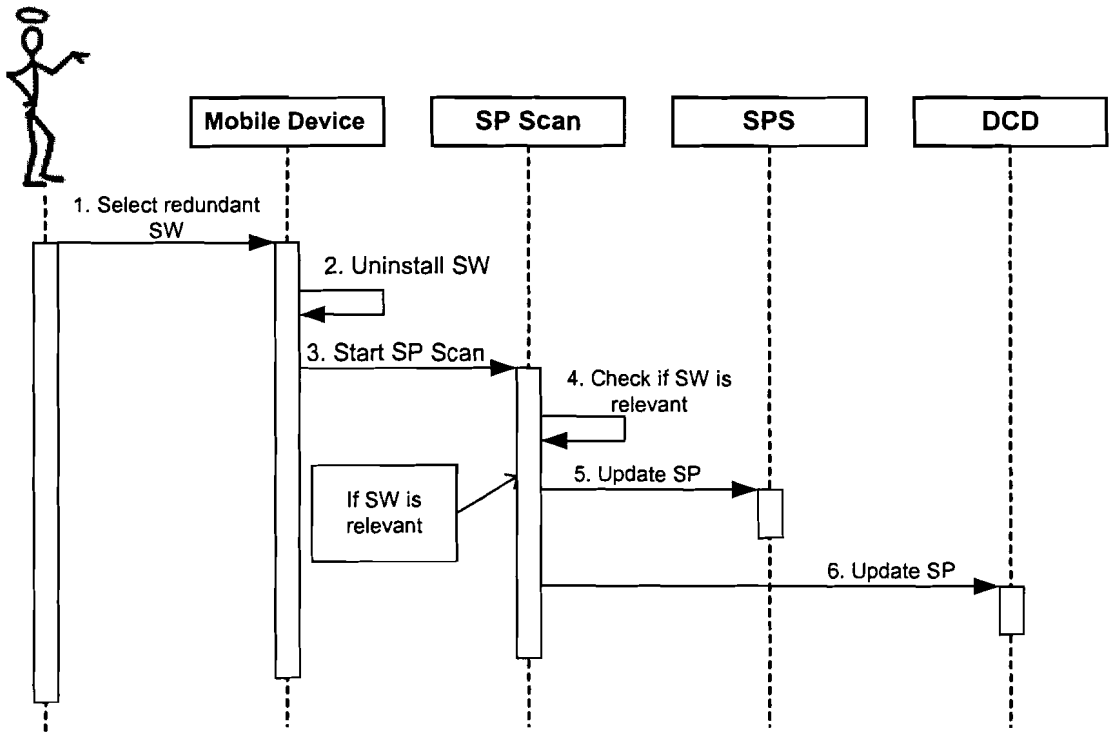


Figure 4-15: *Sequence diagram software deletion*

4.5.3 Architecture

As can be seen in Figure 4-16, the high level architecture of the software download consists of six building blocks: User, Mobile Device, SP Scan, SPS and DCD.

User
The user is the actor that uses the mobile device.

Mobile Device
The entity of which the SP has to be determined and dynamically updated.

SP Scan
As soon as new software is present on the mobile device, it starts a SP scan. One of the responsibilities of the SP scan is to discover which relevant software (the Software Profile SP) is present on the terminal. The result of this query is stored in the SPS. Other tasks of the SP Scan is to control the total process of discovering the SP and the possible updating of the DCD.

SPS
The Software Profile Storage contains the result (SP's) of at least the two latest SP Scans.

DCD
In the case that the DC is updated in the SPS, the DCD needs to be updated as well.

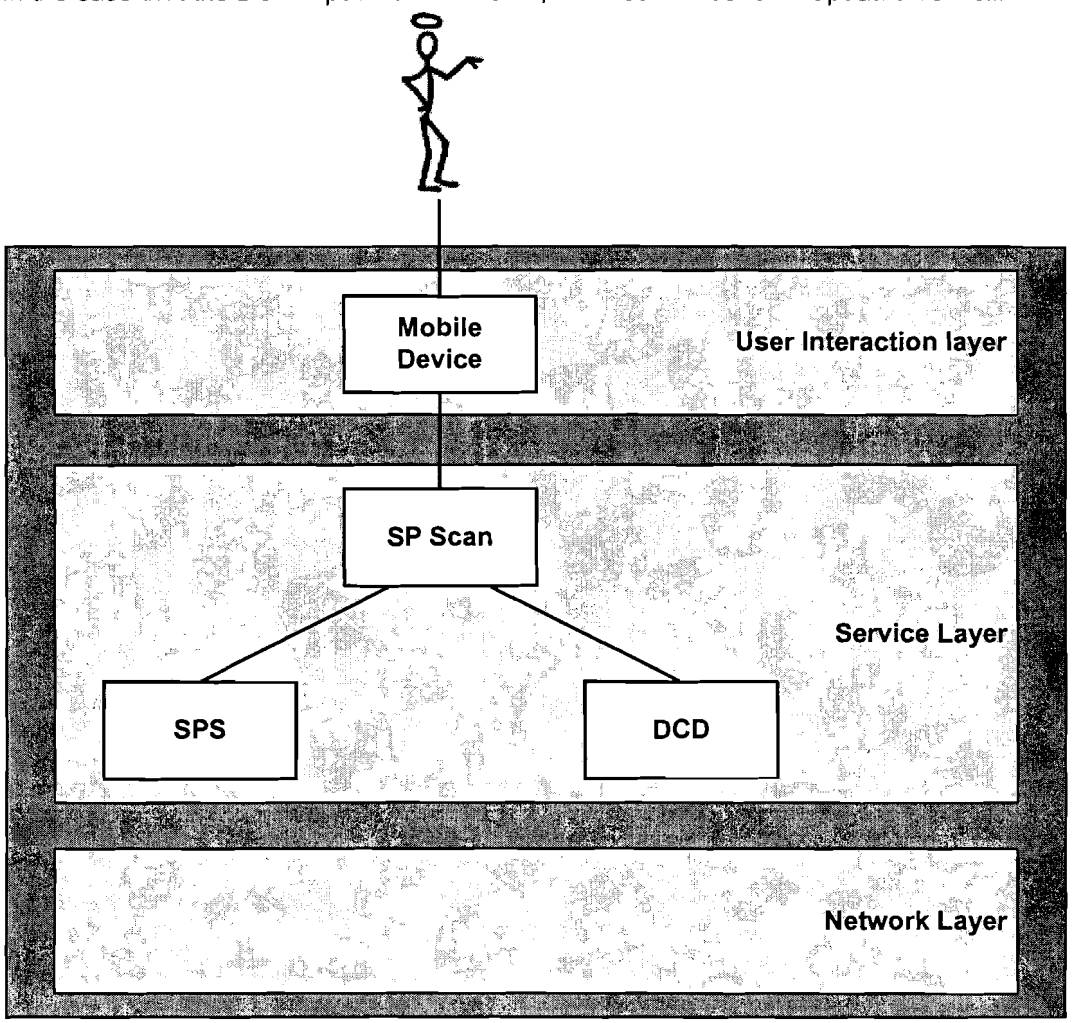


Figure 4-16: *Functional architecture software download*

Figure 4-16 shows as well how the conceptual architecture of the software download fits into the overall conceptual architecture. The mobile device belongs to the user interaction layer, whereas the SP scan, SPS and the DCD are integrated in the service layer. It might seem strange that the DCD is classified as a service layer element, because it is physically present in the network. But since the conceptual architecture just identifies the main elements this classification is valid, because the DCD in the network is used to enhance the user experience and thus belongs in the service layer.

4.5.4 Interface descriptions

User

QrySWIns()	Query the user if the stored software has to be installed
Parameters	DescrSW: The description of the software
Returns	OK or NotOK

Mobile device

UninstallSW()	Delete the redundant software
Parameters	RedSW: The description of the redundant software
	LocRedSW: The location of the redundant software
Returns	Acknowledgement when ready
InstallSW()	The software is installed on the mobile device
Parameters	
Returns	Acknowledgement when ready
SelectSW()	The software to be downloaded is selected
Parameters	
Returns	DescrSW: The description of the software
SelRedSW()	The redundant software is selected
Parameters	
Returns	RedSW: The description of the redundant SW
StoreSW()	The software is stored on the mobile device
Parameters	SW: The software
	LocSW: The location where the software has to be stored
Returns	Acknowledgement when ready
TransferSW()	The software is transferred to the mobile device
Parameters	SW: The software
Returns	Acknowledgement when ready

SP Scan

StartSPScan()	The SP scan is activated
Parameters	CurSP: The current SP of the mobile device
Returns	Acknowledgement when ready
CheckRelSW()	Check if the SW is relevant
Parameters	SW: The software
Returns	OK or NotOK

SPS

UpdateSPS()	The current SP is stored in the SPS
Parameters	CurSP: The current SP of the mobile device
Returns	OK or NotOK

DCD

GetDC()	Retrieve the DC from the DCD
Parameters	
Returns	The DC of the mobile device

UpdateDCD()	The current SP is stored in the DCD
Parameters	CurSP: The current SP of the mobile device
Returns	OK or NotOK

4.6 Non-functional requirements

Non-functional requirements specify aspects of the system other than its capacity to do things. Examples of non-functional requirements include those relating to performance, accessibility, usability, branding and visual style.
One non-functional requirement that is important for this report is the user-experience. Updating the DCD real time leads to the correct delivery of content to the mobile device, which enhances the user experience.

4.7 Technical Architecture

4.7.1 Location of the components

Three locations can be indicated for implementation of the components described in section 4.5.3, i.e. the SIM, the mobile device and the network. Each of these locations have their pro's and cons depending on the component concerned and are described in this section. For each component the selection of its location of implementation is explained, the results are to be found in Table 4-1.

Table 4-1: *Location of Software Update components*

	SIM	*Device*	*Network*
SP Scan	**Yes**	Yes	Yes
SPS	**Yes**	Yes	No
DCD	No	No	**Yes**

4.7.1.1 Location of the SP Scan

SIM pro's: No need to implement in device, device independence, SIM under Vodafone supervision
SIM cons: Not all SIMs are SIM Aplication Toolkit (SAT) enabled or capable of executing Over-The-Air (OTA) commands, SIM space might be scarce
Device pro's: SIM can be spared
Device cons: Some devices might not have the SP scan implemented

Network pro's: No need to update or replace SIMs or devices
Network cons: Capacity is used when no changes have taken place, heavy signalling load over the air

The ease of over-the-air (OTA) implementation on the SIM outweighs the fact that not all SIMs are OTA enabled. It is for sure that implementation on the SIM is easier to introduce than to implement this function in the device. If the function were to be implemented on the device, only the new devices would be able to effectuate a device scan, so a satisfying penetration of this function would span many years.
Implementation obviously would be even more easy, but as the network would be loaded every time a SP Scan would be effectuated, the signalling load would become disproportionate.

4.7.1.2 Location of the SPS

SIM pro's: No need to implement in device, easy interaction if SP scan is implemented on SIM as well, DIDS is implemented on the SIM as well and could share the same location
SIM cons: SIM space might be scarce, when SIM is placed in new device, the SP stored on the SIM is of the old device and most certainly inaccurate

Device pro's: Plenty of memory available, when SIM is placed in new device, the SP stored in the device is most likely more accurate than in the SIM case
Device cons: Some devices might not have an internal database allocated for SP storage

The SPS can thus be implemented either on the SIM or on the device. The implementation on the device might have several advantages like sufficient memory and a better accuracy in terms of device swap. But eventually implementation on the SIM is better when one considers the overall solution. In the previous chapter the DIDS is implemented on the SIM as well, so when a new profile has to be transferred to the DCD, it can be collected at the same place. The DIDS and the SPS might even be on the same memory location.

4.7.1.3 Location of the DCD

Since the assumption is made that the SP is stored in a database in the network, it is clear that the location of the DCD is in the network.

Chapter 5
Personal Area Networks

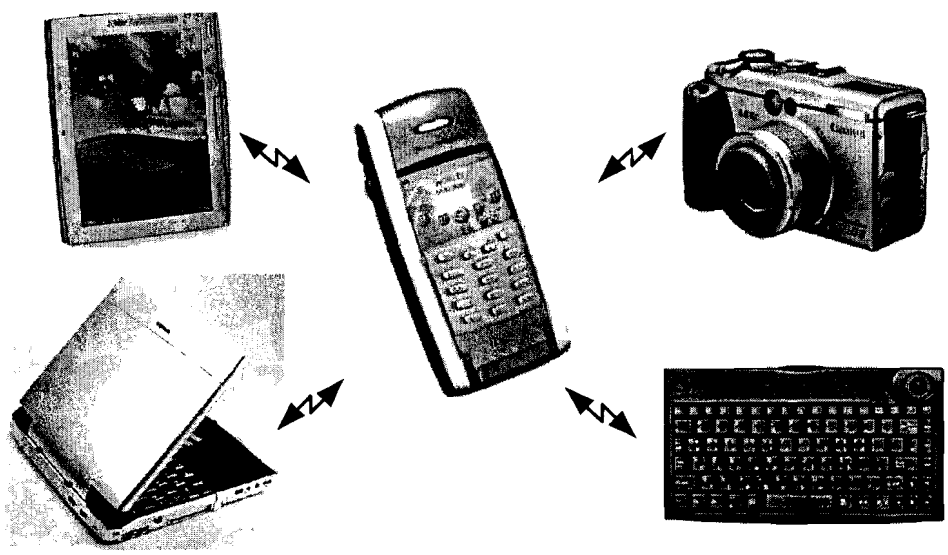


Figure 5-1: *Personal area network*

5.1 Problem statement

The mobile device is able to communicate with external devices, e.g. with digital cameras, external keyboards, external screens, laptops, etc., thus forming a Personal Area Network (PAN). Because of the extra features, the network can send advanced data to the mobile device. However, the network has to be informed over these external devices, otherwise it will send data formatted for just the mobile devices without the external devices.

5.2 Existing technologies

The best known technology for PANs is at the moment the Bluetooth technology [Blu01], [Kha02] though new techniques are emerging that might play a significant role [Sie00], like UWB based PANs [War02]. However, the existing technologies focus on establishing and maintaining PANs, but do not pay significant attention to the real time exchange of the capabilities of the PAN. Therefore some mechanisms are presented in the following sections.

5.3 Conceptual Architecture

Since the business requirements have not changed, the conceptual architecture as described as in section 3.3 is also applicable for Chapter 5 and repeated in

Figure 5-2.The third parties can again be left out, because the main focus of this report is the real time provisioning of the device capabilities. Third parties will benefit from this feature, but will not play an active role in the process, they are the users of it.

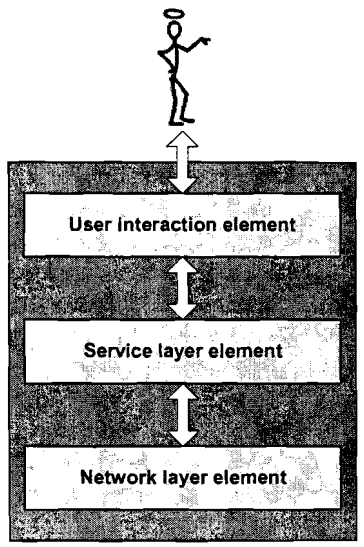


Figure 5-2: *Conceptual architecture external devices*

5.4 Functional requirements

5.4.1 Use cases

First of all, the term PAN will have to be defined in order to obtain the correct understanding of it. The typical range of a PAN is 10 meters, and is mostly used as some sort of cable replacement, providing the user more flexibility and ease of use. Since the goal of this report is to update the DCD in the network with the capabilities of the devices in use, the focus is on this subject.

Two different approaches can be envisaged to communicate the capabilities of the different devices of the PAN to the DCD. In the first approach, it can be imagined that every single device of the PAN sends its capabilities separately to the DCD. Since it is probable that not all devices of the PAN will have direct access to the DCD, it has to be assumed that they have to communicate their capabilities to the DCD by making use of a device that does have access to the DCD. The possibility exists that the DCD receives for a single PAN different values for a specific capability.

The second approach is to let one device act as a master device and the rest of the devices of the PAN as slave devices. Each slave device sends its capabilities to the master device and the master device merges these capabilities with its own capabilities into the capabilities of the PAN

and then sends it to the DCD in the network. The network can then deliver content to the master, which will then route the content to the appropriate device in the PAN.

Due to time restrictions, a closer look will only be taken at the second approach, i.e. the master/slave concept.

First of all, let's take a look at the creation of a PAN. In Figure 5-3a six devices are depicted that have no relationship, i.e. there exists no PAN. As soon as two devices connect to each other (i.e. device C and F in Figure 5-3b), one of the two devices will have to become the master device and the other device will automatically become the slave device. In practice, for the mobile world this master device is most likely to be the mobile phone and the slave device some kind of accessory like a larger screen or a camera.
If the PAN already exists (so the master device has at least one slave device), the situation is more simple, because there already exists a master device. The new device just becomes an extra slave device, as depicted in Figure 5-3c, where device B is the extra slave device compared to Figure 5-3b. If all devices are connected to the master, we have the situation that is depicted in Figure 5-3d.

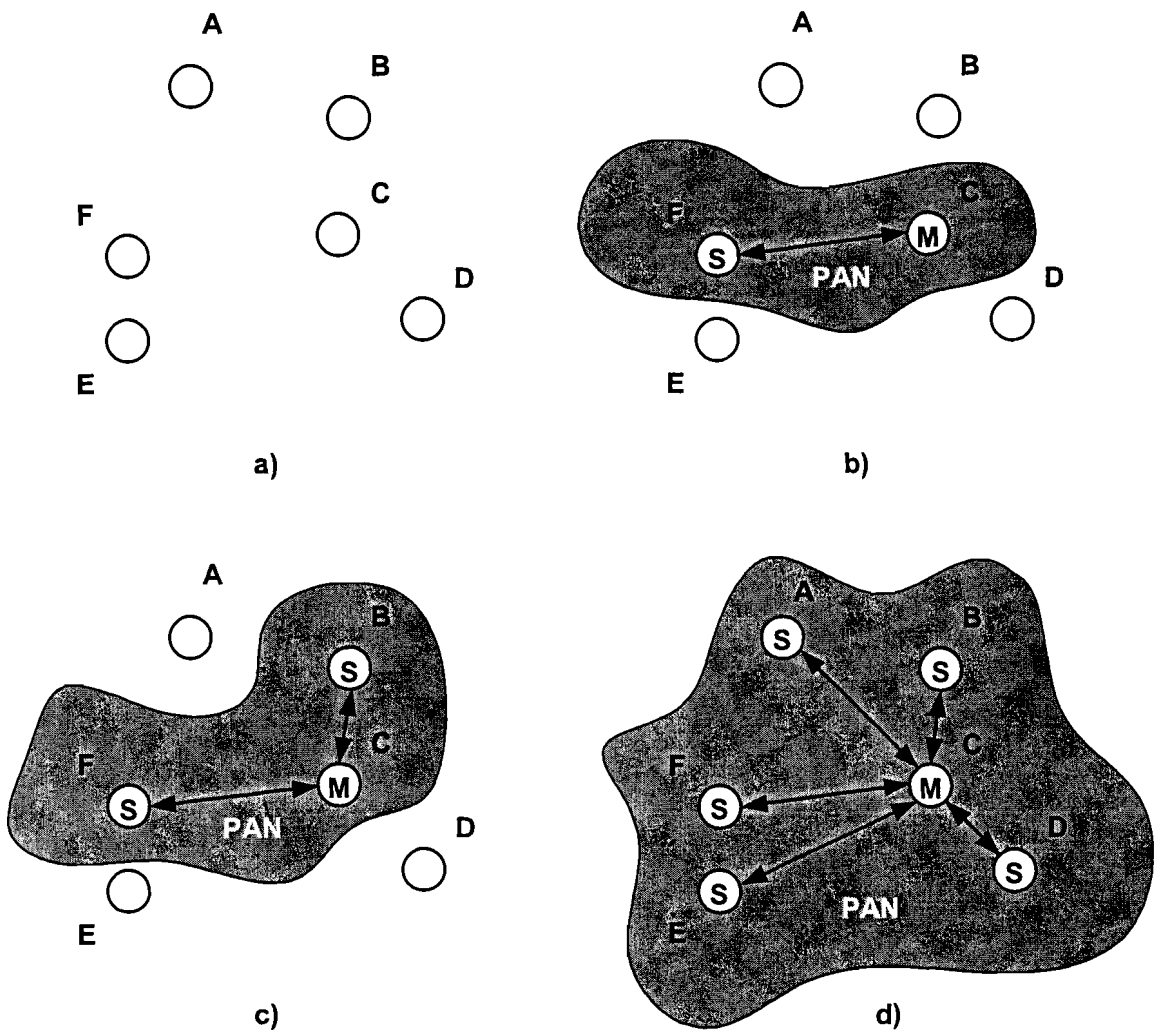


Figure 5-3: *Creation of a PAN*

Of course it is possible that a slave device will be disconnected from the PAN. If, for example, in Figure 5-4a device D is disconnected from the master device, this will lead to the new situation as depicted in Figure 5-4b. The PAN still exists, but has less components. Eventually, a case as depicted in Figure 5-4c can occur. Here, the PAN exists only of two components: the master device and the slave device. If the slave device E is disconnected from the master device C, both devices will lose their respective master or slave status and thus become "ordinary" devices as depicted in Figure 5-4d.

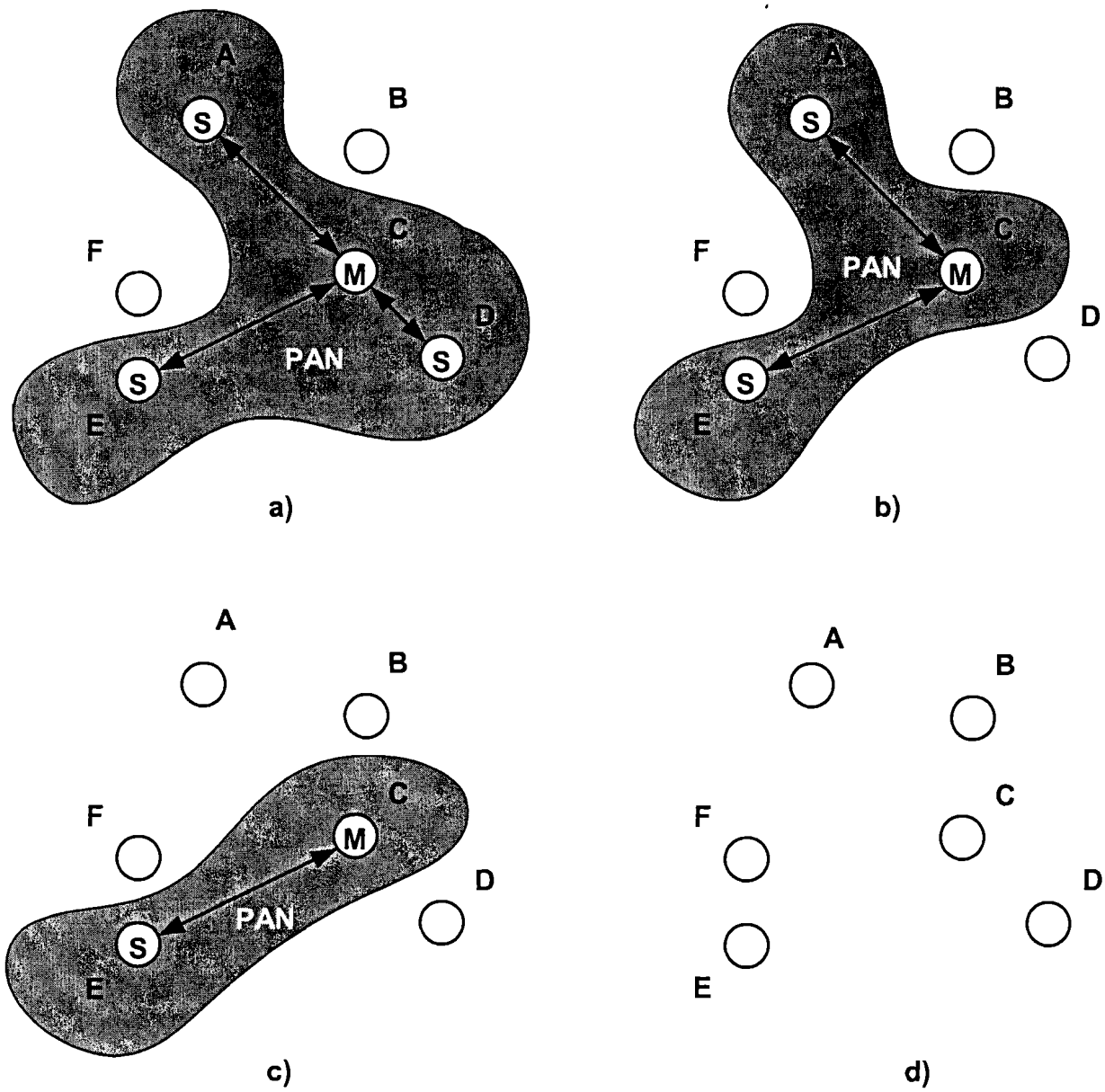


Figure 5-4: *Dissolution of a PAN*

From the information stated in this section one can distinguish four different use cases regarding PANs and the real time update of DC. These use cases are PAN creation, PAN extension, PAN reduction and PAN dissolution. The use cases are depicted in

Figure 5-5 and are described in the following subsections.

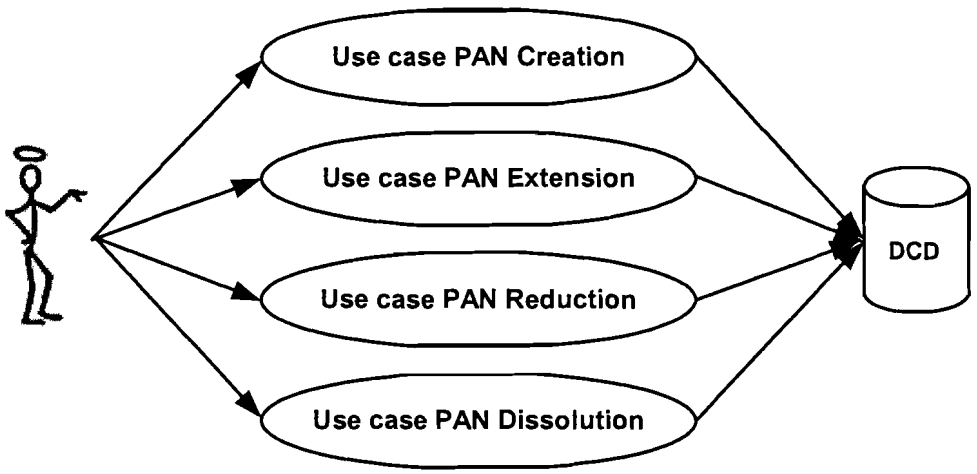


Figure 5-5: *Use cases PAN*

5.4.1.1 PAN creation

5.4.1.1.1 Use case description

Two mobile devices are connected to each other, i.e. a PAN is created that is able to share and communicate information. In this PAN, a master/slave relationship will have to be established.

5.4.1.1.2 Flow of events PAN creation

1) The user connects mobile device 2 to mobile device 1
2) A master/slave relation is established, mobile device 1 becomes the master, mobile device 2 becomes the slave
3) The master queries the slave device for its user agent information
4) The slave sends his user agent information to the master
5) The master merges his own user agent information with the slave user agent information into the PAN user agent information
6) The PAN user agent information is communicated to the network
7) Content is delivered to the PAN according to the new PAN user agent information

5.4.1.2 PAN extension

5.4.1.2.1 Use case description

An extra mobile device gets connected to an already existing PAN. Thus the PAN extends with a slave device that has to share its DC with the PAN.

5.4.1.2.2 Flow of events PAN extension

1) The user connects a new device to the master device of the PAN
2) The new device is designated as slave device
3) The master queries the new slave device for its user agent information
4) The slave device sends its user agent information to the master
5) The master merges the user agent information of the new slave device with the PAN user agent information
6) The PAN user agent information is communicated to the network
7) Content is delivered to the PAN according to the new PAN user agent information

5.4.1.3 PAN reduction

5.4.1.3.1 Use case description

A slave device is disconnected from the PAN. The network has to be made aware of the fact that the PAN in this new situation has a different DC than before.

5.4.1.3.2 Flow of events PAN reduction

1) The user disconnects a slave device from the master device
2) The master orders each slave device to send its user agent information to the master
3) The master merges his own user agent information with all slave devices user agent information into the PAN user agent information
4) The PAN user agent information is communicated to the network
5) Content is delivered to the PAN according to the new PAN user agent information

5.4.1.4 PAN dissolution

5.4.1.4.1 Use case description

The last slave device of a PAN gets disconnected from its master, thus dissolving the PAN and resetting the master device to a stand-alone mobile device.

5.4.1.4.2 Flow of events PAN dissolution

1) The user disconnects the last slave device from the master device
2) The master detects no more slave devices
3) The master device sends his user agent information to the PAN user agent information field
4) The PAN user agent information (equal to the master device user agent information) is communicated to the network
5) Content is delivered to the PAN according to the new PAN user agent information

5.4.2 Analysis functional requirements of the use cases

Studying the use cases of the previous section, it becomes clear that the user has to be put forward as an actor. The user selects whether a mobile device has to be connected or disconnected from the PAN, or whether to create or dissolve a PAN. The mobile dvices in the PAN (i.e. the master and the slave devices) of course cannot be let out. The ultimate goal is to transfer the capabilities of the PAN real-time to the DCD in the network. This DCD is assumed to be a given parameter, so it is a passive actor.

The main problem of this PAN issue is how to present the capabilities of all the distinct devices in the PAN as the one set of capabilities belonging to the PAN. Therefore, a special component is introduced: the merging component. This merging component is able to join the distinct components of the single devices into a PAN DC that can be considered as the capabilities of the whole PAN. Of course, in case of PAN reduction the merging component is able to uncouple the capabilities of the slave device to be disconnected from the capabilities of the whole PAN.

To avoid unnecessary load to the network, a database can be introduced that stores the PAN DC on the master device, this component will be called PAN capabilities.

5.5 Functional Architecture

5.5.1 Sequence diagrams

For every flow of events described in section 5.4 and its subsections, a sequence diagram is presented in the following sections, along with a description of the steps in the sequence diagrams.

5.5.1.1 Sequence diagram PAN creation

First of all the user selects on mobile device 2 that it has to connect to mobile device 1. Once this connection is established, a master/slave relationship will have to be set up. In this case, the mobile phone usually becomes the master device, but in the generic case, this is arbitrary. Once the master is set, i.e. in this case mobile device 1, the master device sets the other mobile device that is connected to it as a slave device. Subsequently, the master device orders the merging component to create configuration profile of the PAN, sending its own DC along with the order to the merging component. The merging component then queries the slave device for its DC and merges the two DCs into the PAN capabilities. This synthesis of DCs is used to update the internal database PAN Capabilities and the DCD in the network.

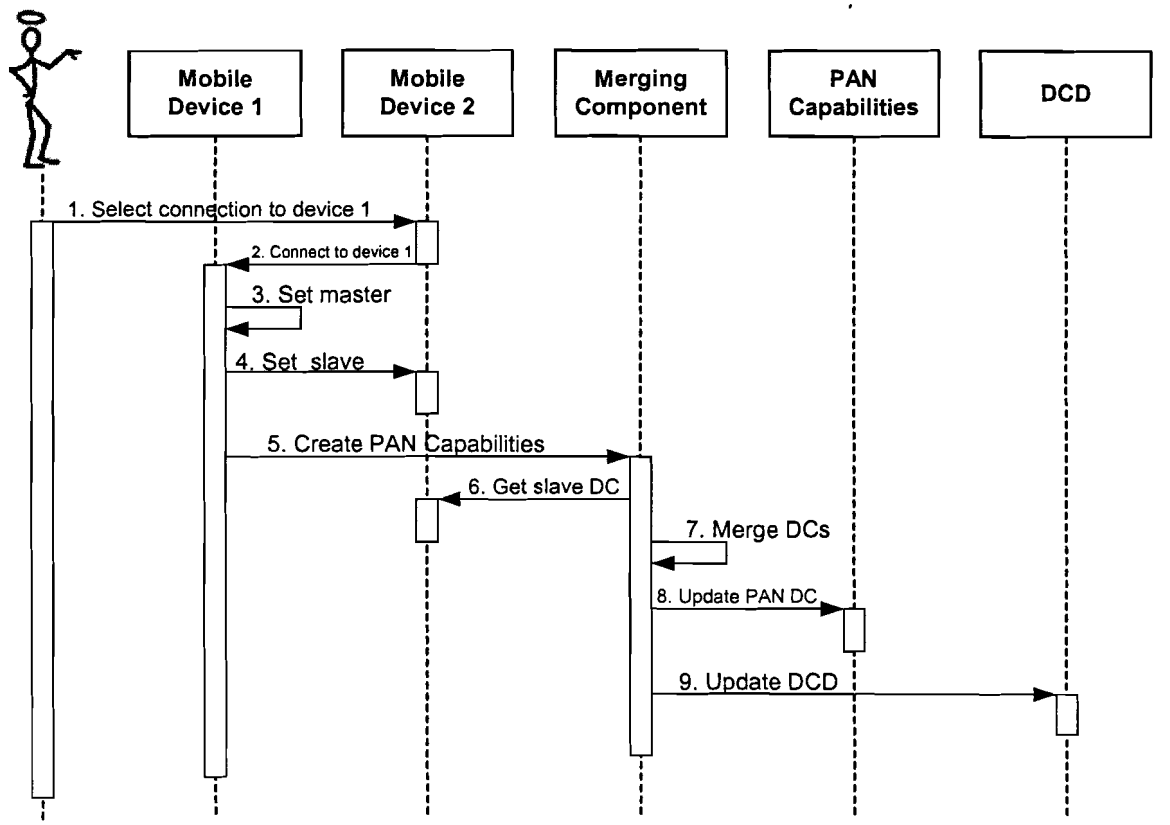


Figure 5-6: *Sequence Diagram PAN Creation*

5.5.1.2 Sequence Diagram PAN Extension

As the user connects a new device an already existing PAN, this new device is connected to the master. The master then sets this new device as a slave device and orders the merging component to create a new PAN configuration profile. Therefore the merging components retrieves the DC from the new slave device and the existing PAN capabilities to merge these into the new PAN capabilities, which eventually update the internal and external databases.

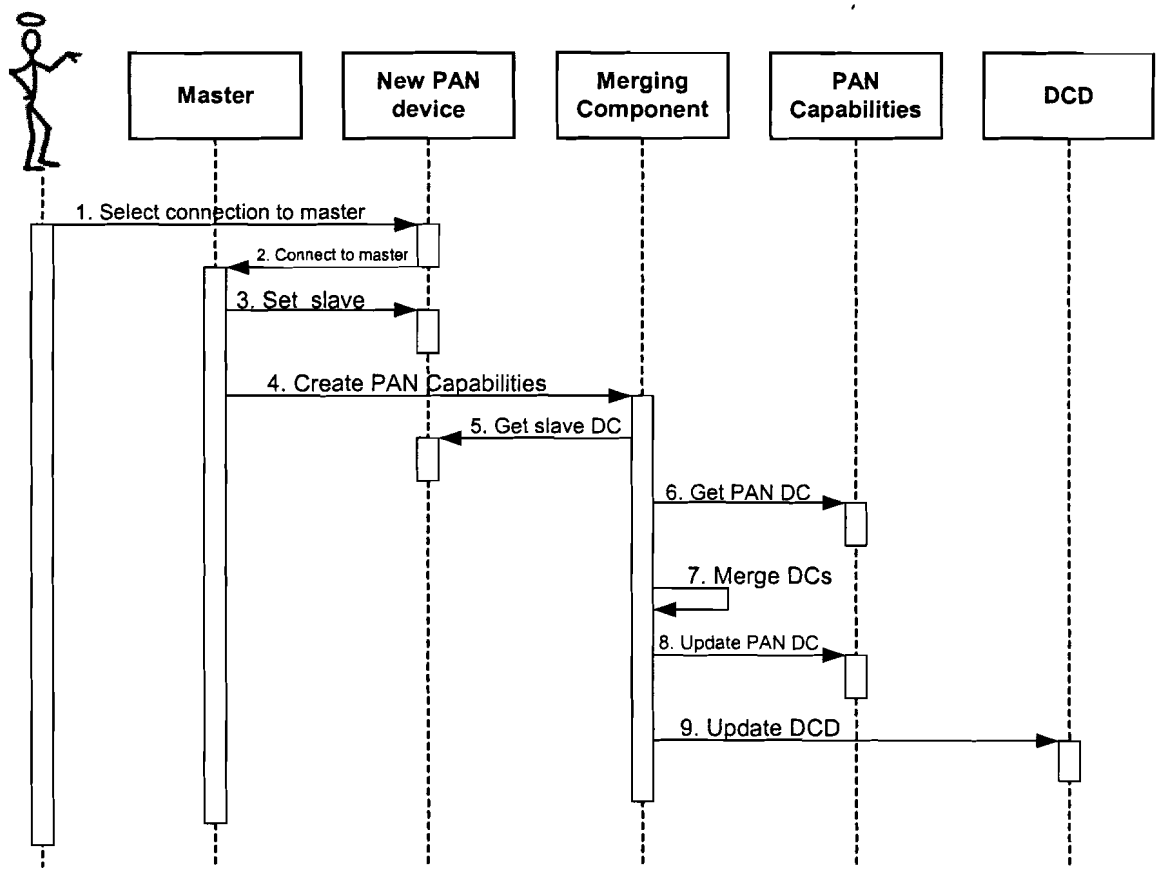


Figure 5-7: *Sequence Diagram PAN Extension*

5.5.1.3 Sequence Diagram PAN Reduction

If the user decides to disconnect a slave device from the master device, the latter disconnects the first device. The master then orders the merging component to detach the DC of the disconnected slave device from the PAN DC. The merging component therefore fetches the PAN DC from the PAN Capabilities database and then removes the DC of the disconnected slave device, thus resulting in an accurate description of the capabilities of the PAN. This PAN DC is then send to the internal and external databases.

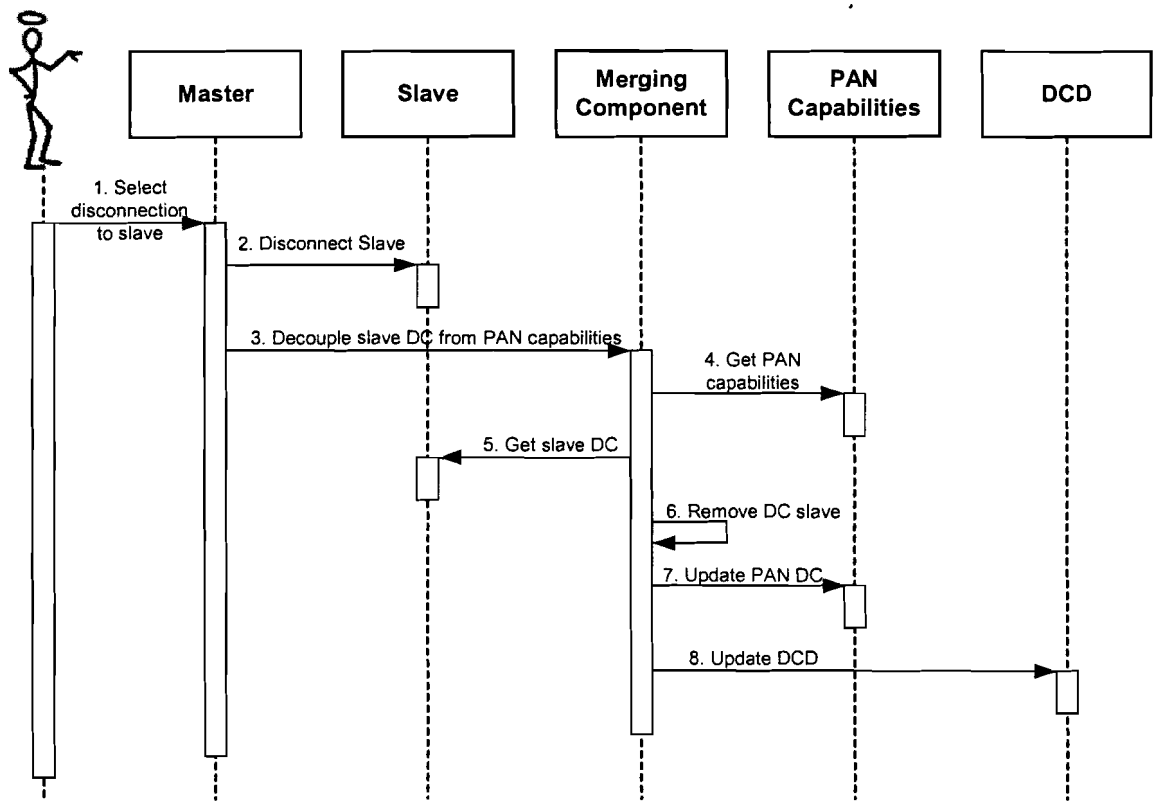


Figure 5-8: *Sequence Diagram PAN Reduction*

5.5.1.4 Sequence Diagram PAN Dissolution

The dissolution of the PAN is a special case of the reduction of the PAN. The PAN exists only out of a master and a single slave. For the reduction of this slave the sequence diagram of the previous can be used. The final result is that the PAN DC contains only the DC of the master device, thus the DC of the mobile device.

5.5.2 Architecture

As can be seen in Figure 5-9, the high level architecture of the real-time PAN capabilities update consists of the following building blocks: User, Master Device, Slave Device(s), Merging Component, PAN Capabilities and DCD.

User
The user is the actor that uses the mobile device.

Master Device
The entity of which the DC has to be determined and merged along with the DCs of the slave devices and then subsequently dynamically updated.

Slave Device n
The entity of which the DC has to be determined and merged along with the DCs of the master device and other slave devices and then subsequently dynamically updated.

Merging Component
This component is responsible for merging the separate DCs of all the devices of the PAN into the PAN DC, which then has to be updated to the internal database PAN Capabilities in the master device and the external database DCD in the network.

PAN Capabilities
The PAN Capabilities database is located in the master device and contains the result of the merging of the DCs of all the devices in the PAN.

DCD
The DCD database is located in the network and contains the result of the merging of the DCs of all the devices in the PAN.

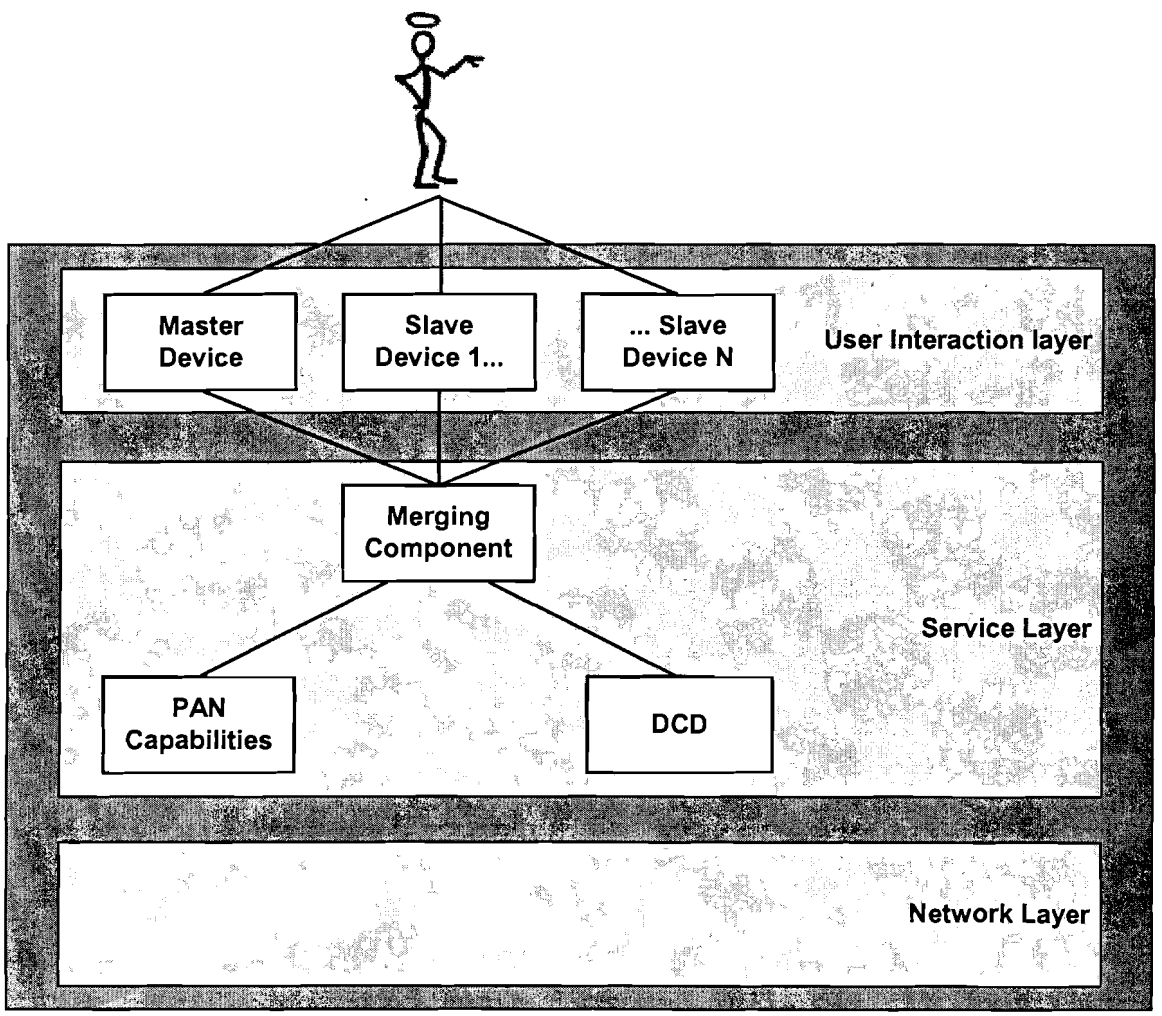


Figure 5-9: *Functional Architecture PAN*

Figure 5-9 shows as well how the conceptual architecture of the PAN capabilities update fits into the overall conceptual architecture. The master device and the slave devices belong to the user interaction layer, whereas the merging component, PAN Capabilities and the DCD are integrated in the service layer. It might seem strange that the DCD is classified as a service layer element, because it is physically present in the network. But since the conceptual architecture just identifies the main elements this classification is valid, because the DCD in the network is used to enhance the user experience and thus belongs in the service layer.

5.5.3 Interface descriptions

Mobile Devices	
SelConMaster()	The user requests a connection of a device to the master
Parameters	MasterID: The identity of the master in the PAN
Returns	Acknowledgement when ready
ConToMaster()	Connect the mobile device to the master
Parameters	SlaveID: The identity of the slave device in the PAN
Returns	Acknowledgement when ready
SetMaster()	Set the master device
Parameters	MasterID: The identity of the master device in the PAN
Returns	Acknowledgement when ready
SelDiscSlave()	The user requests the disconnection of a slave
Parameters	SlaveID: The identity of the slave in the PAN
Returns	Acknowledgement when ready
DiscSlave()	Disconnect the slave device from the master device
Parameters	SlaveID: The identity of the slave device in the PAN
	MasterID: The identity of the master device in the PAN
Returns	Acknowledgement when ready
SetSlave()	Assigns a mobile device its slave status
Parameters	SlaveID: The identity of the slave device in the PAN
	MasterID: The identity of the master device in the PAN
Returns	Acknowledgement when ready
GetSlaveDC()	Requests the DC of a slave
Parameters	SlaveID: The identity of the slave device in the PAN
Returns	SlaveDC: The DC of the slave device
Merging Component	
CreatePANCap()	Create the capabilities of the PAN
Parameters	MasterDC: The DC of the master device
	SlaveID: The identity of the slave device in the PAN
Returns	Acknowledgement when ready
MergeDCs()	Assembles the DCs of the devices in the network into the new PAN DC
Parameters	MasterDC: The DC of the master device
	SlaveDC: The DC of the slave device
	StoPANDC: The stored joint DC of the PAN
Returns	CurPANDC: The current joint DC of the PAN

DecSlaveDC()	Decouple the slave DC from the PAN DC
Parameters	SlaveID: The identity of the slave device in the PAN
Returns	Acknowledgement when ready
RemSlaveDC()	Remove the slave DC from the PAN DC
Parameters	StoPANDC: The stored joint DC of the PAN
	SlaveDC: The DC of the slave device
Returns	CurPANDC: The current joint DC of the PAN
PAN Capabilities	
GetPANDC()	Retrieve the stored joint DC of the PAN
Parameters	
Returns	StoPANDC: The storedjoint DC of the PAN
UpdatePANCap()	The current PAN DC is stored in PAN Capabilities
Parameters	CurPANDC: The current joint DC of the PAN
Returns	Acknowledgement when ready
DCD	
UpdateDCD()	The current PAN DC is stored in the DCD
Parameters	CurPANDC: The current joint DC of the PAN
Returns	Acknowledgement when ready

Chapter 6
Conclusions

In this report the conceptual and functional architecture for a real-time device capabilities management system is presented. In order to enable providers and third parties to tune the content for a specific mobile device to its specific capabilities, it is important to have an accurate description of these capabilities in a place that can easily be accessed by the above parties. In this report it is assumed that this place is a Device Capabilities Database (DCD) that is situated in the network, so that it is accessible in a autonomous way.

The real problem is to make sure that the information in this DCD is always accurate. Therefore, a mechanism had to be developed that updates the DCD as soon as changes in the hardware or software capabilities of the mobile device take place, i.e. at every change the DCD has to be updated real-time.

To tackle this general problem, the solution was divided in three stages of ascending complexity. First of all the hardware change consists of a simple device swap, the solution is then rather straightforward: first detect whether a device swap has taken place and then retrieve the capabilities of the new device and update the DCD. The next stage is a bit more complex, since the change is a software change, which is a bit more complicated to detect, since not all software change is relevant for the capabilities of a device. The most complex stage is where we have multiple mobile devices that form a personal area network (PAN). Since the capabilities of all the devices will have to be known by the network, a master/slave system is used where the master collects all the capabilties of the slave devices , then merges these and its own capabilities in a joint PAN capabilities profile and then communicates this to the network. Since the configuration of a PAN is dynamic, it is once more important that any changes in the configuration of the PAN are real-time communicated to the DCD in order to offer third parties the most accurate description of the capabilities of the PAN.

In the report several technologies have been described that offer more or less ways to gather information about the capabilities of a mobile device. One of the most advanced technologies is the User Agent Profile (UAProf). However, even the UAProf mechanism is rather static, since it needs a WAP session to communicate the capabilities of a device to the network, without the initiation of a WAP session, the new capabilities are not communicated, so the network assumes that the old capabilities are still valid.

The novelty of the mechanisms presented in this report is that the device capabilities (whether these concern hardware or software) are instantly communicated to the network as soon as changes take place in the capabilities of a mobile device. So the update is dynamic and real-time, thus enhancing the user experience, because the user always receives content according to the latest capabilities of his mobile device.

The main problem for implementing the solutions is to gain access to the device capabilities, for example the UAProf or user agent information. Since there seemed to be no straightforward way to access this information on the mobile device, a request for information was send out to the main terminal manufacturers with the question how to access this information. All manufacturers stated that at the moment it is impossible to access the user agent information in an autonomous way which is required for the solutions. However, the three presented mechanisms remain valid.

In the future, the technical architecture can be worked out, thus paving the way for implementation in the form of a trial to further test the conceptual and functional architectures presented in this report.

Appendices

Appendix A
References & Further Reading

[3GP01] "3rd Generation Partnership Project; Technical Specification Group Terminals; Specification of the Subscriber Identity Module – Mobile Equipment (SIM-ME) interface (Release 1999)", 3GPP TS 11.11 v8.6.0 (2001-12).

[3GP02] "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3GPP Generic User Profile – Architecture; Stage 2 (Release 6)", 3GPP TS 23.240 v1.0.0 (2003-03).

[3GP03] "3rd Generation Partnership Project; Specification of the SIM Application Toolkit for the Subscriber Identity Module – Mobile Equipment (SIM-ME) interface (Release 1999)", 3GPP TS 11.14 v8.10.0 (2002-03).

[Blu01] "Specification of the Bluetooth System, Core, Version 1.1", 22 February 2001, Bluetooth Special Interest Group.

[Fam00] D.Famolari and P.Agrawal, "Architecture and Performance of an Embedded IP Bluetooth Personal Area Network", ICPWC'00, IEEE 2000.

[Kaw03] Y.Kawamoto, V.Wong and V.Leung, "A Two-Phase Scatternet Formation Protocol for Bluetooth Wireless Personal Area Networks", IEEE 2003.

[Kha02] J.Khan, J.Wall and M.Rashid, "Bluetooth-Based Wireless Personal Area Network for Multimedia Communication", Proceedings of the First IEEE International Workshop on Electronic Design, Test and Applications (DELTA'02), IEEE 2002.

[MaL02a] D. Martín López and D. Del Ser Bartolomé, "User's Handset Database in Vodafone ES", Report Vodafone ES, April 2002.

[MaL02b] D. Martín López and D. Del Ser Bartolomé, "User's Handset Database Phase II", Report Vodafone ES, November 2002.

[Mar01] M.Martin and M. Van de Bergh, "WAP Push Pre-study", Report Vodafone NL, January 2001.

[OMA02a] "Digital Rights Management". Version 5 September 2002, Open Mobile Alliance, OMA-Download-DRM-v1_0.

[OMA02b] "Generic Content Download Over The Air Specification", Version 12 September 2002, Open Mobile Alliance OMA-Download-OTA-v1_0-20020912-a.

[OMA03] "Device Management Requirements V1.0", Draft Version 2003-04-01, Open Mobile Alliance OMA-REQ-DevMngmt-V1.0-20030401-D.

[Ran00] W.Rankl and W.Effing, "Smart Card Handbook, second edition", John Wiley and Sons, Ltd, Chichester, England, ISBN 0 471 98875 8.

[Ren99] A.Ren and G.Maguire, "A Smart Network with Active Services for Wireless Context-Aware Multimedia Communications", IEEE 1999.

[Sal01] T.Salonidis, P.Bhagwat, L.Tassiulas and R LaMaire, "Distributed Topology Construction of Bluetooth Personal Area Networks", IEEE 2001.

[Sie00] T.Siep, I.Gifford, R.Braley and R. Heile, "Paving the Way for Personal Area Network Standards: An Overview of the IEEE P802.15 Working Group for Wireless Personal Area Networks", IEEE Personal Communications, February 2000, pp.37-43.

[Syn01] "The Business case for Device Management", 13 November 2001, SyncML Initiative Ltd. White Paper.

[Syn02a] "SyncML Device Management Protocol", Version 1.1.1, 2002-10-02, SyncML Initiative.

[Syn02b] "SyncML Device Management Tree and Description", Version 1.1.1, 2002-10-02, SyncML Initiative.

[Syn02c] "SyncML Device Management Standardised Objects", Version 1.1.1, 2002-10-02, SyncML Initiative.

[Syn02d] "SyncML Notification Initiated Session", Version 1.1.1, 2002-10-02, SyncML Initiative.

[Syn02e] "SyncML Device Information DTD", Version 1.1.1, 2002-10-02, SyncML Initiative.

[UAP02] "User Agent Profile 1.1, Candidate Version 12 December 2002", Open Mobile Alliance, OMA-WAP-UAProf-v1_1-20021212-c.

[Wan02] W.Wang, "Bluetooth: A New Era of Connectivity", IEEE Microwave Magazine, September 2002, pp.38-42.

[WAP01a] "Wireless Application Protocol Architecture Specification", Version 12 July 2001, Wireless Application Protocol Forum, WAP-210-WAPArch-20010712.

[WAP01b] "Wireless Application Protocol Wireless Session Protocol Specification", approved version 5 July 2001, Wireless Application Protocol Forum, WAP-230-WSP-20010705-a.

[WAP01c] "WAP Push Architectural Overview", Version 03 July 2001, Wireless Application Protocol, WAP-250-PushArchOverview-20010703-a.

[WAP01d] "Push OTA Protocol", Version 25 April 2001, Wireless Application Protocol, WAP-235-PushOTA-20010425-a.

[WAP01e] "Service Loading", Version 31 July 2001, Wireless Application Protocol, WAP-168-ServiceLoad-20010731-a.

[WAP01f] "Service Indication", Version 31 July 2001, Wireless Application Protocol, WAP-167-ServiceInd-20010731-a.

[WAR02] J.Warnier, "Ultra Wideband: Emerging Spectral Efficiency", Report Vodafone Group R&D, RC_NL_2002_116, September 2002.

Appendix B
Abbreviations

3GPP	Third Generation Partnership Project
CC/PP	Composite Capabilities/Preference Profile
CC/PPex	Composite Capabilities/Preference Profile Exchange profile
CPI	Capabilities and Preference Information
DC	Device Configuration
DCD	Device Configuration Database
DDC	Dynamic Device Configuration
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
HTTP	Hyper Text Transfer Protocol
HTTPex	Hyper Text Transfer Protocol Extension Framework
MIME	Multipurpose Internet Mail Extensions
MMS	Multimedia Messaging Service
OMA	Open Mobile Alliance
OS	Origin Server
OTA	Over-The-Air
PAN	Personal Area Network
PAP	Push Access Protocol
PI	Push Initiator
PPG	Push Proxy Gateway
SIM	Subscriber Identity Module
SMS	Short Message Service
UAProf	User Agent Profile
WAP	Wireless Application Protocol
WSP	Wireless Session Protocol
SAT	SIM Application Toolkit
SP	Software Profile
SPS	Software Profile Storage
RFI	Request For Information

Appendix C
Definitions

WSP
Wireless Session Protocol. The Wireless Session Protocol provides the upper-level application layer of WAP with a consistent interface for two session services. The first is connection-mode service that operates above a transaction layer protocol, and the second is a connectionless service that operates above a secure or non-secure datagram transport service.

SAT
The SIM Application Toolkit is a set of commands which defines how the card should interact with the outside world and extends the communication protocol between the card and the handset. With SIM Application Toolkit, the card has a proactive role in the handset (this means that the SIM initiates commands independently of the handset and the network).

OTA
OTA (Over-The-Air) is a technology used to communicate with, download applications to, and manage a SIM card without being connected physically to the card . OTA enables a network operator to introduce new SIM services or to modify the contents of SIM cards in a rapid and cost-effective way.
OTA is based on client/server architecture, where at one end there is an operator back-end system (customer care, and at the other end there is a SIM card.

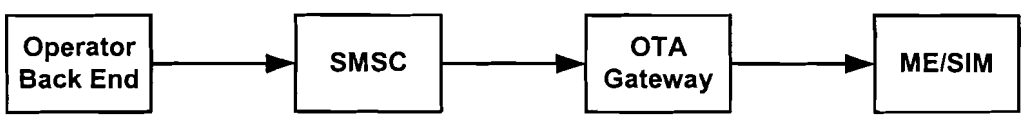


Figure C.1: *OTA SIM update*

The operator's back-end system sends service requests to an OTA Gateway which transforms the requests into SMS and sends them onto a Short Message Service Centre (SMSC) which transmits them to one or several SIM cards in the field, see figure Y.1.
Thus, OTA is a technology that updates and changes data in the SIM card without having to reissue it. Indeed, the end user can receive special messages from the operator, download or activate new services on his telephone, without having to return to a retail outlet.

In order to implement OTA technology, the following components are needed:

- A back end system to send requests
- An OTA Gateway to process the requests in an understandable format to the SIM card
- An SMSC to send requests through the wireless network
- A bearer to transport the request: today it is the SMS bearer
- Mobile equipment to receive the request and transmit it to the SIM card
- A SIM card to receive and execute the request

Back end System
The back end system can be anything from a customer care operator to a billing system, a content provider or a subscriber web interface. The provisioning system has to be connected to the mobile network (either per LAN or via the Internet). Service requests contain the service requested (activate, deactivate, load, modify...), the subscriber targeted and the data to perform the service. The back end system then sends out service requests to the OTA gateway.

OTA Gateway
The OTA Gateway receives Service-Requests through a Gateway API that will indicate the actual card to modify/update/activate. In fact, inside the OTA Gateway there is a card database that indicates for each card, the SIM vendor, the card's identification number, the IMSI and the MSISDN.
The second step is to format the service request into a message that can be understood by the recipient SIM card. To achieve this, the OTA Gateway has a set of libraries that contain the formats to use for each brand of SIM cards. The OTA Gateway then formats the message differently depending on the recipient card.
The third step consists in sending a formatted message to the SMSC using the right set of parameters as described in GSM 03.48. Then the OTA Gateway issues as many SMS as required to fulfil the Service-Request. In this step the OTA Gateway is also responsible for the integrity and security of the process.

SMSC
Services center for short messages (SMS) exchanged between the management system of these messages (OTA Gateway) and the cellular network. A message consisting of a maximum of 160 alphanumeric character can be sent to or from a Mobile Phone. If the Mobile Phone is powered off or has left the coverage area, the message is stored and offered back to the subscriber when the mobile is powered on or has re-entered the coverage area of the network.

SIM Card
Smart cards provide secure user authentication and is mainly used in GSM standard as Subscriber Identification Module (SIM cards). The SIM is the major component of the GSM market paving the way to value-added services. SIM cards now offer new menus, pre-recorded numbers for speed dialling, and the ability to send pre formatted short messages (SMS) to query a database or secure transactions.

Origin Server
Software that can respond to requests from a WAP terminal by delivering appropriate content or error messages. The origin server may receive requests via either WSP or HTTP. Application programs executing on the origin server can use UAProf to deliver content that is tailored in accordance with the CPI that can be found within the provided profile.

Proxy
Software that receives HTTP requests and forwards that request toward the origin server using HTTP. The proxy receives the response from the origin server and forwards it to the requesting client. In providing its forwarding functions, the proxy may modify either the request or response or provide other value-added functions.

DRM
The scope of OMA Digital Rights Management is to enable the controlled consumption of digital media objects by allowing content providers to express usage rights, e.g., the ability to preview DRM content, to prevent downloaded DRM content from being illegally forwarded (copied) to other users, and to enable superdistribution of DRM content.

Appendix D
SAT commands in bytes

Provide Local Info

Byte #	Value (Hex)	Description
1	D0	Proactiv SIM Command tag
2	09	Length
3	01	Command Details tag
4	03	Length
5		Command Number
6	26	Type of Command
7	01	Command Qualifier
8	02	Device Identity tag
9	02	Length
10	82	Source Device Identity
11	81	Destination Device Identity

Select

Byte #	Value (Hex)	Description
1	A0	Class
2	A4	Ins
3	00	P1
4	00	P2
5	02	P3

Open Channel

Byte #	Value (Hex)	Description
1	D0	Proactiv SIM Command tag
2		Length
3	01	Command Details tag
4	03	Length
5		Command Number
6	40	Type of Command
7		Command Qualifier
8	02	Device Identities tag
9	02	Length
10	81	Source Device Identity
11	83	Destination Device Identity
12	06	Address tag

Send Data

Byte #	Value (Hex)	Description
1	D0	Proactiv SIM Command tag
2	X+Y+10	Length
3	01	Command Details tag
4	03	Length
5		Command Number
6	40	Type of Command
7		Command Qualifier
8	02	Device Identities tag
9	02	Length
10	81	Source Device Identity
11	83	Destination Device Identity
12	36	Channel data tag
13 to Y+12		Length (X)
Y+13 to X+Y+12		Channel Data string

Close Channel

Byte #	Value (Hex)	Description
1	D0	Proactiv SIM Command tag
2	09	Length
3	01	Command Details tag
4	03	Length
5		Command Number
6	40	Type of Command
7		Command Qualifier
8	02	Device Identities tag
9	02	Length
10	81	Source Device Identity
11	83	Destination Device Identity

Appendix E
Vendor queries

In order to be able to implement the solution suggested in chapter 3, knowledge is required about the location and retrieval of User Agent or UAProf on a mobile device. Although the OMA specifications of the User Agent Profile [UAP02] define the user agent transport over W-HTTP and WSP, the exact process of storage and retrieval of the user agent information on the terminal is not described. The terminal experts of Vodafone NL as well as Vodafone Group could not provide the desired information, so Vodafone Group issued a Request For Information (RFI) on behalf of this project to the main terminal vendors, i.e. Nokia, SonyEricsson, Motorola and Siemens.

E.1 Request for information

In this section can the RFI be found as it was sent to the terminal manufacturers.

At Vodafone R&D The Netherlands we are working on a project which needs information of the capabilities of a mobile device on a real-time bases. For this project we are interested in where the capabilities of the mobile device are stored on the mobile device and how the description of these capabilities can be retrieved from the mobile device.

Nowadays two types of information on the capabilities of a mobile device exist:
1. User Agent Profile (UAProf) information
2. non-UAProf information (HTTP header user agent information).

The User Agent Profile is stored on the Web (typically at the manufacturers Website). The URL for a specific device's UAProf document is embedded in the terminal and is manifested through the User-Agent-Profile HTTP header. The WAP Gateway/Content Server can then retrieve the document from the manufacturer's site and do content adaptation accordingly.

For our project we want to retrieve this information (UAPROF URL or user agent info) from the mobile device without starting a WAP session, so in an autonomous way.
To make progress on our project we need to know the following:

Where is the UAProf information stored on the device and how can we autonomously retrieve it? And in case that the mobile device is not yet UAProf enabled, where is the device-specific information (e.g. manufacturer and type) stored on the mobile device and how can we autonomously retrieve it?

E.2 Nokia

E.2.1 Contact information

Jouni Kangas (jouni.kangas@nokia.com, +358718021438)
Julian Heaton (julian.heaton@nokia.com, +447802688486)

E.2.2 Response to the RFI

Nokia was very helpful and interested in the topics discussed in the RFI. In an email discussion that resulted in a conference call the points of view of both Nokia and our project were stated. Nokia is in the phase of introducing the UAProf capable terminals i.e. to offer the static device capabilities for the end-to-end architecture meaning that intermediate network elements (gateways, proxies) and also the content provider's application servers/services can utilize the capability information in order to do content selection on behalf of the end user and to be able to

perform content adaptation based on terminal capabilities. This leads to better end user experience. The user agent information is in the device binary code, which cannot be retrieved other than by a client connecting to the server.

The work to understand the dynamic capabilities and the requirements for it has just started so Nokia unfortunately cannot provide an immediate solution for that. They would however be very pleased if we could continue the discussion in order to better understand the requirements and get our ideas to be taken into account.

E.3 Sony Ericsson

E.3.1 Contact information

Kouji Kodera (kouji.kodera@sonyericsson.com)
Magnus Richardsson (magnus.richardsson@sonyericsson.com, +46702194110)

E.3.2 Response to the RFI

According to SonyEricsson, it is not possible to "ask" the phone for the UAProf without establishing a browsing session. The only way to find out information about the phone "offline" would be to use the TAC(Type Approval Code) part of the IMEI number,
which is send to the network as soon as the phone is used in the operators network.
This number says which model of the phone it is. The model name could then be used to download the relevant UAProf file.
The TAC is connected to the HW of the phone. Therefore if there are new revisions to the software(and then also new versions of the UAProf) there is no way of telling which version of the UAProf that should be used.
After furher inquiry, SonyEricsson explained very kindly that the UAProf iformation is sent from the application and not the stack, so in e.g. their P800 phone they could download a new browser which could send adifferent UAProf URL compared to the standard browser.

E.4 Motorola

E.4.1 Contact information

Marcus Watson (marcus.watson@motorola.com, +447801304318)

E.4.2 Response to the RFI

As a response to the RFI, Motorola stated that for its current shipping phones the UAProf URL or user agent info can be retrieved from the User Agent Header. For their phones E380, V500, V600 & E395 and beyond, the UAProf URL or user agent info can be retrieved from either the UAProf or User Agent Header.
After making clear that our aim is to autonomously retrieve the user agent or UAProf information in a situation where no HTTP or WSP is established, Motorola promised to get the responses for our questions.

E.5 Siemens

E.5.1 Contact information

Gary Vincent (gary.vincent@siemens.co.uk, +441344850335)

E.5.2 Response to the RFI

Siemens kindly provided the location of the UAProf xml data for its UAProf phone Siemens S55 on its website (http://communication-market.siemens.de/UAProf/S55_11.xml). However, concrete information about where and how to retrieve the UAProf information on the mobile device was not given. Further inquiry as a result of this response to the RFI did not provide more detailed information.

E.6 Conclusions regarding the RFI

All device manufacturers admit that the problem stated in the RFI is quite complex. At the moment, no other mechanism for fetching the user agent or UAProf information seems to exist than making use of the information expressed in the User Agent Headers or UAProf headers.
As far as our wish for autonomously retrieving the user agent or UAProf information is concerned, the device manufacturers are very hesitating to provide more information. Mostly, they simply state that it just is not possible to do it in another way than by making use of HTTP or WSP headers, which implies session establishment.
Even the simple question where the UAProf URL can be found on the device caused quite a stir for most terminal manufacturers. One easily would suspect that the manufacturers are somewhat anxious to let a provider in on their field of expertise.
As far as UAProf is concerned, this technology is making its way to the consumer market. The first handsets have already appeared among the current shipping phones. However, the use of Profile-Diff headers which are sent along with the UAProf URL to show deviations from the standard UAProf profile is limited to very few phones.
Anyway, the conclusion of the RFI is that according to the device manufacturers it is at the moment not possible to autonomously retrieve the desired user agent or UAProf information.