OPEN Project

STREP Project FP7-ICT-2007-1 N.216552

[1]

Title of Document:	Initial OPEN Service Platform architectural framework
Editor(s):	F.Paternò, C.Santoro,
Affiliation(s):	CNR-ISTI
Contributor(s):	S.Bigi, G.Ghiani, A.Nickelsen, H.Klus, E.Kovacs, A.Jahanpanah, S. Marzorati, R.Olsen, F.Paternò, C.Santoro, A.Scorcia, H. Schwefel
Affiliation(s):	Aalborg University, Arcadia, Clausthal University, CNR-ISTI, NEC, Vodafone Italy
Date of Document:	31 October 2008
OPEN Document:	D1.2
Distribution:	
Keyword List:	Migratory Interactive Services, Architectural Framework, Platform for Supporting Migration
Version:	

OPEN Partners:

CNR-ISTI (Italy)
Aalborg University (Denmark)
Arcadia Design (Italy)
NEC (United Kingdom)
SAP AG (Germany)
Vodafone Omnitel NV (Italy)
Clausthal University (Germany)

---

[1]

Title: Initial OPEN Service Platform architectural framework	**Id Number: D1.2**

Abstract

The purpose of this deliverable is to present the initial OPEN Service Platform architectural framework. For this purpose it indicates the main phases of the migration process, and the high level description of the possible associated components, with a discussion of advantages and disadvantages for the various design options. It will result in the description of the overall architecture of the Migration Service Platform (MSP), which will provide a high level design of the basic components and the interfaces between them. The overall goal is to create a common vocabulary, identify the main parts of the OPEN architecture and discuss how they can interact with each other. More detail on the various parts of the OPEN platform will be provided in a number of deliverables planned for M12 (D2.2, D3.1, D4.2).

Title: Initial OPEN Service Platform architectural framework	**Id Number: D1.2**

Table of Contents

1 Introduction

One important aspect of ubiquitous environments is to provide users with the possibility to freely move about and continue the interaction with the available applications through a variety of interactive devices (including cell phones, PDAs, desktop computers, digital television sets, intelligent watches). In such environments one big potential source of frustration is that people have to start their session over again from the beginning at each interaction device change.

Migratory interactive services can overcome this limitation and support continuous task performance. This implies that interactive applications are able to follow users and adapt to the changing context of use while preserving their state. Namely the knowledge of context, being any information that can be used to characterize a situation of an entity, [Dey00], is useful to ensure that services are migrated at the right time and place. This implies that the system would need not only to consider to migrate the service, but also under which circumstances it happens, e.g. migration of a video stream service to a laptop that is running out of battery, may not be an appropriate option, while a nearby tv screen may be a better option. To summarise:

Migration = Device Change + Adaptation + Continuity.

As we will describe in the following sections, in order to have a real ‘migration’, all such aspects have to be included: no proper ‘migration’ occurs if there is a change of device and an adaptation of the application features to the new device, but there is no continuity in the resulting user activity because, for instance, the user has to restart from the beginning when the new configuration is activated. Likewise, a situation in which there has been a device change and also the state of the application has been saved cannot be properly called ‘migration’ if an opportune step of adaptation is not performed as well.

Therefore, migration encompasses all the three aspects related with: device change, then there is the issue of how such devices are discovered; adaptation, then there is the problem of how the characteristics of the context is taken into account and handled when adapting the application to the characteristic of the new context; and continuity, then there is the issue of how to guarantee continuity in task performance and issues like techniques for maintaining state persistence have to be addressed.

Thus, the OPEN project provides integrated solutions able to address three aspects: device change, state persistence and content adaptation. This is obtained through the Migration service Platform (see Figure 1), a middleware able to consider and integrate various aspects: adapt and preserve the state of the software application parts dedicated to interacting with end users; support mechanisms for application logic reconfiguration; and identify suitably flexible mechanisms available in the underlying network layers. The resulting middleware should be able to interoperate with existing technologies. Ideally, the migration platform should be able to take all existing applications and make them migratory. However, this is clearly too ambitious for a single project, thus we will focus on classes of applications (Web applications and distributed applications in the game and business domains). In

addition, it is likely that such applications should be developed according to some guidelines in order to ease the support given by the migration platform.

Examples of how the migration capabilities could be used in practice for improving e.g. the user experience can be found in Deliverable 1.1 (Requirements for OPEN Service Platform, see [OPEN D1.1]). In other cases, the need and opportunity for triggering a migration can be aimed at other objectives (for instance, a better use of the device resources and/or network capabilities available, etc.). To this respect, in [OPEN D1.1] several scenarios and related requirements have been identified as of interest for the project. For instance, one scenario is a gaming scenario in which migration capabilities are used for making the gaming experience being shared by an additional user, without the need of stopping the game which was already running and being played by another user. Indeed, in this scenario, at some point an additional player wants to join a Pacman game, and such a new player takes the control of a ghost (whose control is in this way shifted from the application to a human player). The new player can seamlessly join the game by using a device currently available, which is equipped with the needed functionalities and user interface for controlling the game without the need of stopping the game. In this case, the Migration Platform should recognise the availability of a new device and, depending on the specific context and device characteristics, migrates the needed services and the related user interface parts on the new device, also taking care of saving the current state of the game and render the new user interface adapted to the new device. In this way, on the one hand the join of the new user is completely transparent for the player(s) who already were playing Pacman (e.g.: there is no need of stopping the game for enabling another user to join), and, on the other hand, the burden needed for suitably modifying the current configuration of the application, in order to match the new requirements (e.g.: saving the state of the game, make all the needed controls available on the device of the new user and adapted to the new device, etc.) is handled by the Migration Platform.

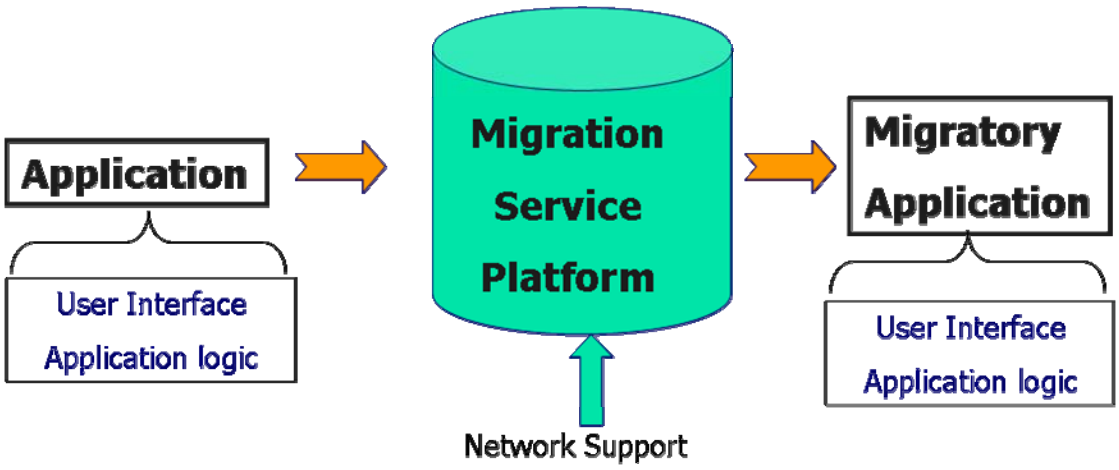

Figure 1. The Migration Service Platform.

1.1 The Migration Process

The migration process can be seen as composed of a number of phases, which can be implemented in various ways. Such phases are specific aspects that characterise the migration process and the migration platform should explicitly support:

- **Device Discovery**. Its purpose is to identify the devices that are available to be involved in the migration process and their attributes that can be relevant for migration (private or public device, their connectivity, their interaction resources, …).
- **When to Migrate**. The Migration Trigger indicates when to migrate. This event can be generated by the user or the system, or through a mixed initiative process (the system proposes migration and the user can decide whether to accept it). Users can request migration when they feel it necessary, while the system can trigger it when specific events are detected (such as the device is getting out of power).
- **Where to Migrate**. Once migration is triggered, it is important to identify the target device for the migration process. Such target should be one of the devices available for this purpose and it should be detected on the basis of its features and how well fit in the new context of use.
- **What to Migrate**. An interactive migratory service is composed of two main parts: the user interface and the application logic. The former is the software dedicated to the interaction with the user while the latter is the functional core independent of how user interaction takes place.
- **How to Migrate**. Since the device to access the application changes after migration, some level of adaptation of the migratory service should be performed, in particular of its interactive part, in order to better exploit the new resources available while preserving usability.
- **State persistence**. One of the main reasons for migration is to continue a session through different devices. This means that the changes made by the user in the source device should not be lost when moving to the new one. Thus, it is important to carry out source state extraction and associate it to the target version.
- **Activation in the target device**. In order to obtain continuity, it is important that the application on the target device is activated not at its usual starting point but at the point in which it was left off on the source device.
- **Optional termination** of the source version, in general after migration the source version should be terminated but there are case in which it can be useful to allow access to the migratory application from both the source and the target device after the migration.

1.2 Adaptation - What can be Adapted

Adaptation can take place at various levels of granularity. For example, Aura [GSS+02] provides support for migration but the solution adopted has a different granularity. In Aura, for each possible application service, various applications are available and the choice of the application depends on the interaction resources available. Thus, for example for word processing, if a desktop is available then an application such as MS-Word can be activated, whereas in the case of a mobile platform a lighter editing application is used. Thus, Aura aims to provide a similar support but this is obtained mainly by changing the application depending on the resources available in the device in question, while we want to have control on various parts of the same interactive application that adapt to the interaction resources available. Indeed, various parts of an interactive migratory service can be adapted:

The **User Interface**, which is composed of the presentation (the choice of the modality, layout, graphical attributes, …), the dynamic behaviour (the choice of the navigation model, the dynamic activation and deactivation of interaction techniques), and content (what information is actually presented). Each of them can adapt according to a change of context. The User Interface can adapt according to different strategies:

- *Conserving* (keep the arrangement/presentation of UI objects)
- *Rearrangement* (UI objects kept during migration but they are rearranged according to some techniques, e.g.: different layout)
- *Increase* (target device can provide more UI features)
- *Reduction* (less UI features)
- *Simplification* (UI objects kept, but with simplified representations, e.g.: images with lower resolutions)
- *Magnification* (opposite of simplification).

The **Application Logic** can be adapted by reconfiguring the access to the functionalities in order to access different implementations of some of them or increase/decrease such functionalities because of the change of device or the change of the connectivity. For example, an access to a data base to retrieve a large amount of data can be performed if the application is using a good connectivity, while the same access should be avoided if the connectivity is too poor to provide results in a reasonable amount of time. In this case, it could be necessary to perform the data base access in a separate phase or in separate application, hence affecting not only the business logic of the migrating application but also the overall procedure the migrating application belongs to.

The **Content**, which is stored in the functional core of the application. The adaptation can also cover aspects related with content. There are two possible alternatives: one possibility is that different representations of the same content can be statically maintained and, depending on i.e. the resources of the device at hand, the most suitable content will be selected. Another possibility is that the

adapted content can be dynamically generated from the content that is already available. For instance, it is the case when an image can be derived from a video, by capturing just the first frame.

The **Network support**, since the connectivity can change, then the network protocol and their quality of service may have to change. We should emphasise that network aspects can cover two different issues. On the one hand, the network can adapt in order to offer better QoS. In this case, the network properties will be adjusted in order to fulfil some requirements. In other cases, the network can be considered itself as an additional contextual aspect (e.g.: it can be considered as an aspect of the physical environment): depending on the current network capability, other aspects of the system (e.g.: the user interface presented on device, ...) can be adapted.

2 Migration Logical Dimensions

2.1 Migration Types

Various types of migration can be identified, depending on the number of source and target devices involved in it:

Total, change from one device to another;

Partial, migration of only a part of the interactive migratory service, e.g.: partial control migration (see Figure 2);

Distributing, the interactive migratory service is totally distributed over two or more target devices;

Aggregating, the interactive migratory service distributed over multiple source devices is then grouped into a single target device;

Multiple, both source and target are multiple devices.

Figure 2. Example of Partial Migration

2.2 Trigger Types and Target Device Selection

The trigger event defines when migration should occur. It should then be accompanied by the indication of the target device(s) for migration. In order to allow for a good choice of the target device, the migration platform should retrieve and store information about the devices that are automatically discovered in the environment. The collected information mainly concerns device identification and their capabilities. On the one hand, such information allows users to choose a target migration device with more accurate and coherent information on the available target devices and, on the other hand, it allows the system to suggest or automatically trigger migrations when the conditions for one arise. Thus, both the system and the user have the possibility to trigger the migration process, depending on the surrounding context conditions. Users can have various ways of issuing migration requests. One example is to graphically select the desired target device in their migration client. Users should have the possibility of choosing those devices that they are allowed to use and are currently available for migration.

Migration can also be initiated by the system skipping explicit user intervention in critical situations when the user session could accidentally be interrupted by external factors. For example, we can foresee the likelihood of having a user interacting with a mobile device that is shutting down because its battery power is getting too low. Such situations can be recognised by the system and a migration is automatically started to allow the user to continue the task from a different device, avoiding potential data loss. Alternatively, the migration platform can provide users with migration suggestions in order to improve the overall user experience. This happens when the system detects that in the current environment there are other devices that can better support the task being performed by the user. For example, if the user is watching a video on a PDA and a wide wall-mounted screen is detected and available in the same room, the system will prompt the user to migrate to that device, as it could improve her/his performance. However, the user can continue to work with the current device and refuse the migration. Since receiving undesired migration suggestions can be annoying for the user, users who want to receive such suggestions when better devices are available must explicitly subscribe in order to allow for this mixed-initiative migration service. In any case, once a migration has taken place, nothing prevents the user or the system from performing a new migration to another available device.

2.3 Context of Use

Context information refers to all information that is relevant to describe a given situation for a given object, and as a general definition we use is the one given by [Dey00]:

> ***Context*** *is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.* [Dey00]

Knowing context information may be a useful asset for an application, service or other networking components in order to adapt to the current situation.

The definition is wide and implies that context may be any general type of information. The key is, as indicated, that it is *relevant* information that falls under the definition of context. Thus in OPEN we need first to define what is relevant information for service migration. Various parts of the context of use can be relevant in the migration process. A change in a single contextual dimension (which, in turn, results in a more general 'context change') might represent a condition to be analysed for activating a possible migration. The context of use can be seen as composed of four main aspects:

- *Device*, its resources for interaction and processing, and its connectivity. A device change is a common example of a context change. A device change can affect different aspects in a migrating application, not only from a user interface point of view (because the new device might offer different interaction capabilities with respect to the old one), but also e.g. from a performance point of view (the new device can support more concurrency in task execution and therefore the module handling the application logic reconfiguration can benefit of this adequately), etc.
- *User*, the preferences, the tasks, and the associated goals. The knowledge of user preferences, tasks and goals can enable the activation of an opportune migration as soon as some conditions occur in the current context.
- *Physical Environment*, example attributes are noise, light.
- *Social Environment*, relations among the users that can affect the interaction.

The information may be measured directly, e.g. information about the physical environment like light or noise may be measured by sensors locally or remote. Other types of relevant information, e.g. with respect to user activity may need to be inferred or derived from existing context information, as such information may not necessarily be directly measurable. Thus, a context management framework for OPEN would need to support not only collecting, and distribution of context information, but also online processing capabilities for inferred context information in order to ensure best triggering mechanism for service migration.

3 Architectural Framework

This section introduces the architectural framework of the OPEN migration platform, which will be detailed in the following sections. The OPEN platform should be able to provide support and transformations at various layers: the user interface, the application logic, and the underlying middleware oriented to manage the context of use, including the network support.

In Figure 3, the different architectural modules at a very high level are visualised. Basically, our architecture foresees a number of devices which communicate with the OPEN Platform. The OPEN Platform provides support for a number of services (user interface migration, application logic migration, etc.) enabling migration. For each device involved in the migration, two main parts can be considered: an Application module (covering UI and application logic functionalities) and another module (called OPEN Client), which is the module that allows for the connection with the OPEN platform server and for gathering information regarding the current state from the application.

The OPEN platform has a service-oriented architecture. The first version will be based on a server supporting it but the possibility of a distributed peer-to-peer (P2P) architecture may be explored in the future ("future P2P?" bullet).

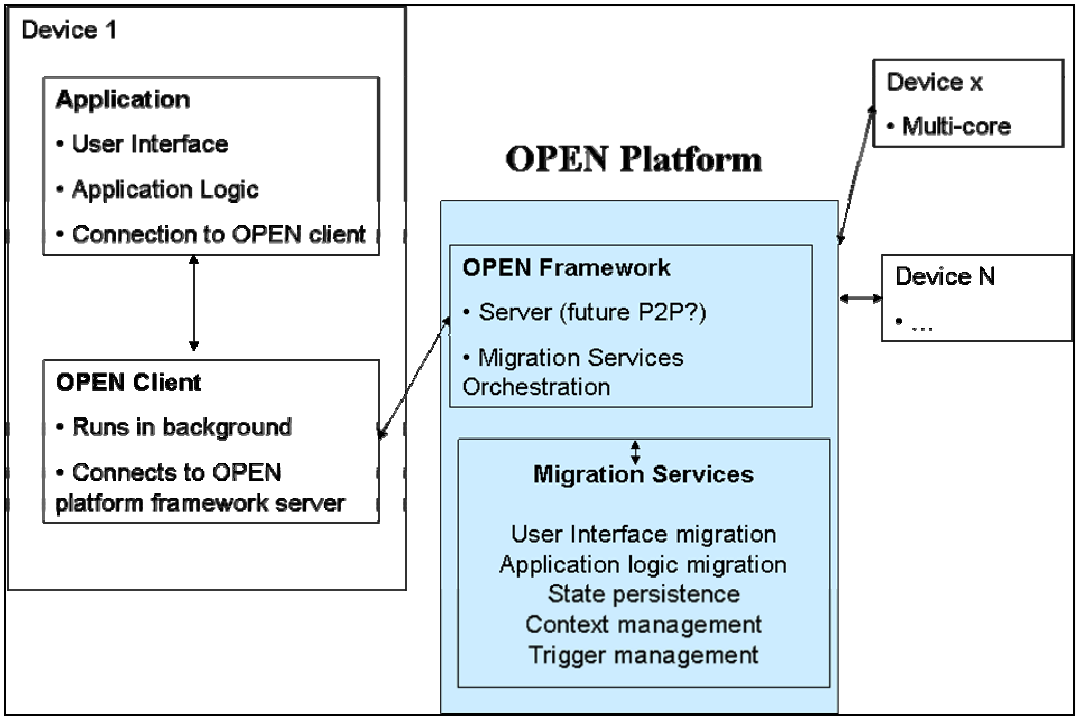

Fig 3: General Overview of the system

The next figure (Figure 4) provides a more detailed indication of the overall architecture of the OPEN Platform. While the figure still shows that we consider applications interacting with users, it provides some more indications on the functionality of the platform. For example the user interface migration is detailed into user interface adaptation + user interface state persistence. Likewise, the application logic migration is refined. The middleware is refined into a number of functionality (device discovery, mobility support, context management, ..). The device(s) involved in migration is expected to include, in addition to the application itself, another piece of software (Migration Client), through which the user can see the available device(s) and select to which device(s) trigger the migration.

In the multi-layer architecture proposed, the lower layers basically provide functionalities to the higher layers, and OSGI framework is currently-considered as a common glue for enabling communication among the various modules.

As we already mentioned in Section 3, different types of adaptation can be handled by the OPEN Platform. For instance, the module "UI reconf/adaptation" in Figure 4 is aimed at adapting the user interface part of an interactive migrating application and, likewise, the module "Appl. logic reconfiguration/orchestration" handles the adaptation of the application logic.

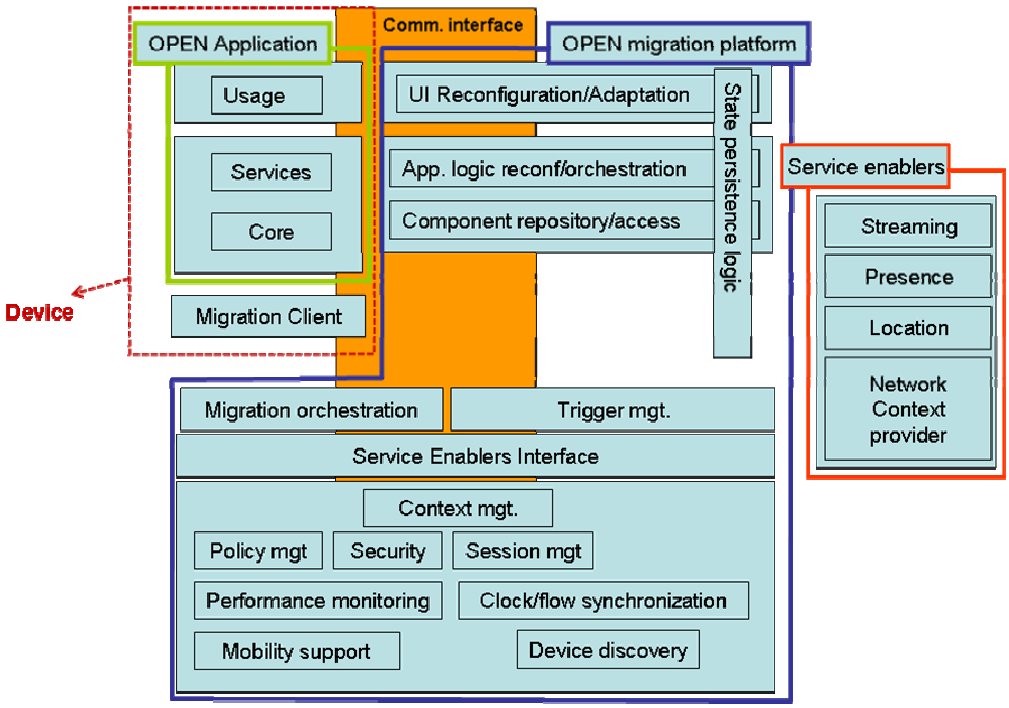

Figure 4. OPEN architectural framework

This architecture has been derived through a mixture of top-down from the application scenarios and requirements, and bottom up by identification of relevant functions from experience of the partners. It will be revised in the second year depending on the project evolution.

In the next sections we will provide more detailed descriptions of the three main parts composing the OPEN Platform middleware identified by the three main aspects that are handled: user interface, application logic, and network and context management.

4 The User Interface Middleware Layer in the architectural framework

WP2 concerns middleware for supporting user interface migration. Its main components, according to Figure 4, are user interface adaptation and state persistence.

The main requirements that we plan to consider in this part are:

- *Avoid manual solutions*, in which developers directly create migratory versions of an application because they give full control on the results but they lack of generality and would be too expensive when applying to various applications;
- *Existing Web desktop Applications*, while we would like to make all existing applications migratory, this can be too ambitious for this project. Thus we decided to focus on existing web desktop applications. These are applications that can be accessed through a Web browser. The reasons for this choice is that this type of application is the most common (there exist millions of Web desktop applications). In addition, we can manipulate and transform the content of such user interfaces for obtaining versions suitable for various types of devices;
- *Targeting a wide variety of interaction platforms*, we want to be able to dynamically create migratory user interfaces adapted for various types of devices and implementation languages (even non Web languages, which are languages that allow to express specifications that can be interpreted by Web browsers) starting with existing desktop Web applications.

4.1 Adaptation Strategies

The adapted version of the application for the target device of the migration can be obtained in various ways:

Pre-computed, in this case it has been defined beforehand the migration process by an ad-hoc development. Usually this implies more control on the resulting version. However, this solution requires considerable development effort (the development of a version for each type of target device for each application considered, consequently it has a limited applicability. Such pre-computed version can be either preinstalled in the target device or loaded on-demand when migration occurs.

Dynamically generated, in this case the application version for the target device does not exist beforehand and it is automatically generated when migration occurs. The new version is obtained by applying some transformation rules coded in the adaptation engine starting with the information of the application version in the source device. This solution requires limited effort (it only requires the activation of the automatic adaptation engine). However, there is less control on the resulting version and it is difficult to find a set of rules that provide optimal solutions in a wide set of cases. One example of strategy implementing this

approach is based on three main phases: reverse engineering for creating a logical description of the interactive application from the existing implementation, semantic redesign to adapt the logical description to the target platform, adapted implementation generation. This is the solution for adapting the user interface of a migrating web application which is currently followed by ISTI-CNR, and it has been implemented in a prototype implemented in Java under further development. In this prototype, the design of the considered application is represented by a XML-based logical description, is dynamically adapted according to the characteristics of the target device, and then it is used to generate the final user interface for the considered platform. Further details of this approach will be better explained in D2.2.

Mix between pre-computed and automated, in this case various levels of automation are considered but starting with some aspects pre-computed. This ranges from developing the source application according to some guidelines that make it more suitable for adaptation to having the structure of the adapted version pre-defined with some aspects that are dynamically filled in it. In some cases there is one generic version of the application with annotations for obtaining adapted versions to different types of devices.

In general, there is no a priori 'best' solution to be selected among such strategies, although on the one hand a pre-computed solution will emphasise the full control by the designers in obtaining the adapted version, but it requires a big effort for addressing various applications and target devices because each application version should be manually created. On the other hand, a completely dynamically generated solution requires less effort but it may generate not optimal solutions in terms of usability because the underlying rules driving the generation may not be sufficiently able to address the specific usability requirements of each application. In any case, in the OPEN Project we will aim to identify the best solution depending on the devices and the applications at hand. In particular, we are working on an automatic generation solution but we will consider the possibility to modify the rules driving adaptation and provide guidelines for designing and implementing the interactive application in order to make it more easily manageable for our platform.

4.2 Where Adaptation Can Take Place

Adaptation can take place at least in three types of hosts:

The Application Server, which recognises the type of target device and activates an adaptation accordingly. The drawback of this solution is that the adaptation engine should be duplicated in all the potential application servers.

Intermediate server, in this case the adaptation engine is located into a single server that acts also as a proxy server monitoring the requests from the target client device and adapting the results according its resources. Potential issues for this architectural approach are congestion or lack of scalability. An example will be better explained in Section 7.4, where we describe the various steps of the

approach which is under development at ISTI-CNR in a prototypal environment for supporting migration.

Client device, in this option the adaptation engine is installed into the target client device and is applied when migration has been triggered. The local installation guarantees that there is full access to information on the local device capabilities. However, with some device with limited capabilities (such as some types of mobile devices) there can be problems in supporting the processing requested by adaptation.

Every architectural approach has also impacts on the business ecosystem to put in place:

- If the adaptation takes place at application server level, the focus will be for the specification and promotion towards the developers community of a framework for a specifying guidelines and development constraints.
- If the adaptation is based on the intermediate server, the effort related to the complexity and computational resources for the user interface adaptation should be put towards ISP or telco operator who should provide the intermediate server.
- If the adaptation takes place in the client device, this could impose some form of device manufacturers endorsement, in order to ensure that the adaptation engine can exploit at its best the capabilities offered by the device.

In addition, it is worth pointing out that the aspects that are related with *where* adaptation should occur have an impact on issues related with adaptation strategies (*how* to adapt) described in the previous section. Indeed, for instance, a heavy adaptation strategy like the dynamically generated one is difficult to be combined with an adaptation that takes place in the client device, when the client platform is a thin device with limited performance capabilities.

4.3 State Persistence and Associated Support

One of the main components of the migration platform is the support for task continuity. The goal is to allow the users to change the device and continue the task at hand from the point they left off in the source device. This implies the ability to take the state of the of the source application and to apply it to the version that is going to be activated in the target device.

The state of the interactive application can be seen as logically composed of the state of the user interface (which is defined by the information entered or selected by the user) and the state of the application logic.

In a Web application, we have identified at least eight aspects that can be relevant for de-fining the state of Web user interfaces and that can have an impact on the overall user experience. The first element is associated with the user input. People make selections, enter text and modify the state of various input controls during a session, and such modifications should not be lost when moving to a new device if

we want to maintain continuity. An associated element is client-side variables associated with small functionalities (e.g. Javascript variables).

Another component that can be dynamically modified is the content of a Web application. While this can be easily managed with dynamic Web sites using PHP, JSP and similar languages because whenever a new request is performed then a different page is uploaded, with Ajax scripts this aspect becomes more problematic. Indeed, in this case the content of the page can vary without requiring the loading of a new page. Thus, it becomes more complex to detect what is actually composing the currently displayed page.

Cookies are more and more used and they allow an application to provide small pieces of information to the client in such a way that whenever the client accesses the application, then the client identifiers are inserted in the HTTP protocol. It is important that if and when a user changes device, then the current application preserves the same cookies in order to be recognised by the application server. A related technique is the session: it is a server-side mechanism, which stores information related to the user session, which is in turn associated with a specific identifier.

Another important aspect is the history of user accesses, which is maintained by the Web browser and drives the behaviour of the frequently used browser back button. Since the user is still the same, even if she has changed device, then she would appreciate still being able to easily return to recently accessed pages, even if through a different device. It is clear that the pages accessed through the new device may be adapted to the currently available interaction resources. In some cases (e.g. migration from desktop to mobile), it may even happen that the original desktop page is split into multiple mobile pages, thus accessing all its content may require further navigation.

Bookmarks are another interesting aspect that can be considered part of the user interface state. Users often use them to quickly find and access favourite pages. In migration, the devices change but not the user, who still has the same interests and may appreciate the possibility to find in the current bookmarks including the pages that were bookmarked in the previous device. Another element that has similar characteristics is the browser home page: in some cases users may be interested to migrate it to different platforms as well.

A last element that can be considered part of the state is the query string included in a URL after the “?” symbol. It is usually used to specify parameters for a dynamic site, which define some data that are presented in the associated page. By modifying the query string we will access the same Web site but since the parameters vary, then the corresponding page varies in terms of content.

In D2.2 we will discuss how the OPEN platform addresses the rich user interface information state discussed in this section.

4.4 User Interface Migration

Figure 5 shows how the migration of the user interface works at run-time. It assumes the existence of a desktop web implementation of the interactive application.

A reverse engineering process builds the corresponding logical descriptions. The logical descriptions and how they are represented in XML will be described in a next WP2 deliverable. This is the starting point for the adaptation part, which creates a logical description tailored for the target platform. Then the state resulting from the user interactions with the source version (selected elements, text entered, …) is associated with the target concrete description, which is finally used to generate the implementation for the target device.

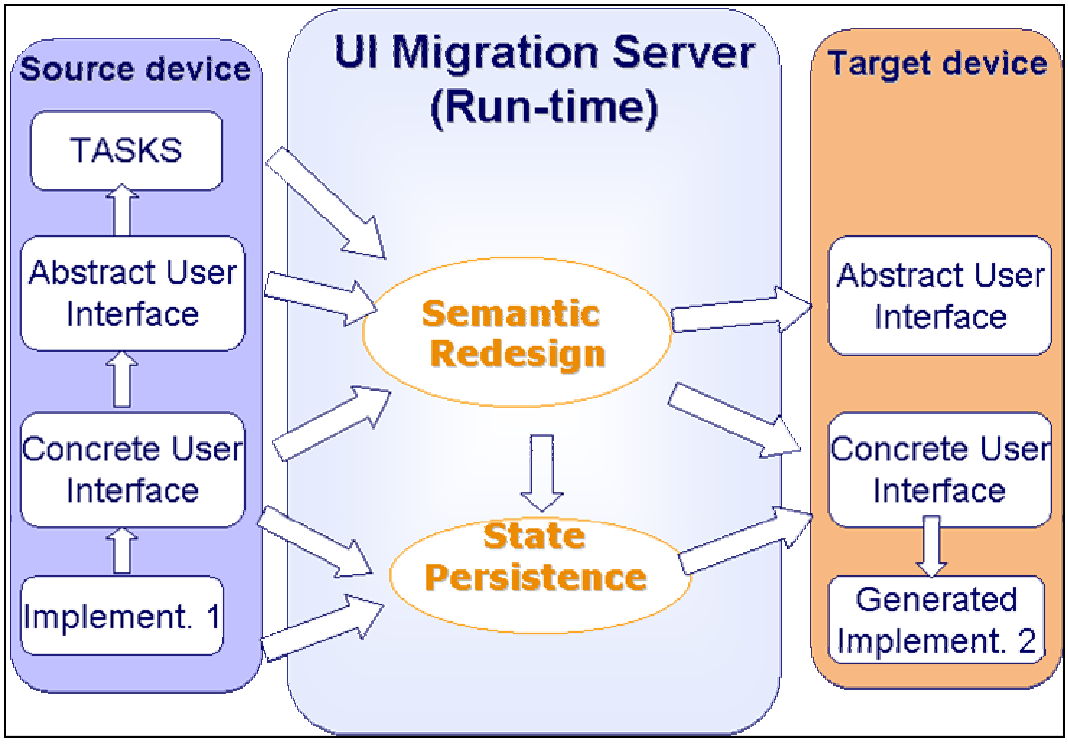

Figure 5. UI Migration Server

4.5 An Example of Target Platform: The Multicore UI Toolkit

In this section we describe a specific target implementation platform that is going to be developed within the OPEN project, the multicore UI framework.

This is a library of classes that enable application programmers to implement visually appealing graphical user interfaces similar in look and feel to existing UIs on mobile devices (e.g. Apple iPhone), game consoles (e.g. Xbox Live! Arcade UI, Playstation™ Store) or set-top boxes (e.g. Channels list, program guides, etc.).

The framework is mainly targeted at multimedia applications and in particular will provide a 3D compositing layer allowing for features such as rotated and scaled windows, transparency and animated widgets.

To achieve these effects and to provide best possible performance the library uses multiple threads internally to leverage multicore CPUs.

The innovative adaptive load-balancing implementation always utilizes the full processing performance provided by the CPU.

Especially the system automatically adapts to changing numbers of available cores at runtime which is a key advantage with respect to application migration.

It also supports disabling of CPU cores at runtime, without interruption of the running application and with no special requirements on the programmer's side. This behaviour is especially useful for mobile and embedded devices as application performance can be adapted with respect to power consumption and battery life.

This is the most important distinguishing point of the toolkit as none of the existing GUI libraries on the market are designed from the ground up to support multicore rendering on CPUs to this extend (Apple and Microsoft focus on GPU acceleration for example).

The second important aspect is ease of adaptation and migration of GUI layouts to different devices: For example, all relevant coordinates, regions and parameters are specified in normalized floating point coordinates which enables GUIs to be resolution independent. Memory footprint and run-time resource requirements are very low in order to ensure smooth interaction on mobile and embedded devices.

The GUI framework is a software development kit similar to e.g. Trolltech Qt / Qtopia (see http://trolltech.com/) or wxWidgets.

Viewed from a higher level it provides the following functions:

- Widget implementations (for windows, buttons, lists, etc.)
- handling state and updates, messaging logic and event handling
    - E.g. when the user moves the mouse, clicks a button, scrolls a window, etc. the corresponding event is stored in a message queue
    - On update, widgets get their events from the queue and react to these events, i.e. change their state
- rendering and compositing
- transformation (rotation, scaling)

A Renderer component could provide lower-level graphics tasks such as drawing of graphical primitives like diagrams, charts, etc.

The Compositor sub-system performs several functions, including

- control of display update
- management of the active windows and update of their transformations
- interaction with the rendering and multicore sub-systems to draw the final layout

The following figure (Figure 6) gives a brief higher-level overview of how various components interact in the generation of the user interface for a multi-core

platform. The ouput of the adaptation part is an XML concrete user interface description, which is used as a starting point for the generation of a user interface exploting the multi-core library.

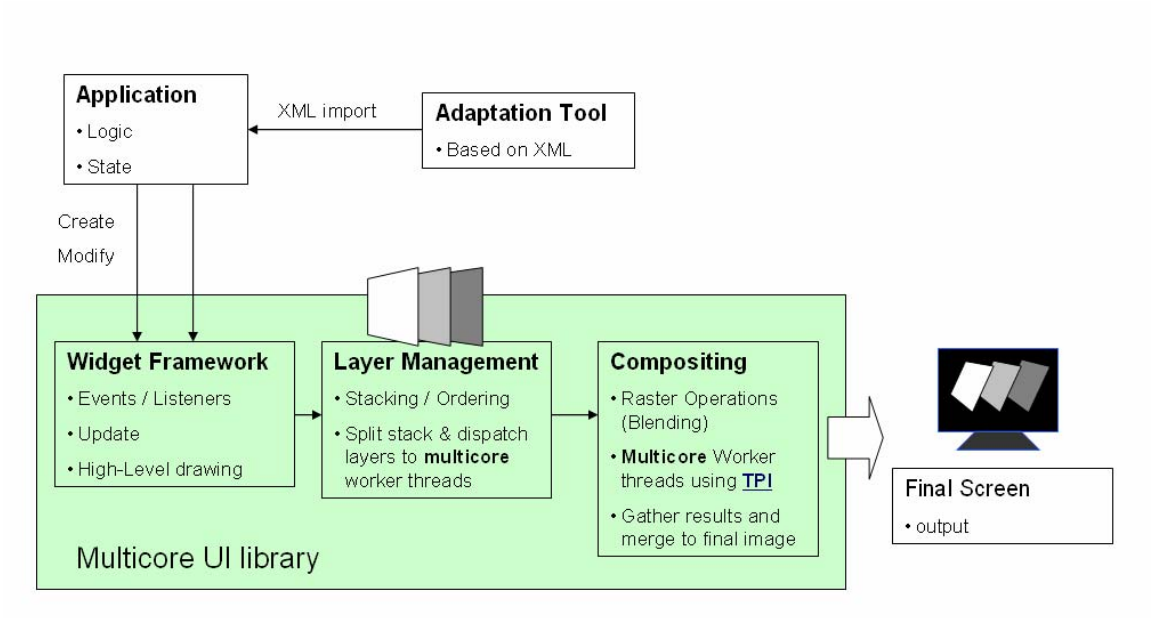

Figure 6. Overview of how various components interact

Besides instantiating new GUI widgets and handling events, applications will also to some extent have to interface with the framework's Renderer and Compositor sub-systems in order to control display update and possibly perform lower-level graphics tasks.

4.5.1 Dependencies

An application creates its GUI elements and layout using the toolkit classes based on the XML description provided by an UI adaptation service. Although some tasks regarding management of state and logic are already provided by the GUI framework, typically large parts of these tasks also have to be implemented on the application side. This results in tight intertwining of GUI and application logic code. However, the GUI library will not perform migration of data and state information stored inside GUI widgets – it is the responsibility of an external entity to get these values from the respective widgets and hand them to the OPEN platform.

To maximize integration and interoperability with various other OPEN components and systems, the toolkit will be implemented in the Java programming language. Additional C++ code is also used in order to gain maximum performance for multimedia applications. E.g. the UI library will use

NLE-IT’s C++ Task Programming Interface (TPI) to distribute workload over all available CPU cores.

5 Supporting Application Logic Reconfiguration in the architectural framework

In the first section we will introduce which types of adaptation we will consider and how they fit into the basic modules shown in Figure 4 (OPEN Architectural Framework). Afterwards we will give a rough overview of the interplay between migration and application logic reconfiguration.

5.1 Application Logic Reconfiguration of Service-based Dynamic Adaptive Systems

We consider dynamic-adaptive systems as systems built from a set of services that work together to perform some kind of tasks that are useful for application users. In addition, to established component-based applications, the behaviour of dynamic-adaptive applications adapts during runtime to the needs of the current user and his environment.

Components are software entities that realize specific services that are described by and accessed through interfaces. Other components can define dependencies to provided services by defining them as required. Provided and required services have to be linked together in order to enable for example method calls from one component to another. The Figure 7 shows a set of component instances running on a PC that offer and require different services, described by interfaces. Required services are depicted as semi-circles, while provided services are depicted as circles. That notion follows the UML 2 standard [UML]. In order to run the application, according provided and required services have to be connected. The decision of which services to connect will be made in the middleware.

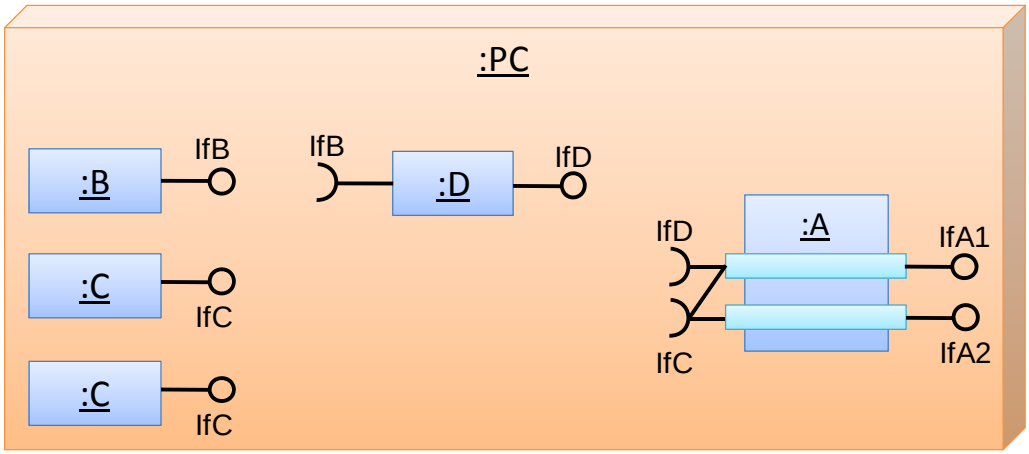

Figure 7. Component instances running on a PC ready for wiring and interacting together. If a component instance holds all required references to service implementations, it becomes runnable and will provide the given services to other components.

Therefore, the task of the middleware is to ensure that the wiring of component instances always fits to the user's needs on the one hand, and to the usage context on the other hand. The wiring can be changed during runtime based on context information, user preferences, or currently available component instances. The rewiring of components is one aspect of considered reconfiguration types.

Currently we distinguish three types of reconfiguration [NKA+07]. The first one we call *Service Implementation Adaptation*, where the behaviour of a single service changes by replacement of the implementation during runtime based on any available context information. One example is that a speech recognition service adapts its recognition algorithm based on the stress of the user. This could be useful because the speech of a relaxed person is very different than that of a stressed person and therefore there would be a lack of reliability in changing situations. That type of adaptation could be triggered by the middleware and performed by the application component itself. The second type of adaptation we call *Service Usage Adaptation* which is depicted in Figure 8.

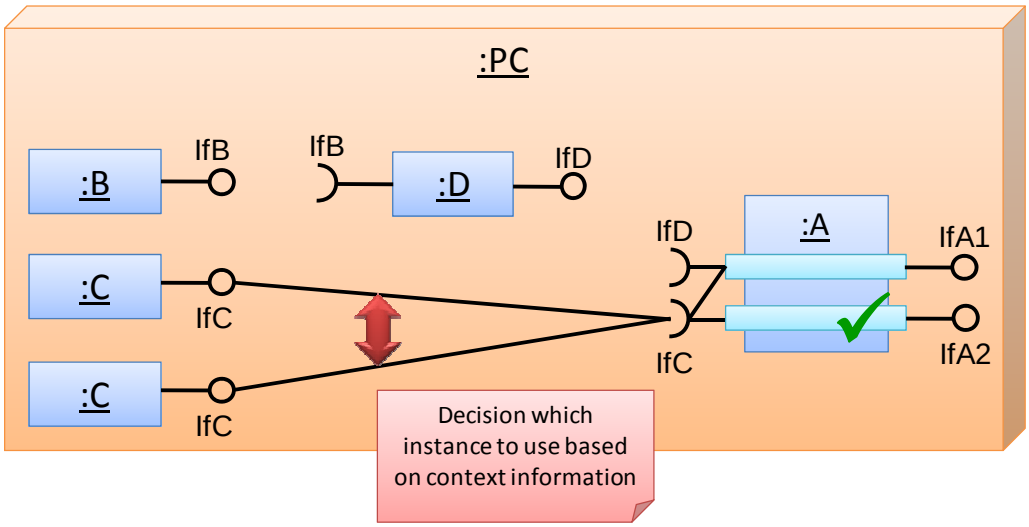

Figure 8. This figure illustrates the *Service Usage Adaptation*. If two implementations of the same service are available, the middleware has to decide which one to use in the current situation.

In this type of adaptation an application component instance, which is currently in use is replaced by another application component instance, which is more appropriate in the current context. A very powerful but resource-intensive component instance could for example be exchanged by a less comprehensive but more energy-saving component instance if the battery of the device is getting low. This type of adaptation is performed by the middleware. The third type of reconfiguration we call *Component Configuration Adaptation* which is illustrated in the next figure (Figure 9).

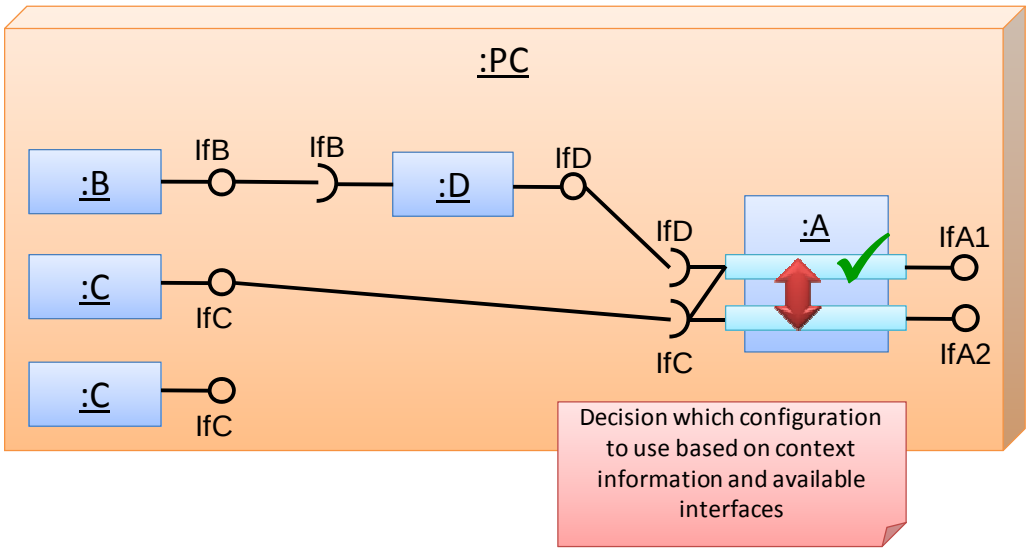

Figure 9. The figure illustrates the component configuration adaptation. If interfaces *IfC* and *IfD* are available, component A provides *IfA1* to other components, if only *IfC* is available, component A will provide *IfA2*. The active configuration is tagged with a green check mark. Depending on which application component configuration is active, the behaviour of that component may be different.

Therefore, a component can consist of several configurations as shown in component A in Figure 9. Each configuration is a mapping between provided and required services. If all required services of a configuration are available, this configuration will get activated and the provided services of that configuration will be available to other components in the system. In addition, the behaviour may change depending on which configuration is currently active. The active configuration is tagged with a green check mark. That means that during reconfiguration the functionality of the application can change due to the instantiation of new components with new services.

In the first version of the middleware provided in WP4 these types of adaptation will be realized. The main trigger for adaptation will be the migration of application components, and the change of context information. To do this, the WP4 will interact with several building blocks from the OPEN architectural framework shown in Figure 4. In the following we will shortly describe how the middleware cooperates with these building blocks.

- **Access to the Component Repository**

  Each component instance within an OPEN system has to be published at the Component Repository. In order to decide which of the three types of adaptation can be used, the application logic reconfiguration unit has to access the component repository in order to decide which instance is the best in the current situation. The application reconfiguration unit is also interested in the information if new component instances, and therefore new services, get available or other component instances disappear. Therefore, it will register for these kinds of events at the component repository.

- **Interaction with the Trigger Management**

  A trigger is an event which tells the system that a migration has to be performed. Therefore, it will also be used to initiate the adaptation of the application. To do this, specific trigger definitions will be stored at the trigger management, that is conditions when to trigger migration and adaptation. A trigger could for example define that the application or parts of it have to be adapted if the battery power is below 50%, below 30%, and below 10%.

- **Interaction with the Context Management**

  The Context Management will be used to obtain high-level context information in order to decide how to adapt the application, that is, to decide which type of adaptation to apply and how to realize it. If the battery power for example is between 50% and 30%, *Service Implementation Adaptation* could be applied. If the battery power is between 30% and 10%, maybe *Service Usage Adaptation* will be useful.

- **Interaction with the Policy Management**

  The policy management is accessed by application logic reconfiguration in order to obtain specific guidelines or preferences for adaptation. Maybe the user for example wishes to slow down the application in case of low battery instead of limiting the functionality.

The first version of application logic reconfiguration will deal with services implemented in Java running on Windows or Linux. Since the addressed applications are distributed applications, where services may run on different devices, an appropriate middleware for supporting those kinds of applications will be used, like CORBA (Common Object Request Broker Architecture) or OSGi (*Open Services Gateway initiative).* Those existing middleware systems will be used as a basis for realizing the communication between all required architectural building blocks used for application logic reconfiguration and application components. Components, which originally were not designed to communicate over a specific middleware, can be integrated through simple wrappers.

In order to perform reconfiguration, each application component will have to implement specific interfaces through which the application reconfiguration unit can control the lifecycle of each component and initiate reconfiguration tasks. The concrete methods will be described in deliverable D4.1.

Application logic reconfiguration is only one part of work to be done in WP4. The second part considers the migration of single or all components of an application. In the next section we will therefore introduce the *Migration Control and Management* unit depicted in Figure 3 (OPEN Architectural Framework).

5.2 Interplay of Application Migration and Application Reconfiguration

The second task of WP4 is to integrate work of WP2, WP3, and Application Logic Reconfiguration of WP4 into a common OPEN migration service platform. Therefore, the *Migration Control and Management* unit will interact with other

modules shown in Figure 4 (OPEN Architectural Framework) using specific interfaces. In order to realize the migration functionality, solutions developed in WP3 will be applied concerning migration handling, context management, policy management, trigger management, and device discovery. Solutions for user interface migration and adaptation come from WP2 and will be integrated together with application logic reconfiguration concerned in WP4.

The following figure (Figure 10) shows one example of how application migration and application logic reconfiguration could interplay. There is for example a running application on the PC which the user wants to migrate to his PDA. Among other actions, the middleware has to transfer the state and in some cases the code of each component to the PDA and start these components with their states on the target device. This part is basically performed by the migration manager. The application logic reconfiguration will then try to adapt the application to the user's needs and the available resources.

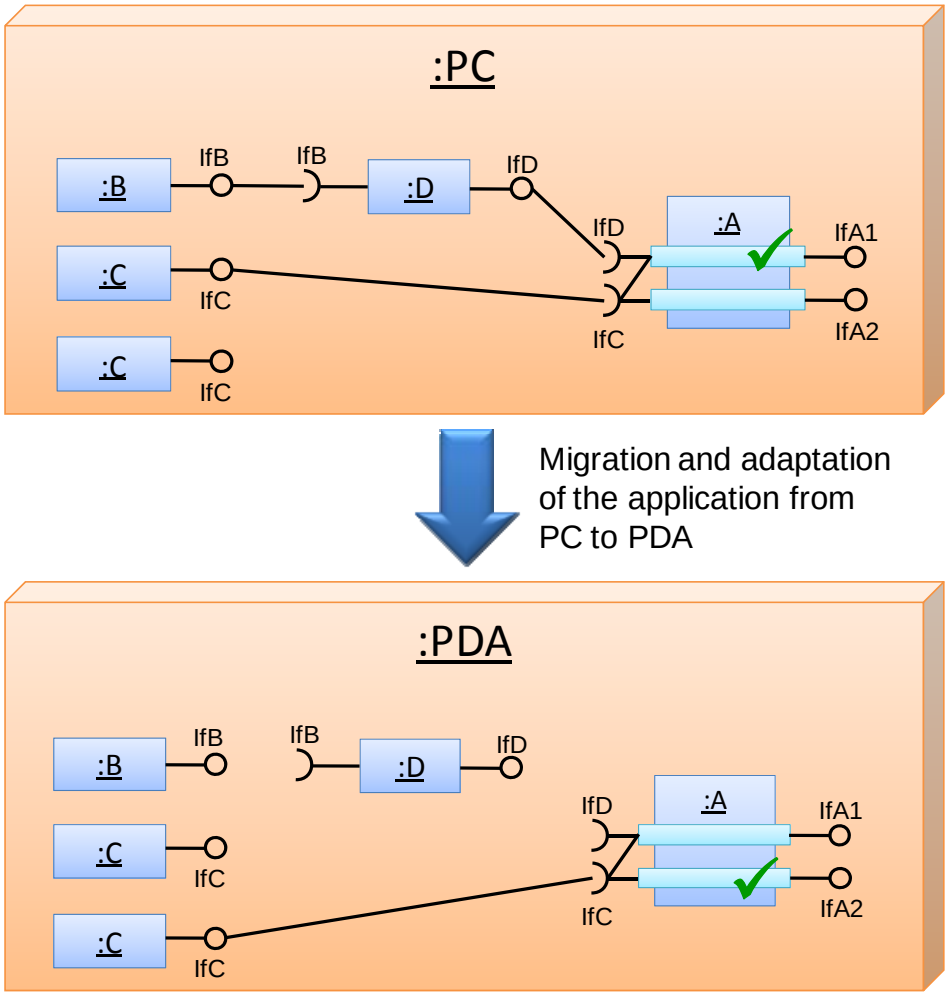

Figure 10. The figure shows the interplay between migration and application logic reconfiguration as described in the section before. In this case *Service Usage Adaptation* and *Component Configuration Adaptation* are performed.

This is only one example of how migration and reconfiguration work together. Many other scenarios are possible. One example is that only parts of the application migrate from one device to another while other components stay at their source device. A more detailed discussion of further migration/adaptation scenarios and middleware solutions will follow in deliverables D4.1 (Solutions for application logic reconfiguration) and D4.2 (Migration service platform design).

Dependencies of the Migration Control and Management unit are the same as described in the previous section. However, the reasons for them are slightly different. The migration control unit is notified by the Trigger Management unit, if migration is required. For migration control and management, the location of specific component instances becomes important. Therefore, information about that has to be managed and stored for example at the component repository. Furthermore, a component, which has to be migrated, has to be stopped, their state has to be stored, their code has to be transferred to the target device, and the component has to be started again.

The communication infrastructure and supported platforms are basically the same as described in the previous section. In addition, the middleware has to initiate and perform migration of services and their code. Some middleware systems like CORBA or OSGi for instance support some kind of mobile code. Such a middleware could be used as basis for realizing the migration of whole services from one device to another. Since we will use Java, there has to be a running virtual machine on the target and source device in order to perform migration. The communication between application components and middleware components could be realized similar as described in the previous section.

6 Communications and Context management Middleware part of the architectural framework

The goal of WP3 is to design and evaluate the detailed communications and context management solutions for migratory services. The communication relations thereby include the message exchange between the devices participating in the UI/application migration, as well as to remote migration-unaware communication end-points (correspondent nodes) participating in the distributed application. From Figure 4 the following functions are described in this document:

- Context management
- Migration orchestration
- Trigger management
- Session management
- Policy management
- Security
- Performance monitoring
- Clock/flow synchronization
- Mobility support
- Device discovery
- Service Enablers Interface

In the following an overall description of each function in the middleware is presented. The description contains; a brief functional specification, an introduction to which other functions a function interacts with and concerning what information, a consideration on what requirements and scenarios are specific for the function and finally potential candidate solutions for the function.

The descriptions in the following are first high-level functional descriptions based on the functional requirements derived in [OPEN D1.1]. In section 7, examples are given of how the functions interact on an overall level. The function details (e.g. interfaces, requirements, candidate systems, specific interactions) for the communication and context management middleware are highly dependent on which scenario and network architecture they run in. For this reason, a detailed analysis is described [OPEN D3.1] of networking scenarios and consequences of using these migration support functions in the scenarios.

6.1 Context management

The context management system generally handles information distributed in the network and provides easy access for services, application and networking components to their needed information. Thus it is responsible for collecting, storing, processing and delivering relevant information from different sources of

context to functions in need information. As described earlier, context refers both to raw sensor data, or higher level, processed context. Context providers can cover anything from environment changes (the user is standing in front of the device) to very device specific information, such as the remaining battery life. In OPEN context it is used for many purposes as envisioned and required in Appendix of [D1.1]. For example, it is used in the trigger management function that is capable of issuing a migration trigger based on a context change (e.g. once the battery capacity is below a certain threshold). In addition, application logic reconfiguration, UI migration and mobility (see e.g. requirement no. 35 in [D1.1]) functional modules also require use of context and thus access to it.

The context management function must be capable of collecting information from a wide range of sources, both on terminal devices and sensors and in the network (e.g. the user's presence status, or information collected from social networks such as Facebook, Twitter, etc.). To accommodate this, the context management function should support distributed collection of information and either distributed or centralized processing of the information.

Context source look up and data retrieval is achieved using a semantic query language, supported by a context broker function and the rest of the management framework.

At present existing solution related to context management have been researched for some time, and in particular a solution originating from the MAGNET Beyond project is under investigation as a possible basis for the OPEN project. The framework here is denoted Secure Context Management Framework (SCMF) and is presented in [MBD2.3.1], [Bauer06], [Sanchez06] and [MBD2.3.2].

Referencing to Requirement no. 65, 89, 135, 136, 34 and 59

6.2 Migration orchestration

Migration orchestration controls the migration process – from received trigger to successful use of the migrated service. This may entail pre-planning of what application parts to migrate where, how and when in the migration process this happens. An example of migration orchestration is when migrating a user interface: then various migration services need to be accessed: the reverse engineering for building the logical description; the adaptation service; the state mapper, which maps the state of the source user interface in the target one; the user interface generator for the target device; and the target device itself for uploading the migrated user interface.

References to Requirement no 6, 13, 82, 30, 48, 51, 79, 80, 139 and 126,

6.3 Trigger management

This module analyses contextual information changes and decides whether or not a migration should be activated through issuing triggers. Simple triggers can be provided by the context management function itself. Here, simple thresholds on context information values can be used or even fusion of different pieces of context information may apply.

Even more complex triggers, e.g. based on inferred knowledge from available context information, network information or based on user-input, may be generated and handled by this function.

References to Requirement no 89, 13, 82, 86 and 37.

6.4 Session management

Ongoing networking, security or application sessions must be respected when performing migration to provide continuous services.

Examples of such sessions are

- application session: logged in on website, a stateful server (http session and corresponding timeout), service/device registration (e.g. from UPnP)
- security session: established security association (authentication, authorization, credential/key exchange)
- network session: NAT connections, stateful firewalls, TCP connections, multicast group assignment

The session management function helps ensure that sessions can continue during migration. This functionality would apply to e.g.

References to Requirement no 118, 119, 126, 43, 77, 45, 54 and 64.

6.5 Policy management

A policy management function handles that in some cases migration is not allowed due to requirements (from user or provider or 3rd party), or conversely that a migration must be effectuated in a given situation. This is typically based on contextual calculations, based on functionality requirements, application context, user context and profile, security parameters, etc.

References to Requirement no 142, 66, 49, 52 and 98.

6.6 Security

In several scenarios, security issues arise. Problem such as ensuring privacy protection, prohibiting eavesdropping on wireless transmissions or preventing malicious interactions during the migration process must be addressed. The security function handles support for secure communication by managing keys and ensuring authenticity of participating entities.

If the application uses encrypted channels for communication, the existence of these must also be assured during migration, so that secure service is not interrupted unintended.

References to Requirement no 38, 15, 23 and 118.

6.7 Performance monitoring

This module is aimed at monitoring the performance of entities that are involved in the migration environment, because such performance could possibly affect the migration process. Indeed, as network performance, device performance and

similar could potentially be important context metrics for migration triggers, performance monitoring functions are needed as context providers for the context management system. Therefore, this module provides input to the Context Management module,

Lower-level system performance such as efficiency of multi-core utilization high-level application QoS metrics are also potential inputs for a performance monitor.

References to Requirement no 21, 18, 32 and 102.

6.8 Clock/flow synchronization

Different streams, existing or new, need to be synchronized in order to provide continuous service when migrating/distributing an application. An example is enhancing an audio call with video capabilities. When establishing the video stream, this should be synchronized to the audio stream already running.

To apply such synchronization some sort of clock synchronization must be in effect. Also, if application state is dependent on time, synchronized time between involved devices must be present in the migration process.

References to Requirement no 145 and 58.

6.9 Mobility support

Mobility support is required in the middleware to handle user mobility so that corresponding nodes do not need to be aware of and handle mobility. Correspondent nodes are application node(s) in remote networks. As these could be non-OPEN-aware application peers, such as regular web- and application servers which are relevant in web-service scenarios and applications, it is not feasible to expect that they can handle user mobility themselves. Thus to support users moving around, e.g. from one network to another or from a fixed infrastructure network to an ad-hoc network, the functionality must be embedded in the OPEN platform and it must provide transparent mobility support to non-OPEN service providers.

References to Requirement no 104, 35 and 91.

6.10 Device discovery

The main aspect in migrating services and applications is enriching user-experience. This is done by utilizing additional resources and/or devices available. To use these additional entities, their presence need to be discovered and their capabilities established. This is handled by the device discovery function.

Discovery can be performed in several manners; locally (short-range communication) or centrally (using a commonly accessible registry). The objectives for discovery are also multiple:

- device discovery (network presence)
- service discovery (what services are provided by a device)
- resource discovery (which resources, e.g. battery lifetime, processing power, storage capabilities, etc. are offered by a device).

References to Requirement no 20 and 33.

Regarding the communications with other modules, the Device discovery can interact with Context Management module in the following way: whenever there is a change of device, the Device discovery module provides such information to the Context Management module, which is in charge of maintaining updated the information about the current context.

6.11 Service Enablers interface

OPEN Platform and migratory applications access network capabilities through a Service Enablers Interface. This interface provides several adapters for every network elements type that has to be accessed. The network architecture and the low level interface are hidden abstracting the underlying layer and providing common functionalities to OPEN middleware and to all the applications that use the middleware.

Service Enablers can be resources external to the OPEN platform providing some network capabilities that can be accessed by the OPEN platform using the Service Enablers Interface component. Such resources might be:

- Presence system where customers and services or applications can publish their presence status or verify presence status of their buddies
- Location through which location information of SIM cards are accessed
- Network Context and access layer information such as:
    - type of network in use (in terms of GSM, GPRS, UMTS, WLAN, Bluetooth, …),
    - type of service in use
    - information regarding the URL accessed by the customer.

Service Enabler Interface performs privacy related functionalities enabling or denying access according to general privacy rules, customers preferences and profile.

References to Requirements are indirectly through many other subsystems, since e.g. location is most often provided through such an interface.

7 Introduction to a few Scenario Realizations

Figure 11, Figure 12 and Figure 13 illustrate the anticipated action flows between middleware functions when context changes and a trigger is generated (Figure 11) and when a trigger is issued and the migration is performed (Figure 12).

7.1 Context change

Figure 11 illustrates the relations between the described middleware components during a context change and a subsequent issue of a migration trigger.
The sequence leads to a migration trigger. The scenario includes the situation in which the user is moving around. The user part considers not only the actual user behaviour but also how it is detected through some technology, e.g. a user localization infrastructure (through GPS or RFIDs or something else). The user movement has an impact on the devices available for migration because such devices should be nearby the current user location. Thus, the user movement event should be sent to the device discovery, which will update the current list of devices and services available for migration and send such information to the context management, which can then send the event of context change to the trigger manager.

Thus, the user mobility implies a change in the set of devices available for migration. This change is detected by the device discovery, a kind of context provider, which is continuously active, communicating with the context manager, which is the entity that generates high-level events for the trigger management, continuously calculating if the context change will result in a migration trigger, as it happens in the figure.

In general, some context changes may go about unnoticed by the user, and others may results in a migration trigger. In addition, the trigger management may be complex, depending on various combinations of events.

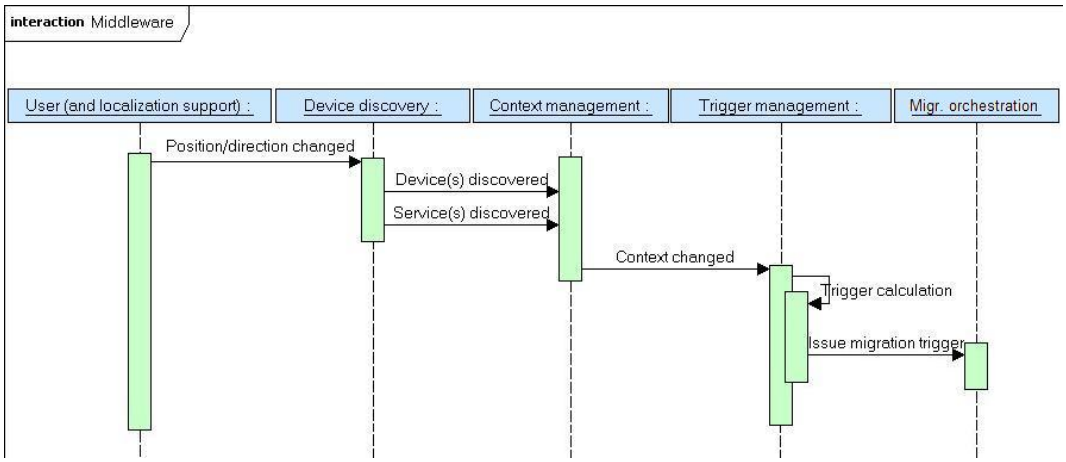

Figure 11: Example interactions between middleware functions on context change and migration trigger

7.2 Performing migration

The second example, depicted in Figure 12, illustrates interaction between middleware components during the actual migration, i.e. reaction to a migration trigger. This trigger may be automatically inferred by the system or manually issued by the user, as previously described.

The sequence is initiated upon the migration control function receiving a migration trigger from the trigger management function. The migration control function uses a set of support functions to carry out the migration. In this example the function handling migration policies, security, sessions and network are used during the migration to allow for the application flows to be migrated. As a later step in the migration, several communication/message flows of a running application might be in need of synchronization. This is illustrated by the Sync function referring to the clock and flow synchronization functions. Descriptions of all of the applied functions can be found in section 6.

7.3 Performing adaptation and reconfiguration

In addition to migration, a trigger can also initiate adaptation of the application logic to the target device. That includes adaptation to available resources, user's needs and other aspects. Figure 13 illustrates the interaction between involved middleware components during the adaptation and reconfiguration phase. Depending on the application, different kinds of trigger are needed. Therefore, the reconfiguration control unit registers for specific events at the trigger management component specific for each running application. The reconfiguration control also has to be aware of available component instances which can be used for reconfiguration, e.g. for deciding which of two instances of a component to use in the current situation. Therefore, it registers for according events at the component repository. These events can also initiate reconfiguration and adaptation of the application. The use of both types of adaptation triggers is shown in Figure 13. In both cases, the reconfiguration control (see Section 5) may change connections between component instances, replace instances by other more appropriate instances in the current situation or reconfigure single component instances by adapting their behaviour to the current usage context. The gathering of context is done by accessing the context manager during the reconfiguration and adaptation phase. This context information is then used to find the best configuration and adaptation actions in the current situation.

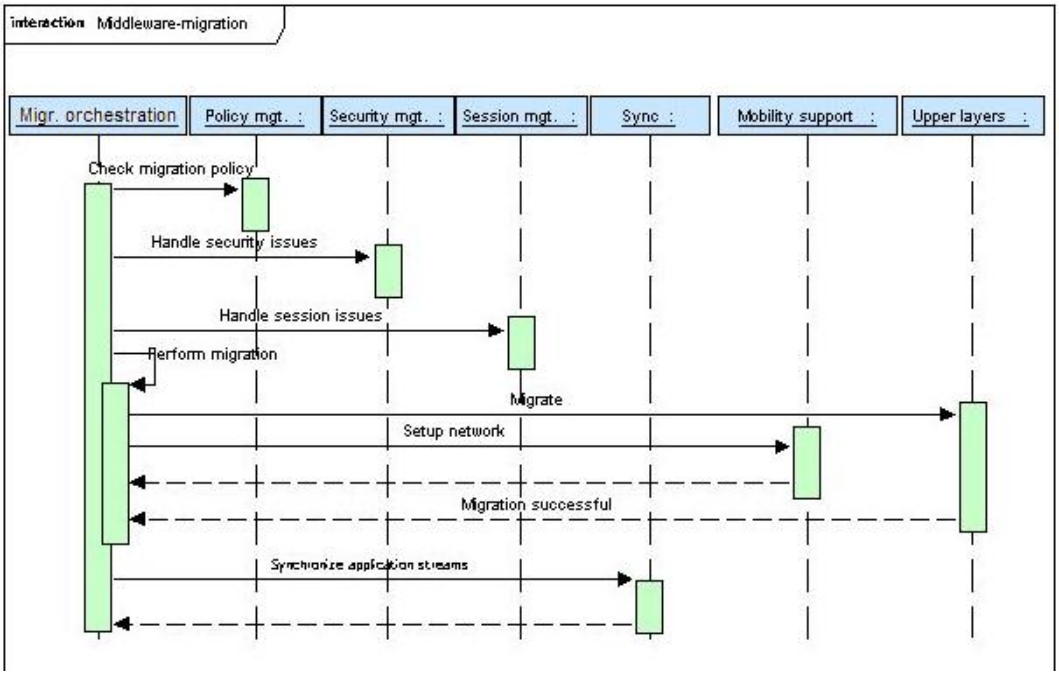

Figure 12: Example interaction between middleware functions during migration

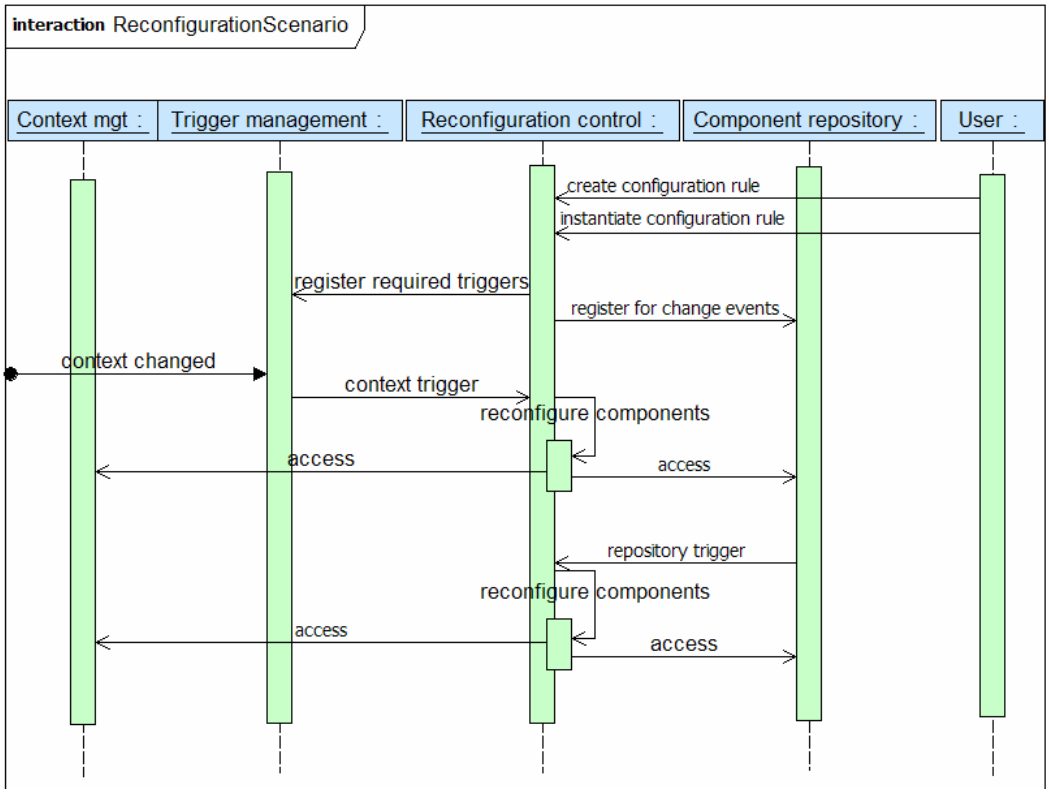

Figure 13: Example of interaction between reconfiguration components and the trigger management for triggering logic/application reconfiguration

7.4 Example of a Server-Supported Migration Scenario

In this section we analyze a scenario in which a user is interacting with a Web application through a browser on a computer (PC). Differently from the first example, in this case we consider a migration triggered on explicit request of the user.

In this scenario realisation the application is migrated from the PC to a PDA through the use of a migration server, which is one of the possible solutions for the OPEN Platform (as it has been highlighted in Figure 3) and the one currently implemented in the migration prototype developed at ISTI. However, further developments of the prototype could also foresee other solutions, different from the server-based one. In any case, it is worth pointing out that, since this is an example of a networking-based migration scenario, further networking scenarios will be further elaborated in next D3.1.

At first, there is a device discovery phase, which allows the migration environment to identify the devices available for migration. If we consider the access to a web application from a desktop system, such an access could go through an intermediate server, which also has proxy capabilities. Before passing the requested Web page to the client, the server includes, within the page retrieved from the Application Server, a JavaScript excerpt whose purpose is accessing the state of the page at migration time, so that such a state can be preserved for being used afterwards. In order to be migration-enabled, the devices should run a migration client, which is used for supporting the device discovery phase (through such a thin client the devices announce their presence to the others) and allow the user to select the target device from the list of devices currently available and then trigger migration. When the migration is triggered, the script inserted sends the state of the Web application, which depends on the user interactions to the migration server. Then, the migration server creates an adapted version for the target device (a PDA in our scenario), associates the state to the corresponding elements of the target version, generates the corresponding implementation and activates it at the point the user left off in the source device.

In the sequence diagram shown in Figure 14 we depict the communication flows occurring between various migration components during a Desktop to PDA Migration. The process has been divided into different phases, distinguished into ”Service Use” (which refers to a normal use of the service by the user), ”Pre Migration” (which are all the operations that have to be carried out for preparing the actual migration), ”Migration” (the step in which the migration is actually carried out), ”Post-Migration” (all the actions that might be done just after migration, before coming back to a normal use of the service).

The scenario starts with a user who is currently using a service through a desktop platform. The module dedicated to discovery the devices available in the current environment and to update the list of devices accordingly together with information regarding their characteristics (the ”Discovery mgt” in Figure 14), discovers in this case a desktop and a PDA. At a certain point the user asks for migration, then a migration trigger is sent by the currently used platform (desktop)

to the ”Migration Mgt” module. Then, the currently saved state is sent to the migration server, which also asks to the Discovery module information about the capabilities of the target device involved in the migration. Afterwards, the reconfiguration of the application logic, the adaptation of the user interface to the new device and the association of the state to the adapted interactive application are carried out, and the reconfigured result is sent to the target device (the PDA in this case) for being loaded. Then, in the Post Migration phase there is a step in which the application device currently running in the source device is terminated, so that the user can continue the interaction with the service through the new device.

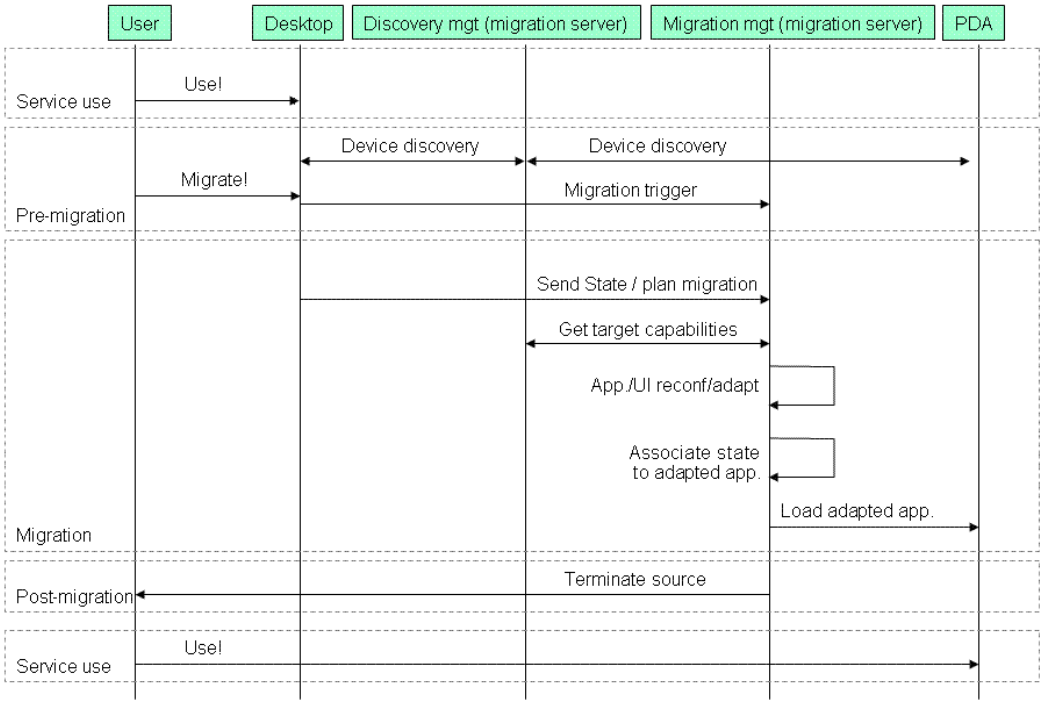

Figure 14. Sequence diagram describing the interaction between middleware migration functions for device discovery, migration triggering, state preservation and application logic reconfiguration

8 Conclusions and Future Work

In this deliverable we presented our first proposal for the OPEN Service Platform architectural framework. After describing the main concepts and phases involved in the migration process, we specified the overall architectural framework of the Migration Service Platform (MSP), with a high level design of the basic components and the interfaces between them. Finally, we outlook some scenario realisations in order to better describe how the middleware functions work together in more concrete examples.

The architecture described in this deliverable is a first draft and it will be subject to change in the continuation of the project. More in detail, further modifications to the various architectural parts will be elaborated and documented in a number of planned deliverables: D2.1 (“Early infrastructure for migratory interfaces”), D2.2 (“Document about architecture for migratory user interfaces “), D3.1 (“Detailed Network Architecture”), D4.1 (“Solutions for application logic reconfiguration”) and D4.2 (“Migration service platform design”). Such modifications will also be taken into account for elaborating a revised version of Deliverable D1.2, which has been planned for Year 2.

9 References

[GSS+02] Garlan, D., Siewiorek, D., Smailagic, A., Steenkiste, P. Project Aura: Toward Distraction-Free Pervasive Computing. IEEE Pervasive Computing, Vol 21, No 2 (April-June 2002), 22-31.

[PSS08] F. Paternò, C. Santoro, and A. Scorcia, Preserving Rich User Interface State in Web Applications Across Various Platforms, Proceedings EIS'08, Springer Verlag.

[Dey00] A. K. Dey, "Providing Architectural Support for Building Context-Aware Applications", PhD thesis, Georgia Inst. Tech., USA, Nov. 2000.

[OPEN D1.1] Requirements for OPEN Service Platform, OPEN Consortium, June 2008, Faatz, A. and Goertz, M. (eds.)

[OPEN D3.1] Detailed network architecture, OPEN consortium, Feb 2009

[MBD2.3.1] Martin Jacobsen, et.al., Specification of PN networking and security components, Deliverable 2.3.1, December 2006, MAGNET Beyond, IST-027396

[MBD2.3.2] Martin Jacobsen, et.al., PN secure networking frameworks, solutions and performance, Deliverable 2.3.2, June 2008, MAGNET Beyond, IST-027396

[NKA+07] D. Niebuhr, H. Klus, M. Anastasopoulos, J. Koch, O. Weiß, A. Rausch. "DAiSI – Dynamic Adaptive System Infrastructure". IESE-Report No. 051.07/E, Fraunhofer Institut für Experimentelles Software Engineering, 2007

[Bauer06] M. Bauer, R.L.Olsen, M. Jacobssen L. Sanchez, J. Lanza, M. Imine, N. Prasad, Context Management Framework for MAGNET Beyond, IST Summit 2006, Mykonos, Greece, 2006

[Sanchez06] L. Sanchez, J. Lanza, M. Bauer, R. Olsen, M. Girod-Genet, "A Generic Context Management Framework for Personal Networking Environments", Proceedings from 1st Workshop on Personalized Networks, San Jose (CA), July 2006

[UML] Object Management Group. UML 2.1 Superstructure and Infrastructure Specifications. November 2007.