OPEN Project

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

Title of Document:	Final application requirements and design
Editor(s):	Giancarlo Cherchi, Francesca Mureddu
Affiliation(s):	Arcadia Design
Contributor(s):	All Open Partners
Affiliation(s):	All Open Partners
Date of Document:	January 2010
OPEN Document:	D5.3
Distribution:	EU
Keyword List:	Prototypes, Requirements, Design, Guidelines
Version:	1.0

OPEN Partners:

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

OPEN Project

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

ABSTRACT

This document presents the final requirements for migratory applications, provides the general guidelines for designing and developing them in compliancy with the OPEN technology, and describes the integrated prototypes that are being developed as concrete examples of migratory applications integrated with the OPEN Migration Service Platform.

OPEN Project

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

TABLE OF CONTENTS

1. INTRODUCTION

The main objectives of this deliverable are three: first, to list the final requirements for migratory applications; second, to provide general guidelines for designing a migratory application, making it compliant with the OPEN technology and able to take advantage of the capabilities of the Migration Service Platform (in the following "MSP"); third, to describe how the development of the integrated prototypes have to be carried out to fulfill all the platform specifications.

The starting point was the analysis of the requirements presented in D1.1 [1], from which the applications related requirements have been extracted and split in generic and application-specific ones. From the generic ones the first guidelines came off, whereas from the specific ones a first design of the applications has been done and described in D5.1 [5].

Besides the work in D1.1, other sources of information have been used to obtain the results of this document, including D5.1, D1.3 [2], D1.4 [3], D4.2 [4], and D6.5 [6].

From D5.1 some specific requirements have been extracted and they will be prioritized according to the evaluation results and to the experience on the integration with the migration platform.

From D1.3 all the common agreed definitions of the terms used throughout the project have been taken.

From D1.4 the OPEN platform capabilities have been considered in order to describe how to design applications that exploit them.

From D4.2 the technical requirements and specifications for building a migratory application integrated with the OPEN platform have been taken into account, as well as the definition of the interfaces among the MSP modules and the objects and messages exchanged.

From D6.5 evaluation results have been exploited for defining the final user requirements and for drawing into the final design of the integrated prototypes.

The outline of this document is as follows. The first chapter is dedicated to illustrating the final requirements for a migratory application; the second chapter discusses how to design a migratory application, describing its aspects, how to make an application OPEN aware, and some usability related issues that must be considered; the third chapter presents some application use cases, which serve as concrete examples of development of migratory applications integrated with the MSP.

2. FINAL REQUIREMENTS FOR A MIGRATORY APPLICATION

This section contains the final requirements for a generic migratory application. The requirements that are specific for each prototype can be found in the corresponding use case section. The main sources from which these final requirements have been extracted, elicited or elaborated are D1.1 (for general application requirements) and D4.2 (for OPEN compliancy requirements).

2.1. GENERIC MIGRATORY APPLICATION REQUIREMENTS

The first set of requirements that has been addressed for the development of the integrated prototypes was extracted from D1.1 considering only the application related ones. Among these, the requirements applied to a generic migratory application have been selected and reported below without any modification. Moreover, some requirements that were originally tailored to a specific scenario have also been selected, since, after an in-depth analysis and the first integration experiences and evaluations, it has been deemed that they can also be applied to a generic migratory application. A synthesis of the selected requirements is reported in the following tables, which consider only the more relevant fields for the scope of this document.

ID	155
Description	The application must support the pause feature
Type	Functional/ application
Typology	Application Logic
Rationale	The user needs time to change device and UI

ID	59
Description	Application should be able to adapt its behavior to the context / user's needs
Type	Context
Typology	Application Logic, Migration Service Platform

Rationale	Not only the change of an interface-implementation is relevant, but also the change of behavior of single components during runtime without replacing them.

ID	53
Description	Web applications must be able to control the middleware installed in the device via the browser. Alternatively the browser must be OPEN-aware
Type	Control
Typology	N/A
Rationale	The applications must interact with the OPEN work just like any other one

ID	73
Description	Applications must allow multiple concurrent heterogeneous input sources
Type	Input/Output
Typology	User Interface, Migration Service Platform
Rationale	The multi-player variant requires cell-phone and PC keyboard at the same time

ID	10
Description	The application must support being split into components, such that different pieces can be sent to different devices

Type	Reconfiguration
Typology	N/A
Rationale	Splitting is a core innovation, and this scenario requires to split the game

ID	107
Description	We need good applications that involve long operations making it worthy to migrate the application. It's arguable if such long operations would be present in good applications. (The applications must be long enough to make the migration worthy)
Type	Usability
Typology	N/A
Rationale	No matter how nice, migration is a hassle, so we need to make it worthy

2.2. REQUIREMENTS FOR OPEN COMPLIANCY

The second set of requirements that has been addressed for the development of the integrated prototypes was elicited from D4.2, considering the issues related to the integration of an application with the MSP. A list of the selected requirements is reported in the following tables.

ID	164
Description	The application must be able to send and receive XML-RPC messages
Type	Interface/Application
Typology	MSP

Rationale	The XML-RPC protocol Is the common accepted protocol for handling the communication between the MSP and the application

ID	165
Description	The application must be able to handle asynchronous requests and receive asynchronous answers
Type	Interface/Application
Typology	MSP
Rationale	The MSP needs asynchronous communication as well, since many communications with the application are asynchronous

ID	166
Description	The application must be able to retrieve and restore its state.
Type	Functional
Typology	Application Logic, User Interface
Rationale	The continuity of a task must be ensured even when changing device

ID	167
Description	The application must provide policy rules

Type	Functional, Operational
Typology	Policy Management and Policy Enforcement
Rationale	The platform needs policy rules to ensure safe migration

ID	168
Description	The visible components must contain information about the UI elements which they belong to.
Type	Interface/Application
Typology	User Interface, Application Logic
Rationale	In partial migration, only the UI of migrated components must be adapted

ID	169
Description	The component must provide hardware, software, network and UI requirements to work properly in the target device
Type	Functional
Typology	Application Logic, Mobility Support, User Interface, Context Management
Rationale	The MSP must perform a matching between component's requirements and device's capabilities

3. DESIGN OF A MIGRATORY APPLICATION

As defined inside the OPEN project, migratory applications are special kind of applications that are able to follow users, sense the users' context, and adapt to the changing context, e.g. set of available devices, while also preserving the continuity of application sessions, thereby ensuring the continuity of the tasks supported by the application.

In short: Migration = Device Change + Adaptation + Continuity.

To ensure the correctness of the whole migration process, the design of a migratory application must consider not only the mere application's functionalities and its basic requirements, but also a set of specific aspects that are related to these particular kinds of applications. To be more precise, the application designer must consider all the parts of the application involved in the migration process, the guidelines for the integration with the migration platform, and the usability of the migration experience.

In the following, all these topics will be addressed in subsequent dedicated sections.

3.1. ASPECTS OF A MIGRATORY APPLICATION

As mentioned above, the MSP is able to support device change, adaptation and continuity. Therefore, a migratory application that wants to exploit the platform features must be adapted or designed in an appropriate way. The support to migration and adaptation offered by the MSP involves different aspects of the application that have to be considered and designed in a suitable way, in order to properly integrate the application with the platform. In particular, the aspects involved in the migration process are: the Application Logic, the User Interface, the Network, the Context and the Policy. Each of them is separately handled through a specific module inside the MSP. For this reason, a migratory application should be organized taking into account this kind of conceptual partitioning during the design phase.

An important point to be considered while designing a migratory application is the user perception and awareness of the migration process. Since the migration process may involve different devices and different parts of the application, the user might not be totally aware of what is happening, thus the application must provide suitable mechanisms to inform her/him about its actual configuration (e.g. through alerts, icons, text messages, etc.).

In general, an application is made of two main parts: the Application Logic and the User Interface; the first one performs the required data processing, whereas the second one is aimed at presenting the content and providing input controls to the user.

In the migration process the Application Logic has to adapt to the different contexts, including change of resources that occurs among different device types and to support all the migration types (see D1.4, section 1.3), including partial migration. Thus, the Application Logic has to be reconfigurable and splittable in parts separately executable in different contexts and devices.

As for the User Interface, it must be adapted, entirely or partially to the device capabilities, during the migration process. There are several possibilities (see D1.4, section 1.7), each of them requiring a different design from the application point of view. For instance, in case of pre-computed adaptation, the UI design must take into account all the possible target devices and provide for each of them a specific version of the UI.

In order to guarantee the continuity, both the Application Logic and the User Interface parts need to support state persistence. Therefore, the application must be able to save and restore the state of the two parts before and after the migration process.

Modern applications usually require one or more network connections, all the more so a migratory application must handle this aspect in a multi-device environment. Hence, migratory applications should consider the possibility to be switched to different devices and different networks without the interruption of their network sessions.

A migratory application may have different configurations according to the context, i.e. any information that can be used to characterize a specific situation. This is influenced by changes in devices, network connections, user location, environmental conditions, and so on. Therefore, the application design depends on the considered contexts and their possible variations.

In a migratory application it is important to define what can be migrated and where, especially in a multiuser environment. The policy rules, which provide the preferences for the adaptation in the migration process, may be defined at MSP level, at application level, or by the user. All the alternatives must be considered by the application during the design phase.

3.2. GUIDELINES FOR MAKING AN APPLICATION OPEN-COMPLIANT

All the aspects described in the previous section have been examined during the course of the project with the aim of providing a concrete support to handle them in the migration process. To support a wide range of applications, the MSP has been designed keeping into account different configurations, both in terms of client-server architecture and application-platform interaction. According to the description of the general configuration in D4.2, from the MSP point of view the applications look like a single Open Client to the platform, even when they have a server and a client part.

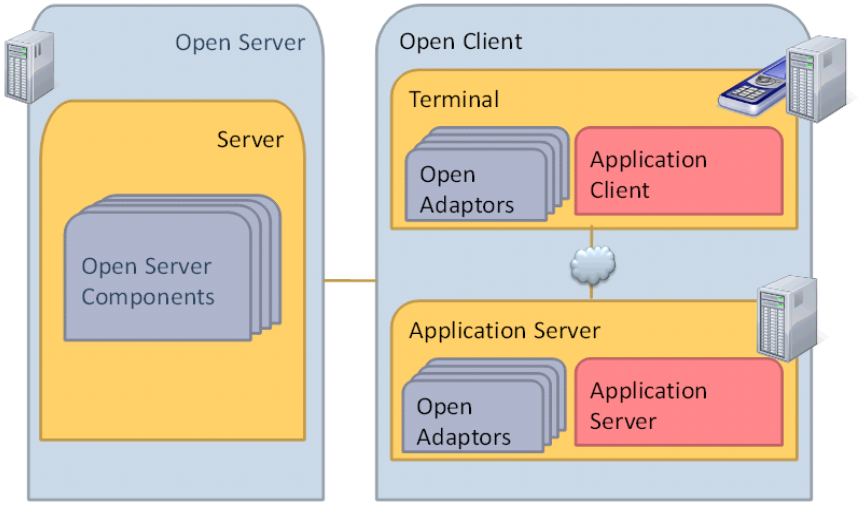

Figure 1: general client-server architecture configuration.

Referring to Figure 1, the Client is made of a terminal running an application and a set of Open Adaptors. The adaptors implement the part of the migration functionalities that are common across applications. Because of the variety of the application ecosystem, to provide more flexibility to developers, applications can integrate with MSP by using the Open Adaptors exclusively, or by reimplementing them and offering the Open Interface directly. Any combination in between it is also supported, with applications reusing some adaptors, and overriding other ones.

A special case of client-server configuration is illustrated in Figure 2, where the Application Server is not OPEN-aware and all the adaptors are in the client side of the application. In this configuration, the network management requires specific support from the platform that will be explained in details in D3.4.

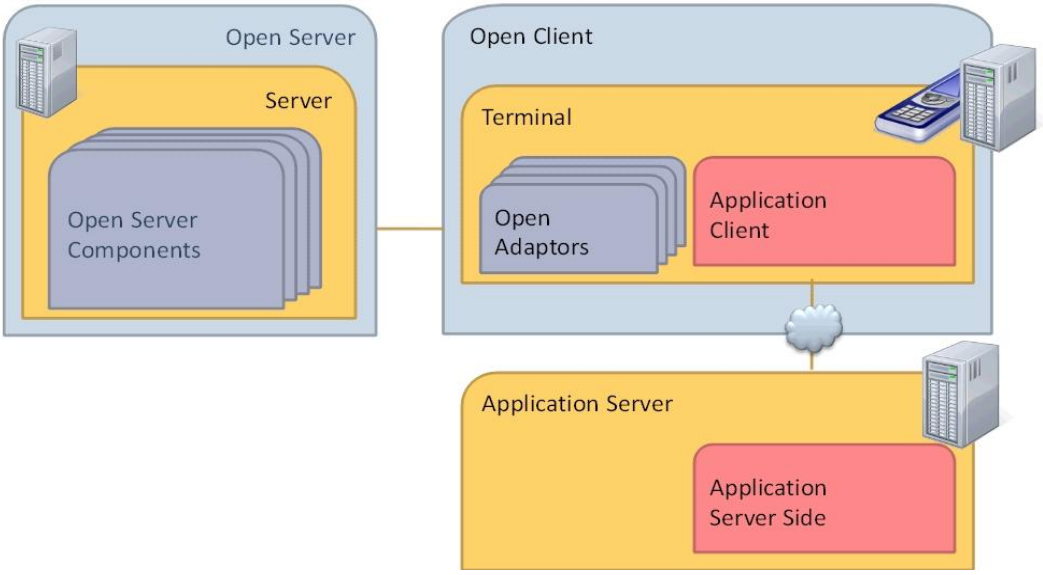

Figure 2: variation of client-server architecture configuration.

Regarding the communication between the MSP and the application, it is handled through the Open Dispatchers, as shown in Figure 3.

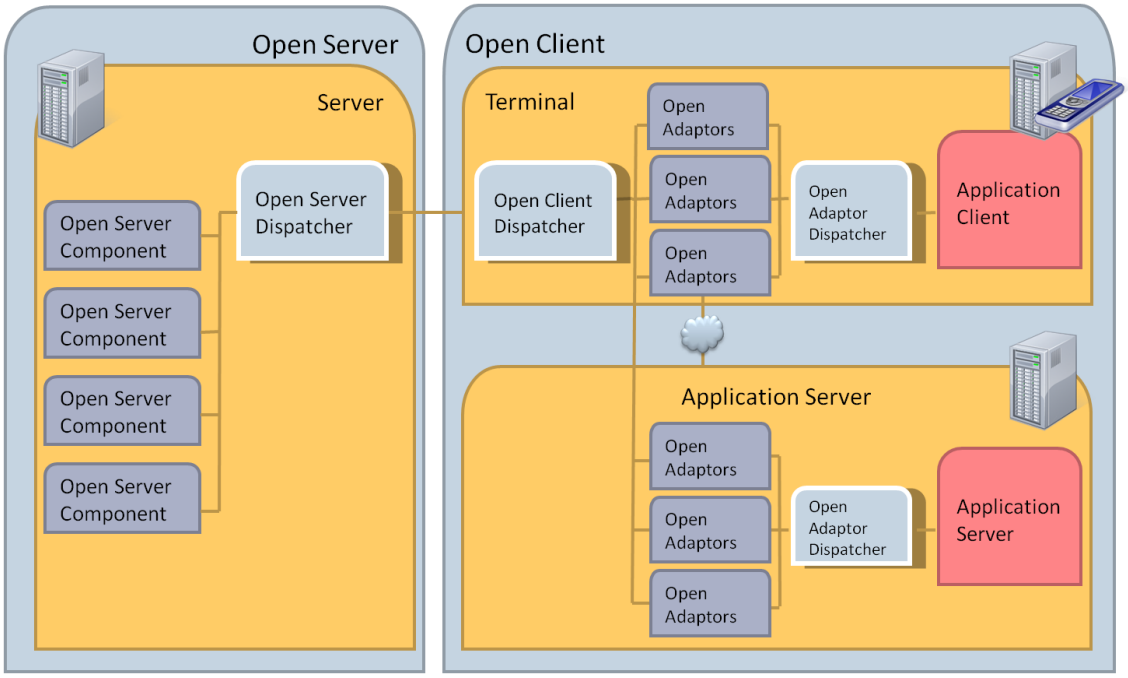

Figure 3: generic client-server architecture configuration with Open Dispatchers.

They are modules acting as single endpoints for routing messages through the different modules of the MSP and through the MSP and the application. The application side dispatcher, i.e. the OPEN Adaptor Dispatcher, is bound to the specific application/device pair considered. It is meant to provide the whole

functionality of the MSP to the application itself, according to the communication protocols that have been adopted. In particular, all the communications use the XML-RPC protocol. This choice implies that the application must be able to exchange messages in that format regardless of its platform and technology. Similarly to the previous case, described in Figure 2, the Application Server might not be OPEN-aware and therefore external to the OPEN client.

In the following, more details are provided, regarding how a migratory application must be designed and developed in order to be OPEN compliant, considering the various aspects of migration described so far. It is worth noting that the subsequent considerations are widely based on the results obtained from D4.2.

3.2.1. APPLICATION LOGIC

During the migration, an application may be subject to a reconfiguration process, due to adaptation to different devices, partial migration, policy management, and so on. To concretely support this possibility, it has been chosen to consider the application as organized in reconfigurable and splittable logical parts, named components.

A component is an entity with internal properties and external dependencies on other components. Each of them has specific requirements in terms of software, hardware, network and UI capabilities.

Depending on the used components and their mutual relationships, an application may assume different configurations. The list of allowed configurations must be provided by the application to the MSP and they are strictly related to the context's conditions. Based on these configurations and on the context input, the MSP (specifically the ALR module) computes the most suitable configuration, and therefore the application must be able to modify its components configuration accordingly.

To be able to exploit MSP's automatisms, a specific procedure must be followed. First of all, the MSP must know the application, its main features and its requirements; hence the application must be registered to the MSP, providing such information. Referring to D4.2 (Appendix C, Object type definitions), the Application object has been introduced containing the required information. More precisely, it includes: the name of the application, the application launch script, the used components, and the possible configurations. The application launch script is needed by the MSP in order to be able to launch another instance of the application on the target device. The configurations represent all the possible configurations of components depending on specific requirements (e.g. screen size, CPU frequency, etc). The application must be registered to the MSP when launched and unregistered when it is closed.

Once the Application has been registered, the MSP needs to acquire the required information about the components. To this end, the Component object has been introduced, containing a description, the running status, the migration status, the internal state, the relationships with other components, the ID of the application which belongs to, and a set of requirements. The description is provided in terms of input, output, interfaces, etc. The running status is the execution state of the component at a given time and it can assume several values such as Paused, Running, Launched and Terminated. It is used by the MSP during the migration process: once the migration for a specific component is triggered, its internal state must be frozen in order to be restored in the corresponding migrated component on the target device. For this reason, the application must be able to pause the component on the source device, retrieve its

internal state, launch it on the target device, restore its state and terminate it in the source device. The migration status indicates whether a component is migrating or not and it allows the MSP to identify the active instance of that component. The internal state is application dependent and contains all the needed information that allows preserving continuity after the migration of one or more components is performed. The component relationships indicate the kind of relationships that a component has with other components within their application, e.g. if a component needs another one in order to function properly. The requirements are a set of hardware, software, network and UI requirements that the application developer must provide to the platform in order to identify a set of criteria for components configurations according to the capabilities of the involved devices. These requirements are used by the ALR and the Trigger Management to select the most appropriate configuration with respect to the specific context.

After the registration of the Application, each component must be registered to the MSP that assigns to it a unique identifier. The application developer must provide all the needed information described above related to the components. In particular, each component must contain a list of Requirement objects, specifying the technical requirements needed by the component itself. These will be used by the MSP to find the best match with device capabilities that are described through the *Capability* object. Moreover, the dependencies among the components must be specified through the *ComponentRelationship* object. Further details regarding the objects containing all the information about the components can be found in the tables filled in D4.2, Appendix C.

To ensure a correct migration of the components, the application must be able to retrieve and restore the internal state of each component, whenever required by the MSP. Thus, the application developer is in charge of choosing her/his own serialization format for handling state management, depending on the specific application features. The MSP's only responsibility is to convey unchanged the state information between the source and target devices. Moreover, the application developer has to support the change of the running status for each component, as well as communicating to the MSP the result of the switching of the status. For example, the MSP terminates the component on the source device only after receiving the confirmation that the running status of the component in the target device has been successfully switched to “running”.

When the context changes, the MSP may require the entire application or a set of components to be reconfigured. Therefore the application developer must implement a suitable mechanism for activating the required configuration, and then inform the MSP about the result of the operation.

It is application developer responsibility to communicate to the MSP, through the unregistration procedure, when one or more components or the entire application are terminated. In this way, the MSP knows which components and applications are still active and migrable.

3.2.2. USER INTERFACE

During the migration process, an application may be moved from a device to another with different capabilities. This can affect the way the UI is presented to the user and how the user interacts with the application. Therefore, an adaptation of the interface may be required to preserve the user experience. In

the OPEN project the migration of the User Interface has been approached considering two different domains: the Web based UI, and the Java based UI that exploits the API provided by Namuco toolkit.

WEB BASED UI

For the web domain, the MSP provides a general approach based on a dynamic generation strategy for adapting migratory Web applications, which has been described in details in WP2 deliverables. From the application developer point of view, the MSP can be exploited to adapt the UI of a Web application following some basic guidelines. Since the approach is based on a reverse engineering process, in order to properly perform this transformation, all the application's web pages must be valid, namely they have to conform to the W3C XHTML specifications (see [8]).

Other, non-mandatory requirements can be identified for allowing the automatic adaptation tool to produce better results. One of these consists in the use of the most appropriate UI technique for supporting a particular task, and avoid the temptation of using instead alternative UI techniques/elements (or combination of them) just to the aim of obtaining more graphically appealing UIs. For instance, an example of this misuse is when in a web page, for supporting the a mutual exclusive choice between a number of elements, instead of using the XHTML input element of type “radio” (which is specifically meant for supporting this kind of activity) other techniques are used in order to simulate the same effect (e.g. using a number of more graphically appealing buttons grouped in a table row). In this situation, it may happen that the semantic of the interaction becomes more difficult to be identified (e.g. by the reverse engineering module, which has to produce a more abstract description of the UI, the so-called Concrete User Interface), and this can lead to less effective results of the overall migration process.

Another non-mandatory requirement, which can lead to more usable results in the migration process, is the use of meaningful labels for identifying the various parts of the user interface to be migrated, in order to help the user to easily recognize which part of the page s/he is referring to. For instance, by including in the XHTML page composition elements having identifiers which provide some information about their content, will allow obtaining sections and/or (eventually split) pages still having meaningful names/titles when adaptation is performed and such pages are rendered on a device with different capabilities. Furthermore, the use of meaningful identifiers for naming the various parts of a page (e.g. for identifying elements in an XHTML page) will also contribute to support a more effective partial migration from the user’s point of view, since the user can more easily select which parts of the user interface s/he wants to migrate if such parts have been identified through meaningful names.

Regarding the state of the UI elements, it is handled by the MSP through State Mapper, which is part of the Web UI adaptation module (see D2.2): indeed, its role is to update the Concrete User Interface for the target device with latest information regarding the state of the user interface contained in the source page just before migration.

JAVA BASED UI

For the Java UI, the MSP provides the Namuco GUI Toolkit, a programming library that allows application developers to easily create visually appealing user-interfaces that utilize multicore processing to provide a

smoother and richer end-user experience. The library helps to achieve a consistent appearance across different devices with varying screen layouts and CPU performance (e.g. net-books, desktop PCs, HDTVs, mobile phones). Namuco targets applications that need to provide extensive graphical elements - e.g., in the multimedia or Web 2.0 area - on a wide spectrum of different devices.

The toolkit consists of a native C++ shared library that implements the low-level multicore rendering functions using NEC's Task Programming Interface (TPI) and a Java package that makes this functionality available to Java developers by integrating into Sun's AWT UI framework.

To exploit Namuco's features, the application developer must use the extensions of Java AWT UI framework provided by the toolkit. In particular, Namuco hooks into the existing AWT class framework by exposing new Canvas and Component classes that integrate the new multicore drawing mechanism and custom event handling code (mouse, keyboard, etc.). To use the new functionality provided by Namuco, the application developer has to use these new classes instead of java.awt.Canvas and java.awt.Component.

Namuco provides also a support to animation, through a set of Animation classes (e.g. BlendAnimation, WipeAnimation) that implement different display effects in native C++ code.

3.2.3. NETWORK

Modern applications use networking for various communication tasks. Depending on the context, network conditions may change during the application execution, making it difficult to maintain the continuity of the services that use the network. In migratory applications, network management becomes even more complex, since not only the network connections change, but also the involved devices. In the OPEN project this problem has been addressed by making transparent to the application the change of terminal and network connections, through the Mobility Support module. In this module, the change of network conditions is supported by Terminal Mobility, whereas the change of device is supported by Session Mobility. Figure 4 illustrates the client-server configuration and the MSP modules involved in the Mobility Support.

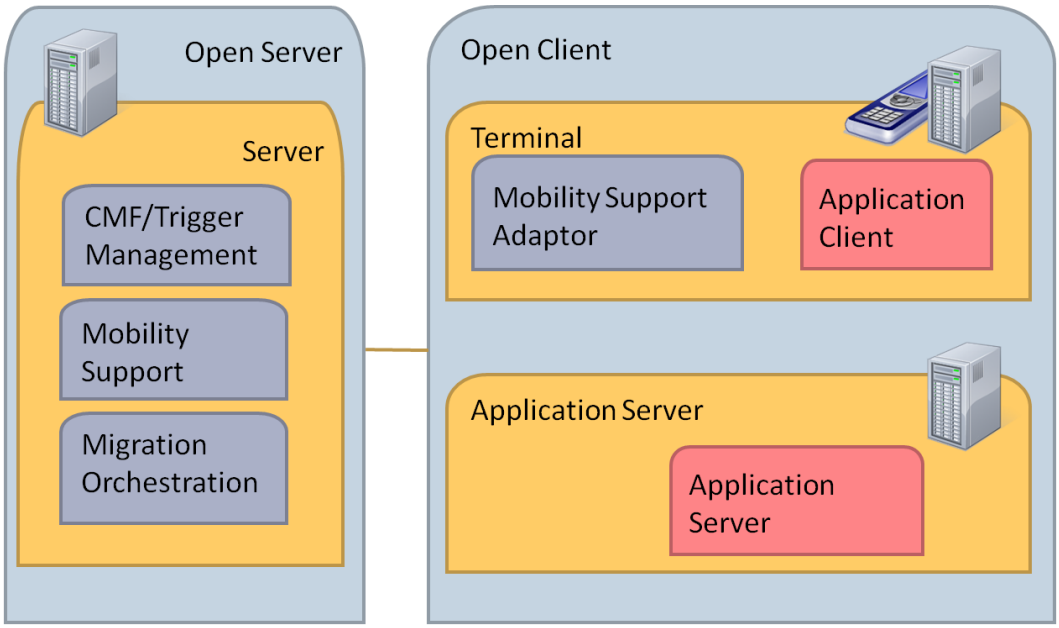

Figure 4: client-server architecture for Mobility Support.

The Terminal Mobility uses the Mobile IP protocol and it is transparent to the application.

The Session Mobility is supported by the implementation of an extended SOCKS proxy and interaction between the Mobility Support and the application is required. To this aim, the MSP provides a Mobility Support adaptor on the OPEN client. For the application developer it is sufficient to use the adaptor for network access instead of system's standard direct connections.

To handle the exchange of information between the application and the Mobility Support Adaptor, the *NetworkConfiguration* object has been introduced, which stores the connection specific information that can be used by the application to establish a network connection between a source device and a target device.

It is worth noting that network requirements are handled at components level. In particular, each component contains information about bandwidth, QoS, etc.

3.2.4. CONTEXT

For a migratory application the context information is very relevant, since the migration process provokes a context change even in the simplest case. Context information refers to all data that is relevant to describe a given situation for a given object. This information is managed by a specific module inside the MSP named Context Management Framework (CMF), which is described in D3.1. The MSP, through the CMF, is in charge of monitoring context changes and communicating with the other platform modules to take the appropriate decisions regarding reconfiguration of Application Logic, Network, UI, etc. Since the application may have the need to access the context information, the MSP offers this possibility by providing a suitable interface with the CMF (see D4.2, Section 7.1.1). Anyway, in the application design, it

is important to take into account that the context may change during the execution, and the application must preserve the perceived user experience as much as possible. Moreover, the user should be aware of what is happening and understand the actual situation and effective configuration of the application.

The interaction between the application and the context can follow different approaches:

- context variations are handled entirely by the MSP; the application is aware only of reconfigurations and adaptations requested by the platform;
- context variations are notified by the MSP to the application through events containing information about the changed context. The application must be able to interpret and show this information to the user, who takes decisions about possible reconfigurations;
- context variations are notified by the MSP to the application, which presents the new situation to the user, but it is the MSP which is in charge to handle and take decisions about the reconfiguration.

Note that these three approaches can be simultaneously supported inside the same application, if needed.

To exploit CMF functionalities, the application must connect to an appropriate Context Agent, which can eventually behave as a client side adaptor for context management. The application communicates with the Context Agent via XML-RPC calls containing queries expressed in CALA (Context Access Language, see D4.2, Section 7.1.1).

The CMF can inform the application about changes in the context in two ways: a) reactively, i.e. the application sends a request to the Context Agent about the current status of the context, a link or which information it needs to know; b) proactively, i.e. the application sends first a subscription to the Context Agent, from where after the Context Agent sends updates to the application as desired.

Through the CMF, the application can obtain different kind of information. In principle, any measurable or inferable types of data can be supported as long as it follows the context model; e.g. in case of network data, the following information is supported: Round Trip Time, Jitter, Packet loss ratio, Throughput and/or available bandwidth.

3.2.5. POLICY

The MSP handles policy issues through two processes: the Policy Management and the Policy Enforcement.

Policy Management is the process of ensuring that a policy rule is sent or updated to relevant recipient(s) inside the MSP.

Policy Enforcement is a process that, given some input in terms of a proposed action, it determines whether that action can be performed or not.

In principle, there is the need of policies in different modules of the MSP, e.g. there are (conceptually) privacy policy enforcements also in the CMF to ensure that private context information is either not communicated or anonymized in some way.

The application's concerns in terms of policy are related to the allowed configurations resulting from a migration process, which depends on context conditions. To exploit the Policy Enforcement mechanisms provided by the MSP, the application must communicate its policy rules, which entail constrains on migration and adaptation under certain conditions. The application may define policy rules at design level, according to context, user profile, security parameters, and functional requirements. The user can also define policy rules about allowing or disallowing migration, e.g. for privacy issues.

The application defines a list of policy rules, each of them containing: a valid configuration of the application, a list of context conditions that must be met in order to allow the migration, and the authorization for the action (in Yes/No format). Details about the formats of configuration and context condition will be described in D3.4.

One of the main usages of the policy is the privacy management that could also be handled at component level. In fact, a component can be classified according to a privacy level and, for each level, a different set of policy rules can be enforced. For instance, an application can have a particular component with high level of privacy that can be migrated only with explicit authorization from the user.

3.3. USABILITY OF MIGRATION PROCESS

As previously mentioned, an important aspect to be considered while designing a migratory application is the user perception and awareness of the migration process. In fact, the migration involves different devices and different components, and the user has to be aware of the whole migration process and the current configuration. The application must then provide suitable mechanisms to inform the user about its actual configuration (e.g. through alerts, icons, text messages, etc.). Although general hints about the information to be shown to the user can be identified, the way it is actually presented is strictly dependent on the specific application. For example, a game application and a business application will present the information about migration using different styles and layouts.

Regarding the migration process, it can be divided into three main steps:

1. choice of components and devices involved in the migration
2. migration triggering and execution
3. migration completion.

As for step 1, two cases are possible: either the user selects the component(s) and the corresponding target device(s), or the platform automatically takes decisions about them. In the first case, the user must have the possibility of choosing the components and the devices. In the OPEN project, two solutions are supported: the application developer can provide to the user a suitable interface inside the application or leave the task to the MSP, which offers a simple GUI for that. In the second case, the application must inform the user about the decision taken by the MSP regarding the involved devices and components.

During step 2, which is a transition phase, the user should be aware of the fact that there is a certain amount of time during which the involved components of the application will not be available in any device, until such a phase is not completely finished. Therefore, during this phase, an adequate and significant feedback must be provided in order to inform the user that, although some components are temporarily unavailable, the MSP is correctly working in the way s/he expects. In particular, the running status of each component has to be clearly perceived by the user, who must be able to distinguish between active and inactive components.

As for step 3, the fact that the migration has completed has to be clearly shown to the user, especially the final configuration of devices and components. This is particularly important in partial and/or multi-device migration, since user's attention is split among different parts of the application. Moreover, the user must be informed about the result of the migration process, both in the case it has succeeded and it has failed.

Another aspect related to usability of migratory applications is the possibility of showing context information to the user in order to help her/him in making choices or to inform her/him about context variations. Also in this case, the type of information and the way it is presented to the user depends on application's specific design choices.

Moreover, it is important that the application informs the user whenever components with privacy issues or sensitive information are involved in the migration, e.g. the case of a web page containing credit card details being migrated to another device.

4. APPLICATION USE CASES

This chapter describes the prototypes that have been chosen as test cases of migratory applications that exploit the OPEN platform. The chosen use cases are: Emergency Prototype, Social Game, Twitter Wall, and Pacman.

These prototypes have been designed to explore migration cases in different domains and contexts, in order to help defining the needed functionalities of a migration platform. The first version of the prototypes has been described in D5.1, where complex scenarios had been used as a starting point to identify application requirements. A simplified version of the prototypes inspired by these scenarios has been first developed in form non OPEN aware applications in D5.2, where almost all the migration procedures were simulated. The next version of the prototypes, whose design is described in this chapter, focuses on aspects related to migration rather than prototype-specific optimizations and improvements. In fact, the scope of the project is to develop a middleware for migratory applications and the prototypes serves as support for the platform in:

- understanding the required capabilities
- testing the platform functionalities
- demonstrating platform's potential and advantages

The prototypes presented apparently show a few improvements with respect to the previous version from the final user point of view. However, their main changes are related to their internal structure, since they have been adapted to be integrated with the MSP, in order to exploit the offered migratory functionalities. This adaptation work, including the related design, is still in progress and not all the described features might be fully implemented, due to design choices or time constraints, although they are in principle supported by the platform.

4.1. EMERGENCY PROTOTYPE

The Emergency prototype is based on the Emergency Scenario where governmental agencies, organizations and companies work together in order to provide public security in emergency situations (e.g. flooding, large fires, and huge accidents). A detailed description of the scenario can be found in D5.1. The initial prototype of the emergency application has been implemented and specified in D5.2.

The planned design and functionality enhancement of the prototype is introduced in the current section. This enhancement complies with the guidelines from section 3.2 and reflects the results of the conducted usability evaluation that is described in D6.5. Furthermore, the requirements listed in D1.3 and D5.1 have also been considered for the final version of the prototype.

4.1.1. DESCRIPTION

The context of the Emergency Scenario is that a number of different experts assemble at an emergency control center to plan the response to a flood. To do this, the experts migrate their applications or parts of

them to a smart wall, where all of the migrated parts are either merged into one application or visualized in a proper way.

Each expert should be aware of the way the migration process will be executed and should know how to control the migrated application. These aspects are particularly addressed in the final version of the Emergency prototype. Further, the user feedback from the already carried out exploratory study and usability assessment test, which are described in detail in D6.5, delivered deep insight on certain points. Most of the suggested functionality that was related to the migration process rather than to the application itself is being reflected in the final version of the prototype.

Before the proposed changes are explained, it should be shortly defined what a component of the emergency prototype is. Components include application logic and/or an input/output widget. For example, components in the current prototype are the traffic and flood control panels, the map with its input and output capabilities. All these components and the design of the current application that serves as a starting point for all proposed enhancements are shown in Figure 5.

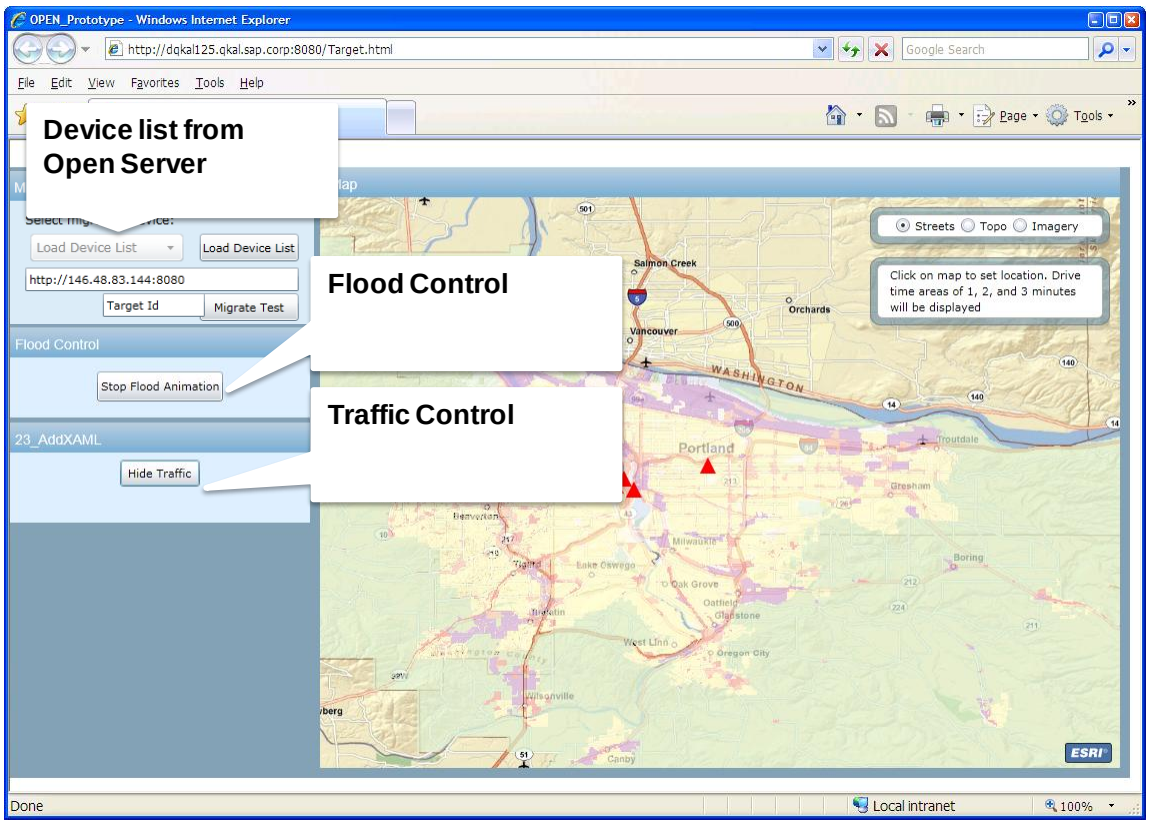

Figure 5: current Emergency prototype

Several new features are planned for the new version. In the next sections the reasons for each suggested modification will be explained, the corresponding requirement(s) will be pointed out, and a hypothesis for the usefulness of each feature will be proposed. These hypotheses can then be tested in the validation

usability test. The suggested changes affect the migration control, the component control design, the component migration, the visualization on the target device, as well as some minor details.

4.1.2. SPECIFIC REQUIREMENTS

The requirements listed in D1.3 and D5.1 have been considered to a great extent in the final version of the application prototype. Furthermore, some new and specific ones were elicited from user suggestions made during the usability tests described in D6.5. The new requirements are listed below.

ID	1
Description	OPEN server addresses should be automatically recognized and presented to the user for choice.
Type	Functional
Rationale	Coming from user suggestions

ID	2
Description	START/STOP indications improve the readability of the traffic as well as of the flood simulation.
Type	Look and Feel
Rationale	Coming from user suggestions

ID	3
Description	There is a legend for each simulation.
Type	Look and Feel

Rationale	Coming from user suggestions

ID	4
Description	The flood simulation color should be stronger and yet allow for identification of affected regions and streets.
Type	Look and Feel
Rationale	Coming from user suggestions

ID	5
Description	After migration, the user has the choice between a map merge or a side-by-side display of migrated components on a target device.
Type	Functional
Rationale	N/A

4.1.3. APPLICATION LOGIC

Following the guidelines described in 3.2.1, the Emergency prototype has been organized into a set of components, each of them having its own requirements in terms of hardware, software, network and UI.

The main components are: migration control panel, traffic control panel, flood control panel, map with its input and output capabilities. The following tables provide detailed information about each component and its relations to the other components.

Flood Control Component		
ComponentDescription	• Component to control the flood visualization. • Is part of the Flood Manager application.	
ComponentRelationship	• Dependence on Map Component to display visual results. Control commands are forwarded to the Map Component for visualization.	
State	• State management is done by the component itself. During migration the state is transferred together with the component to the target device/application using the OPEN Server.	
Requirements	Hardware	• No significant hardware requirements
	Software	• Web Browser with support of XHTML, JavaScript, AJAX, Silverlight
	Network	• This component itself does not need a network connection. Network connections are realized via the Migration Control Component.
	UI	• Minimum screen size should be about 400x400 pixels to allow minimum usability.

Table 1: Flood Control Component

As mentioned in Table 1, the Flood Control Component is the controller of the flood visualization. To visualize the flood the Flood Control Component sends the content and commands to the Map Component (which is only used for visualization). During migration the state is transferred via the OPEN Server to the target application/device. The Flood Control Component interacts with other components (e.g. Map Component) via the "Local Controller" which is part of the Migration Control Component.

Traffic Control Component	
ComponentDescription	• Component to control the traffic visualization. • Is part of the Traffic Manager application.
ComponentRelationship	• Dependence on Map Component to display visual results. Control commands are forwarded to the Map Component for visualization.
State	• State management is done by the component itself.

	During migration the state is transferred together with the component to the target device/application using the OPEN Server.	
Requirements	Hardware	• No significant hardware requirements
	Software	• Web Browser with support of XHTML, JavaScript, AJAX, Silverlight
	Network	• Requires a network connection to the traffic service (e.g. yahoo traffic service).
	UI	• Minimum screen size should be about 400x400 pixel to allow minimum usability.

Table 2: Traffic Control Component

Table 2 gives an overview over the Traffic Control Component, which is the controller of the traffic visualization. To visualize the traffic, the Traffic Control Component sends the content and commands to the Map Component (which is only used for visualization). During migration the state is transferred via the OPEN Server to the target application/device. The Traffic Control Component interacts with other components (e.g. Map Component) via the "Local Controller", which is part of the Migration Control Component.

Map Component		
ComponentDescription	• Map component containing geographical imagery (streets, topographic, satellite imagery). • Ability to display several layers. • Receive commands and content from control components and display them as individual layer on the map.	
ComponentRelationship	• Doesn't have a dependent relationship to other components. Is just "used" by other components to display results.	
State	• Stateless.	
Requirements	Hardware	• No significant hardware requirements
	Software	• Web Browser with support of XHTML, JavaScript, AJAX, Silverlight
	Network	• Network connection necessary to download GIS imagery (Requires

		connection to ESRI map server).
	UI	• Minimum screen size should be about 400x400 pixels to allow minimum usability.

Table 3: Map Component

The Map Component, described in Table 3, is only used for visualization of results on a geographical map. It is connected to the "Local Controller" which is part of the Migration Control Component to receive incoming content and commands from Control Components (e.g. Flood Control Component). To display the GIS imagery the Map Component requires a network connection to the ESRI map server.

Migration Control Component		
ComponentDescription	• Controls the migration process of individual components/applications.	
ComponentRelationship	• Needs a connection to the OPEN Server to control the migration of individual components/applications.	
State	• Stateless.	
Requirements	Hardware	• No significant hardware requirements
	Software	• Web Browser with support of XHTML, JavaScript, AJAX, Silverlight
	Network	• Network connection necessary. Bandwidth requirements depend on mode of collaboration (always online or occasional synchronization)
	UI	• Minimum screen size should be about 400x400 pixels to allow minimum usability.

Table 4: Migration Control Component

Table 4 depicts the Migration Control Component. It is connected to the OPEN Server and is used to manage the migration process. All incoming and outgoing connections regarding migration are routed via the Migration Control Component. The Migration Control Component doesn't have any dependencies to other components. It is the first component instantiated during application start and is responsible to instantiate all necessary additional components during runtime.

4.1.4. USER INTERFACE

In the Emergency Scenario a PC to a smart wall migration is implemented. Until now, there is no UI

adaptation foreseen because the capabilities of both devices are similar enough to provide a comparable visualization of the same interface. Therefore, there is no need to exploit the MSP capabilities for UI adaptation in the final prototype.

Requirement Nr. 27 from the list of final requirements for the OPEN service platform described in D1.3 states that the OPEN platform must enable applications to clearly show who has control in a multi-user scenario. Complying with that requirement is very important for the Emergency Scenario since it assumes multiple users. Thus, some parts of the Migration Control panel will be exchanged for a dropdown list that provides the user with the possibility to decide from which device the application should be controlled.

The parts that should be exchanged are the input field for the server IP address and the "Migrate Test" button. They would be removed from the Migration Control, since the registration to the OPEN server should take place upon starting the application. A start page would ask the user either to type in the server IP address or to choose it from a list of last used or nearest servers. After registration the user can work with the application and set it for migration.

Further, the button "Load Device List" can be exchanged for "Set for Migration". Initially all migration settings (e.g. selection of components) are grayed out. But, after pressing the "Set for Migration" button, the device list gets loaded from the server that was specified when the application was started. Then the user can chose settings for the upcoming migration.

As shown in Figure 6, there is a dropdown list in the Migration Control that offers different control modes. These control modes shape the different possible scenarios for UI and component control migration.

Figure 6: changes in the Migration Control

The first possible choice among the supported control modes is the Home control mode. In that mode the UI of selected components is migrated but the control panel remains at the current (home) device. Figure 7 illustrates the concept of that control mode.

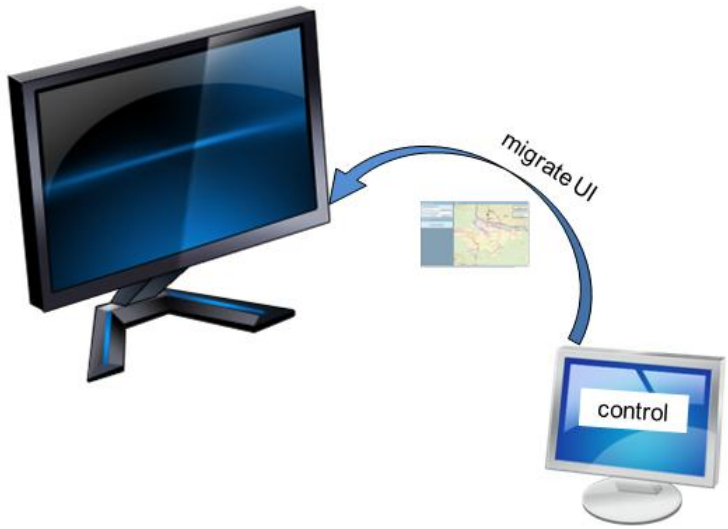

Figure 7: Home control mode

The second option is the Manager control mode. It indicates that only one user (the manager of the current session) can ask some experts to migrate their simulations to a smart wall and that user has the control over the migrated parts because he/she has initiated the migration process. The manager can also migrate one or more components to the smart wall. The experts have the possibility to migrate back and thus update selected components of their own application. The Manager control mode is shown in Figure 8.

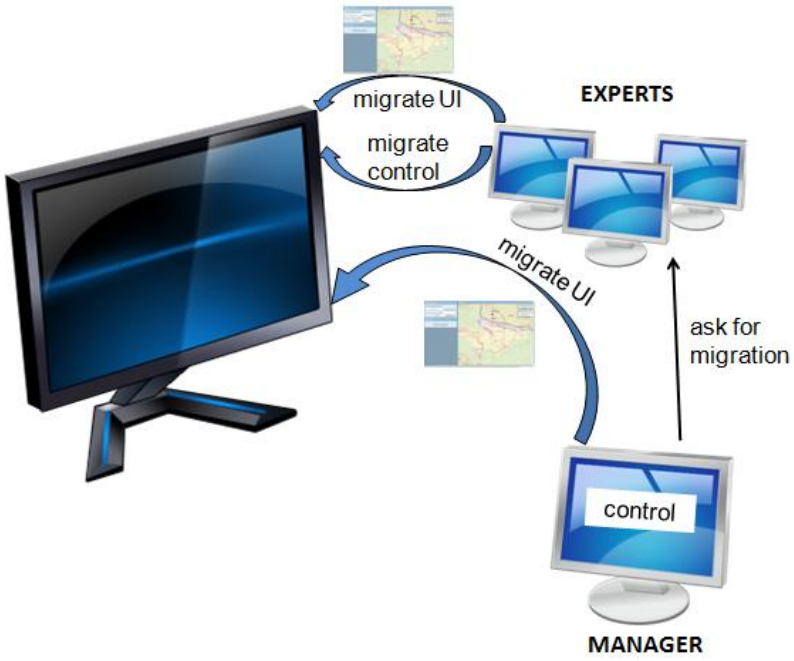

Figure 8: Manager control mode

The goal of the third option Separate control mode is to have two independent, stand-alone applications after completion of the migration process. That means that the user can control the migrated application from the target device and also continue using the application on the source device without having any changes transferred to the other device. Both devices work independently and no synchronization is foreseen here. Figure 9 illustrates the Separate control mode.

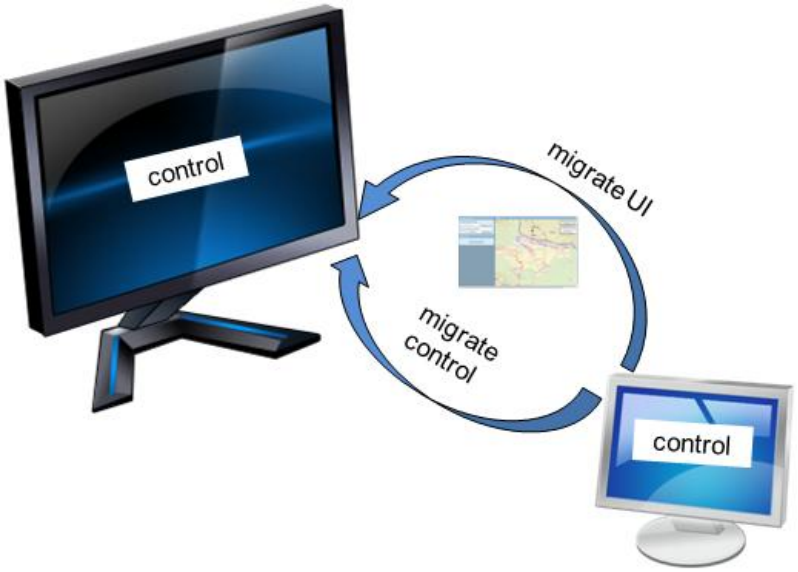

Figure 9: Separate control mode

The last option is the Synchronization control mode, which offers full control flexibility. In that mode both devices are kept synchronized after migration, so all changes made on one device are automatically transferred to the other. Figure 10 shows the synchronization mode in detail.

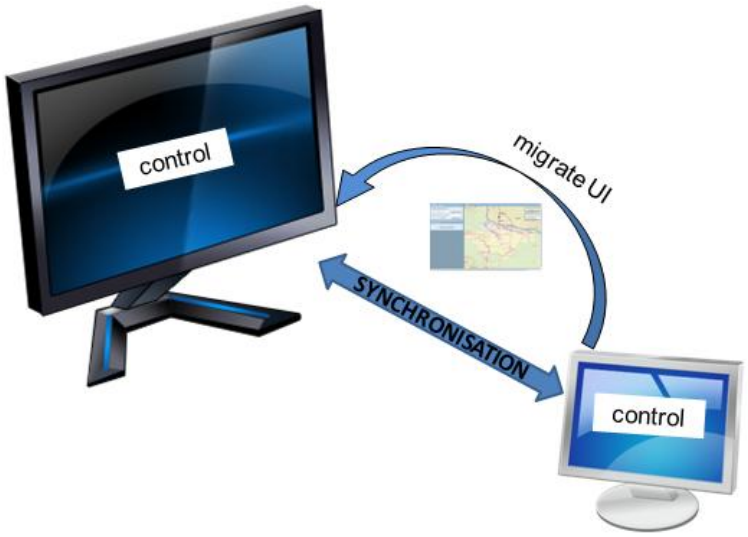

Figure 10: Synchronization control mode

It is worth noting that the functionality described here is not entirely the result of the received feedbacks but its implementation is regarded as quite important since the end user should be aware of how the migration is going to affect the control possibilities.

Hypothesis 1: The control mode possibilities are clear to distinguish from each other and offer great flexibility for the end user.

Hypothesis 2: The "Set for Migration" button makes it clear that there are some parameters to be set up before migration is possible.

COMPONENT MIGRATION

From the set of requirements listed in D1.3, Nr. 57 refers to partial migration. In order to comply with that requirement in the final version of the prototype, it is planned to implement the selection of one or more components for migration. The current prototype does not offer the possibility to select components for migration. So a goal for the final version is to comply with the requirement number 57 and to give the user the possibility to select the components he/she wants to migrate. Since that functionality needs to be presented to the user in an understandable and easy to use way, the design of the component controls eventually needs to be modified.

Further, the traffic and flood controls of the current prototype take up a lot of screen space for only one button for each control. In a real emergency situation there might be also some other important parameters, e.g. population density, potential danger areas, etc. A space-saving design would be more suitable in such cases.

In Figure 11 the differences between the current and the proposed prototype version are illustrated.

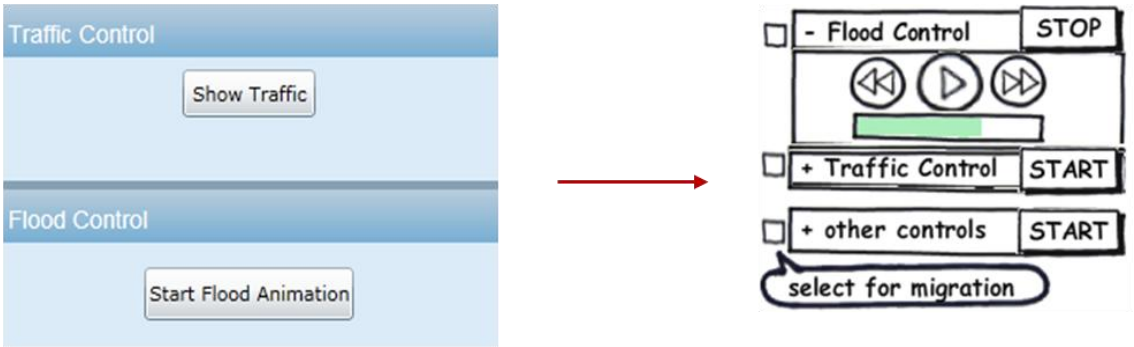

Figure 11: changes in the Flood and Traffic Control

A click on one "+" before "Flood Control" would expand it just as a tree structure and show the controls and progress bar for the flood animation as shown in Figure 12. By clicking the "-", the user can contract the menu. To show or hide e.g. the traffic on the map, the "START"/"STOP"-button should be clicked.

The tooltip "select for migration" already indicates that the choosing of components for migration can be done by selecting the checkboxes. That is possible only after the button "Set for Migration" has already been pressed. After selecting all needed components, the user can initialize the migration.

Before migration, as soon as the mouse pointer comes over a component control, the correspondent simulation is shown and highlighted on the map, no matter whether it was active or not at that time. In such a way the user can better relate a certain control to a simulation shown on the map. An example of a possible means for highlighting can be seen in Figure 12.

Figure 12: simulation highlighting

All listed component controls represent a layer on the map and can be moved up or down by drag&drop. If a component control is on top of the list, it is also shown as the first layer in the foreground. If it goes down the list, it will be put into the background. In such a way, the user has full control over the layers shown on the map and can decide which ones should be put into the foreground and which into the background. This feature should significantly improve the readability of merged simulations, which was regarded as "very low" to "low" in the assessment test.

Some users who took part in that test also mentioned that on/off indications for the traffic simulation would be very helpful. Considering the potential complexity of the map when several simulations are shown, it is reasonable to have on/off indications for each simulation. The highlighting is very important in situations when many different simulations are active or the user needs help in relating a control to a simulation on the map.

Hypothesis 3: The design for the simulation controls proposed here is comprehensible and makes it possible to easily choose one or more different components for migration.

Hypothesis 4: The highlighting function helps the user to relate controls and simulations to each other.

Hypothesis 5: The draggable layers contribute to a flexible and more readable presentation of the needed data on the map.

VISUALIZATION ON THE TARGET DEVICE

The Emergency Scenario foresees the visualization of one or two components on a target device. According to requirement Nr. 114 from D1.3 the user should be enabled to compare data sets in different ways. In the context of the Emergency Scenario there are two ways for displaying migrated components on one screen. Either they would be merged into one map and would be eventually laid one over another, or they would be shown next to each other. The decision for the one or the other way has to be taken by the end user. The visualization process will be described with the help of one example.

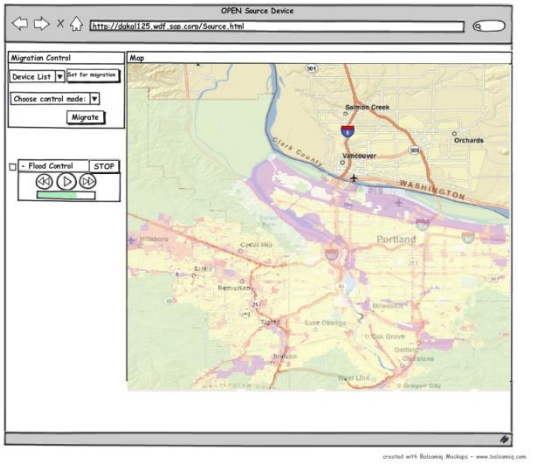

Figure 13: device A showing a flood simulation in Portland

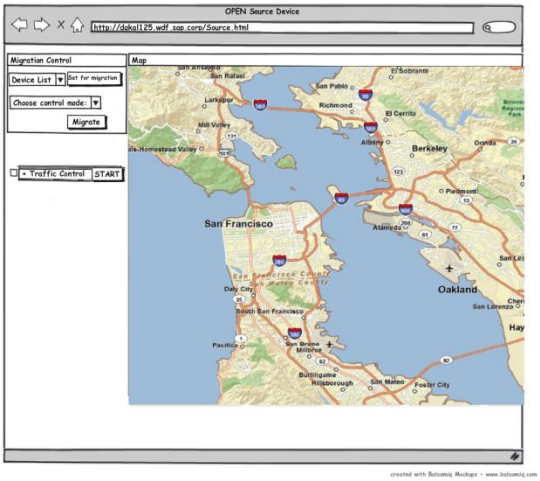

Figure 14: device B showing San Francisco

On device A (Figure 13) one can see a flood simulation focused on the region of Portland. Device B (Figure 14) shows San Francisco. The user of device B initiates a migration to target device A. Since the maps on both devices have different scales and are focused on different regions, a pop-up appears on the target device asking whether the user wants a map merge or a side-by-side display.

The side-by-side display needs no further explanation because in that case no transformations in the map scale or region are necessary. The case of a map merge is more complicated and brings about questions like which map scale is most appropriate, which region should be displayed etc. A possible solution for the complexity of a map merge could be to keep the focus and map scale of the target device but to offer, after migration, a second, small map in the right upper corner of the target map. That second map should have a scale big enough to show both regions. These would be marked, e.g., with a rectangle as shown in Figure 15.

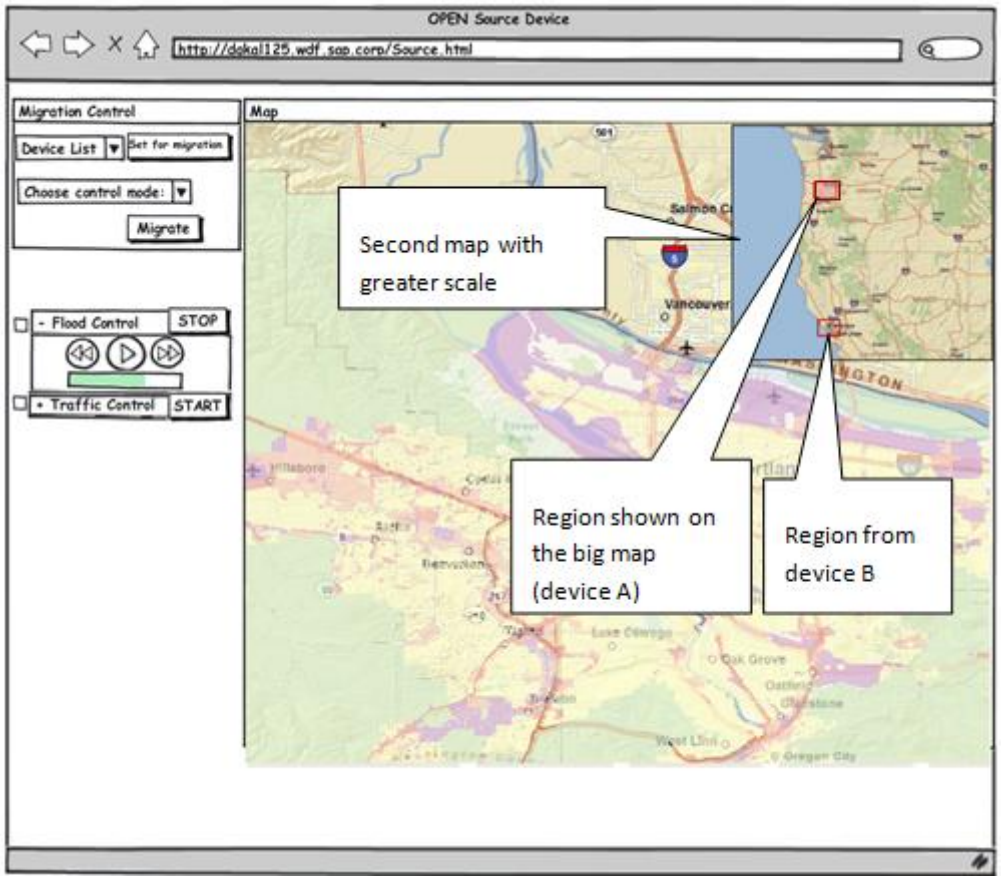

Figure 15: merged map

Navigating in the merged solution should be quite easy; by clicking on a point of the second map, the selected point becomes the focus of the big map. Zooming in and out in the big map display is easily done by scrolling up and down with the mouse within the map borders, so the map scale can also be regulated by the user.

Some minor changes are planned for the new prototype version. For example, in the current prototype a driving time simulation can be started so that one can see how far he/she could drive in one, two or three minutes. The assessment test showed that many users had problems using that simulation - they did not understand its usage and could not turn it off. The driving time simulation will be not included in the final prototype version. The main reasons are that it will take a lot of time to make it more user-friendly, and the added value of including it is not large enough to justify the investment. Thus, the final version will include the traffic and flood simulations that directly correspond to the sketched scenario.

The assessment test also revealed that there are some problems with the color of the flood simulation. It was suggested that a stronger color would make it clearer. Furthermore, there should be a contrast between the map and the flooded areas so that the latter can be unmistakably recognized. For these reasons the flood areas will be marked with a nuance of blue that is strong enough to be clearly

distinguished from the rest of the map, but that is still a bit transparent so that flooded regions and streets can be clearly identified.

In the final version of the prototype also a legend for both simulations is planned to be implemented. That point was addressed during the assessment test and it seems to be quite useful in terms of better readability of the map.

As a whole, the rapid prototype for the final version looks currently as shown in Figure 16.

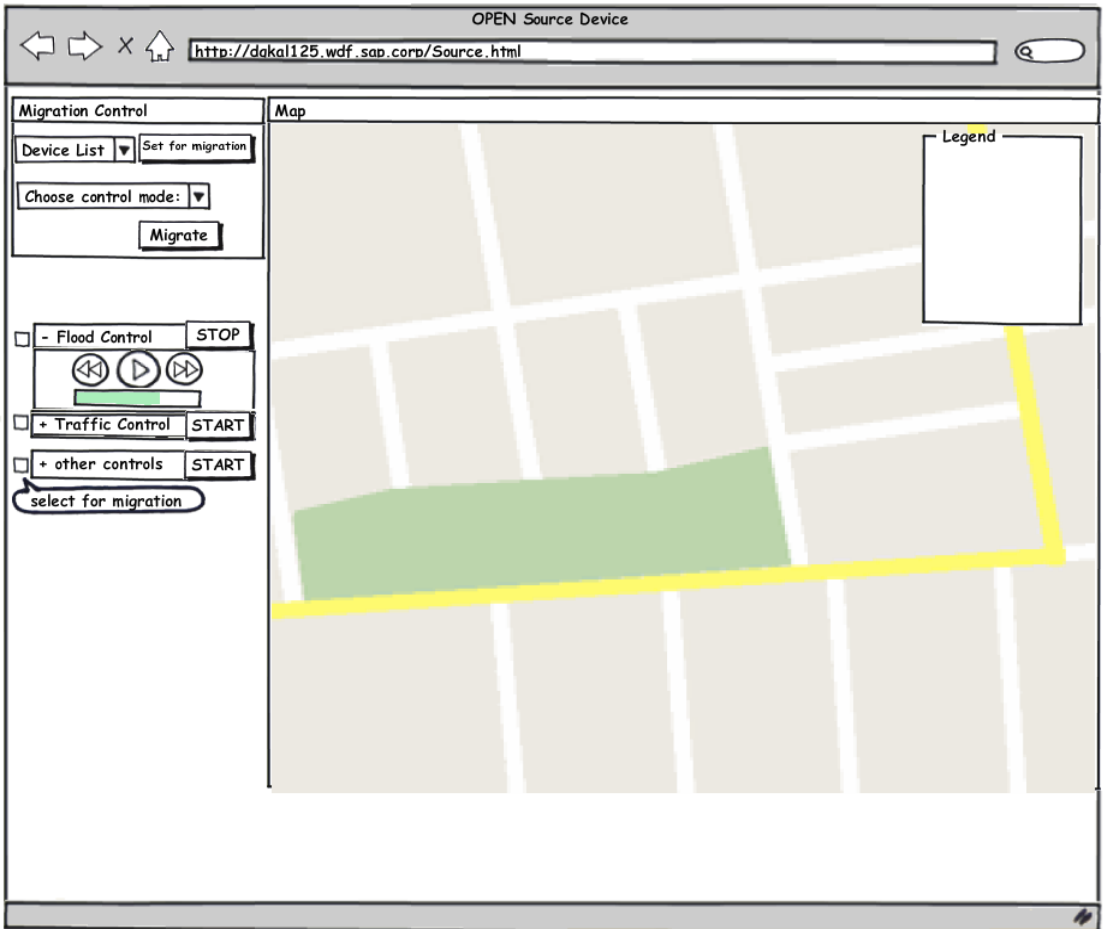

Figure 16: rapid prototype of the final emergency application version

Hypothesis 6: The new color for flooded areas increases the readability of the simulation and at the same time allows for clear identification of affected regions and streets.

Hypothesis 7: A legend for each simulation improves the map readability.

As already mentioned in Section 3.3, the user should perceive the migration process as being intuitive and should be aware of the actual migration configuration. The Emergency prototype provides a suitable interface inside the application for selecting components and corresponding target devices. Furthermore, the configuration made by the user is visible during the whole process so that no additional information mechanisms such as icons, text messages, etc. would be necessary. Figure 17 shows the UI while a migration is taking place.

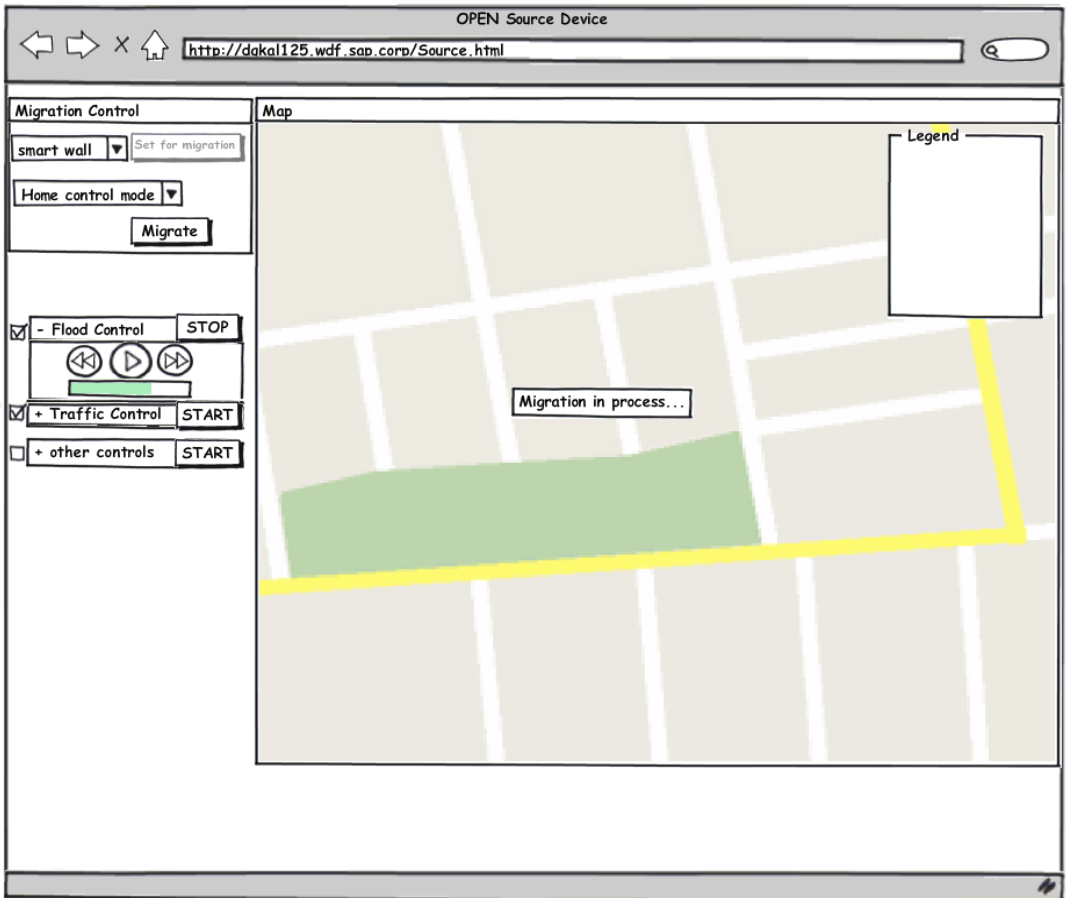

Figure 17: configuration visibility while a migration process is taking place

While the migration process is taking place, a status message appears, keeping the user aware of the actual situation. In such a way, the user is informed that the application is currently not available on both devices for the time of migration. As soon as the migration process is completed, another status message informs the user of its completion. The configuration of the last carried out migration would be still visible, so that no confusion arises about the migrated components and the selected target device.

The user interaction with the migration process has been minimized in a way that does not constrain the user`s right to shape it and to be informed, but that at the same time does not bother him/her with many

details, settings and alert/status messages. Thus, the prototype complies with requirement Nr. 71, also listed in D1.3, stating that the interaction with migration phases should be as unobtrusive as possible. Furthermore, the proposed interface and process design consider requirement Nr. 1 from D5.3, according to which the OPEN platform must be easy to use.

4.1.5. NETWORK

The emergency prototype uses standard network connections like LAN and WLAN. A more specific analysis of the network support for the prototype and how it could benefit from the Mobility Support in the future will be conducted once the final version of the Mobility Support module will be available, which is currently under development.

4.1.6. CONTEXT

In the Emergency Scenario it is vital to have a list containing the available target devices at certain points of time. That list should be communicated from the OPEN server to the application. The interaction between both of them is realized through a client side Context Agent.

Two possibilities of getting information about context changes are considered in the Emergency prototype. The first one is a client daemon that listens to changes in the context information and thus updates the device list in the application (proactive way of informing the user about context changes as described in section 3.2.4). The second possibility is to ask directly the MSP for current target device information (reactive way of informing the user about context changes as described in section 3.2.4). In both cases the available devices are shown as a list to the user when he/she wants to start changing the migration settings (button “Set for Migration”, Figure 6).

4.1.7. POLICY

Currently, no private information is displayed or gathered by any component of the Emergency prototype. Therefore, there has been no need for the adoption of policy rules regarding that issue so far.

4.1.8. ARCHITECTURE OF THE INTEGRATED PROTOTYPE

The Emergency Scenario components are all developed in Silverlight and communicate between each other using an internal event bus. The communication to the MSP is realized through XML-RPC messages. The prototype uses several OPEN functionalities: Register Device, Register Application, Register Component, Trigger Migration (the Migration Orchestration adaptor), and the Open Polling Interface, that allows the application to receive server-side events on the client. The architecture is shown in detail in Figure 18.

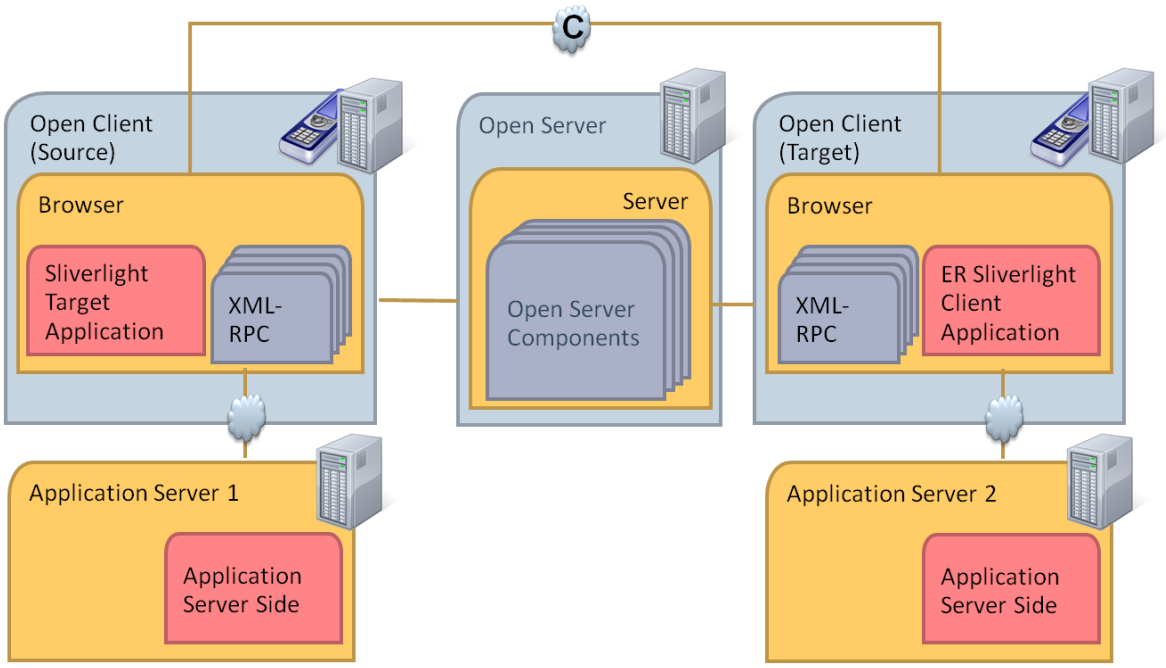

Figure 18: Emergency Prototype client-server architecture

On the left in Figure 18 the Open Client migration Source is depicted. The Open Client Source hosts a simulation application. The simulation is a Rich Internet Application using Silverlight which is requested from the Application Server 1. By using the Open Server in the middle of Figure 18, the Open Client Source can signal its migration intention to the Open Client Target, which is depicted on the right in Figure 18. After the request for migration has been accepted by the Open Client Target, the Open Client Server sends the current state of the simulation application from the Open Client Source to the Open Client Target. On the Open Client Target another Silverlight application is already running. The selected components Open Client Silverlight applications are integrated in the already running application on the target-side.

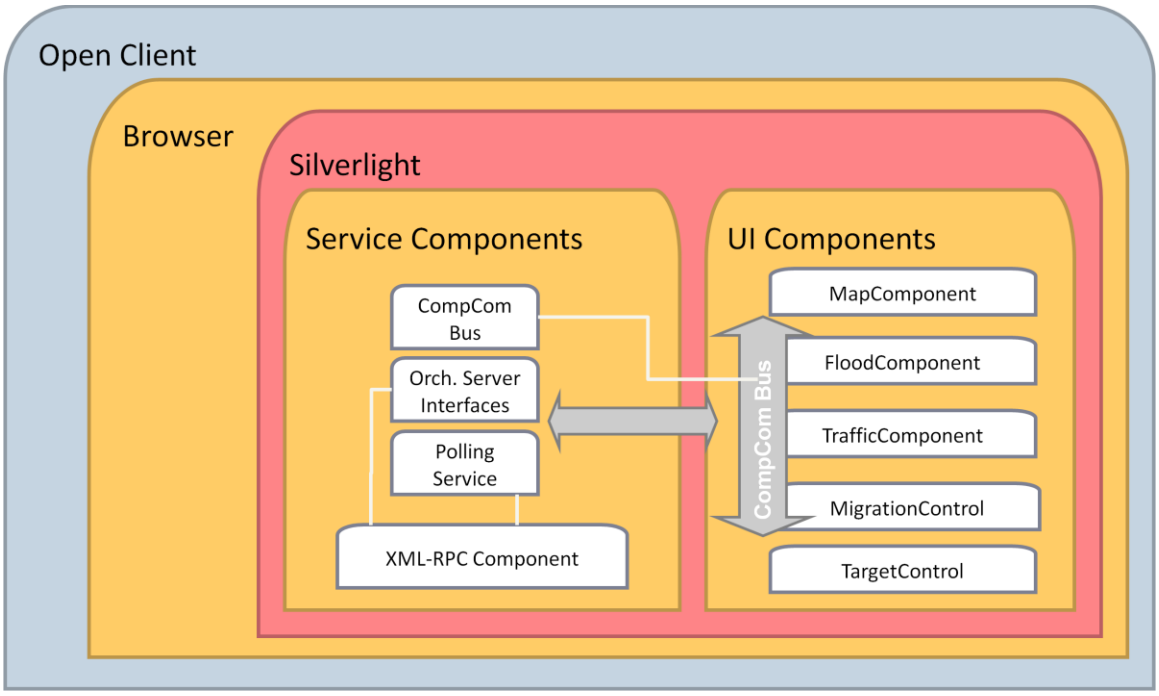

Figure 19: Emergency Prototype client side architecture

Figure 19 depicts the internal architecture of the Silverlight Open Client for the Emergency prototype. The Open Client is a Silverlight application running inside a web browser. It is composed of Service Components and UI Components. The Service Components are listed on the left in Figure and the UI Components are listed on the right. The components communicate by using an internal event bus. Any migration requests are polled from the Open Server by using the Polling Service.

4.1.9. EXAMPLES OF MIGRATION

The purpose of the migration functionality is to improve the user experience when completing different tasks and changing devices. In the emergency application that functionality can be exploited in a way that enables the user to have full flexibility in deciding what, where, and how to migrate.

The migration cases that may be implemented in the Emergency Scenario are enabled through the four different control modes: Home, Manager, Separate, and Synchronized. That means that there are four possible migration cases not regarding the number of allowed components and possible target devices. The here given examples are based on the business domain case study described in D5.1.

Professor Schneider from the Center for Disaster Management decides to migrate the control of his forecast software from his PC to his PDA just before he sets off to Cologne. On the way he would like to try out different algorithms and compare their simulation results, so the migrated controls and the source application should work independently. In that case, he uses the separate control mode, because with that mode he can easily assess different flood scenarios without having all changes transferred to his PC.

After his arrival in Cologne, Professor Schneider meets Professor Fischer from Potsdam Research Center and Dr. Mueller from the Emergency Operations Center. They want to compare the forecasts of both professors and use for that purpose a smart wall. On the smart wall they still want to experiment with different scenarios and modify the parameters for both forecasts. They can control the parameters and the simulations using their own devices (laptop and PDA) if they opt for the home control mode. The manager mode is useful in the case that e.g. Dr. Mueller asks Professor Schneider and Professor Fischer to overlay their forecast results so that the overall impact of the emergency cause on the affected area can become clear to the whole emergency team.

After closely analyzing the emergency situation, Dr. Mueller needs to plan the routes for the emergency services. If the situation changes unexpectedly it would be important that Professor Schneider and Professor Fischer also have these routes. In such a way they can adequately modify their forecasts and warn Dr. Mueller if the routes are not safe and should be changed. All three of them migrate their simulations to a smart wall using the synchronization mode and discuss the possible routes for the emergency services. The synchronization mode keeps their own devices up-to-date with the simulations on the smart wall and they do not have to worry about further information exchange.

These migration examples show that there is enough room for improvement of the user experience when exploiting the MSP functionalities. In all examples, the three main experts involved do not have to worry about how to exchange information or keep it synchronized. Instead, they can fully concentrate on their work and seamlessly transfer the data they need from one device to another. The four control modes enable end users to have full flexibility in terms of information exchange and modification when coping with their tasks.

4.2. SOCIAL GAME

The Social Game is a rich web application offering several functionalities, including: chatting, betting, watching IPTV, playing a multiplayer racing game. The design of the Social Game has been described in D5.1, with particular attention to possible migration scenarios. A first version of the application, in form of stand-alone prototype with migration processes simulated, has been implemented and described in D5.2.

In this deliverable, the design of an improved version of the Social Game, integrated with the MSP, based on the guidelines covered in 3.2, is described. It is worth noting that the design has been improved with respect to the initial one described in D5.1, considering the suggestions arose from the usability evaluation that has been collected in D6.5.

4.2.1. DESCRIPTION

The Social Game is inspired by a scenario in which the user has the possibility to compete against real pilots while watching a live grand prix event, and to interact with other members of a gaming community through a chat service. Some additional information about the track and the participants is also available, as well as pilots' performances and lap records. Moreover, the user has the possibility to bet on the live race's results according to different parameters. The prototype currently under development considers a simplified version of the scenario, thus some functionalities are missing or simulated.

As shown in Figure 20, the Social Game is organized into several elements, each taking up a different area of the screen, namely (from left to right): IPTV, Additional Content, Chat, Betting, and Racing Game. The access to some of these functionalities is handled by an authentication system.

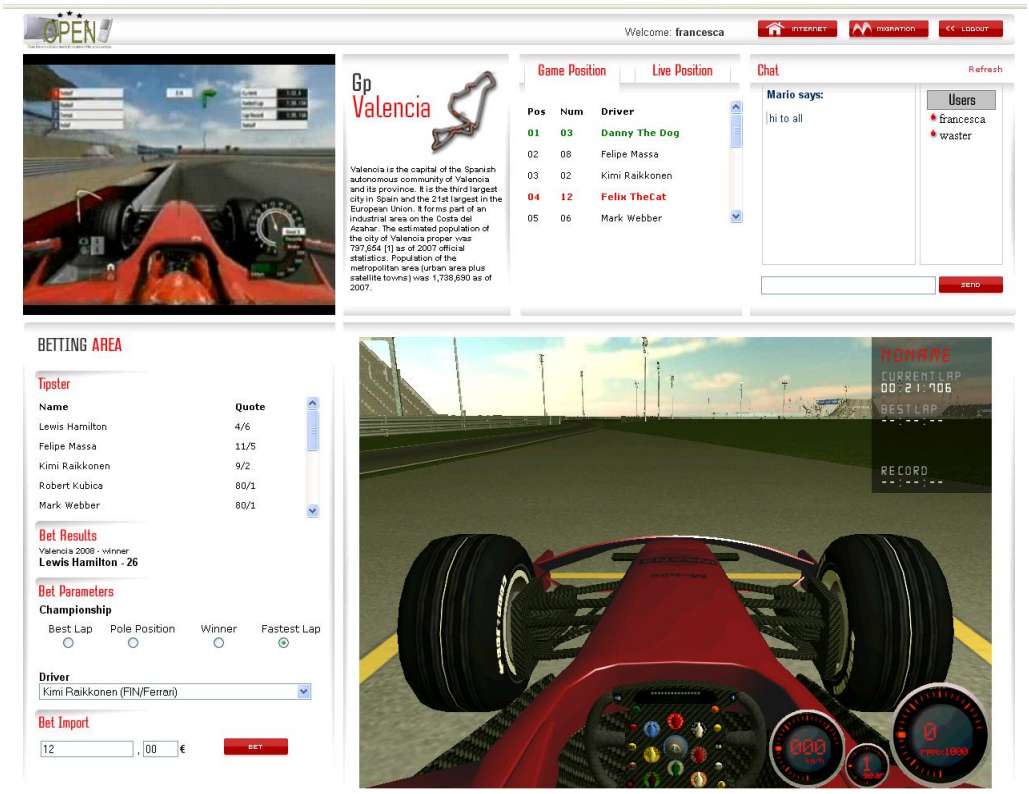

Figure 20: screenshot of the Social Game.

The elements exploit different technologies and this makes the application an interesting use case for experimenting with migration functionalities across various platforms.

The IPTV (on the top-left) is simulated through a Flash player and allows to watch in streaming recorded videos, to switch among the channels, to pause and restart the video being watched, as well as to control the volume and to switch to full-screen mode. With respect to the previous version, the GUI will be improved by taking into account the suggestions by the users involved in the usability testing. For example, the buttons for switching among the channels will be inverted.

The Additional Information (on the top-middle) contains details about the track, the ranking of the real race and the ranking of the real race merged with that of the racing game.

The Chat (on the top-right) is Javascript based and provides basic chat functionalities, such as a dialog window and the buddy list. Several suggestions for improving the service arose from the usability evaluation, especially because the chat is a very familiar and widespread tool, and users have high expectations from it. However, since the requested changes require a significant effort and they do not

add anything relevant to the project, it has been decided to focus on issues more related to migration rather than improving this particular tool.

The Betting (on the bottom-left) is a Javascript and Ajax service, which allows the user to bet on the live race results, according to different parameters. The first version used a set of pop-ups for showing the various betting steps to the user. To make the process smoother and to guarantee the continuity of it after a migration, especially when different devices are involved, it has been chosen to remove the pop-ups and to present the various steps in the same area of the screen. During the usability evaluation, users suggested some improvements that will be considered when implementing the next version of the prototype, e.g.: a clearer indication of bet parameters when a user performs a bet or a back button for allowing the modification of inserted data.

The Racing Game (on the bottom-right) is a 3D multiplayer game in which players compete against each other in scoring the best lap time, while driving a F1 car. The technology adopted is based on an embedded browser object in form of plugin implemented in C++. The game can be controlled by the keyboard and the mouse of a desktop PC or, alternatively, by the keyboard of a mobile phone. Being a prototype, the users found room for improvement, especially regarding car's controls and manageability, probably because they compared it to existing commercial products. Nevertheless, also in this case, it has been decided to concentrate the effort in the general migration aspects instead of refining this part of the prototype, since this would not be relevant with respect to the scope of the OPEN project.

As for the authentication system, to access the Racing Game, the Betting and the Chat, the user needs to log into the application using her/his own username and password. Most of the users, in the usability evaluation, asked to remove the confirmation pop-up and to allow the login procedure to be done entirely using the keyboard. These and other similar requests will be addressed in the final version of the prototype.

Regarding the migration controls (in the topmost-right), they will be adapted for supporting partial migration, by allowing the user to select which component(s) s/he wants to migrate and where.

4.2.2. SPECIFIC REQUIREMENTS

Other than the generic application requirements listed in Section 2, from the Social Game scenario, more specific requirements have been elicited. Some of these were extracted and selected from D5.1, while some others have been defined starting from the usability evaluation results described in D6.5. In the following tables the selected requirements are listed. It is worth noting that some of them will not be entirely addressed, since they come from a very complex scenario that goes beyond the actual implementation of the final prototype, or go beyond the scope of the OPEN project (e.g. they are too related to usability issues, being more important for commercial aims rather than for research purposes).

ID	1
Description	If the application is running on the mobile Phone, then: • IPTV is not available in the mobile screen and it is eliminated from the application's options • Several tabs enable the user to switch to social betting, messaging, gaming or internet access options from the mobile phone
Type	Functional
Rationale	There can be usability issues and network resources constraints for having IPTV service on mobile

ID	3
Description	The migration can be: • Total, all components of the application (i.e. betting, messaging, gaming or internet access option) migrate • Partial, only a part of the application migrates (i.e. the dashboard of the game application or the betting option)
Type	Functional
Rationale	N/A

ID	4
Description	The visualization for large screen can be: • Single user

	• Multiple user
Type	Functional
Rationale	N/A

ID	5

Description	For Single user visualization:
	• The tabs displayed are:
	  ◦ Game
	  ◦ Messaging (chat, betting with tab to switch and chat as the default)
	  ◦ Internet access
	  ◦ IPTV
	  ◦ Display settings
	  ◦ Picture in Picture (PIP) between IPTV and game
	  ◦ Fixed schema
	  ◦ For PIP switch function between IPTV and gaming
	  ◦ Migration, with the previous requirements
	• If no migration has been performed before, then the default visualization is IPTV
	• After a migration towards the large screen all previous selected services are activated

Type	Look and Feel
Rationale	N/A

ID	6
Description	For Multiple user visualization:

	• If no migration has been performed before, then the default visualization is IPTV • When other players join, the screen is partitioned according to the total number of them, with a maximum supported • Private contents, such as chat and betting, automatically migrate to mobile phone (private phone)
Type	Look and Feel
Rationale	N/A

ID	8
Description	As for betting, the user should be able to: • See tipsters and betting parameters • Check bet results • Pay by credit card or a mobile operator prepaid wallet
Type	Functional
Rationale	N/A

ID	9
Description	The game can be played by a single or multiple users. The game dashboard can be visualized through widgets. Some migratory rules exist, being the following an example: when the user has done a pit stop, only the game dashboard is migrated automatically. One or more of the game opponents are represented by avatars, with positions

	determined by real telemetry data.
Type	Functional
Rationale	N/A

ID	10
Description	For both large and small displays, the messaging features allow the user to: • See the buddies list (with presence information such as: on-line, off-line, gaming, watching TV with channel info) • Start chat sessions • chat
Type	Functional
Rationale	N/A

ID	11
Description	IPTV capabilities are: • streaming • EPG (Electronic Program Guide) with channel selection • additional contents linked to TV channels • change view (switching cameras during race streaming)
Type	Functional
Rationale	N/A

ID	12
Description	Every part that cannot be migrated to the new device because of hardware constrains, could be migrated in an alternative way. For instance, when Brad is helping his father and carrying the mobile phone with him, he could receive a vibration for every lap that he is missing
Type	Functional
Rationale	N/A

ID	13
Description	During the migration process, the application should have a standby or auto-drive option that allows the user to have no disadvantage for migrating the application.
Type	Functional
Rationale	N/A

ID	14
Description	When the game is migrated in a device with no resources for the rendering, the game is rendered in a server in the network and the rendering will be streamed towards the device.
Type	Functional
Rationale	N/A

ID	20
Description	Social Game application components should offer the option to be resized, moved or closed. For example, a user could use the betting component to bet on his preferred driver, and then close it when it is no longer needed.
Type	Functional
Rationale	Coming from user suggestions

ID	21
Description	The chat should offer the option to view a buddy list with the status of every buddy
Type	Functional
Rationale	Coming from user suggestions

ID	22
Description	The chat should have the option to send a private message to other users
Type	Functional
Rationale	Coming from user suggestions

ID	23
Description	The racing game should be paused during migrations and at the web application startup
Type	Functional
Rationale	Coming from user suggestions

ID	24
Description	In the racing game classification, besides the user position, also the time employed to complete a lap should be indicated.
Type	Functional
Rationale	Coming from user suggestions

ID	25
Description	During the betting procedure a “back” button for the modification of inserted data is needed
Type	Functional
Rationale	Coming from user suggestions

ID	26
Description	The betting component should offer an indication of the amount of money that the user could win with his bet.
Type	Functional
Rationale	Coming from user suggestions

ID	27
Description	The quote indication of betting should indicate the type of bet performed (best lap, winner, etc.).
Type	Functional
Rationale	Coming from user suggestions

4.2.3. APPLICATION LOGIC

Following the guidelines described in 3.2.1, the Social Game has been organized into a set of components, each of them having its own requirements in terms of hardware, software, network and UI. For partitioning the application into components, the main criterion that has been adopted is based on the provided functionalities. Therefore, one component for each element in Figure 20 has been introduced, plus two input components for controlling the racing game: one using the keyboard and the mouse of a desktop PC, the other one using the keyboard of a mobile phone. In principle, each component is independent from the other ones and can be separately migrated, except for the racing game which needs at least one input component to function correctly.

Being the Social Game a web application, the container for all the components is a web page. Hence, for the registration phase it must be provided to the MSP a launch script containing the command for opening the web browser at a specific url that links to the web page.

The Social Game supports several configurations according to any combinations of active components. The combinations are numerous and can be expressed during the registration phase, either listing the

allowed combinations, the forbidden ones, or a mix of them. In this case, the easiest way is to forbid the configurations that use the racing game with more than one input component or no input at all, and to permit all the other combinations.

The introduced components are: IPTV, AdditionalInfo, Chat, Betting, RacingGame, RacingGamePCInput, RacingGameMobileInput. In the following, the information provided during the registration of each component is described.

The IPTV component does not directly interact with the other components. To correctly initialize the IPTV component, the MSP must know the url(s) of the stream source: this information is provided at registration time in the component description. In order to allow continuity after the migration, both user settings and video state must be preserved. In particular, the component must be able to restore the current seek position of the video stream together with all the interface related settings. Table 5 summarizes the information needed at registration time.

IPTV Component		
ComponentDescription	• verbose description of the component in human-readable format • list of URLs containing the video streams for each channel • id of the div containing the component	
ComponentRelationship	• this component does not depend on other components	
State	• initial state of the component, in terms of: volume level, selected channel, full-screen mode, seek position	
Requirements	Hardware	• CPU able to decode smoothly the adopted video format • support for stereo sound
	Software	• Flash support
	Network	• internet connection with adequate bandwidth for streaming the video
	UI	• minimum video area size that allows for video content to be intelligible (e.g. 160x120 pixels)

Table 5: IPTV component

The AdditionalInfo retrieves racing information from the application server, thus it does not need to interact with the other components. During the registration phase, the url of the application server must be provided to the MSP. The state to be preserved in case of migration is related to the current UI state. In particular it contains the selected tab and the displayed inner data. There are no particular requirements since it is a relatively simple component. Table 6 summarizes the information needed at registration time.

AdditionalInfo Component		
ComponentDescription	• verbose description of the component in human-readable format • URL of server containing the racing data • id of the div containing the component	
ComponentRelationship	• this component does not depend on other components, since input data stay on the application server	
State	• initial state of the component, in terms of selected tab and displayed data	
Requirements	Hardware	• no significant hardware requirements
	Software	• XHTML and JavaScript
	Network	• network connection available
	UI	• no significant UI requirements

Table 6: AdditionalInfo component

The Chat needs a chat server, which can be eventually the same of the application server and must be provided at registration time. The Chat component does not interact with other components, but the user must be logged in order to use the chat service. The state of the chat must contain the session information and, to provide continuity, the last message written by the user and not yet sent. There are no particular requirements for this component to be run, except for the Ajax support inside the browser. Table 7 summarizes the information needed at registration time.

Chat Component	
ComponentDescription	• verbose description of the component in human-readable format • URL of chat server (eventually the same of application server) • id of the div containing the component

ComponentRelationship	• this component does not depend on other components, since input data stay on the application server	
State	• initial state of the component, in terms of user session and the last message not sent to the server	
Requirements	Hardware	• no significant hardware requirements
	Software	• XHTML, JavaScript and Ajax support
	Network	• network connection available
	UI	• no significant UI requirements

Table 7: Chat component

The Betting component is independent from the other components. To be initialized, it needs the application server address, which is provided in the component description, Moreover, to guarantee the continuity of the betting process, the state must contain the current step and corresponding data inserted by the user. Regarding the requirements, considering that sensitive data are transferred, the browser must support secure connections. Furthermore, the user must be logged to access the service. Table 8 summarizes the information needed at registration time.

Betting Component		
ComponentDescription	• verbose description of the component in human-readable format • URL of application server • id of the div containing the component	
ComponentRelationship	• this component does not depend on other components	
State	• initial state of the component, in terms of user's session, current betting step and values of the filled fields in the current step	
Requirements	Hardware	• no significant hardware requirements
	Software	• XHTML and JavaScript • Browser supporting secure connection
	Network	• secure network connection available
	UI	• no significant UI requirements

Table 8: Betting component

The RacingGame needs at least one of the two input components between the RacingGamePCInput and the RacingGameMobileInput. Moreover, it needs to know the addresses of the physics and the game servers, which are provided to the platform during the registration procedure. In case of migration, the user settings must be preserved, while the game state stays on game server. In order to be run, the RacingGame needs some requirements to be fulfilled, regarding hardware, software and network. Table 9 summarizes the information needed at registration time.

RacingGame Component		
ComponentDescription	• verbose description of the component in human-readable format • URLs of game server and physics server • id of the div containing the component	
ComponentRelationship	• this component needs at least one input component, between RacingGamePCInput and RacingGameMobileInput.	
State	• initial state of the component, in terms of user's settings regarding view point and level of detail.	
Requirements	Hardware	• Intel or AMD CPU running at 2Ghz with 1Gb RAM, • ATI or NVIDIA GPU supporting Shader Model 2.0, • 100Mb of free disk space • display with 1280x1024 resolution
	Software	• Windows XP SP2 • Mozilla Firefox 3.x • JavaScript
	Network	• network connection with at least 2 Mbit bandwidth and a set of open TCP and UDP ports
	UI	• minimum game area size of 640x480 pixels

Table 9: RacingGame component

The RacingGamePCInput is one of the possible inputs of the RacingGame component. During the registration phase it needs the url of the game server. The state of this component contains user settings regarding controls. Table 10 summarizes the information needed at registration time.

RacingGamePCInput Component		
ComponentDescription	• verbose description of the component in human-readable format • URL of game server	
ComponentRelationship	• this component can be used only in a configuration which includes the RacingGame component	
State	• initial state of the component, in terms of user's settings regarding controls	
Requirements	Hardware	• Standard PC Keyboard and Mouse
	Software	• Windows XP SP2 • Mozilla Firefox 3.x
	Network	• network connection with at least 2 Mbit bandwidth and a set of open TCP and UDP ports
	UI	• component does not have a visible interface

Table 10: RacingGamePCInput component

The RacingGameMobileInput is the other possible input of the RacingGame component. During the registration phase it needs the url of the game server. The state of this component contains user settings regarding sensitivity of controls. Since it runs on a mobile device, specific hardware and software requirements must be fulfilled, which are summarized in Table 11 together with the information needed at registration.

RacingGameMobileInput Component	
ComponentDescription	• verbose description of the component in human-readable format • URL of game server

ComponentRelationship	• this component can be used only in a configuration which includes the RacingGame component	
State	• initial state of the component, in terms of user's settings regarding sensitivity of controls	
Requirements	Hardware	• A mobile phone supporting Connected Limited Device Configuration (e.g. Symbian OS based Nokia Series 60 Phones)
	Software	• Mobile Information Device Profile (MIDP) 2.0
	Network	• UMTS, H3GA or wifi network connection
	UI	• Minimum screen size of 128x128 pixels

Table 11: RacingGameMobileInput component

4.2.4. USER INTERFACE

Following the guidelines described in 3.2.2, the User Interface of the Social Game has been adapted by creating a correspondence between each component and its interface representation. Being the Social Game a web application, this correspondence has been implemented by explicitly indicating in the description of each visible component the id of the div containing the html code of the considered component. This parameter is passed during the registration procedure, as summarized in the previous tables (from Table 5 to Table 9). In order to support the web adaptation functionalities provided by the MSP, all the web pages of the Social Game are being validated according to the W3C XHTML specifications (see [8]).

Moreover, to improve the effectiveness of the UI adaptation, all the pop-ups are being removed and replaced with more standard interaction forms. See Figure 21 that shows the subsequent screenshots of the Betting steps, from left to right and from top to bottom.

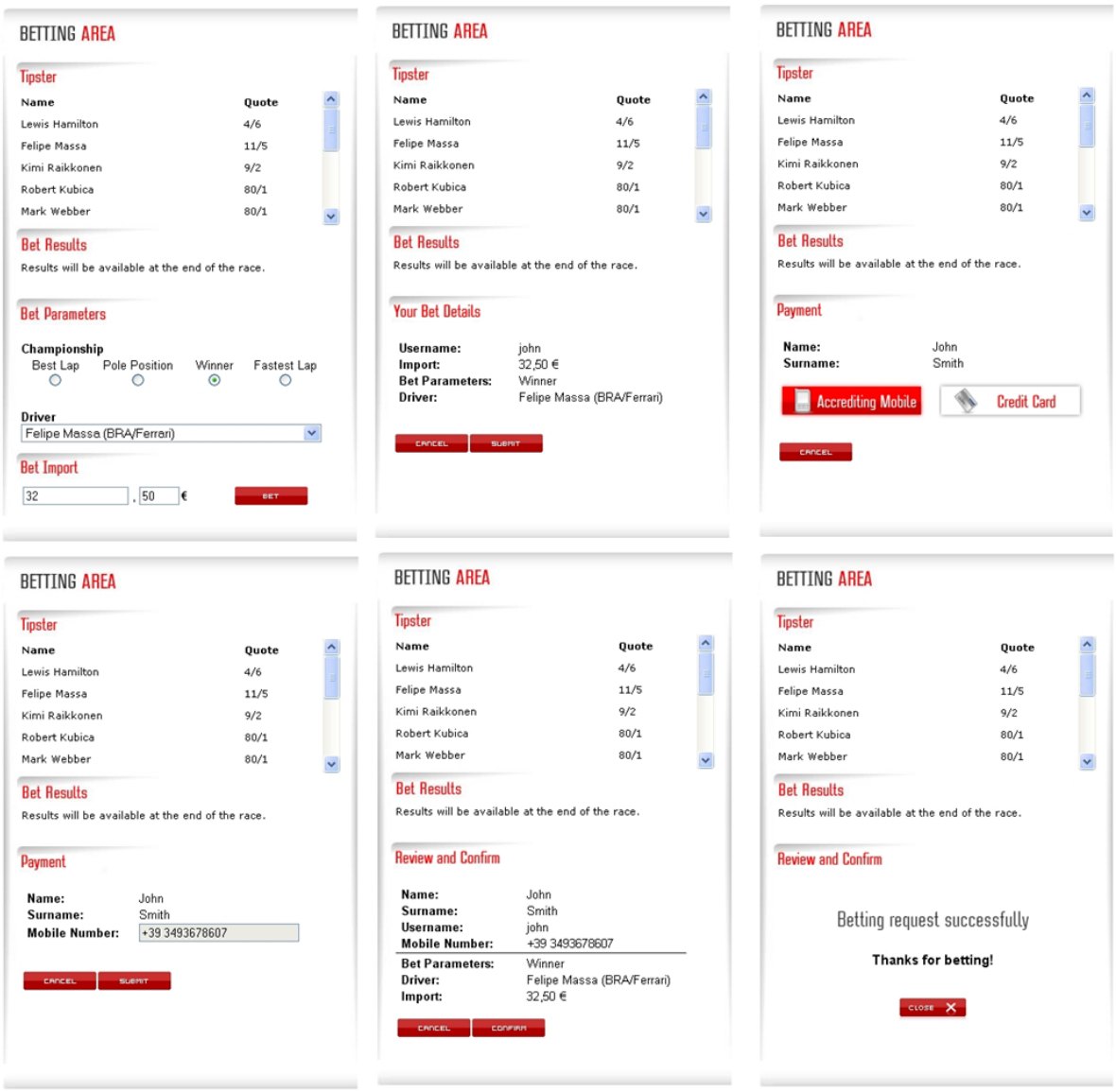

Figure 21: steps of the Betting procedure in the Social Game

As described in 3.3, the user must be aware of which components and devices are involved in the migration process and of the status of the migration process. In the Social Game the user can select which component s/he wants migrate and to which device. To this end, the UI is being improved by additional elements allowing to select the components to be migrated and the available devices, as shown in Figure 22.

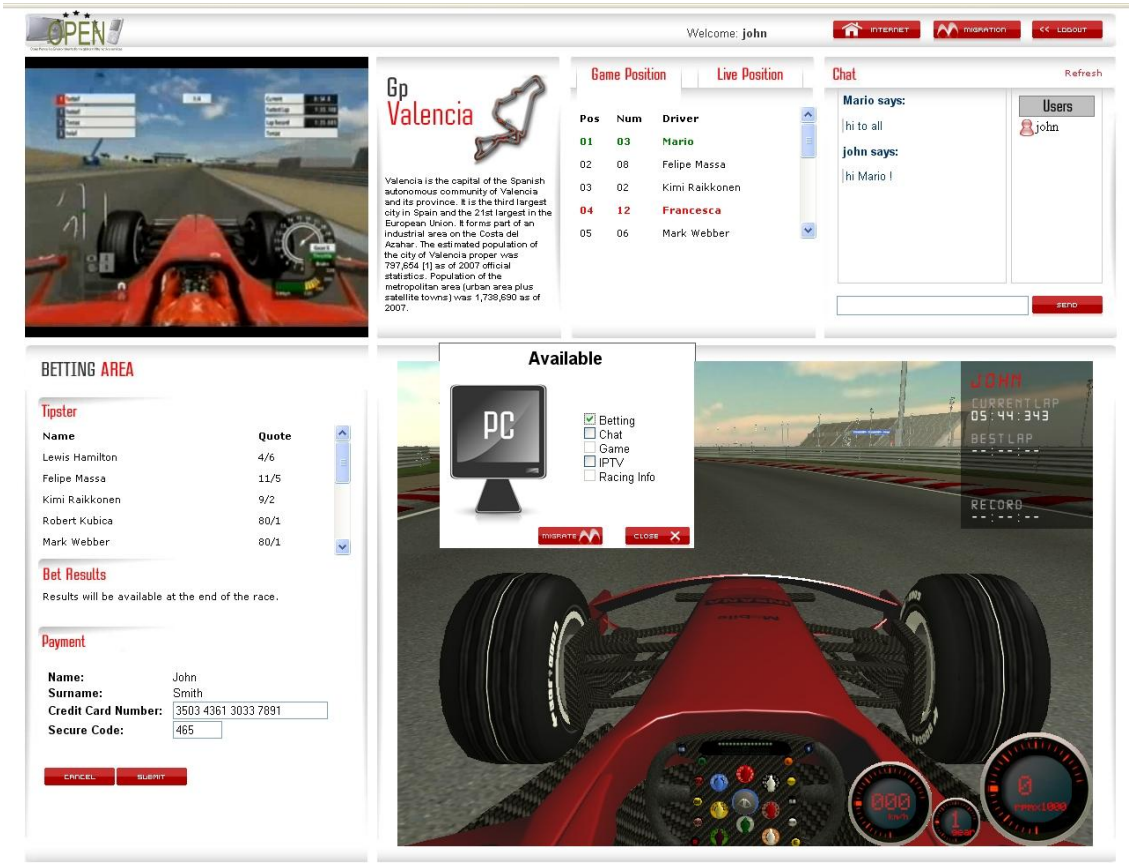

Figure 22: components selection menu in the Social Game

Regarding the migration process, to make the user aware of what is happening, the inactive components are colored in gray (see Figure 23) and a status message showing the current migration step is displayed.

Figure 23: inactive Betting component shown in dotted green

Moreover, the interface of the inactive components is frozen and the user cannot interact with them, until the migration process has completed.

4.2.5. NETWORK

Social Game's components use different network connections. For example, the Chat is connected to the Chat server, the Racing Game is connected to the physics server and the game server. To make these connections more flexible and transparent with respect to devices and network change, the Social Game will exploit the adaptor provided by the MSP for network access, instead of using system's standard direct connections. In particular, each component that needs network connection will provide its network requirements during the registration phase. In this way the Mobility Support can provide to the application the most suitable network configuration according to context information and available network capabilities.

A more specific analysis of the network support for the prototype will be conducted once the final version of the Mobility Support module will be available, which is currently under development and will be described in D3.4.

4.2.6. CONTEXT

The Social Game needs to know information about the devices available at a certain time. The prototype is being arranged to support interaction with the CMF through a suitable client side Context Agent. Two different modalities are feasible: in the first one, the application requests directly to the CMF the needed context information; in the other one, the application listens to context change information notified by the CMF. In particular the Social Game asks to the MSP the list of the available devices supporting the needed technologies for the specific components. These are shown on demand through a migration menu to the user, who is always in charge of taking decisions regarding the migration process.

4.2.7. POLICY

As suggested by users during the usability evaluation, who underlined the necessity of keeping secure certain kind of information, the Social Game needs a support from the MSP for handling with privacy and security issues, since it contains elements that manage private information and sensitive data. In fact the Chat component conveys private conversations among the users, the Betting component manages both personal data and sensitive information such as credit card number, banking account, etc. When a migration occurs, it must be ensured that personal data are kept private by allowing migration only to devices whose identity is reliably identified, and sensitive information are kept safe by migrating to devices that support secure connections. These mechanisms are handled by the Policy Management and the Policy Enforcement modules, which are currently under development.

A more specific analysis of the policy support for the prototype will be conducted once the final version of the Policy modules will be available.

4.2.8. ARCHITECTURE OF THE INTEGRATED PROTOTYPE

Recalling that the MSP supports different configurations in terms of client-server architecture, whose general form has been depicted in Figure 1, the Social Game exploits the configuration illustrated in Figure 24.

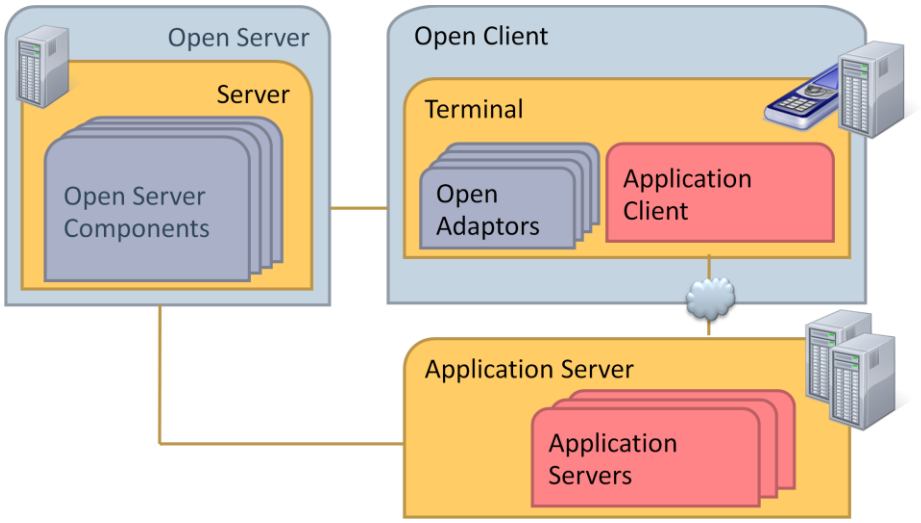

Figure 24: client-server architecture configuration of the Social Game.

The Social Game uses different network connection for communicating with a number of servers (Chat server, game server, physics server, application server, video streaming server), and some of them require specific protocols, e.g. for real time constraints. Each connection might be handled either by the MSP through the Mobility Support module or by a direct connection between the client and server parts of the application. Since the Mobility Support is still under development, and it will provide support only to a limited number of protocols, the Social Game has been designed to support both the solutions. The final choice will then depend on the actual implementation of the Mobility Support module.

The communications among the application and the MSP modules is based on XML-RPC protocol, thus the application must be able to send and receive XML-RPC messages. As previously mentioned, the Social Game components are implemented in different technologies and languages and some of them do not have native support for such a protocol. To allow all the components to communicate with the MSP and to provide a unique interface to the MSP, a dispatcher between the components and the MSP is needed. This dispatcher listens to incoming messages from the MSP, takes care of distributing them to the proper component(s) and vice versa.

Being the Social Game a web application, the dispatcher is being implemented in form of a Java Applet, which can be easily integrated within a browser and allows a smooth interaction with the web page and its components. The architecture of the client side of the Social Game is expanded in Figure 25.

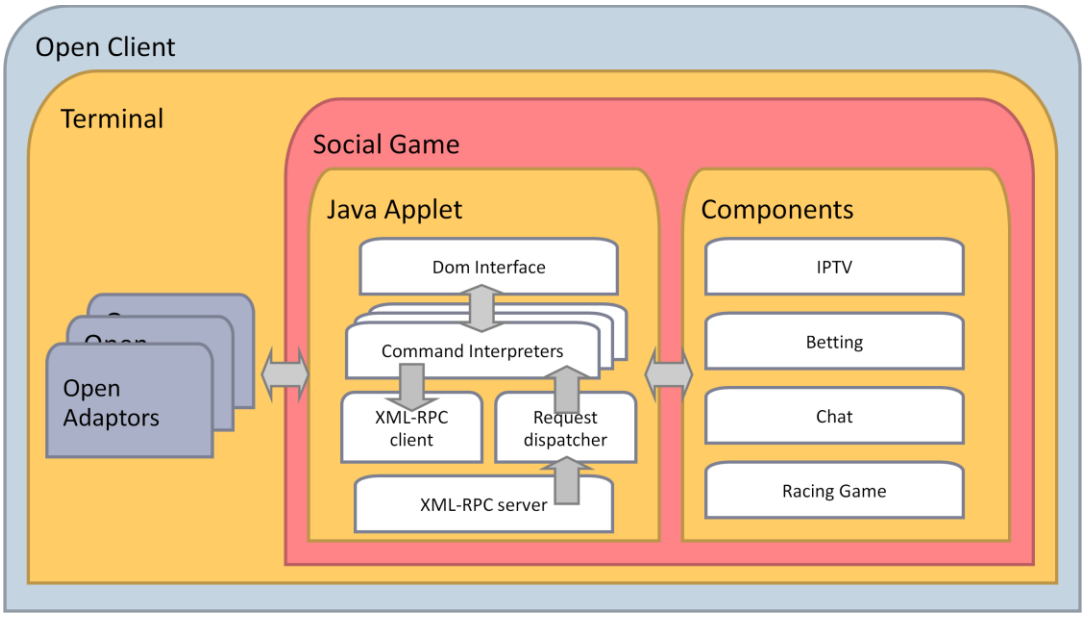

Figure 25: details of Social Game client side architecture.

The Java Applet provides a XML-RPC server for receiving XML-RPC messages from the needed Open Adaptors and a XML-RPC client for sending messages coming from the components. For each component there is a Command Interpreter, which translates the messages received from the MSP in a suitable format that is understandable by the addressed component, according to its specific technology. The involved interpreter is selected by the Request dispatcher, which analyzes the received message. Each interpreter is able to modify the web page through the Dom Interface, e.g. for changing the visual appearance of the controlled component, according to its current migration status.

The Social Game client supports the interaction with several Open Adaptors. In particular, the Migration Orchestration for handling all the migration steps, the CMF for getting information about available devices, the Mobility Support for masking network connections, the Web UI Adaptation for adapting the User Interface of some components.

4.2.9. EXAMPLES OF MIGRATION

The possibility to exploit migration capabilities in the Social Game enhances the user experience, by introducing new usage scenarios and giving continuity to ongoing tasks when switching from one device to another.

The Social Game has been devised to support different migration scenarios and subsequently adapted to be integrated with the MSP to exploit its migration functionalities. Being the Social Game organized into separate components, as well as some parts of it being usable in different devices, a large amount of possible configurations exists when considering the allowed combinations of components and devices. In this section some examples of migration that are significant for illustrating the benefits of the migration functionalities offered by the MSP, are described.

One interesting migration case arises when the user is following an event and s/he performing a bet on it, and s/he needs to change device, but s/he does not want to restart the entire betting procedure in the new device. In this case, the Social Game allows migrating the Betting component to the target device, preserving the already inserted data and starting from the point the procedure was interrupted before the migration had started. Figure 26 illustrates this process, which is an example of partial migration support with state persistence.

Figure 26: example of Betting migration from a PC to a laptop

Another exploitation of partial migration functionalities is illustrated in Figure 27, where the user wants to migrate one or more components of the Social Game to a mobile device. There can be several reasons to move parts of the application from a PC in a fixed workspace to a portable device, for example the need to continue the ongoing task in another place or the need to have private data in a device that is not shared. In the Social Game, it is possible to migrate the web components from a PC to a PDA, using the MSP functionalities that allow adaptation of UI according to the target device's capabilities, without any additional effort for the application developer. For example, the user could decide to migrate the race information to keep following the race results and the betting component to bet on the race events, while moving around with her/his mobile device.

Figure 27: example of partial migration from PC to PDA

A significant migration case for the Social Game is the migration and adaptation of the IPTV component from a laptop with limited network bandwidth and a small area dedicated to IPTV display, to a desktop PC with larger screen and a better network connectivity. Figure 28 shows the migration of a video between two devices with different capabilities. This case is useful when the user wants to enjoy the details of a video watching it in better quality and, at the same time, to interact with the community and the other features of the Social Game. The integrated version of the Social Game makes this possible, since it supports migration of videos preserving the seek position and all the user settings. In other words, the user can continue watching the video from the point s/he left it before s/he had started the migration.

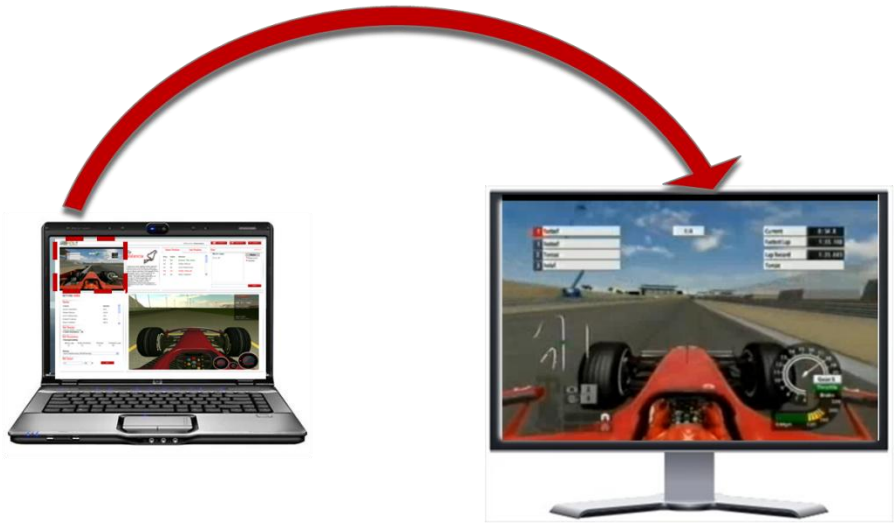

Figure 28: example of migration of IPTV from a laptop to a big screen

A further possible application of migration capabilities in the Social Game regards the ability to select an alternative control for the racing game. This case is illustrated in Figure 29, where on the left part of it a girl is interacting with all the elements of the Social Game through the keyboard. On the right part, the situation when a second user arrives is shown. The new user takes the control of the racing game with his mobile phone, whereas the girl continues using the other parts of the application through the keyboard. In particular, she can continue her chat session while the boy enjoys the game at the same time. In this way different users can share the application interacting with different parts of it.

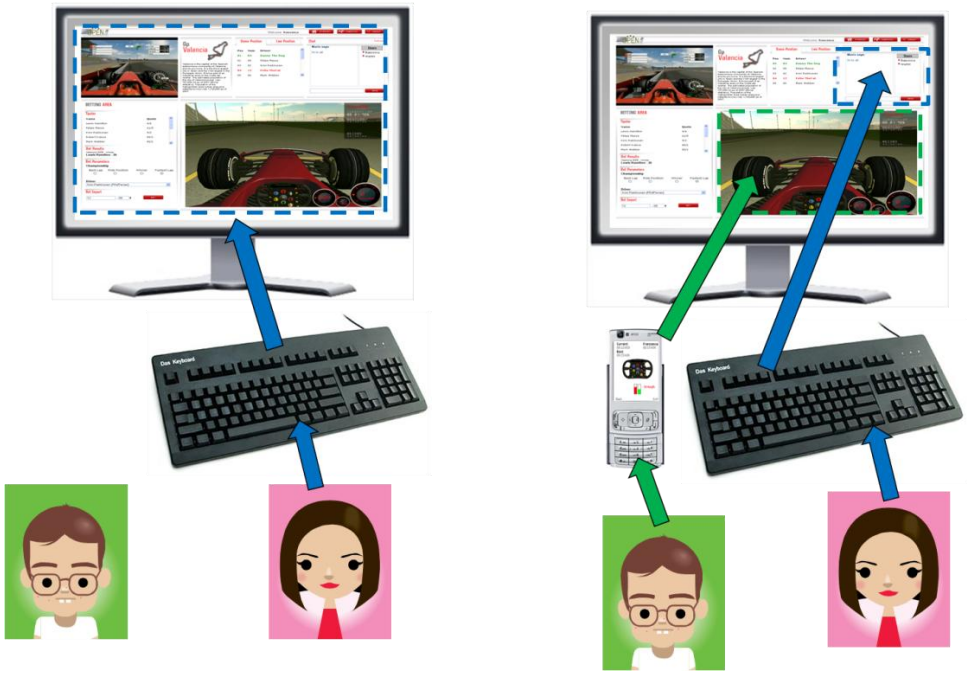

Figure 29: example of migration of controls for sharing the Social Game interface

The presented cases are just a sample of possible ways of exploiting the migration capabilities offered by the OPEN MSP. Many other combinations are of course feasible and those described so far have been selected to demonstrate, through the integrated prototype, some of the most important features offered by the MSP.

4.3. TWITTER WALL

The TwitterWall application is a social interaction tool where strangers can discover their common interests, thus starting a conversation.

When working individually, the application on the terminal constitutes a regular Twitter client. After migration, however, the joint large screen mode highlights the common interests of the two users, thus identifying ice breaking topics.

4.3.1. DESCRIPTION

The TwitterWall is a modular application the usage of which greatly differs between its multiuser and single user modes. The two modes will be separately described in the following.

TWITTERWALL IN SINGLE USER MODE

When in single user mode, the TwitterWall application behaves as a regular Twitter client, searching for specific keywords, as shown in Figure 30.

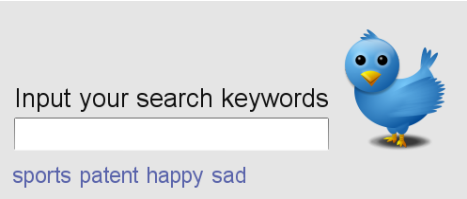

Figure 30: the TwitterWall in single user mode

When the user inputs a search keyword, the Twitter search API [7] is used to retrieve the most recent Tweets concerning that particular keyword, together with the Avatars corresponding to those users.

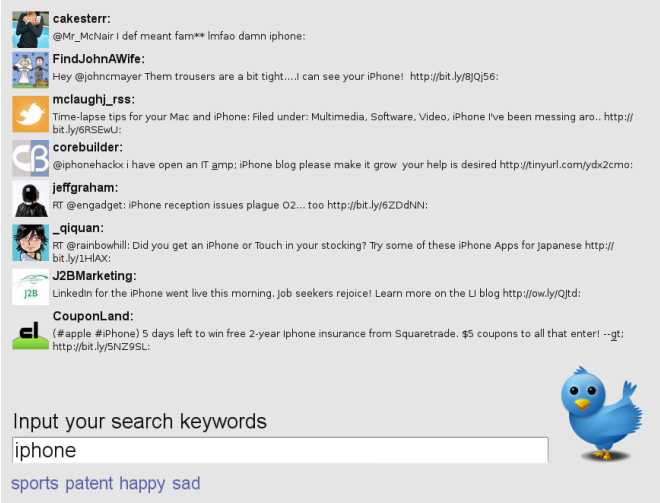

Figure 31: the TwitterWall after resolving a query, in single user mode

Figure 31 shows a sample result for the query keyword “iPhone”. It is important to understand that, in terms of Application Logic Reconfiguration, both the “input” and “output” functionalities of the TwitterWall are on the terminal, since both the queries and the visualization of the results occur there.

TWITTERWALL IN MULTI USER MODE

In the multi user mode the TwitterWall shows all its functionalities. Upon being partially migrated to a multiuser environment (from the terminal to a big public display), the input and output functionalities become separated: the input remains on the terminals of each involved user, while the output is moved to the shared display, where the results of all the queries are integrated.

Figure 32 shows the input screen that would run on the terminal of each participating user when in multiuser mode. Note that there is no visualization area, but rather a very simplified interface with a large search key (the t icon) to query for the chosen keywords.

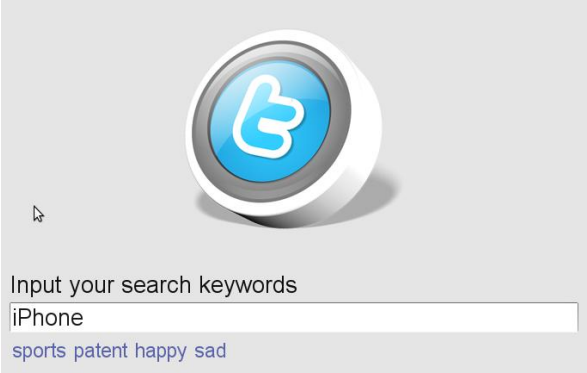

Figure 32: input mode of the TwitterWall application on the terminal when in multi-user mode

Figure 33 on the other hand, shows the shared screen that will be displayed on the public display.

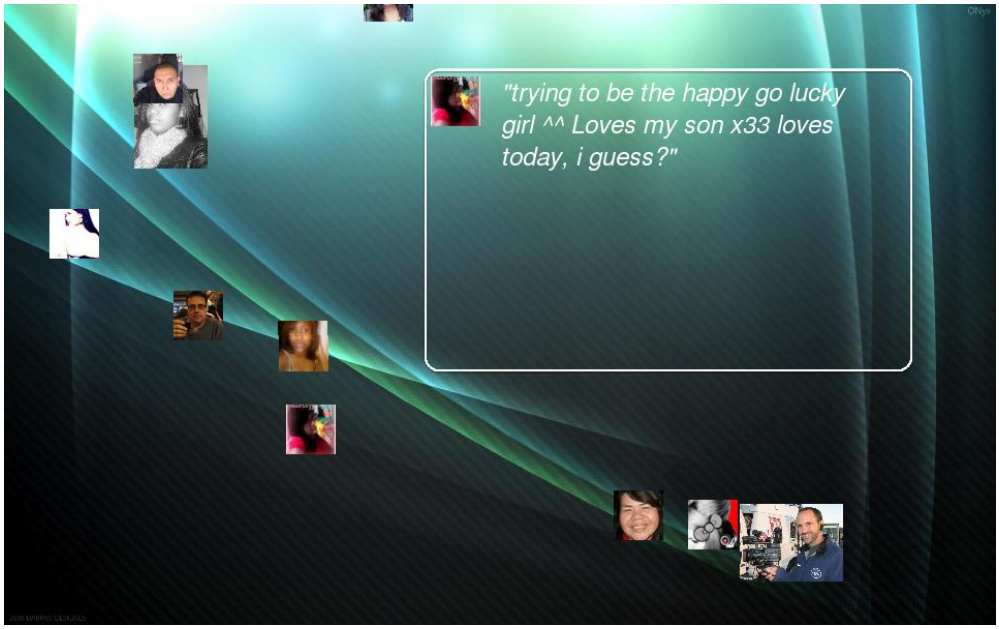

Figure 33: the TwitterWall in shared mode on a public display

This output module collects all the Tweets relevant to the keywords of the multiple users, and grabs the avatars of the users whom the tweets belong to. These ones are shown floating downwards and rightwards on the screen. Additionally, highlighted tweets are shown on the square on the top right. When an overlap is detected between the tweets obtained from the keywords of two users, that tweet is immediately highlighted, providing the users with feedback on their common topics.

In summary, the TwitterWall application is a single user application which, under the right conditions (see section 4.3.3) will be split into an input and output module. When in split mode, the application becomes multiuser, and prompts users to start a conversation by exploring their topics of common interest.

4.3.2. SPECIFIC REQUIREMENTS

The TwitterWall requires that users run the Open Client on their terminals. As far as sensing is concerned, to trigger the application startup at least Bluetooth is required, with RFIDs on the terminal being an additional mechanism for users to express their willingness to migrate, as explained in section 4.3.3.

Additionally, if the target public display is connected to a multicore PC, the Namuco library will optimize the output module for enhanced graphics that makes full usage of the capabilities of the processor.

In more detail, the following tables show the requirements of the application:

ID	1
Description	The public display needs to be connected to: • An RFID reader • A Bluetooth dongle
Type	Hardware
Rationale	These sensors are used by the Device Discovery and the Trigger Management for their operation

ID	2
Description	The Input and Input+Output components need to be installed in the user's terminals. The Output component needs to be installed on the Public Display

Type	Functional
Rationale	The application components are assumed to be pre-installed in the devices

ID	3
Description	Internet access is required for the Twitter data retrieval
Type	Functional
Rationale	Tweets are retrieved using the Twitter.com API

4.3.3. APPLICATION LOGIC

Following the guidelines described in 3.2.1, the TwitterWall has been organized into a set of components, each of them having its own requirements in terms of hardware, software, network and UI. For partitioning the application into components, the main criterion that has been adopted is related to the implementation of functions such as “input” or “output”, which in the TwitterWall application correspond to the search for keywords and the display of the Tweets associated to those keywords respectively.

The main components are: Input, Output, Input+Output. The different components have different requirements in terms of hardware. For instance, the Output component runs on a powerful PC likely connected to a big display and, when possible, with multicore capabilities. The following tables provide detailed information about each component and its relations to the other components.

Input+Output Component	
ComponentDescription	This component runs on the mobile terminal and has two main functions. From the input point of view, it allows users to type in keywords for which they want to find Tweets. From the output point of view, a list of related Tweets and the speaker’s avatar is shown in a list.
ComponentRelationship	When using this component, the application is already complete and does not need any other application components.

State	The state is persisted using the normal Open methods. Eventually, the searched keywords and user session are stored in the CMF, where they can also be accessed by the Trigger Management for Keyword blacklisting as explained in section 4.3.7	
Requirements	Hardware	This is a very lightweight component which runs on Windows machines. A port to a mobile phone would also be feasible
	Software	Java SE >= 1.5
	Network	The same as the Open Client (an IP connection with access to the Open Server)
	UI	SWT (Standard Widget Toolkit)

Table 12: Input+Output Component

Input Component		
ComponentDescription	This component is analogous to the Input+Output component, but the list of Tweets is not present. It has, however, the same field to input keywords and a search button.	
ComponentRelationship	When the application uses the Input Component, it requires a corresponding Output component. Multiple Input components can share the output one.	
State	The state is persisted using the same Open methods as in the Input+Output component	
Requirements	Hardware	This is a very lightweight component which runs on Windows machines. A port to a mobile phone would also be feasible
	Software	Java SE >= 1.5
	Network	The same as the Open Client (an IP connection with access to the Open Server)
	UI	SWT (Standard Widget Toolkit)

Table 13: Input Component

Output Component		
ComponentDescription	This is a graphically appealing component meant to display and highlight Tweets in a large public display. It highlights them according to the user, and points out overlaps.	
ComponentRelationship	This component requires one or more Input components that provide the keywords on which to search	
State	This component is stateless, or rather, depends directly on the state of the Input component	
Requirements	Hardware	This component requires a powerful machine in terms of graphics. For our demonstration, both a Multicore and a Singlecore machine are used to show the difference given the usage of the Namuco library
	Software	Java SE >= 1.5
	Network	The same as the Open Client (an IP connection with access to the Open Server)
	UI	SWT and the Namuco library

Table 14: Output Component

A possible example of supported configuration would involve a list of components with an input component for keyword search and an output component to output Tweets, plus an additional component that includes both input and output. For an application instance, only an output is allowed, but multiple inputs or input+output are allowed. The split functionality with an input component running on a device and an output component running on another one is preferred.

4.3.4. USER INTERFACE

The user interface of the TwitterWall application was designed with migration and multiple users in mind. As such, each of the modes has a distinct interface to help the user understanding what is going on, as shown in Figure 34.

Figure 34: User Interface for the TwitterWall application, showing from left to right the input+output component, the output component and the input component

Each of the components possesses only the necessary UI widgets, e.g. preventing the user from typing on the public display or not showing any output at all in the input only module. This is meant to improve usability by drawing clear boundaries between the different functionalities available to the user at each step.

MULTICORE PLATFORMS

The TwitterWall specifically tries to show the advantages of the Namuco library that makes the most out of the capabilities of the target platform. Specifically, the output module is graphically intensive, and will automatically detect the capabilities of the device it is running on, limiting its complexity to what is available.

In a multicore platform, the component will exploit the full advantage of the Namuco library and therefore engage in more appealing visual effects, while on a single core machine, this will be simplified to guarantee that the application can run smoothly enough.

This features, therefore, an adaptation of the UI based on the capabilities of the target platform, which are automatically detected by the Open aware Namuco library, which will report on capabilities so that the application can choose its operating mode.

4.3.5. NETWORK

The TwitterWall application works on heterogeneous networks, with the only condition that the different components should be able to see each other on an IP level.

Because the application is not bandwidth intensive, no further adjustments are needed.

4.3.6. CONTEXT

To handle context information the TwitterWall needs to interact with the CMF, which plays a double role. Firstly, it reads physical sensors, but just as importantly, it acts as a database for accumulated data, such as the available devices. In this second role, the CMF behaves as a message buffer to connect the different modules.

The physical sensors read by the CMF are:

- An RFID reader installed next to the display, where users can swipe their mobile phones to indicate their willingness to migrate
- A Bluetooth scanner used to discover nearby devices, such as the mobile terminals of users or the big display

There is also another special sensor read from the CMF, shown in relation 4: the keywords that users are searching. This is funneled into the CMF because the Trigger Management enforces privacy policies based on what is being displayed.

The following table summarizes the data that is made available through the CMF.

Information available through the CMF	
RFID sensor	This retriever will report on RFID cards that go through the field of an RFID reader installed under the public displays
Bluetooth sensor	This retriever continuously scans for devices. It allows to discover whether the user's mobile terminals are physically near the public displays
Keywords searched	The keywords that users search are sent over to the CMF for storage and later retrieval by the Trigger Management
Device proximity	This module will combine Bluetooth information with any prior knowledge of a location to provide a list of available devices

Table 15: information provided by the CMF

4.3.7. POLICY

A specific, dedicated policy module is not used in this application. Instead, given the heterogeneity of required policies, a modular approach through the Trigger Management scoring functions has been taken.

One such example is the privacy scoring function, which will remove the application from public view when blacklisted words are about to be displayed.

4.3.8. ARCHITECTURE OF THE INTEGRATED PROTOTYPE

The TwitterWall application involves the interplay of most of the modules of the MSP. To facilitate explanations, we will continuously refer to Figure 35 and the numbers above the arrows, which refer to

the different interactions. We will further split the explanation by modules, and put it all together at the end of the section.

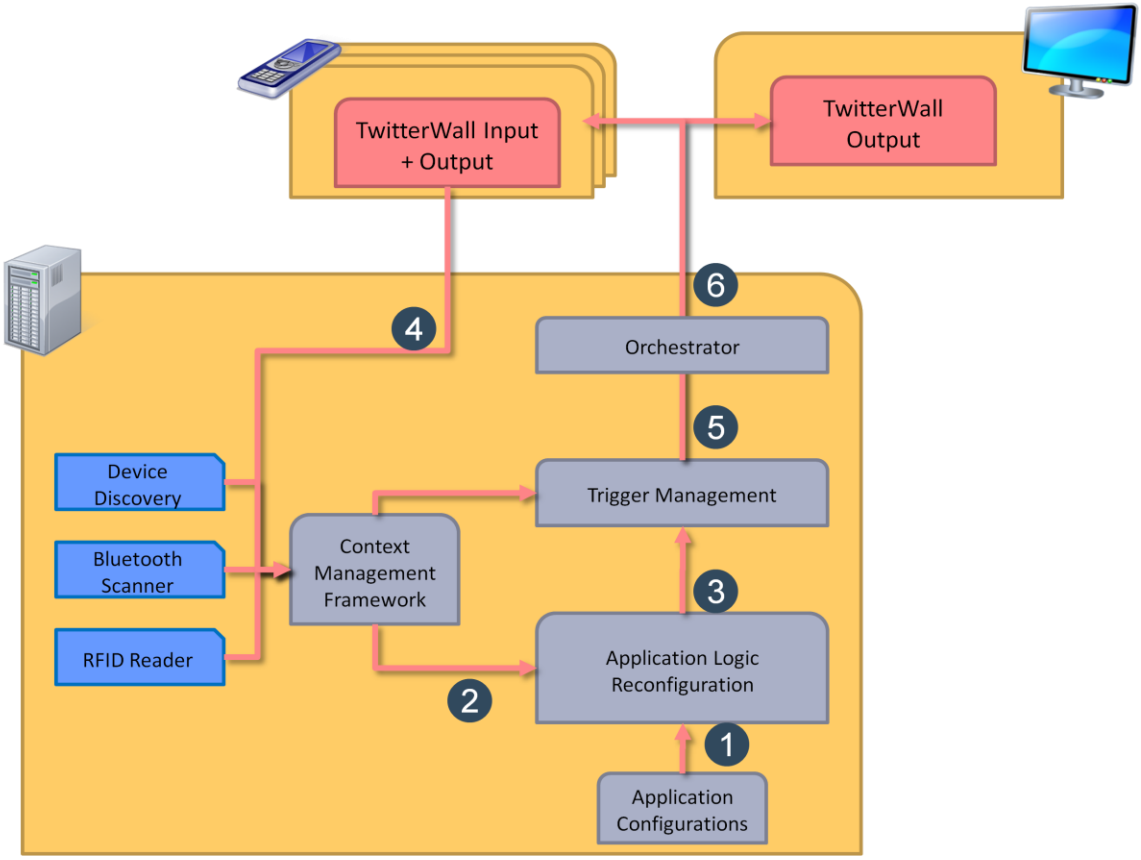

Figure 35: interactions between Open modules in the TwitterWall application

APPLICATION LOGIC RECONFIGURATION

Upon installing an application, the Orchestrator will store the list of components and their requirements for the application. Relation 1 (in Figure 35) shows the income of the list of components and their requirements, while relation 2 shows the input from the device discovery adaptor.

The function of the Application Logic Reconfiguration is to consider the components involved in the application and what they require, and compare them with the devices that are available around the user. It then proposes (Relation 3) the list of possible configurations, prioritized in a way that maximizes the usability of the application.

The Device Discovery then provides the list of devices (Relation 2), which in this example could include two mobile devices and a big display.

Through its internal logic, a sorted list is provided (Relation 3): the top priority would be to put an input component on each mobile terminal and an output component for both (multiuser) on the big display. Barring that, the second choice is to run an Input+Output component on each mobile terminal, and leave the big display be. This, however, is less preferred because it neglects the multiuser capabilities of the application which is, after all, meant as an ice breaker.

CONTEXT MANAGEMENT FRAMEWORK

The interaction with the CMF has been already described in 4.3.6.

TRIGGER MANAGEMENT

The trigger management is responsible for triggering the migration. This is done by taking in the list of possible configurations from the Application Logic Reconfiguration (ALR) in relation 3 and choosing the best one. Now, "best" is calculated according to an array of modular scoring functions, which provide a weighted score for each configuration.

From a high-level overview, the Trigger Management is a framework for modular scoring functions that evaluate the goodness of the possible configurations (Relation 3) and output a trigger (Relation 5) to the orchestrator. The modularity of the scoring functions is what guarantees the reusability and flexibility of the trigger management.

In the specific case of the TwitterWall application, the following scoring functions come into play:

- Application preferences: the ALR sends a list of possible configurations to the Trigger Management. Each configuration is tagged with a score, so that the application developer can propose the best configurations for its application. In the case of the TwitterWall, for instance, the multiuser, split mode (partly on the terminal, partly on the display) is preferred because it provides the Ice Breaking functionality intended for the application.
- Privacy policies: whenever users input keywords, these are communicated to this module, which has a configurable black list. If certain private words are chosen, the score for the "split mode" (input on the terminal, and public output on the big display) will plummet. This will force the application to revert to the privacy of the user's own terminal, and away from the public view. This is to prevent private information from being displayed to everyone.
- User willingness: this module scores the likelihood that a user would like to migrate the application from the privacy of his own mobile phone to a public display. Several factors are considered here: firstly, the "openness of the user" which is, in essence, a preference for sharing data or not. Secondly, the actions the user performs to indicate his willingness to migrate. One example of the latter is the user swiping his mobile phone by the RFID reader installed under the public display, which indicates he would like to use it.

Notice that only the preferred configuration is used to trigger migration, so only when this changes the orchestrator will be notified.

ORCHESTRATOR

The orchestrator here performs the usual functions it does in the MSP. It is worth noting that the migrations it orchestrates (relation 6) are the result of the combination of all the previous modules (relation 5).

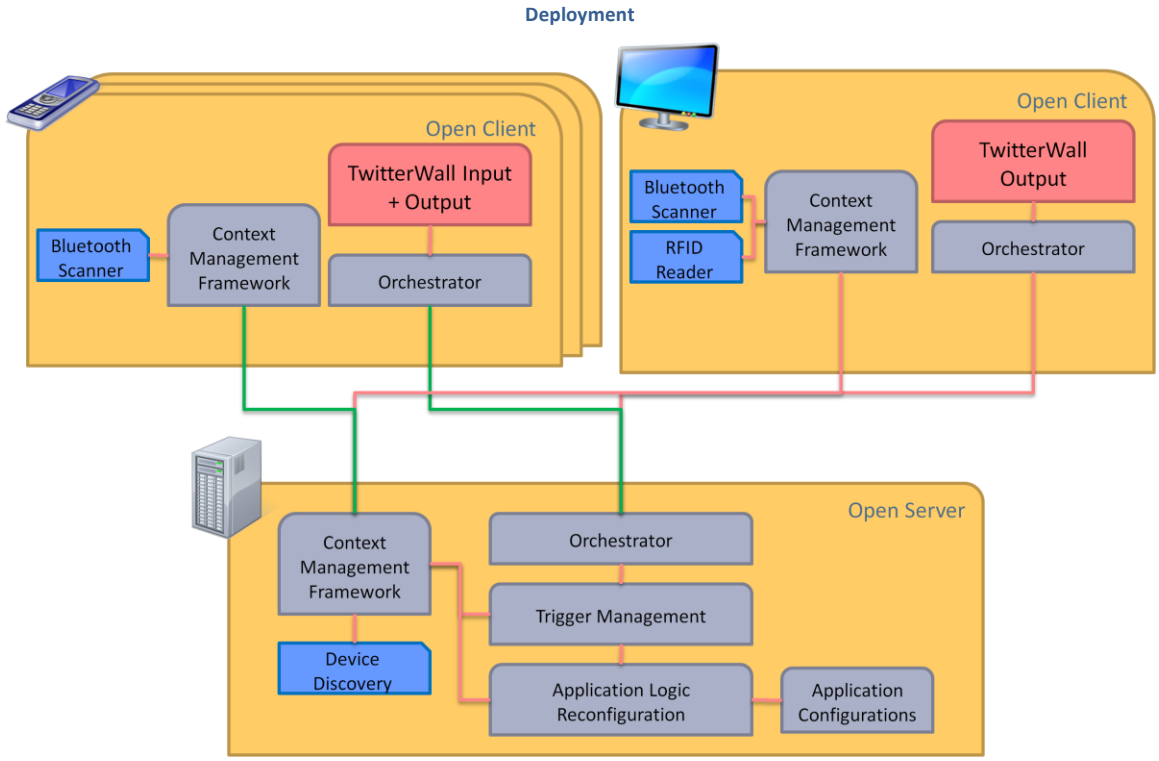

Figure 36 illustrates how the different components are deployed in an Open scenario.

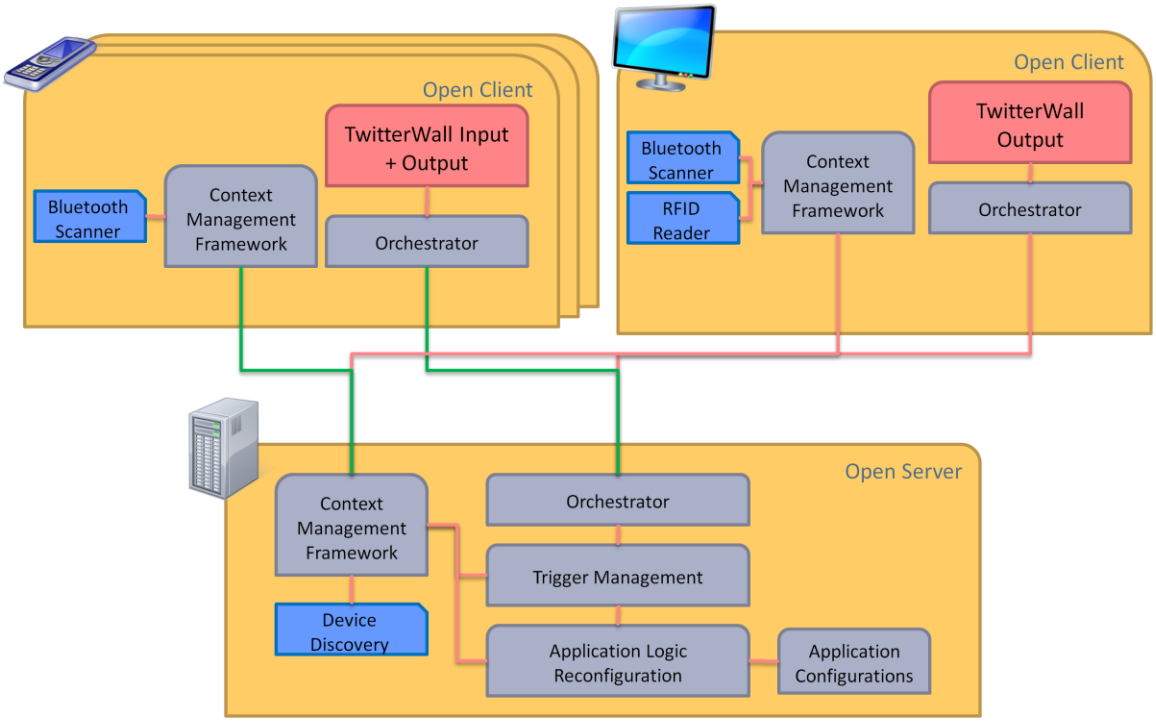

Figure 36: architecture of the TwitterWall application

The typical deployment of the TwitterWall application will involve one Open Server and any number of Open Clients. However, two types of Open Clients with different capabilities are featured. The first is a mobile terminal, which can accept user input, and provides a rather poor output. The second is the Open Client that runs on a big public display. This one does not have any input capabilities, but has a very rich output. It can be further subdivided into one that has multicore support and another one that does not support it, to illustrate the differences when this type of UI adaptation is used.

4.3.9. EXAMPLES OF MIGRATION

The TwitterWall will feature three different migrations, all of them partial, which are shown in Table 16.

Migrations in the TwitterWall application		
Initial State	**Final State**	**Description**
Input+Output on the user's terminal	Input on the user's terminal, output on the public display	The user is working on his mobile terminal, when the Trigger Management accepts the ALR configuration that migrates his output in a public display

A user has input+output on his terminal, and another one has input on the terminal and output on the public display	Each user has the input on his terminal, but the output is shared, becoming multiuser	A user is added to the public display, so that a single output component is used for two (or any number) of different users
Multiple users with output on the public display and input on each terminal	One of the users is removed from the public display (both input and output go to his/her terminal) while the rest of users remain unaffected	The Trigger Management decides the user is not suitable to use the public display anymore (e.g. usage of blacklisted words, walking away from the display…), so that user is removed

Table 16: migration examples in the TwitterWall

4.4. PAC-MAN

The PacMan prototype is a web application based on Javascript. A first version of the application, in form of a stand-alone prototype that uses the Application Logic Reconfiguration functionality has been implemented and described in D4.3.

In this deliverable, the design of an improved version of the PacMan prototype, integrated with the UI adaptation module is described.

4.4.1. DESCRIPTION

Pac-Man is a game where a character called Pacman, which is steered by the user, has to collect dots in a maze, as shown in Figure 37.

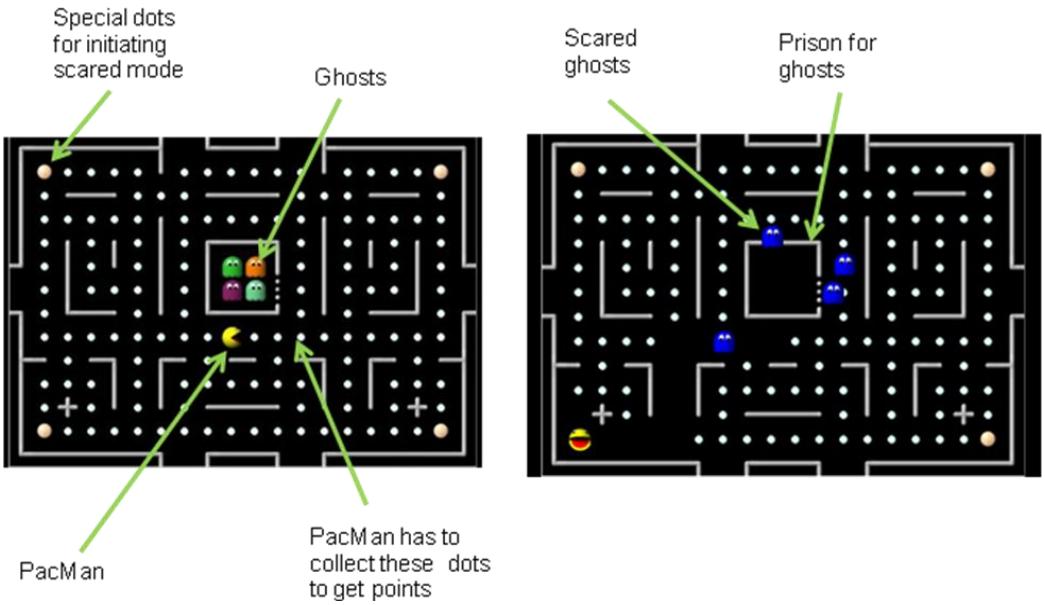

Figure 37: the two modes of a single PacMan game. On the left hand side the game is in normal mode where the ghosts try to catch the PacMan. On the right side the game is in scared mode where the PacMan can catch the ghosts.

Ghosts, who are controlled by the computer, are running around with the goal to catch the Pacman. If the Pacman collects special dots, ghosts and Pacman change roles for some seconds, as depicted on the right hand side of Figure 37. That means that the Pacman now can catch ghosts and that the ghosts try to run away. Caught ghosts will be imprisoned in the middle of the maze for some seconds. After some seconds the roles change back again. The goal for the player is to get as much scores as possible by collecting dots and catching ghosts.

4.4.2. SPECIFIC REQUIREMENTS

In the development of the PacMan prototype several requirements listed in D1.1 section 7.2 are being considered. The following requirements are fully or partially satisfied by the PacMan prototype:

ID	65
Requirement	The user should only get presentations with possible migration options.
Type	Look and Feel
Rationale	Having a general list of 50 items with 40 not working is not fun.

ID	78
Requirement	Gaming anywhere, anytime, anyhow.
Type	Functional
Rationale	People like to spend time in games.

ID	156
Requirement	The input devices must be able to support the same actions
Type	Functional
Description	If the game need a 4 directions input, all the devices must support it.

ID	74
Requirement	Users must be able to migrate identified parts of the application to other devices e.g. with high score list.
Type	Look and Feel
Description	User has control

4.4.3. APPLICATION LOGIC

On a desktop device the user can control the PacMan in an optimal way since s/he can see the playing field of the PacMan game in big size, the keyboard has big keys and it can be placed in an optimal position. On a mobile device the control of the PacMan is much more difficult since the playing field is very small, the control panel is fixed directly below the display and the keys are relatively small. The PacMan prototype can be migrated from the desktop to the mobile device. To ensure that the usability and the fairness of the game are maintained, the PacMan game has to be easier to play on the mobile device. For this reason, the ghost logic of the PacMan prototype has to be adapted in the migration process, by selecting intelligent ghost logic in the desktop device and easy ghost logic in the mobile device.

Regarding the state of the game, it is preserved and then re-activated in the mobile device at the point it was when the migration was triggered. Thus, the current PacMan and ghost positions are saved, together with the current level, the score and the other settings that the player had already selected in the desktop version. The player can continue from this point when s/he accesses the mobile version.

The PacMan prototype game is organized into several components, which communicate among them through a web service. In general a game consists of two main elements: UI and application logic. The game UI is realized by the PacMan UI component and the application logic is represented by the ghost logic component. The components of the PacMan prototype are described in the following tables.

PacMan UI component		
ComponentDescription	• It provides the UI of the game. • It shows the playing field of the Game and processes the input of the users.	
ComponentRelationship	• It depends on the ghost logic component. The Pacman UI is connected to the ghost logic component through a web service.	
State	• Initial state: game is ready to start. Component is connected to the ghost logic component.	
Requirements	Hardware	• Device with network connection.
	Software	• Browser with Javascript enabled.
	Network	• Internet connection to the required web service.
	UI	• no significant UI requirements

Table 17: PacMan UI component

Ghost logic component		
ComponentDescription	• The ghost logic component computes the movement of the ghosts. • It provides an interface for the Pacman UI component which is realized by a web service. • From this interface the UI gets the direction in which the ghosts have to move next.	
ComponentRelationship	• it depends on special ghost logic components (e.g. intelligent or easy ghost logic)	
State	• initial state: intelligent ghost logic	
Requirements	Hardware	• CPU able to compute intelligent ghost logic.
	Software	
	Network	• Internet connection for the provided web service.
	UI	• component does not have a visible interface

Table 18: Ghost logic component

Intelligent ghost logic component		
ComponentDescription	• The ghost logic component computes the movement of the ghosts. The movement depends on the position of the Pacman.	
ComponentRelationship	• Component depends on the ghost logic component, which provides the web service for the UI.	
State	• initial state: active	
Requirements	Hardware	• CPU able to compute strategic ghost logic • A big display to ensure the fairness of the

		game
	Software	
	Network	• component does not use network
	UI	• component does not have a visible interface

Table 19: Intelligent ghost logic component

Easy ghost logic component		
ComponentDescription	• The ghost logic component computes the movement of the ghosts. The movement of the ghost is random.	
ComponentRelationship	• Component depends on the ghost logic component which provides the web service for the UI.	
State	• initial state: inactive	
Requirements	Hardware	• No significant hardware requirements
	Software	
	Network	• component does not use network
	UI	• component does not have a visible interface

Table 20: Easy ghost logic component

4.4.4. USER INTERFACE

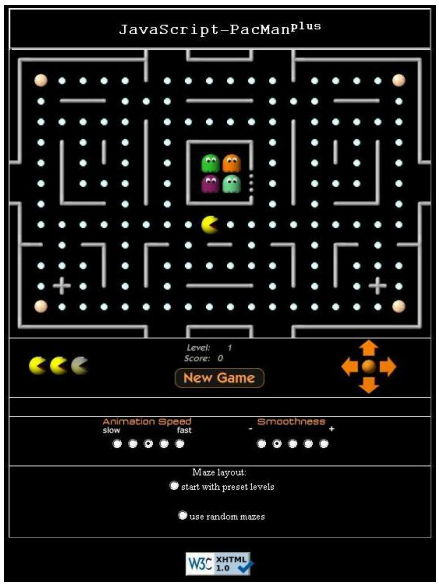

Figure 38: the original desktop version of the PacMan.

As shown in Figure 38, the desktop UI of the game includes different parts: the maze of the game with the different characters (the ghosts and the Pacman), a part devoted to visualizing the current state of the game (the number of PacMan lives still available, the current level of the game, the score), the 'New Game' button to start a new game, the settings for specifying the general configuration of the game (e.g. the animation speed of the PacMan, the possibility of using random maze layouts for the various levels, etc.), and the interaction elements for controlling the game. Indeed, in the desktop version, the PacMan can be controlled either using the keyboard (e.g. arrow keys can be used to specify directions) or by using an imagemap (visualized beside the maze), in which five regions are identified: four of them are associated to the north/south/east/west directions, the central one is for pausing the game. Then, the user can click a specific region of the image or even pass the mouse over such a region (event "onMouseOver") in order to activate a function allowing to set the new direction that the PacMan should take. The version of the PacMan prototype for the desktop platform is transformed by the MSP in order to obtain a new version adapted for the mobile device. In order to do this, a reverse engineering of PacMan for the web desktop platform is needed in order to obtain its logical UI description, which will be then redesigned to finally generate the adapted user interface for the mobile device.

When migrating to the mobile device, appropriate page splitting is performed. On the one hand, all the controls for the general configuration of the game are grouped together into a newly generated presentation, which is accessed by activating the "Settings" button (see Figure 39) in the first presentation. On the other hand, given the wide (relatively to mobile devices) screen available (an iPhone with a screen with 320x480 pixels) and since the PacMan is a quite interactive game, the design should not force the user to use zooming/scrolling during the game. Therefore, in this case the maze should fit entirely in the iPhone screen, without forcing the user to scroll the page for visualizing parts of the maze (see Figure 39).

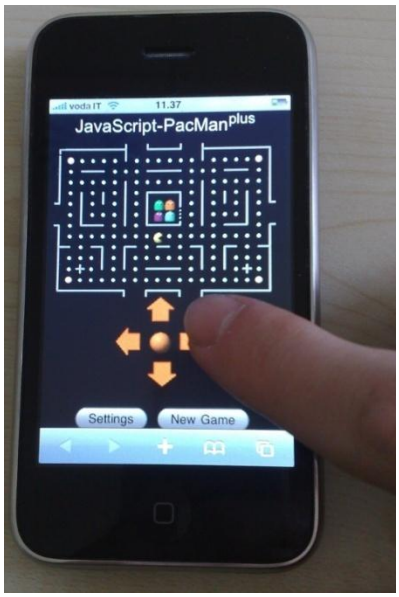

Figure 39: the adapted iPhone version.

The adaptation takes into account that touch-based interaction will be performed, thus the selectable elements should be large enough (or distant enough from each other) to be touchable without creating confusion about what the selected element is. Therefore, when adapting the image map supporting the game controls to the iPhone platform, it will be ensured that the adaptation engine provides controls large enough to support an easy-to-tap interaction. In addition, in the desktop version there were some actions activated by "onMouseOver" events (e.g. for moving the PacMan), which are not supported on the iPhone, because there is no direct equivalent of this event on the iPhone (as well as other touch-based devices). Therefore, in the adapted version the onMouseOver events have been removed and the interaction has been made more explicit (by tapping on the corresponding element).

Moreover, in the desktop version of the game, there were two possibilities for controlling the game (through the keyboard or by using the imagemap). Since a touch-screen mobile device is used as migration target, the keyboard controls are considered by the adaptation engine not suitable on this type of platform and they are removed from the mobile version. Indeed, the iPhone does not have a physical keyboard, and triggering the appearance of the virtual keyboard would clutter the screen and obscure some parts of the user interface (e.g.: the maze), which is not fair for the player who is currently using the mobile device. Therefore, in this case, the support for controlling the PacMan is obtained through touch-based interactions, with the five buttons that have been made available on the IPhone platform (see Figure 39) and that allow the user to control the game. Such buttons are automatically derived from the original image map of the desktop version, by selecting the different areas assigned by the image_map and mapping them to a set of buttons.

4.4.5. NETWORK

The components of the PacMan prototype use several network connections. The browser of the clients has to connect to the web server of the PacMan prototype over a HTTP connection. During this connection the Javascript code of the PacMan prototype is transferred to the clients. To get the directions of the ghost the UI component of the Game has to connect to the application logic component (ghost logic) through the web service.

4.4.6. CONTEXT

To retrieve context information, the PacMan prototype needs to connect to the context agent provided by the CMF module of the MSP. In particular the game needs the screen size of the current device In order to perform the UI adaptation.

After the migration of the PacMan prototype, the actual screen size is updated in the CMF. The application logic reconfiguration (ALR) is registered at the CMF to obtain the screen size value. So it gets informed when this value changes.

4.4.7. POLICY

Currently, no private information is displayed or gathered by any component of the PacMan prototype. Therefore, there has been no need for the adoption of policy rules regarding that issue so far.

4.4.8. ARCHITECTURE OF THE INTEGRATED PROTOTYPE

In the PacMan prototype two Open Clients are used: the first one for the mobile device, the second one for the desktop device. The PacMan prototype runs in a browser both on desktop and on mobile device. Like depicted in Figure 40, the browser of the devices are connected to the Application Server with the PacMan web server to get the PacMan game as Javascript embedded in HTML.

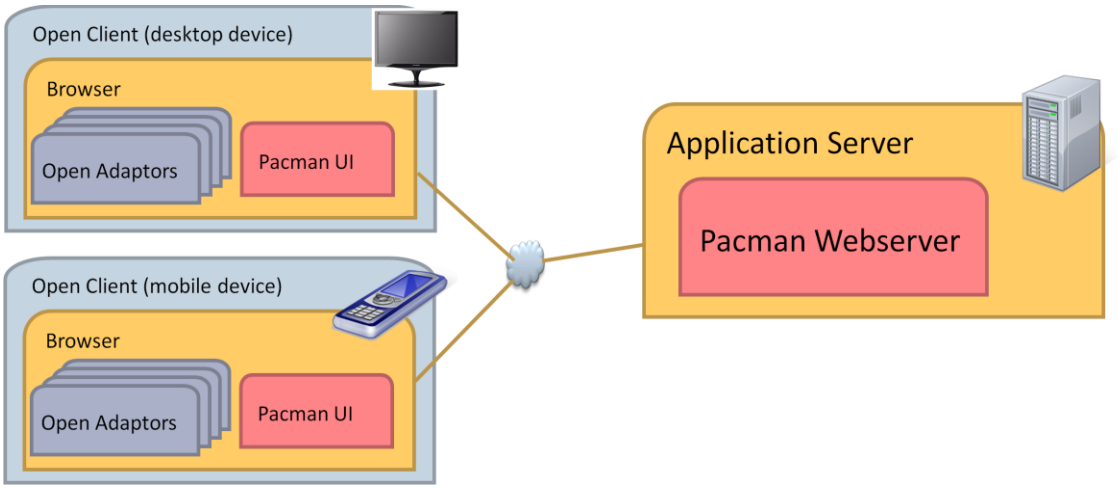

Figure 40: The UI of the Pacman Game on the mobile and desktop device

In addition to desktop and mobile Open Clients, another Open Client is used for the ghost logic component. As depicted in **Errore. L'origine riferimento non è stata trovata.**, the browser with the active acMan UI is connected to the ghost logic component. The Open Clients of the PacMan UI and the ghost logic component are also connected to the Open Server. When the PacMan prototype is migrated to another client (e.g. mobile to desktop device) the Open Adaptors reconfigure the PacMan UI and the ghost logic component.

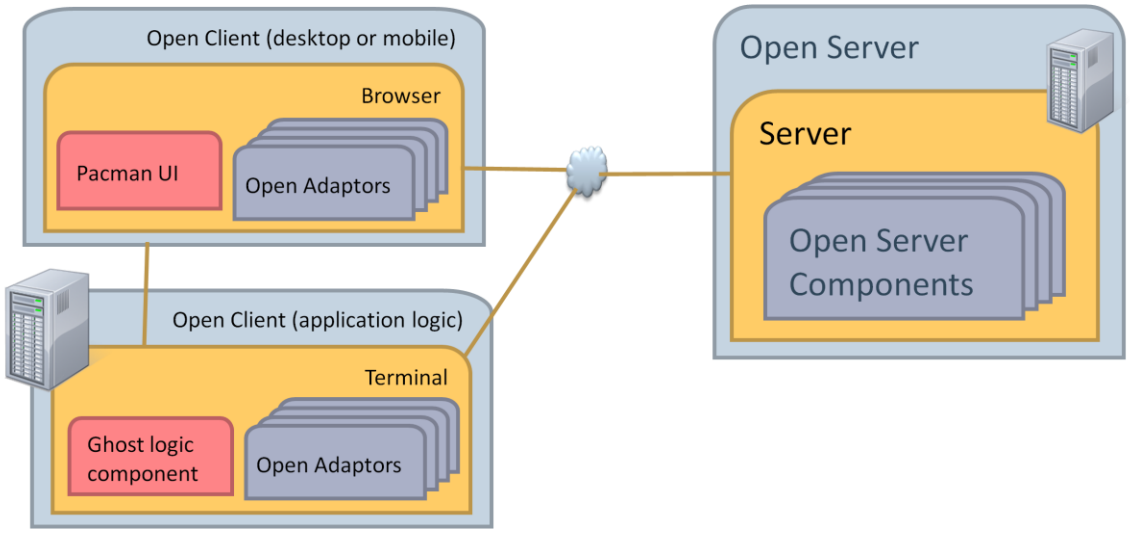

Figure 41: the Open Clients and the Open Server of the Pacman prototype

When the PacMan UI is migrated, the Context Agent commits the type of the target device. The ALR gets informed about all the updates of the device type value. After the update the ALR recovers the screen size value and computes a table of all the possible configurations sorted by most suitably and the best configuration is then forwarded to the Orchestrator. The Orchestrator initiates the reconfiguration of the ghost logic component depending on the configuration received from the ALR.

4.4.9. EXAMPLES OF MIGRATION

The prototype is designed to work on two different device types: a desktop and a mobile device. The desktop device has a big display and a keyboard to control the game. The mobile device has a small display and just a small control panel. The game can be migrated from the desktop to the mobile device and vice versa.

The benefit of the migration with the Open Platform is the optimized adaptation of the game logic and the UI to the current available devices. Therefore, the application developer has only to develop the different components and to define the rules for the adaptation. For instance the developer could only implement different ghost logic components and define rules for the Open Platform to decide which configuration is optimal for a specific circumstance. The reconfiguration logic and the game logic are completely separated and can be easily adapted. Without the support provided by the MSP, it would be needed to implement a large number of branches of the PacMan game and change the source code of it for every new possible configuration.

5. CONCLUSIONS

In this deliverable, the final requirements for migratory applications have been provided, together with the general guidelines for designing a migratory application compliant with the OPEN technology and able to take advantage of the capabilities of the Migration Service Platform. Moreover, the design of four integrated prototypes, namely Emergency, Social Game, Twitter Wall and Pac-Man, has been presented. The prototypes serve as examples of migratory applications that exploit MSP' capabilities. The final version of the prototypes will be released in D5.4 and evaluated in D6.7.

6. REFERENCES

[1] " Requirements for OPEN Service Platform", EU FP7 ICT project OPEN, Deliverable D1.1

[2] " Final requirements for OPEN Service Platform", EU FP7 ICT project OPEN, Deliverable D1.3

[3] " Final OPEN Service Platform architectural framework", EU FP7 ICT project OPEN, Deliverable D1.4

[4] "Migration Service Platform design", EU FP7 ICT project OPEN, Deliverable D4.2

[5] "Initial application requirements and design", EU FP7 ICT project OPEN, Deliverable D5.1

[6] "Evaluation results", EU FP7 ICT project OPEN, Deliverable D6.5

[7] "Twitter API", http://apiwiki.twitter.com/

[8] "W3C recommendation", http://www.w3.org/TR/xhtml1/