PersOnalized Smart Environments to increase Inclusion of people with DOwn’s syNdrome

Deliverable D5.1

Development framework

Call: FP7-ICT-2013-10
Objective: ICT-2013.5.3 ICT for smart and personalised inclusion
Contractual delivery date: M6
Actual delivery date: 15.05.2015
Version: V2
Author: Lars Thomas Boye, Tellu AS
Contributors: Andreas Braun, Fraunhofer
Reviewers: Andreas Braun, Fraunhofer
Juan Carlos Augusto, MU
Dissemination level: Public
Number of pages: 29
Contents

Executive summary ........................................................................................................... 4
1. Introduction .................................................................................................................... 5
   Framework content ........................................................................................................ 5
   First version of deliverable .......................................................................................... 6
   Second version of deliverable ....................................................................................... 6
2. Requirements .................................................................................................................. 7
   Overall .......................................................................................................................... 8
   Fr1 – Support POSEIDON ......................................................................................... 8
   Interconnection ............................................................................................................. 9
   Fr2 – Interconnection .................................................................................................. 9
   Fr4 – Mobile use .......................................................................................................... 9
   Fr19 – Real-time communication ............................................................................... 9
   Fr20 – Asynchronous communication ....................................................................... 9
   Fr22 – Extensibility ....................................................................................................... 10
   Interconnection strategy ............................................................................................ 10
   Client .............................................................................................................................. 11
   Fr3 – Devices ................................................................................................................ 11
   Fr14 – App platforms .................................................................................................. 11
   Fr16 – Distribution ...................................................................................................... 11
   Functionality support .................................................................................................. 12
   Fr5 – Safety ................................................................................................................ 12
   Fr6 – Privacy ................................................................................................................ 12
   Fr7 – Context-awareness ............................................................................................ 12
   Fr8 – Interface customization ..................................................................................... 13
   Fr9 – Data storage ........................................................................................................ 13
   Non-functional properties ........................................................................................... 13
   Fr17 – Robustness ........................................................................................................ 13
   Fr18 – Efficiency .......................................................................................................... 14
   Fr21 – Scalability ......................................................................................................... 14
   Developer support ....................................................................................................... 14
   Fr10 – Development environment ............................................................................. 14
   Fr11 – Documentation ................................................................................................. 15
   Fr12 – Server-side pluggability .................................................................................. 15
   Fr13 – Client-side pluggability ................................................................................... 15
   Fr15 – App components ............................................................................................... 15
3. Framework components ............................................................................................... 17
   Tellu SmartPlatform .................................................................................................... 17
   SmartPlatform client libraries .................................................................................... 19
   universAAL ............................................................................................................... 19
   Context awareness middleware .................................................................................. 20
   Mobile application strategy ......................................................................................... 20
   Android ....................................................................................................................... 21
   Stationary system ....................................................................................................... 22
4. Framework architecture ................................................................................................. 24
   First iteration ............................................................................................................... 24
   Design process ............................................................................................................ 24
   Prototype 1 architecture ............................................................................................. 24
   Second iteration ......................................................................................................... 25
Design process....................................................................................................................... 25
Prototype 2 architecture........................................................................................................ 26
5. Conclusions......................................................................................................................... 29
Executive summary
This deliverable documents our work with defining and selecting the technology infrastructure and development tools needed for the POSEIDON system. It is associated to Task 5.1 - Establishment of a development framework. This deliverable will inform and drive all further developments within WP5, most notably tasks 5.2 - 5.6 that develop the single components. The task is also connected to the technology development of work packages 3 and 4, most importantly Task 3.1 on middleware for context awareness. Context awareness middleware must form a key component of the framework.

This document first introduces its subject, giving an overview of what we mean by POSEIDON Development Framework. We have thoroughly analysed and discussed the framework requirements first listed in deliverable D2.1. These are the requirements which guide the framework design process.

A number of potential framework components have been identified, and the most important ones are described in chapter 3. At the time of the initial version of this deliverable, a first framework for the first POSEIDON prototype had been defined. This early framework was based on the SmartTracker service as the main component and central hub, with applications connecting to the APIs of this service over the internet. It had become clear that the initial plan from the project DoW – to have a framework finalised in the first six months of the project – wasn’t feasible. Framework design needs to be an ongoing process iterating with the prototype development. We continue developing the framework, testing and evaluating components and finding the borders between the framework and the applications built on it. For the second prototype the framework was extended with middleware for the stationary and mobile systems, primarily to support context awareness directly in these systems. An important task in the second iteration has been to learn about the universAAL framework and experiment with its usage in POSEIDON.

Work on the framework task has so far provided the support needed for the first demonstrator prototype, and the second prototype for the first pilot. However, there is still much to be done in this task, and the work continues in the next iterations, with the goal of having a well-defined framework at the end of the project.
1. Introduction

The POSEIDON development framework is the technology infrastructure and development tools which form the necessary basis for developing and running the POSEIDON system. This system is to consist of services and applications to support people with Down’s syndrome and, to some extent, also those who interact with them on a daily basis.

The framework has dual roles. Firstly, it will enable and form the basis of the implementation of prototypes in the project. But it is also to be generally usable outside the project prototypes. It is a goal in the project to provide a technological infrastructure to foster the development of services to support inclusion of people with Down’s syndrome and others in closely similar situations. We want to identify a set of components, services and tools which together provide the needed basis for building assistive applications and services, and document and make available this framework to others. The hope is to attract interest and help others develop applications for this sector of the society. If others build on the same framework, it also enables better compatibility and synergies between applications. This means that we need to think bigger than just the prototype requirements when selecting and designing the framework. As a framework, it needs to be generic. And those parts which will be needed by developers of new services and applications have to be made freely available (Open Source if source code is needed) as well as be properly documented. The prototypes we create in the project, in addition to providing value from their own functionality, will serve as tests and illustrations for framework use. Figure 1 gives a simple illustration of the role of the framework in the project, being the basis for the prototypes but also extending beyond the domain of the POSEIDON system.

![Figure 1: POSEIDON and its framework](image)

Framework content

The framework will have several parts and levels. On the conceptual level, it includes methodology, system architecture and API and protocol specifications. On the implementation level it includes hardware, server-side sub-systems and software components for client applications. And it includes the developer tools and documentation needed to create applications for the system.

We will not be creating this framework from scratch, implementing all its software components in the project. The main pillars have to be sought in existing technology and frameworks, both because of the scope and size of our task, because it would be a waste of resources to create something new.
where technology meeting the requirements already is available, and because we need mature, tried and tested technology for the basis of our solution.

The types of client hardware devices to be used in the POSEIDON prototypes are given from the onset, described in the Description of Work. The virtual reality and interactive table components will be connected to the system through PCs. Then there will be apps for mobile systems, originally with tablets as the primary form factor. From the DoW, we have the aim of having apps on tablets of at least three different brands, and on supporting Android and iOS (and not rule out additional mobile platforms in the future). We want to make an app framework available for the two selected platforms, with common base functionality and the connection to the communication infrastructure of the POSEIDON framework. In the requirements phase, we have selected Android as the primary focus, and iOS will follow later. So the app framework must include a multi-platform strategy.

Of middleware and servers, what is needed to support the wanted functionality is a more open question. Tellu’s SmartPlatform (or SmartTracker service) is mentioned in the DoW. We also look at the universAAL framework. What is certain is that we need middleware for context awareness.

A main task is to consider existing technologies and frameworks for inclusion in the POSEIDON framework, based on our requirements. We will strongly prefer software components which are free and which follows formal or de-facto standards. Where necessary we must compile technologies together, or extend them with our own additions.

First version of deliverable
This deliverable was originally scheduled for month six of the project. The ambition in the project planning was to have a complete framework defined early on, and then use that for developing the three main prototype iterations. It soon became apparent that this was not feasible. Establishing a complete framework is a much larger task, and it needs to iterate with prototype iteration. Of the two main tasks completed for the original deliverable, the first was to specify and analyse framework requirements. This is presented in chapter 2. The second task was to get a first framework in place to support the first prototype. The components were those outlined in the DoW, and they were described in the deliverable. We also gave some plans for the framework development after the first prototype, such as investigating the use of universAAL.

Due to the limited time, the framework presented was a minimum needed to be able to rapidly build the first prototype, and not something which met all the requirements. In this iterative process of selecting framework components, building prototypes and assessing the components it also takes time to find the right border between framework and applications. So it was clear that the framework task would have to run through most of the project rather than be completed in the first six months, and that we would need to revise this deliverable to document the later results.

Second version of deliverable
The second version of this deliverable has been updated to reflect the state of the framework as it is to be used for the second prototype iteration and first pilot. The original chapter 4 (Framework roadmap) has been split into two chapters, chapter 3 describing various framework components and chapter 4 describing the framework architecture as a whole, for the two prototype iterations. The executive summary, introduction and conclusion have also been updated to reflect the changes. It is important to point out that the framework is still very much a work in progress at this point in the project, and that there will be further versions of this deliverable.
## 2. Requirements

A listing of technical requirements for POSEIDON and the framework was produced, and listed in the requirements report, D 2.1. Here we repeat the list of framework requirements, and analyse what each of them means for the framework selection and design. For this discussion, we have grouped the framework requirements into some categories. See the category column added to the table – the rest of this chapter is organised according to these categories.

<table>
<thead>
<tr>
<th>Label</th>
<th>Requirement</th>
<th>Category</th>
<th>Prototype</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fr1</td>
<td>Support the development of the POSEIDON prototypes according to the POSEIDON requirements and constraints.</td>
<td>Overall</td>
<td>0– Seen as mandatory at all times</td>
<td></td>
</tr>
<tr>
<td>Fr2</td>
<td>Enable connecting client devices with server-side services, as well as server-side services to each other, to form a POSEIDON system.</td>
<td>Interconnection</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Fr3</td>
<td>Support various client devices. The required set in the project is tablets, interactive table and virtual reality set, but the framework should be open for any type of device running a major operating system or connected to a computer running a major operating system.</td>
<td>Client</td>
<td>3– Seen as mandatory at all times</td>
<td></td>
</tr>
<tr>
<td>Fr4</td>
<td>Support mobile use – mobile devices, and mobile networks such as 3G.</td>
<td>Interconnection</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Fr5</td>
<td>When live, support the safety of the end users.</td>
<td>Functionality</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Fr6</td>
<td>When live, support the privacy of the end-users. POSEIDON to provide optional user privacy settings to enable customization. Default settings will be provided.</td>
<td>Functionality</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Fr7</td>
<td>Support context-awareness.</td>
<td>Functionality</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Fr8</td>
<td>Support for optional interface customization to suit an end-user’s needs. Several sets of default settings will likely be provided.</td>
<td>Functionality</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Fr9</td>
<td>Storage of user data at the server-side with appropriate backup to safeguard. Optional user settings to customize data storage requirements with a default setting provided.</td>
<td>Functionality</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Fr10</td>
<td>Provide a mature, stable and well-integrated software development environment for writing, testing and deploying POSEIDON software components.</td>
<td>Developer</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Fr11</td>
<td>Documentation must be provided, to enable project participants and third parties to develop POSEIDON components.</td>
<td>Developer</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Fr12</td>
<td>Allow new server-side services to be developed and plugged into a POSEIDON system by third party developers, giving a clean, documented and standards-compliant API to do so without knowing the inner workings of the framework or the other components in the system.</td>
<td>Developer</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Fr13</td>
<td>Allow new client devices and applications to be developed and plugged into a POSEIDON system by third party developers, giving a clean, documented and standards-compliant API to do so without knowing the inner workings of the framework or the other components in the system.</td>
<td>Developer</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Label</td>
<td>Requirement</td>
<td>Category</td>
<td>Prototype</td>
<td>Priority</td>
</tr>
<tr>
<td>-------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td>workings of the framework or the other components in the system.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fr14</td>
<td>For tablet and smartphone client applications (“apps”), the framework must be</td>
<td>Client</td>
<td>iOS at 3</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>able to support all platforms with a significant (15 %) market share. This</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>means Android and iOS at the current time, but the framework should not</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>exclude future mobile platforms.</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Fr15</td>
<td>Software components for connecting an app to a POSEIDON system will be made</td>
<td>Developer</td>
<td>iOS at 3</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>available for Android and iOS, to support POSEIDON app development on these</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>platforms.</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Fr16</td>
<td>Enable distribution of apps to tablet and smartphone end users, in a way</td>
<td>Client</td>
<td>iOS at 3</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>that makes it possible for non-technical users to install and update the</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>apps with little effort.</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Fr17</td>
<td>When live, framework components should have robustness and fault-tolerance</td>
<td>Non-functional</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>comparable to non-vital commercial systems.</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Fr18</td>
<td>Enable efficient implementations of client applications and communication,</td>
<td>Non-functional</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>so that end users experience performance comparable to other applications</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>running on the same hardware.</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Fr19</td>
<td>Enable near real-time communication between components in the POSEIDON</td>
<td>Interconnection</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>system.</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Fr20</td>
<td>Enable asynchronous communication, where server-side services can initiate</td>
<td>Interconnection</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>communication with a client application.</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Fr21</td>
<td>When live, enable the POSEIDON system to scale to large numbers of connected</td>
<td>Non-functional</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>end users without significant performance loss.</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Fr22</td>
<td>Extensible, allowing integration of new functionality not yet foreseen</td>
<td>Interconnection</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>without breaking existing functionality.</td>
<td></td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

**Overall**

A category for requirements too wide to fit in the other categories.

**Fr1 – Support POSEIDON**

Summary: Support the development of the POSEIDON prototypes according to the POSEIDON requirements and constraints.

Discussion: While the framework must be generic enough to support services and hardware beyond what we consider in the project, the most important requirement is that it can support the prototypes we want to build in the project. It must therefore support the prototype-specific functional and non-functional requirements listed in deliverable D2.1. Many of the rest of the framework requirements are also guided in part by Fr1.

See also: Fr22, Fr5-9
Interconnection
A system such as POSEIDON can consist of various server-side services and client applications/devices, and these must be interconnected to form a cohesive system. The requirements in this category specify requirements for interconnection and integration, including protocol requirements.

Fr2 – Interconnection
Summary: Enable connecting client devices with server-side services, as well as server-side services to each other, to form a POSEIDON system.

Discussion: Facilitating the interconnection of sub-systems to form a cohesive system is a core task for the framework. We will discuss the interconnection strategy after looking at the other, more specific requirements in this category.

See also: Fr12-13

Fr4 – Mobile use
Summary: Support mobile use – mobile devices, and mobile networks such as 3G.

Discussion: Mobile use needs both a mobile device and a mobile network connection. Support for the first is more specifically handled in the client requirements Fr3 and Fr14. Considering network support, internet over cellular networks (3G/4G) will be part of the infrastructure the system needs for mobile use. All relevant mobile device platforms support such network connections. A mobile internet connection may be slow or lost in some cases. The framework should support efficient communication (Fr18), without unneeded overhead. It should also support robustness with regards to network fallouts – in the case where software components dealing with network transactions are provided for mobile client development, these should include local caching to handle intermittent network unavailability.

See also: Fr14-15, Fr18

Fr19 – Real-time communication
Summary: Enable near real-time communication between components in the POSEIDON system.

Discussion: Near real-time communication between components means that when a state changes in one component, this change is communicated to all other components which are interested in such a change, within a few seconds. Not all communication will be in real-time, but the framework must include technology and protocols which support it. One necessary element of real-time communication in this type of system is asynchronous communication, so that a server-side service can push the state change to client applications right away. This is discussed in the next requirements (Fr20). Another element is speed: A message from one component must be processed and passed on quickly. This means there should never be more than a few intermediary points between two components connected to the system, and all of these must be able to process the message in real-time.

See also: Fr20, Fr18

Fr20 – Asynchronous communication
Summary: Enable asynchronous communication, where server-side services can initiate communication with a client application.

Discussion: The classic client-server pull model, where the client application requests data from services and gets a reply in return, is not sufficient for all POSEIDON needs. Firstly, we need to be to
push state changes from server-side services to client applications as they happen. High-frequency pulling of the server is a possible way to meet the near real-time requirement, but would put a big load on servers even when there is nothing to communicate. One potential technology is HTTP push, such as WebSockets\(^1\) which is part of the emerging HTML5 standard. HTTP push is available for all modern HTTP implementations, including web browsers for web applications. However, it requires the connection to be kept open.

For mobile applications, we want to be able to send a message to the app without requiring the app to use resources by running in the background and keeping a connection open; even without the app running at all. Both Android and iOS, the two primary mobile platforms for POSEIDON prototypes, have push services on the system level which can be used to start an app and deliver a message to it. For Android, the service is Google Cloud Messaging\(^2\). These are platform-specific communication channels. The framework can hide the differences behind a generic communication interface, with implementations plugged in as needed.

Another aspect of asynchrony is that it may not be possible to deliver a message right away. The message must then be stored by the framework, and delivered as soon as possible.

See also: Fr19

Fr22 – Extensibility
Summary: Extensible, allowing integration of new functionality not yet foreseen without breaking existing functionality.

Discussion: This is a key requirement to make the framework something more than a part of a POSEIDON prototype. It means that the framework must be generic, not built to handle just a specific functionality set. It should be pluggable, allowing new components to be plugged in without affecting unrelated components in the system. And the framework itself should be extendable, allowing modules to be added at the framework level.

See also: Fr12-13

Interconnection strategy
Having looked at the individual requirements for integration and interconnection, we can discuss a cohesive strategy to meet them all. This is basically a question of the communication infrastructure. The simplest model to build a system of client and server sub-systems is that of servers providing APIs, with clients and other servers communicating with a server through its API. But this means that each sub-system must be built for specific APIs. Components should have as few direct dependencies as possible. To provide a more fixed structure and framework functionality, we can use a central framework server. This will then act as a communication hub, and other components only need to communicate directly with this server. However, it’s still not very flexible. As it bundles the infrastructure into a single component it’s more difficult to modify and extend it. It’s also a potential bottleneck, as all communication needs to go through this point.

For a more modular and less centralized approach, a bus or message broker middleware is a natural choice. Other framework functionality, such as data storage, can then be separate modules to be plugged into the bus as needed. New components can be plugged into the bus, and process relevant

\(^1\) http://tools.ietf.org/html/rfc6455
\(^2\) http://developer.android.com/google/gcm/
messages from other modules. Our conclusion in this section is therefore to seek a bus architecture and a generic, standardized message format for the communication infrastructure.

Client
Here we look at some requirements on the framework’s support for client application platforms.

Fr3 – Devices
Summary: Support various client devices. The required set in the project is tablets, interactive table and virtual reality set, but the framework should be open for any type of device running a major operating system or connected to a computer running a major operating system.

Discussion: For any form of computer with an internet connection, running a major operating system, it should be possible to develop the necessary software to connect it to the communication infrastructure. Major operating systems today are Windows (7/8), Linux, OS X, Android, iOS and Windows Phone. Since programming APIs with network functionality are available for all these platforms, we see no big barrier to the fulfillment of this requirement. If a specific bus API is chosen for the framework, it’s also a question of whether this bus API has an implementation available for the platform. If not, connecting the platform to the framework requires either developing an endpoint implementation for the bus, or connecting to it through some standard protocol like HTTP. So it’s important that at least one of these options is possible for all mentioned platforms.

See also: Fr13, Fr14

Fr14 – App platforms
Summary: For tablet and smartphone client applications (“apps”), the framework must be able to support all platforms with a significant (15 %) market share. This means Android and iOS at the current time, but the framework should not exclude future mobile platforms.

Discussion: This requirement stresses that all major mobile platforms should be possible targets for apps connected to the framework, although this is a sub-set of the Fr3 requirement. There are generally two strategies for mobile app development today. One is to use the programming APIs available for each individual platform, to make so-called “native” apps (note that these are only native in a software sense, not necessarily in the traditional sense of being compiled for a specific hardware platform, as f.e.x. Android lets the apps run on a wide range of processor architectures). With this strategy, all the implementation must be done separately for each platform. The other strategy is to use web technology (HTML, JavaScript and CSS), so that at least a significant part of the implementation can be used on all platforms. A framework such as PhoneGap helps by automating the wrapping of the web code as an app for each platform. The web technology is mainly suited to implementing the user interface, light-weight logic and HTTP-based communication. For full access to device hardware and the rest of the system, or for better performance, a “native” back-end can be implemented separately for each platform in a hybrid approach.

See also: Fr3, Fr15

Fr16 – Distribution
Summary: Enable distribution of apps to tablet and smartphone end users, in a way that makes it possible for non-technical users to install and update the apps with little effort.

---

3 http://phonegap.com/
Discussion: This is an important aspect of client device support, as there is little value in developing apps for a device if the users aren’t able to get and install those apps themselves. Installation could be done by a carer, but in any case it needs to be user friendly or it won’t be successful. If the framework includes its own distribution system, and this is to be used, it must be available for all significant platforms (see Fr14), and it must be easy to use. Otherwise, all the mobile platforms have app stores, and these are a natural choice for distribution channel. These are user-friendly, mature and stable, and they tend to have good system integration, giving the developer installation statistics and crash reports.

See also: Fr14

Functionality support
The requirements in this category deal with aspects of the services and applications to be built which the framework should support. Since it is the services and applications built on the framework which are ultimately responsible for the functionality provided to the end users, these requirements do not rest solely on the framework, and it is a question of how the responsibility for meeting them is distributed. As a minimum, the framework can’t jeopardize their implementation. And functionality which is needed for most services and applications should ideally be implemented as part of the framework.

Fr5 – Safety
Summary: When live, support the safety of the end users.

Discussion: The safety of the end users is very important in our case, and the framework must play its part in supporting it. The system should be able to detect the level of safety of the user, so that unsafe situations can be avoided, or the user or carers can be notified if they do occur. This relates to context-awareness (Fr7). The notification of carers must be dependable, so this places requirements on the communication infrastructure. If one line of communication fails, there should be a back-up. And the general robustness and dependability of the system is important (Fr17).

See also: Fr7, Fr17

Fr6 – Privacy
Summary: When live, support the privacy of the end-users. POSEIDON to provide optional user privacy settings to enable customization. Default settings will be provided.

Discussion: For the framework, this requirement primarily means that the communication infrastructure and any data storage done by the framework must be secure, such as through encryption. The framework should also avoid storing sensitive information, such as information to easily and uniquely identify the user or medical information. And what data to be stored should be customizable.

See also: Fr9

Fr7 – Context-awareness
Summary: Support context-awareness.

Discussion: Context-awareness is one of the pillars of the POSEIDON project. Key user context aspects, such as location, time and what the user is doing, should be generally available to system components. A new component developed for the system should be able to rely on getting context information.
This can be significant for APIs and protocols in the communication protocols. We may also want the deduction of context information and storage of such information to be included in the framework.

Fr8 – Interface customization
Summary: Support for optional interface customization to suit an end-user's needs. Several sets of default settings will likely be provided.

Discussion: The user may have user interface preferences, such as font size, high-contrast and input and output modes. Rather than needing to configure this separately for each application in the system, the framework should store a general set of interface preferences. Various components can then access these and make use of them.

See also: Fr9

Fr9 – Data storage
Summary: Storage of user data at the server-side with appropriate backup to safeguard. Optional user settings to customize data storage requirements with a default setting provided.

Discussion: We have seen some need for data storage at the framework level, such as for privacy settings, context and interface preferences. Other information that should be generally available within the system may also be discovered during the project or after. So the framework needs to include data storage, and this must be done safely and securely. This could be implemented with a database framework component.

See also: Fr6, Fr8

Non-functional properties
These are non-functional requirements for the framework itself, such as for the communication infrastructure.

Fr17 – Robustness
Summary: When live, framework components should have robustness and fault-tolerance comparable to non-vital commercial systems.

Discussion: It is essential for user acceptance that the system is dependable. This means that both the framework and the applications and services must be available and working very close to all the time. As some applications will be running on mobile systems, important resources such as network and GPS may be unavailable from time to time. This is not an error in the sense this requirement addresses, being outside our control, but rather a case we must plan and design for. Mobile applications must be designed to fail gracefully, taking care to work as well as they can with the resources available and not crash or give the user a technical error message. This aspect only relates to the framework as far as app components delivered as part of the framework are concerned (Fr15). The main concern for the framework, as it is the basis for the system, is that all framework components must be robust and free of serious bugs. They should be able to run with an uptime of at least 99%. This means that framework components should be mature, well-tested technology. As the services and applications to be developed will be prototypes, and they will depend on the framework, it is important that the framework is never the weak link. In addition to contributing to dependability for the end user, the framework should be dependable for developers, so that when a new application is failing during development and testing, the developer can be fairly certain the fault lies within his own code.
Fr18 – Efficiency
Summary: Enable efficient implementations of client applications and communication, so that end users experience performance comparable to other applications running on the same hardware.

Discussion: Firstly, efficiency is needed in the communication infrastructure, including any nodes the data must pass through before reaching its destination. This is needed for the responsiveness in the system, and to meet the real-time requirement. It means that communication protocols and processing should impose minimum overhead. Secondly, framework components such as database and mobile app components must be efficient to achieve responsiveness. And in addition to itself being efficient, the framework must support the development of efficient applications. Efficiency is especially important for mobile applications, as the hardware is relatively weak, and battery use is also an issue. This means we don’t want to enforce a heavy, high-level framework on mobile platforms.
See also: Fr19

Fr21 – Scalability
Summary: When live, enable the POSEIDON system to scale to large numbers of connected end users without significant performance loss.

Discussion: This requirement doesn’t mean that a single server needs to be able to support a huge amount of simultaneously connected users, or that the initially deployed system must be able to support whatever amount of users we can hope for in the future. Server-side hardware and infrastructure will always need to be scaled to fit the user base – the requirement means that such scaling must be possible. The framework must have the potential to support any size of user base we can imagine for it, through enabling scaling with multiple server instances and load-balancing to maintain good performance for all users. However, our efficiency requirement (Fr18) means that the system should support a reasonable number of simultaneously connected users seen in relation to the deployed hardware.
See also: Fr18

Developer support
Finally we have requirements dealing with how the framework should support developers creating new applications and services for the system.

Fr10 – Development environment
Summary: Provide a mature, stable and well-integrated software development environment for writing, testing and deploying POSEIDON software components.

Discussion: Developers need well-working tools to be able to produce software components for the system. This is an important part of the framework, though it is for the most part not something we should produce in the project. We need to ensure that the necessary tools exist for whatever platforms and APIs we include in the framework. The tools should be mature and relatively bug-free – if the developers need to struggle to use the necessary tools, this will affect the productivity in the project and we can’t expect the framework to be used outside the project. It is also a big plus for tools to have a large user base, so that if there is a problem, a solution or someone to ask can be found online. In addition to writing and compiling the software, the development environment must support testing and deployment.

Different platforms require different tools. For instance, Android has an extensive package of Development Tools freely available for the different desktop operating systems, which also integrates
well with the major Software Development Kits. Eclipse\textsuperscript{4} is a highly configurable and extendable software development platform with a lot of available components, including integration for the Android tools. It is a natural basis for a development environment. New Eclipse components can also be developed as part of the POSEIDON framework if necessary.

See also: Fr15

**Fr11 – Documentation**

Summary: Documentation must be provided, to enable project participants and third parties to develop POSEIDON components.

Discussion: Whether a framework component is reusing existing technology or produced in the project, the developer documentation needed to make use of it must be available. For instance, a communication API through which applications can integrate with the POSEIDON system must be well documented.

---

**Fr12 – Server-side pluggability**

Summary: Allow new server-side services to be developed and plugged into a POSEIDON system by third party developers, giving a clean, documented and standards-compliant API to do so without knowing the inner workings of the framework or the other components in the system.

Discussion: This and the next requirement specify how the framework and resulting system will be extendable. It is not enough that it can be extended; it must be feasible to do so for third-party developers without prior knowledge of the system, and without needing to learn about the inner workings of the framework. To easily be able to plug in a new component, the developer must be given a well-defined interface for integration, and be insulated from the rest of the system. It is very challenging to make services pluggable to this extent – to be useful a service must be used by other components, but at the same time we don’t want anything to depend on the service specifics directly. This is where a bus architecture will work well. A service can subscribe to messages based on what type of data it can process, and send messages on the bus for any interested clients.

See also: Fr13, Fr22

---

**Fr13 – Client-side pluggability**

Summary: Allow new client devices and applications to be developed and plugged into a POSEIDON system by third party developers, giving a clean, documented and standards-compliant API to do so without knowing the inner workings of the framework or the other components in the system.

Discussion: This is the client-side version of Fr12, and the same comments apply, except that pluggability is more straight-forward than for server-side services. As long as there is a well-defined API, and this is generic enough to support an open-ended set of services for the domain, this pluggability is achieved.

See also: Fr12, Fr15, Fr22

---

**Fr15 – App components**

Summary: Software components for connecting an app to a POSEIDON system will be made available for Android and iOS, to support POSEIDON app development on these platforms.

\textsuperscript{4} http://www.eclipse.org/
Discussion: In addition to making sure that all relevant platforms can be connected to the framework (Fr3, Fr14), we want to produce framework software components for Android as a primary app platform and for iOS as a secondary app platform. Framework software components for app platforms will take care of the connection to the communication infrastructure of the framework, and handle tasks all such apps will need to include, such as framework session/authentication and data retrieval and storage. This will provide a basis for developing new apps, giving the app-specific code a well-defined API to the provided framework implementation. A platform strategy must be selected (see Fr14). We can choose to either provide separate, native components for Android and iOS, or provide web technology components usable on both platforms (or a combination of both). If for native Android, the components will be implemented either as a jar-file or as an Android library project source code, and this will be freely available for third party developers. If using web technology, the framework can also include user interface components, so that apps developed by different entities as well as for different platforms can share a common look and feel.

See also: Fr13, Fr14
3. Framework components

This chapter describes framework components used in the POSEIDON prototype solutions. These are considered for inclusion in the final POSEIDON development framework. The next chapter describes how these components are combined for the framework of the prototype iterations.

Tellu SmartPlatform

Tellu SmartPlatform is a generic and highly configurable platform for sensor-based services. Its core is the storage and processing of data from sensor devices. Sensor data can in the broad sense be any type of data received as observations of limited size. Figure 2 gives an overview of the platform. Refer to the figure for the following presentation. Note that in the DoW and early project work, only the term SmartTracker is used. Tellu SmartPlatform has later been adopted as the name for the platform, while the service provided by this platform and used in POSEIDON is still called SmartTracker. Both terms will be used in the project, and the distinction is not usually important.

![Tellu SmartPlatform architecture overview](image)

On the left side of the figure is the input to the platform – the connected sensor devices. The Sensor Platform connects the heterogeneous sensor devices with the core. It consists of edges which implement the different protocols of the different devices, translating between the device-specific protocols and SmartPlatform’s internal format. In addition to receiving data from the devices and passing it on to the core, an edge typically support commands to the devices, such as for configuration, so there is communication both ways. In the commercial applications of SmartPlatform, most connected devices are small purpose-built sensor devices such as GPS, communicating using mobile network connection and SMS. Edges have been developed for many such devices, as well as for some sensor gateways. For the cases where we have control of the communicating device, such as for SmartPlatform-specific mobile apps, there is also an edge more directly based on SmartPlatform’s internal format. Otherwise, a device-specific edge needs to be developed to support a new type of
device or protocol. As long as the protocol is known and the device has an internet connection, we can connect it to the SmartPlatform.

Moving on to the core of the system, SmartPlatform has its own data model with some generic concepts. The most important concept is that of the asset, which is a tracked entity, i.e. the thing we care about. In our case it is the primary user, but it could also be a car or a house. A sensor device is usually associated with a specific asset, and all data from this device will then pertain to that asset. Core data entities such as assets and the incoming sensor observations are processed by a rule engine. This important component of the platform uses Drools$^5$ by JBoss, an open-source production rule system with an inference engine at its core. The rule engine is a place to implement server-side logic. The action of a triggered rule can be one of the external actions of the platform, such as sending an SMS or email to notify someone of an abnormal condition, or it can be to update the internal state of an asset. More advanced logic such as machine learning and artificial intelligence can be achieved by designing larger rule systems and state sets.

A framework for developing SmartPlatform rules is a potential task within the POSEIDON project. Rule code is inserted into the system through the administration web interface, but a development platform is wanted to write and test the rule code so that it is verified to work before it is deployed. The rules operate on facts, which are Java objects based on the data model, so a Java library is needed to write and test rules. The rule development framework needs to include the fact classes, the Drools implementation used in SmartPlatform, and utility classes for simulation. If SmartPlatform’s rule engine is going to be used as a reasoning component of the POSEIDON system, this rule development framework will be a deliverable to be made available to project partners, and later third party developers in general if relevant, so that any developer will be able to write and test rule code.

A core concept for many of SmartPlatform’s use cases it that of alarm. In traditional use, the rule engine is mainly used to detect abnormal conditions or other conditions that needs human attention. A result of rule triggering is then to raise an alarm with a specified level. Alarms are available through SmartPlatform’s web application, or they are passed on to alarm centres through the platform’s APIs. Messaging actions such as SMS and email are used as additional notification channels.

All data is of course stored in databases. This is mainly for the current state, but long-term historical storage is also supported by the platform. The data objects of the SmartPlatform data model are available through a rich REST API. Such an API gives access to the data to external systems, for reading it, and also for adding, updating and deleting where it makes sense to do so. This allows connected systems to for instance query asset state. A push-based API for subscribing to data updates is also in development. Data access is of course based on authentication and permissions. SmartPlatform has a role hierarchy for user entities and a detailed permission system to fine-tune what data such a user entity can access. Communication can be secured with HTTPS. The data model, data storage and access policies are described in more detail in Deliverable D5.4 – Databases for integration of services.

SmartPlatform has a web interface for administration and access to the data. This is highly configurable. The available functionality depends on the permissions of the logged in user, and it can be white labeled to suit a service provider. The web component also adds a layer on top of the rule code, to allow for user-friendly configuration of rule logic. It involves parameterized Drools template code, with placeholders for values to be filled in when instantiating a set of Drools rules, and a form-style web user interface for instantiating such rules.

$^5$ http://drools.jboss.org/
SmartPlatform is in active development, and will continue to be for the foreseeable future. However, it is currently used commercially in a variety of domains, where it must meet strict demands on uptime, security and error-free operation, so it has a high degree of maturity. It is usually offered as Software-as-a-Service, with the main commercial instance hosted by a professional hosting company providing 24-7 monitoring and support. Tellu offers this service to companies which act as service providers in their respective markets, and it is these partner companies which deal with end users. Tellu has also delivered server instances as disc images for virtualization. In relation to our framework requirements, it is important to note that SmartPlatform is closed source, with the exception of the open-source components it uses.

**SmartPlatform client libraries**

Client code components for connecting to Tellu SmartPlatform have been developed as part of the project work in the prototype 2 iteration. These handle connecting to the REST API and the edge for submitting raw data. These components will be made freely available, and can be included in client applications, middleware and server applications, enabling these to connect to the SmartTracker service.

Here is a list of the artifacts:

- **findit-rest-lib**: A Java library (jar file) for the SmartPlatform REST API, to facilitate data access.
- **findit-rest-android**: An Android-specific extension of the rest-lib, adding asynchronous transaction handling.
- **findit-edge-lib**: A Java library (jar file) for the SmartPlatform edge, to facilitate posting events to SmartPlatform.
- **findit-edge-android**: An Android-specific extension of the edge-lib, adding asynchronous transaction handling.
- **findit-lib-demo**: Source code for an Android application which demonstrates the use of both the rest and edge libraries.

Documentation is also provided for the libraries.

**universAAL**

universAAL\(^6\) is an open platform for Ambient Assisted Living technologies and services that was created based on the input gathered in a number of previous projects and aims to consolidate aspects into a sustainable, semantic, interoperable platform that is open and developed by a growing community. It is a natural candidate for the POSEIDON framework, as it is an extensive framework for AAL solutions which consolidates numerous previous platform projects into a shared, generalized architecture and middleware.

universAAL allows for multiple communication methods allowing for interoperability between different instances of universAAL in different locations, and external clients. These include the following:

- **Web services**: universAAL enables the ability of external clients to invoke services within AAL spaces SOAP web services. Service requests sent by the client are formatted as Turtle Strings in RDF. Each request is accepted and translated into universAAL service requests by the server. These requests are then sent and handled over the Service Bus before service responses are returned to clients using SOAP responses.

---

\(^6\) [http://universaal.org/](http://universaal.org/)
• **AAL Space Gateway Communication:** This type of communication allows different geographically positioned AAL spaces to communicate. Using this communication, services and contexts can easily be shared among different AAL spaces. Each AAL space is connected through an AAL space gateway.

• **Node-to-Node Peering:** Different universAAL nodes can connect to each other in the same network/AAL space. This is handled by service announce and discovery using the SLP protocol. This protocol is handled using jGroups and jSLP, and allows for AAL spaces to be created and announced, discovered, and connected to.

The process of learning about universAAL was started after the first prototype iteration. The project has signed an agreement to be associated with the ReAAL project, which is working on commercialisation of the universAAL framework. This agreement means we will try to use universAAL, and that the ReAAL project will provide us support in doing so. We will assess universAAL with respect to each of our framework requirements. If not used as the main framework, POSEIDON should still seek to be compatible or connectable with it in some way.

**Context awareness middleware**

Context awareness and the middleware to support it is the subject of Work Package 3. Context awareness support is an important goal for the POSEIDON framework. universAAL includes context awareness mechanisms, and its use is investigated in WP3. Finding the right context awareness middleware is still a work in progress. Refer to the WP3 deliverables for more on this subject:

• D3.1: Ontology and Language
• D3.2: Learning and reasoning module (this deliverable will have four iterations).

**Mobile application strategy**

As was discussed for framework requirement Fr14, there are multiple strategies for mobile app development today. Generally, either apps are programmed specifically for each platform, such as Android or iOS, or web technology is used for platform independence. We have done a brief review of the state-of-the-art, and identified a total of four possible multi-platform strategies:

• **Native applications:** This means targeting each platform individually. It is the only way to fully exploit the possibilities in each platform, but it means building a separate codebase for each platform. For a POSEIDON framework, we could supply framework components for multiple platforms, such as for connecting the app to the SmartPlatform. Application developers could then use the supplied components to develop their own POSEIDON-enabled applications.

• **Web applications:** A web application is an application which runs in a web browser rather than in a specific operating system. It is built on the web technologies of HTML, JavaScript and CSS. Built the right way, a web application should be able to run in the browsers of all modern platforms. Web applications are seen by some as the future of multi-platform applications, but they are still limited to mainly being a user interface layer, connected to online servers for data. So it is not a strategy for all types of applications.

• **Hybrid:** Web technology can be combined with native components for a hybrid strategy, where at least the user interface is platform-independent. The mobile platforms allow wrapping a web application as a traditional app which is installed locally. It can then be connected to back-end, native components, such as for background processes which are independent of the web UI. There are also frameworks to facility developing this type of
application, such as PhoneGap\(^7\). While this strategy may at first seem promising, we feel it is as much a combination of drawbacks as it is of strengths. Both web technology and frameworks built on it have their limitations compared to native applications, and you end up having to make platform-specific adaptations anyway.

- **Multi-platform frameworks**: To complete the list, we should also mention that there exist development frameworks which can compile a single source code to applications on all the major platforms. Two of the best-known are Xamarin\(^8\) and Qt\(^9\). So in this approach, the application is developed for a single platform, such as Xamarin, and the tools handle the mapping to the target platforms. We do not see this as a feasible approach for the POSEIDON framework, primarily because using these frameworks require costly licenses. You also lock yourself into a proprietary platform, so you need to be very sure it is the right long-term strategy, both in terms of features and in terms of economy.

It is clear that some of the things we want to do in POSEIDON, such as using the sensors of the devices to track context and provide navigation, needs a native approach. But we are also building a web application in the project, and we want this to be responsive and run on all devices. So we will investigate both the native and web strategies. Making platform-specific additions to web applications will also be considered. The choice of strategy will also depend on the other framework components, if they include components for any of the mobile platforms.

**Android**

Android has been selected as the primary mobile platform for POSEIDON. For the development work, the Android SDK\(^10\) is the base, containing the code libraries, build tools and much more. Eclipse\(^11\) used to be the development environment of choice, with plugins for the Android SDK. It is also what was used initially for POSEIDON Android development. However, a new development environment, called Android Studio\(^12\), has taken over as the officially supported environment. Support for the Eclipse toolset was dropped in POSEIDON’s second prototype iteration.

The Android system includes a data provider pattern. Data provider APIs for calendar and contact data are included in the Android APIs. The data providers act as central repositories for such data on the device, and any application can retrieve and update data through the standardised APIs. Various data synchronisation services exist for the data providers, taking care of synchronising local data with cloud services. Using this pattern, the application developer doesn’t need to know which cloud service is used. For instance, if we initially use Google Calendar to store events for POSEIDON, we can later switch to our own calendar service without any change being needed in applications interacting with calendar data through the data provider API. All that is needed is that we provide a synchronisation service which plugs into the Android system.

Tellu has a code base with an application framework and modules based on this framework, which is used as a base for Tellu’s POSEIDON application prototype. This in turn builds on Android’s own frameworks. Figure 3 shows the Android framework layers. The green and orange layers are what come with the Android system. It has a rich set of programming APIs, enabling easy use of all the phone’s resources and providing the components needed to build feature-rich applications. The blue

---

\(^7\) http://phonegap.com/
\(^8\) http://xamarin.com/
\(^9\) http://www.qt.io/
\(^10\) http://developer.android.com/sdk/index.html
\(^11\) http://www.eclipse.org/
\(^12\) http://developer.android.com/tools/studio/index.html
layer is Tellu’s framework with patterns and functionality for Tellu’s own apps, adding a high-level layer to help quickly create complex apps and reuse code modules. This layer is not meant to be included in POSEIDON’s framework, being Tellu’s own code base and not something most application developers would want to learn in order to be able to provide a POSEIDON application. However, we won’t rule out that elements from it may be useful to provide POSEIDON framework components on Android. Whether the Android APIs will be considered part of the POSEIDON framework depends on the mobile device strategy chosen.

Figure 3: Android and Tellu application framework layers.

**Stationary system**

The stationary system is the system continuously present at the user’s home. It is a PC attached to a large screen and a set of input devices that allow a natural interaction of the system, based on different forms of gesture input.

A conceptual sketch can be seen in Figure 4. The PC (2) is connected to the large screen (1) and handles all processing requirements. A depth camera, such as the Microsoft Kinect is placed next to the large screen (3) in order to acquire full-body gestures that may be required in some scenarios. An interactive table (4) is attached to or integrated into a living room table and captures hand gestures performed above it. Therefore, the stationary system provides the following inputs, outputs and data connections:

- Capture of full-body gestures for interaction
- Capture of hand gestures for interaction
- Output of graphically rich user interfaces
- Connection to POSEIDON platform and visualization based on acquired context
As shown in Figure 5 the different input systems are interfaced by the POSEIDON platform using their respective APIs. The platform is not comprised only of the PC, but also includes information acquired from the web, mobile systems, etc. Thus, the input information can be enriched with any additional context that is acquired. Finally, this is translated to commands to the Output APIs that control the content of the large screen. This could be 3D APIs such as X3D or Unity, but also include 2D contents that might be better suited to transmit some content.
4. Framework architecture

Here we present the evolving picture of the architecture of the POSEIDON solution and framework. The current version of the deliverable covers the first two prototype iterations. These are iterative attempts at an architecture, driven by the prototype needs.

First iteration

Design process

After the requirements gathering and analysis, the framework process leading to the initial version of this deliverable consisted of providing the necessary development framework for a first prototype, as well as planning ahead for the richer framework needed for the final POSEIDON solution. The Milestone First Integrated Prototype (MS8) was scheduled for month 9 of the project – the first of a total of four prototype iterations. This first tentative prototype comes very early in the project, and is an important starting point for further development. The idea is to get both a tablet application with some functionalities as well as a first version of a home system implemented so that we gain experience and have something to show the user groups and can start getting feedback. These two parts also need to be connected through a server side.

It is important to keep in mind that the experimental nature as well as the very short time frame for implementation means that the first prototype didn’t try to put into place a definitive POSEIDON development framework. The focus in this stage was to provide what was necessary to support the implementation of this first prototype, using the technology with which the implementing partners had prior experience. Then the framework process would continue after developing the first prototype, given our experience from this implementation.

Prototype 1 architecture

Prototype 1 consists of a mobile system and a stationary system for the primary end users to interact with. As these are demonstration systems and not for real use, they are not connected together. However, an architecture connecting these systems to Tellu’s SmartPlatform was designed. This architecture is shown in Figure 6.

Tellu’s SmartPlatform (SmarTracker service in the figure) is the main framework component in this architecture. It is the central hub in the solution, with mobile and client applications being able to connect to it to share data. It is a place to implement server-side storage and logic. As was described in the previous chapter, applications can post data to the sensor edge. This data is stored, and it is processed by the rule engine. Applications can access the aggregated data in the SmartPlatform data model through the REST API. So the SmartTracker service was provided for server-side data storage, for server-side logic and context awareness and as the hub in the system, connecting the mobile and stationary clients. The web interface of the service is used for configuration and administration.

A first prototype mobile application was developed for the Android platform. The primary features are tracking and navigation, in addition to being a test-bed for context awareness. While navigating, the user’s position, as well as the battery/power status of the device, is sent to SmartTracker for processing. The application was also connected to the SmartTracker REST API. For additional services in the application, such as routes for navigation and calendar for day planner, it can be connected to external services such as those offered by Google.
The conceptual architecture also shows how a web application can be added to the system. Based on SmartTracker’s API, such a web application can offer monitoring of the primary user by presenting the tracking and context data, and administration of the SmartTracker account, for instance editing preferences.

The stationary system is what is running on the home computer. Here prototype applications were developed for virtual reality navigation training and augmented reality guidance. The idea in the conceptual architecture is that this part can also connect to SmartTracker, to access preferences and context data, and possibly also to post context observations.

**Second iteration**

**Design process**

An early roadmap for the architecture and framework process was made in the first iteration, mainly stating the intention to add universAAL to the framework. Figure 7 shows a rough sketch of the prototype 1 and 2 systems. While the prototype 1 part roughly corresponds to the final prototype 1 conceptual architecture, the idea was that universAAL would take over the role as communication hub from SmartTracker. As SmartTracker is a service in its own right, not just middleware, it is not ideal as the communication framework in POSEIDON. It is better if it can be a pluggable component in the system, which could also be replaced or augmented by other services.
Figure 7: Early system architecture roadmap

The idea for prototype 2 and beyond, based on our requirements discussion, was to use some form of message broker so that new components could be connected without relying on SmartTracker’s APIs and without going through SmartTracker when this is not necessary. So rather than being directly connected to SmartTracker, we saw the components all connected to some form of bus. It was thought that universAAL would play this role. However, as we got to know universAAL and its components in the second iteration, it became clear that this wasn’t a role it was suited to. The primary purpose of universAAL is smart-home middleware. universAAL instances are deployed on home computers to act as gateways, for the interconnection of different sensors and applications. As such, it is mainly a framework for the stationary system. While it has mechanisms for connecting specific universAAL instances together, it is not suited to provide server-type services such as central storage of data.

The result has been to divide the framework for prototype 2 into two main components. Tellu’s SmartPlatform continues to be the cloud service part of the framework, being available online for all POSEIDON applications and users to connect to. The other main part is the middleware running locally, on the stationary and mobile systems.

Prototype 2 architecture

Figure 8 shows the resulting second iteration conceptual architecture. The box labeled Context middleware is the middleware running on the stationary and mobile systems, mainly for context awareness. This framework component is still in heavy development. universAAL is still being considered, and will be deployed on both the stationary system and Android device for the second prototype. However, the ontology and rules are currently running independently of universAAL.

Note that SmartTracker’s rule engine is no longer indicated in the conceptual architecture, as the context awareness logic is now placed in the locally deployed context middleware. Moving this from
server to client side is wanted to avoid the lag and possible fallout of connectivity associated with an internet connection. However, SmartTracker’s rule engine may still be used at some point, if there is a need for logic which aggregates data from multiple users. The locally deployed middleware instances may be connected to SmartTracker. They can send context observations there so that they are stored in the cloud, independently of the device. This allows sharing this context data between instances belonging to the same user, retrieving the data with the REST API, authenticated by a SmartTracker account set up for the user.

Figure 8: Prototype 2 conceptual architecture

Three types of applications can be connected to the framework in this model. Stationary applications are those running on a PC or laptop, at home or at another location the primary user regularly stays at. Such applications may be connected to the context middleware deployed on the same machine. As in the first prototype, virtual reality training and augmented reality guidance are the focus of development here, and the interactive table is used as a control device.

Mobile applications are those running on a phone or tablet, and meant to be used anywhere, especially when not at a location with a stationary system. Context middleware is deployed on the same device. A mobile application can subscribe to context changes, and the middleware will monitor context on the device and deliver updates to the application. The mobile application can also provide input to the context reasoner of the middleware, posting data regarding its own state and context. Mobile applications may also connect to SmartTracker over the internet. They can send observations directly to SmartTracker, for the online monitoring. And they can access the SmartTracker data through the REST API, mainly for the user preferences stored here.

The conceptual architecture also states that there needs to be a web application, primarily for secondary users to be able to do monitoring and personalisation from any device and location. This is an end user interface for SmartTracker functionality, presenting the tracking data and letting the user
interact with the preferences stored here. It is not yet clear if parts of this web application should be considered as a part of the framework or not. Tellu will make available our web application framework as open-source. This builds on some state-of-the-art open-source web application libraries and frameworks, adding an overall structure and modules for SmartTracker interaction. Using this, other developers will be able to create their own SmartTracker-enabled applications.

The applications need more services than we are currently supplying with POSEIDON framework components. These can be provided by external services – free cloud services such as those provided by Google. In the POSEIDON solution, the following services are especially important:

- **Calendar data storage**: Secondary users are to be able to plan activities for the primary user with the POSEIDON system, and the planned events need to be available throughout the system. This means we need cloud storage of the events. For the second prototype we are using Google Calendar to provide this, but the goal is to develop our own calendar storage.
- **Route planner**: The navigation aspect is another of the primary functionality set selected for POSEIDON. It requires both automated route planning and the ability to augment plans with personalised instructions. Google Directions will be used as the automated route planner for prototype 2. Personalisation is done in the stationary system. An automated route planner is outside the scope of POSEIDON, and will always be an external service.
- **Media storage**: Media such as photos and video is primarily needed for instructions and guidance to the primary user, configured and personalised by the secondary user. Here we are currently exploring options. It is not yet clear whether a specific service should be specified as part of the framework.

All types of applications can connect to external services, and use them to share data between applications. For the second prototype, calendar functionality is added to the web application, so that it can be used for planning. The mobile application prototype will also have calendar functionality, connecting to the same service. The route planner is used by the stationary system, where routes are personalised. Personalised routes are transferred to the mobile application for navigation. The mobile application can also request new routes directly from the route planner service, for navigation from any starting point, but without the personalisation.
5. Conclusions

In our framework process we have so far put together what was needed for developing the first and second prototypes. We have thoroughly analysed and discussed the framework requirements first listed in deliverable D2.1. These requirements continue to guide the framework design process, which is still very much on-going. We have identified and described a number of potential framework components. We defined a framework for the first prototype iteration, based on the SmartTracker service as the central hub. For the second prototype the framework was extended with middleware for the stationary and mobile systems, primarily to support context awareness directly in these systems.

Work on the framework task has so far provided the support needed for the first demonstrator prototype, and the second prototype for the first pilot. However, there is still much to be done in this task, and the work continues in the next iterations, with the goal of having a well-defined framework at the end of the project.