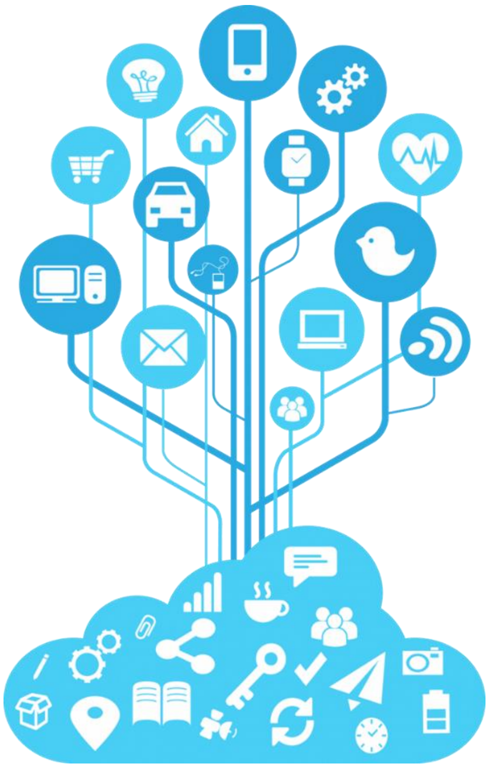
POSEiDON

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

New Report R4

Developers Guide



Call:	FP7-ICT-2013-10
Objective:	ICT-2013.5.3 ICT for smart and personalised inclusion
Delivery date:	February 2017

Author(s): The developers of POSEIDON
Lars Thomas Boye, Tellu
Juan Augusto, Middlesex Univ
Dean Kramer, Middlesex Univ
Unai Alegre, Middlesex Univ
Silvia Rus, Fraunhofer IDG
Andreas Braun, Fraunhofer IDG

Dissemination level:	Open
Number of pages:	42

Content

1. Introduction

This document is the printable and pocketable version of the POSEIDON Developer Guide.

What is this about?

We share our experiences and tools gathered during the POSEIDON project with anyone who would like to develop an App for persons with Down's syndrome.

At this point we strongly recommend to consult our case-study "Developing using the POSEIDON framework by example of a Healthy Eating App"[1]. This document illustrates how we used the POSEIDON framework in order to develop a healthy eating app.

Why is it interesting for me?

Here we refer to useful documents and provide a step-by-step guide on how to get started.

What does the Developer Platform offer to me?

We offer requirement gathering methodologies, information about the user group, ethical and privacy checklist, a basic app to start with, special libraries and APIs and best practices.

What does it cost to use it?

Using the POSEIDON framework is free. All software is open source.

Structure

We use the POSEIDON developer guide in order to convey to you lots of information on the POSEIDON framework and findings. Throughout the document lots of links will point to more detailed resources. Our case-study "Developing using the POSEIDON framework by example of a Healthy Eating App" accompanies this developer guide illustrating how the POSEIDON framework was used to create a Healthy Eating app.

In the following the content of the sections of the developer guide are shortly presented:

Understanding the POSEIDON framework. This is important in order to get to know the existing architecture of technology and methodologies how to execute a fine-grained requirement analysis.

Getting started tutorial. We take you step by step through the basic usage of the POSEIDON infrastructure. You will use the Starter App[2] as working example. As the main theme of this tutorial is the communication with the infrastructure, you will go through code examples from the Starter App which explain the basic connection to the POSEIDON infrastructure. Links to the different resources assure that you can start developing.

Extending your app with context awareness. This tutorial explains the process on how you can integrate generated code from the context awareness methodology into a basic app. This is important in case you want to develop an app where different automatically detected situations (contexts) trigger functionality in your app.

Info on user group. Here you can find a concise description on persons with Down's syndrome as computer users.

[1] http://www.poseidon-project.org/wp-content/uploads/R42-Healthy-Eating-App_v2_final.pdf
[2] http://www.poseidon-project.org/wp-content/uploads/StarterApp.zip

Ethical, privacy and security principles. Once you have developed your first app idea, please read these, and check if all of the principles are taken into account.

Design user interfaces for persons with Down's syndrome. In the first part you will find the general principles and how POSEIDON recommends to implement them. In the second part you will find developer guidelines with concrete examples what to use and what to avoid.

Lessons learned during our cooperation with the persons with Down's syndrome. We address the requirement gathering, the app design and the app testing phase.

2. Understanding the POSEIDON development Framework

The POSEIDON developer framework is a collection of methodologies, infrastructure, middleware, tools, specifications, etc. These are things which were used in the development of the POSEIDON prototype system and part of it. This is partly methodologies, open-source code and free tools of general use, and partly the POSEIDON infrastructure which can be used for other applications.

A goal of the POSEIDON project was to foster development of inclusive services for people with Down syndrome, and a commercial POSEIDON solution needs to be extensible with new services. The development framework is the project's way of addressing these goals. Here we give an overview of the framework for new developers, along with references to more detailed information. The summary is based on project deliverable D5.1, which gives the full description of the framework.

Methodologies

A system development methodology specifies structured ways of creating ICT systems. The framework describes the following methodologies:

- UC-SDP: User Centered Software Development Process. This software development process focuses on facilitating stakeholders to co-create intelligent environments.
- eFriend: A set of engineering principles to guide and encourage developers to secure that ethically sensitive aspects of technology are explicitly addressed in the development of a product, especially within the Ambient Assisted Living field.
- R4C-AS: A methodology for requirements elicitation in Context-Aware systems. It provides a structured approach to gather and define the requirements of such systems, including diagrammatic support, which can be then linked to design and correctness analysis.

The framework also includes tools to support the methodologies. See the Tutorial for context-aware systems for a practical introduction. Documentation references:

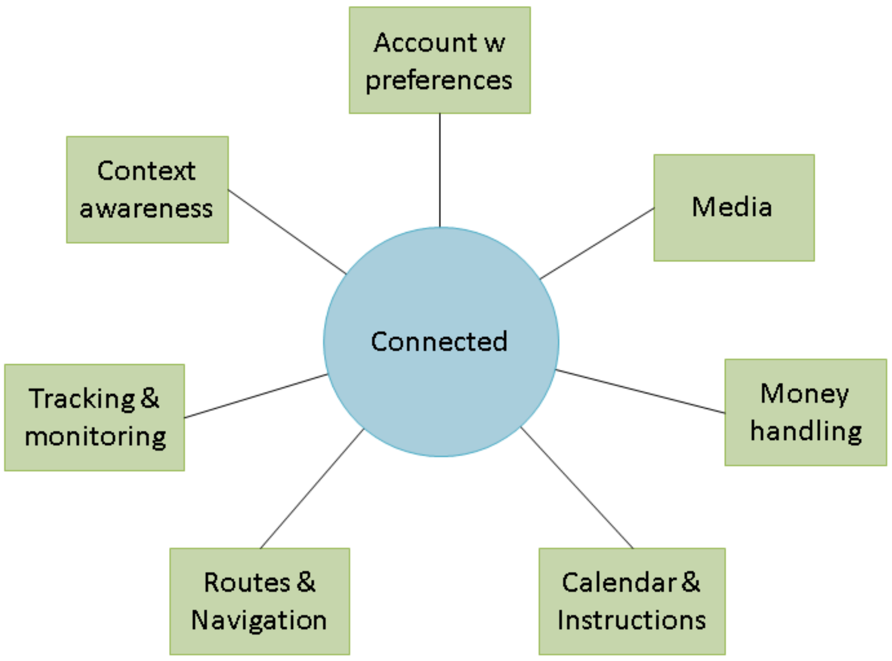
- D5.1 Development framework chapter 3.
- Ethics in POSEIDON: Pdf document available from web.

Architecture and platforms

The POSEIDON solution is a system of various end user applications and different platforms connected by an infrastructure. The framework architecture describes this infrastructure and how applications are integrated in this system. Based on the type of services we wanted POSEIDON applications to provide we have defined a set of services the underlying infrastructure needs to provide to applications. The figure below shows an abstract view of the services the infrastructure provides and integrates. Server-side services for holding data and providing shared access is an important part of this infrastructure.

Individual personalisation of services is especially important for people with Down syndrome, as there are big differences between individual capabilities and needs. Personalisation needs be possible in all forms of user interaction, and supporting this is an important aspect of the POSEIDON framework. This support is two-fold: providing a shared store of personal preferences for applications, and providing shared storage and standards for content.

Figure 1: A service view of the infrastructure



Another key part of the architecture is the supported platforms and devices for end-user applications. We make a distinction between stationary and mobile application platforms. Stationary platforms are for using applications at home or in other stationary locations and are based on laptop and desktop computers. These have large screens and various input devices, and are suited to primary user training and secondary user management tasks. Mobile platforms are smaller devices such as phones and tablets, suited to bringing with you wherever you go and to use outside. These are needed to provide notifications and guidance to the primary users. We specify a framework for web applications, responsive to run on all devices. For more specialised functionality, we provide additional support for stationary training applications developed in the cross-platform framework Unity, and mobile applications for Android devices. The architecture includes middleware and interaction devices for the supported platforms.

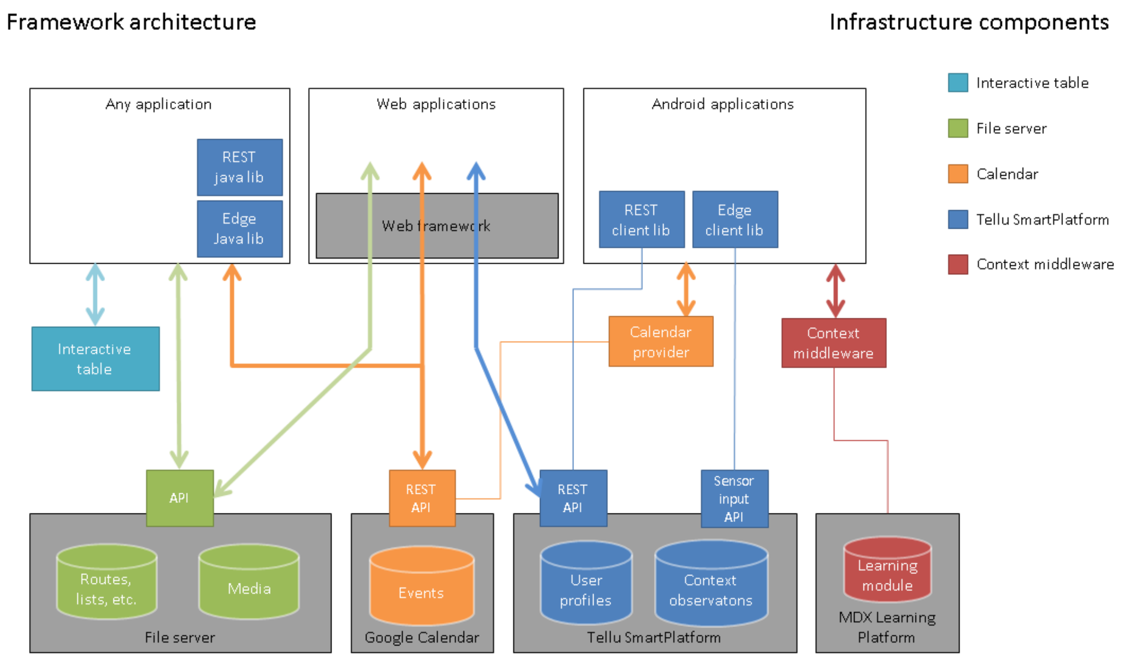
Documentation references:

- D5.1 Development framework chapter 4.
- D5.2 Prototypic systems of the POSEIDON concept: Describes the integrated prototype systems produced in the project. It describes the developments in each prototype iteration and the functionality in the two pilots, giving an overview of the system as a whole. It also includes technical documentation for components not part of the framework or documented elsewhere.

Technology infrastructure

The technology infrastructure is provided to implement the architecture. Going through the main components, we refer to the framework infrastructure overview below, and start with the server-side services along the bottom of the figure.

Figure 2: POSEIDON framework infrastructure



The file server provides cloud storage of data for POSEIDON applications. Secondary user applications are used to define personalised instructional content with multi-media, and this is stored here for the primary user applications to access. The framework includes data specifications for routes for navigations, shopping lists for money handling and video play lists for instructions, so that new applications are interoperable with the existing ones. A calendar system is used to connect instructions to points in time and give notifications. The infrastructure uses the Google Calendar service for cloud storage of calendar events, but specifies an extended data format including instructions and multimedia, only supported through POSEIDON-enabled user interfaces. The Tellu SmartPlatform service is where POSEIDON user accounts are stored, along with a user profile. Tracked data such as mobile device position is sent here to be available for monitoring.

Another key part of the infrastructure is the Context Reasoner Middleware. It provides context awareness to mobile applications. Applications connect to it, causing it to perform acquisition of and reasoning over context, notifying the interested applications of changes. The context awareness part of the framework also includes a tool for producing context rules and a server-side for collecting data from the middleware.

The interactive table is a part of the infrastructure for stationary applications. It is a prototype of a new interaction device, which combines the size of a multitouch table with 3D hand position recognition. It is intended to be unobtrusively built into tables, allowing the user to control applications via hand gesture on or over the table.

Documentation references:

- D5.1 Development framework chapter 5.
- D3.2 Reasoning and Learning module: Describes context ontology and definition language for the context awareness middleware.
- SmartPlatform POSEIDON developer documentation: Developer documentation & API, pdf available from web.

- Online SmartPlatform documentation: https://smarttracker.no/documentation
- File server API: Pdf available from web.
- Data specifications: Data specifications for calendar events, routes for navigation, shopping lists and video list. Pdfs available from web. API and data specifications are also collected in D5.4.
- Interactive table (CapTap) – API: Pdf available from web.

Developer tools and components

The framework includes software and code libraries provided to developers to enable them to develop POSEIDON applications. Here is an overview of components:

- Tool support for R4C-AS: A set of tools for keeping track of requirements and creating diagrams.
- Context modelling tool: Tool for creating new rules for the context reasoner of the infrastructure.
- Android SDK: Mobile application development is facilitated by the Android application framework and Android Studio development environment.
- Web framework: Web applications are based on JavaScript and the latest HTML/CSS standards. Our recommended framework includes the AngularJS JavaScript framework and Bootstrap CSS library.
- Code libraries: Libraries provided include API for connecting to the interactive table, Java and Android libraries for the two SmartPlatform APIs, and a Unity library for connecting to the SmartPlatform and file server.

The Mobile application tutorial of these guidelines shows how to develop a POSEIDON mobile application, making the connections to the infrastructure, including references for detailed documentation. General documentation references:

- D5.1 Development framework chapter 6.
- Android developer documentation: https://developer.android.com
- POSEIDON code components: http://www.poseidon-project.org/developers/code/

User interface design

The framework also provides specifications, information and graphical elements for user interfaces. Developer documentation includes guidelines for developing accessible user interfaces, and information about mobile and web application UI development. A user interface strategy is specified, considering user requirements. A colour palette and icons are provided, to create applications with a shared look and feel.

User interface guidelines are collected here, in the User interface section. Documentation references:

- D2.3 Report of Design of HW, Interfaces and Software: Chapters 2-5 give the basis for the user interface guidelines.

- D4.1 Interface strategy: Outlines the user interface strategy for the POSEIDON system, and is a resource for user interface developers. Documents the different iterations of UI work in the project.
- D4.2 Adaptive tablet interface: Presents our work on adaptivity in the user interfaces. It describes adaptivity and personalization in the pilots, and the user interface of the main POSEIDON mobile application.
- D4.3 Interactive table: Describes the interactive table – the novel input method developed in the project and used with some of the applications – including prototype implementations and communicating with computer applications.
- D4.4 Virtual Reality System: Discusses use of Virtual Reality in the POSEIDON solution, and the prototype implementation, which is mixed reality navigation training.
- D4.5 HCI user and developer manuals: Documentation on user interface design and implementation. User manuals for pilot 1 and pilot 2 applications are included in this deliverable, along with developer documentation for user interface implementation.
- POSEIDON icon repository: http://www.poseidon-project.org/product/symbols/

3. Getting started - Mobile application tutorial

This tutorial shows how to create an Android application connected to the POSEIDON infrastructure. POSEIDON provides an Android application project with source code, which we call the POSEIDON Starter App. This barebones example app connects to the infrastructure, and can serve as an example and/or a basis for your own app. To get the most out of this tutorial, you should have access to the Starter App code.

The tutorial goes through the basic usage of the infrastructure, often referring to the Starter App code. The APIs, data models and libraries used have their own documentation, and there are parts of the framework not covered by the tutorial. The final section of the tutorial, *Further reading*, provides references to all the relevant documentation.

Android SDK

POSEIDON Android applications can be developed much like other Android applications, using the official Android SDK. The Android platform has a solid application framework and development environment, used by a very large community of developers. It is well documented on the official developer website:

http://developer.android.com/

To develop a POSEIDON Android application, it is a big advantage to have Android development experience. If you have no prior experience, it is a good idea to start with some of the official tutorials on the developer website. User interfaces and interaction uses the normal Android APIs, and so is not a focus for our POSEIDON-specific tutorial. Our primary concern is connecting to the server-side infrastructure, and for this part of application development you will learn enough in this tutorial to make POSEIDON applications.

Development environment

For the development work, the Android SDK[3] is the base, containing the code libraries, build tools and much more. Android Studio[4] is the officially supported development environment, and comes with everything you need. The Starter App is provided as an Android Studio project. If you have not installed Android Studio already, doing so is the first step to start development.

POSEIDON Starter App

The POSEIDON Starter App is available from the developer code page of the POSEIDON web site:

http://www.poseidon-project.org/developers/code/

It is provided as a zip archive with all the needed files. Download and unpack this Android project. Open Android Studio. You'll want to import the project. If you currently have an open project, you can use the *File* menu, select *New* and *Import Project…* Otherwise, *Import project (Eclipse ADT, Gradle, etc.)* is one of the main choices from the starting screen. This brings up a file browser; select the Starter App folder. It will then ask to use the Gradle wrapper, which you should OK. It may also ask you to update the project or download the correct version of the Android SDK.

Looking at the project, the two main parts are *java* (the code) and *res* (the resource files). The Java code is a small collection of classes, mostly Activities, which are the user interface modules. The resource files are mainly layouts for the activities and graphical resources such as icons. There is also

[3] http://developer.android.com/sdk/index.html
[4] http://developer.android.com/tools/studio/index.html

a manifest file – AndroidManifest.xml. This provides a basic definition of the application as a whole. Important are the *uses-permission* elements, which name protected features the application needs, such as internet access. And each Activity needs to be listed under the *application* element.

Once everything is set up, you may want to build and run the application (press the green triangle "play" icon on the tool bar). If you have an Android device, connect it to the computer first, so that you may run the application there. Otherwise it will run in a virtual device.

The Starter App connects to the SmartPlatform and file server of the POSEIDON infrastructure, which requires a POSEIDON account. Running the app, it starts with a login screen. If the login is successful, you are presented with a main menu with some actions which test the infrastructure connections. It has no functionality other than showing some debug information from the actions. Its main value is as a basis for new applications and as an example for this tutorial.

Connecting to the server-side POSEIDON infrastructure, as the Starter App does, requires the following information:

Infrastructure	Information	Value
SmartPlatform File server	POSEIDON account user name and password	Currently created by the SmartPlatform administrator (Tellu).
SmartPlatform	Server address	POSEIDON pilot server: https://ri.smarttracker.no
File server	Server address	The pilot file server is on the same address as the SmartPlatform server: https://ri.smarttracker.no

User interface

As illustrated by the Starter App, a POSEIDON app will typically consist of user interfaces and a little bit of background logic. Communication with the infrastructure is an important part of the background logic, and is the theme of much of this tutorial. User interfaces are technically implemented as in other Android applications, using the mechanisms of the Android platform. The Starter App is a basic example. Note that this does not attempt to follow POSEIDON guidelines for usability, as it is mostly background logic and has minimal user interaction.

Activities and layouts

In Android, the Activity is the basic user interface component, meant to represent some activity where the user interacts with the application. The Starter App has an Activity class for each activity in the app, containing the user interaction code. Which user interface elements to show on screen and how they are laid out, is defined in an XML file. These layout XML files are located in res/layout in the project structure. To make a new function in an application, you can start by making an Activity class and accompanying layout file (copy an existing set as a starting point).

User interface elements

To get started with your own user interface, you can start with one of the layout files from the Starter App, and add, remove and move elements. Android Studio has a graphical tool for working with a layout, letting you drag elements around and see the result.

When you are ready to go beyond the basic functionality of the Starter App, look at the design pages of the Android developer website, if you are not already familiar with Android UI development. And read the user interface guidelines in this document to learn about how to make good, user-friendly interfaces for the primary user group.

The POSEIDON user interface guidelines recommend using the POSEIDON visual theme to create a family resemblance between applications. The Starter App includes definitions of the POSEIDON colours, in res/values/colors.xml. A POSEIDON icon set is available from the project web site, here:

http://www.poseidon-project.org/product/symbols/

SmartPlatform connection – account with preferences

The SmartPlatform server keeps the POSEIDON accounts, and is responsible for authenticating the user and storing preferences. If you wish to use the file server but not the SmartPlatform server directly, it is possible to log in with the POSEIDON account through the file server. However, it is recommended to connect to the SmartPlatform server, to have access to the user profile. The Starter App logs in to the SmartPlatform server and retrieves the preferences.

The Starter App project includes the SmartPlatform API Android library (findit-rest-android.jar). This is an Android/Java wrapper for the HTTP REST API. Data is exchanged in JSON format, and the library uses the org.json Java JSON representation, which comes with Android. In the Starter App, we manage the use of this API through the class StarterApp. This class extends the Android Application class, a class which represents the application as a whole, which is created when the application is created, and which is always accessible to Activities. To use this class for the application object, we must specify it in the manifest file (android:name under application). We have put the SmartPlatform interaction there so that it stays in memory regardless of Activities coming and going. In a larger application, you may consider putting it in an Android Service object.

Logging in

The LoginActivity is the launcher activity of the Starter App – the initial Activity started when launching the application. It brings up a login screen, for entering the POSEIDON user name and password. When the user clicks to log in, we call a login method on the StarterApp, which in turn handles the API transactions. On success, it launches the next Activity, with the main menu.

StarterApp.login shows how we typically start a session with the SmartPlatform server. We use the AsyncService class from the API library, which handles the transactions in an asynchronous way, with callback objects being called on success or failure. It is important not to block the main thread with a network transaction, as that would make the user interface unresponsive.

```
AsyncService apiService = new AsyncServiceImpl(API_URL);

AsyncService.AccountCallback callback = new AsyncService.AccountCallback() {
    public void onAccountOK(long customerId, String customer) {
        // Login OK
        // now we want to retrieve the asset object representing the primary user
        getAsset();
    }
    public void onAccountError(int code, String message, Exception e) {
        Log.e("StarterApp", message, e);
    }
};
apiService.initiate(USERNAME, PASSWORD, callback);
```

In the code above, we create the AsyncService object, and run the first transaction. This transaction sends the user name and password to the server, and if accepted it gets back an authentication token. This token must be included with all requests to the SmartPlatform server, as well as the file server, which uses the same account and authentication. The AsyncService object handles this for us. Once the initiate transaction is successful, the object can be used to do other transactions, and we must keep this object as long as we want to stay logged in. At this point we can retrieve a ServiceSpec object with various information about the session. We can get the authentication token, if we need it for the file server.

```
ServiceSpec ss = apiService. getServiceSpec();
String token = ss.getAuthenticationToken();
```

Retrieving the asset – primary user profile

Asset objects in SmartPlatform represent tracked entities – in our case primary users. Note that it is possible for an account to have access to any number of assets. In POSEIDON pilots we have had accounts with a single asset, for the primary/secondary user pair, as well as tertiary user accounts with access to a larger set of assets.

The Starter App is made for accounts with exactly one primary user, which means it expects there to be exactly one asset (if there are multiple, it just selects the first). It retrieves the list of assets directly after the initiate transaction succeeds. This is a GetTransaction, resulting in a JSONArray of JSON objects.

```
AsyncService.DataRetrieveCallback retrieveCallback = new
AsyncService.DataRetrieveCallback() {
    public void onRetrieveOK(JSONArray objects, GetTransaction transaction) {
        JSONObject jo = objects.getJSONObject(0);
        assetId = jo.getLong("id");
    }
    public void onRetrieveError(int code, String message, GetTransaction
transaction) {
        Log.e("StarterApp", message);
    }
};

GetTransaction trans = new GetTransaction(Resources.ASSET);
apiService.doDataRetrieve(trans, retrieveCallback);
```

When retrieving a list of object, each entry will by default have a minimum of data, to identify the object but not give details. We are interested in the asset ID, which uniquely identifies the asset and can be used to retrieve the detailed asset object. In the Starter App, we also get the ID of the “positionProvider”, which represents a device which can post observations and events into the SmartPlatform. We show how to post such data later in this tutorial.

Each asset in the list also has a name. If you want to add support for having multiple primary users, you need to consider the whole list returned by this transaction. If there are more than one, you could list the names in the user interface, for the user to select one.

Asset properties – user preferences

When we have the asset ID, we can get the complete asset object for this primary user by setting the ID on a GetTransaction. Note that we use a different callback, for a single JSONObject.

```
AsyncService.TransactionCallback callback = new AsyncService.TransactionCallback()
{
    public void onTransactionOK(JSONObject object) {
        JSONArray ja = object.getJSONArray("properties");
        // Get name and value of each JSONObject in the array
    }
    public void onTransactionError(int code, String message, Exception e) {
        Log.e("StarterApp", message);
    }
};

GetTransaction get = new GetTransaction(Resources.ASSET);
get.setId(assetId);
apiService.doTransaction(get, callback);
```

The asset object holds various information about the primary user, including the last known position. We are mainly interested in the properties, where we store POSEIDON user preferences and other settings, to be shared between applications. This is an array of JSON objects. For reading the preferences, we are interested in the "name" and "value" properties. The Starter App shows how to retrieve these, listing them all in the user interface when selecting the LIST PREFERENCES option in the main menu (PreferencesActivity in the code).

An application can update much of the information in an asset, such as a property value, writing the changes back to the server. To update an existing asset, we must use a PutTransaction.

```
PutTransaction put = new PutTransaction(Resources.ASSET);
put.setObject(asset);
apiService.doTransaction(currentTrans, callback);
```

In this code, *asset* is a JSONObject, like the one returned above. It does not need to be complete; it only needs the "id" field and whatever data we want to update. But a property object needs to include a field "typePropertyIdentifier", as that identifies the property in the SmartPlatform system. You may modify the JSON retrieved, and then send back the whole thing.

File server connection – cloud storage

The POSEIDON file server provides a common cloud storage for user data. It is used for instructional content, which is typically entered by a secondary user through the POSEIDON web application or a special-purpose app, and used by the primary user through primary user apps. Pilot content includes media, routes for navigation and shopping lists.

The file server has an HTTP API. Files are uniquely identified with a resource ID, and they are categorized with a type, which is intended to work much like directories in a file system, giving us a way to organize the files. Uploading a new file assigns it a resource ID. We use this ID to retrieve the file, and to provide a new version of an existing file. We can get a list of all files, or those of a specific type.

There is no Java wrapper for this API, so the recommendation is to use java.net.HttpURLConnection to do HTTP transactions. The Starter App code shows examples of this. The transactions themselves are all found in the class FileServerTransactions. A number of Activities, available from the main menu, run these and show the result.

File content

Before you start interacting with the file server, you need to have a clear idea of what you want to store and/or retrieve. It can be any type of file, but there are some types already defined by the developer framework and used by other applications. In the POSEIDON project, we found that instructional content typically has two aspects: some structured data with text, and some associated media. The structured data could be a shopping list – a set of products with quantities and prices – or a route for navigation – a list of geographic coordinates with instructions on how to get to the next point. This is typically stored in JSON or XML format. As we want to use more than just text to convey the instructions, the structured data may also refer to images, sound or video files. Each of these media files are uploaded to the file server, where it is given a resource ID. This resource ID is entered in the XML or other structured data, to use the media file in the instruction. In this way, an instructional content such as a shopping list will consist of a set of files.

You can look at the various data specifications of the framework, to see if and how to make use of one of the existing forms of instructional data (see Further reading at the end of this chapter). Or you can define your own for a new type of functionality.

Authentication

The POSEIDON account is used for authentication, with the file server connecting to the SmartPlatform to authenticate the user. The requests must contain the same authentication token as used for the SmartPlatform API. We have already seen how to get this from the AsyncService if we have logged in that way. If we only want to use the file server, we can send the user name and password there, and get back the authentication token. See the method FileServerTransactions. authOnFileServer for how this can be done.

Retrieving files

To retrieve files, we typically first do a GET request for a list of files, specifying a type if we want to limit it to that type. This returns a JSON listing the files, with a resource ID and original file name for each. An MD5 hash is also given, if we already have files cached locally, and we want to check if the local copy matches that on the server or not.

METHOD	GET	**<BASE_URL>/files/resource.php**
Headers	X-Auth-Token	<TOKEN>
Parameters	type (TEXT)	(OPTIONAL) Category of the file. Will list only resources in this category if provided.
	assetID (TEXT)	(OPTIONAL) indicating which asset the resource should be associated with. This only applies to users that control multiple assets.

Example reply:

```
{
   "image":[
   {
      "resourceID": "b0.jpg",
      "name": "b0.jpg",
      "mime": "jpg",
      "md5": "0A212AE3D160DFB219863D0B22192EC5"
   },
   {
      "resourceID": "b2.jpg",
      "name": "b2.jpg",
      "mime": "jpg",
      "md5": "5A45BCE032195AEFF13C3CAB714002CF"
   }
   ]
}
```

Starter App implementation: FileServerTransaction.listResources

Knowing the resource IDs, we can retrieve specific files one by one. This is a GET request with a resource ID, which returns the stream of bytes making up the file.

METHOD	GET	**<BASE_URL>/files/resource.php**
Headers	X-Auth-Token	<TOKEN>
Parameters	resourceID (TEXT)	The UUID of the resource to be fetched

	assetID (TEXT)	(OPTIONAL) indicating which asset the resource should be associated with. This only applies to users that control multiple assets.

The StarterApp method FileServerTransactions.fetchResources shows how this can be done in Java code, in this case getting an image and showing it directly. Alternatively, the byte stream could be saved to a file, so that we have it stored locally and don't need to download it each time it is used.

Uploading files

The File server API specification[5] shows how to add a new file, posting with the multipart/form-data content type. This combines several different entities in one payload, specifically the file type and file name as well as the file itself in this case. It works well for web applications, but we have not been able to find a good way to do this in Android code. In Android we therefore have to split this transaction into two stages, and use the alternative resource2.php on the server.

First, we post the type and filename. This returns a JSON which contains the resource ID assigned to the new file.

METHOD	POST	**<BASE_URL>/files/resource2.php**
Headers	X-Auth-Token Content-Type	<TOKEN> application/x-www-form-urlencoded
Payload	type=<TYPE>&filename=<NAME>	

Starter App implementation: FileServerTransaction. addResource

We can now upload the file itself in a POST request, specifying the resource ID in the URL.

METHOD	POST	**<BASE_URL>/files/resource2.php?resourceID=<resID>**
Headers	X-Auth-Token	<TOKEN>
Payload	File content	

Starter App implementation: FileServerTransaction.uploadFile

In the Starter App code, the AddResourceActivity shows how this is done, using both FileServerTransaction methods.

If we change a file which is already on the file server, we should upload the new version of the file using the existing resource ID, so that it replaces the old version. This is exactly the same as the second stage when uploading a new file, specifying the resourceID in the URL. So this is done with the FileServerTransaction.uploadFile method.

Deleting files

If a file is no longer used, it should be deleted from the file server, to free the space. This is done with a simple DELETE request, specifying the resource ID of the file to delete.

METHOD	DELETE	**<BASE_URL>/files/resource.php**
Headers	X-Auth-Token	<TOKEN>
Parameters	resourceID (TEXT)	The UUID of the resource to be deleted
	assetID (TEXT)	(OPTIONAL) indicating which asset the resource should be associated with. This only applies to users that control multiple assets.

[5] Available on http://www.poseidon-project.org/developers/developer-documentation/

FileServerTransaction.deleteResource implements this transaction. The main menu item for deleting a resource invokes PrepairDeletingActivity to get a resource ID, with DeleteResourceActivity running the delete transaction.

SmartPlatform edge connection – sending observations

In addition to the SmartPlatform REST API where data objects such as the asset can be retrieved and updated, this server has another API with its own Android/Java wrapper library. This is the so-called edge API. A core function of the SmartPlatform is to collect and aggregate data. This may be sensor data from various sensor devices and gateways, and the platform has various edges to talk to various device types. From an Android application we can post data in a JSON format to an HTTP edge. A wrapper library for this is provided as part of the POSEIDON framework, and included with the Starter App project. The ObservationActivity class demonstrates its usage.

In SmartPlatform terminology, a set of data posted in this way is known as an *observation*. All observations posted to the SmartPlatform server is stored in a history database. Both the latest values and the history may be shown in the monitoring part of the POSEIDON web application. The main POSEIDON pilot app posts various forms of information. It tracks the position of the user, and regularly posts the current position. This also includes the battery status of the device. It also posts log and error messages, as this was used in the research project to see how much each function was used and which problems occurred.

Device entry

Posting to an edge does NOT require authentication with an account. Instead, the device posting observations must be registered in the SmartPlatform with a Device entry, with a unique ID used to identify the source of the observations. Each POSEIDON asset has a device associated with it, representing the device running the apps. In the Starter App, we retrieve a device ID in the “positionProvider” field of the asset. This is not the ID we can use to post observations, but rather a foreign key referring to a device object. StarterApp.getDevice retrieves the device object:

```
GetTransaction get = new GetTransaction(Resources.DEVICE);
get.setId(deviceId);
apiService.doTransaction(get, callback);
```

In the device object, we need the field called “uuid”:

```
public void onTransactionOK(JSONObject object) {
    String uuid = object.getString("uuid");
    AsyncEdgePoster edgePoster = new AsyncEdgePoster(EDGE_URL, uuid);
}
```

Posting observation

The class AsyncEdgePoster from the edge library handles edge transactions. As shown in the code above, we need a URL and a device UUID to instantiate an object.

The class Observation in this library represents the data we can post. It has some built-in fields, such as timestamp and position. The rest of the data is simply a list of key-value pairs, so here we can put whatever we want to track. The Starter App ObservationActivity posts some test data:

```
Observation obs = new Observation();
obs.setPropertyAsString("testProp1", "1");
obs.setPropertyAsString("testProp2", "something");
```

edgePoster.doPostAsync(obs, this);

Context middleware connection

The POSEIDON Context Reasoner Middleware is an Android component which can run in the background on Android devices and provide context information to POSEIDON applications. It is a separate installation, available on Google Play:

https://play.google.com/store/apps/details?id=org.poseidon_project.context

The source code is available on GitHub:

https://github.com/deankramer/POSEIDON-Context

Here we will give a summary of how to use the context reasoner from a POSEIDON application. We refer to the project deliverables (see Further reading below) for a more detailed description.

Context Observers

The application using the middleware subscribes to specific contexts, to receive a message each time there is new information about these contexts. The following contexts are available:

- **BatteryContext:** Monitors the remaining amount of device battery in percent
- **WifiContext:** Monitors the status of the WiFi receiver on the device, whether it is not active, disconnected, connecting, or connected to an access point
- **TelephonyContext:** Monitors multiple parameters of the device's telephony receiver. This currently includes connection status, and whether the device is currently roaming
- **CompassContext:** Monitors the current orientation of the device in degrees.
- **LightContext:** Monitors the current amount of ambient light, measured in lumens.
- **ExternalStorageSpaceContext:** Monitors the amount of storage space remaining on the device shared storage area including the SD card, measured in megabytes (MB)
- **GPSIndoorOutdoorContext:** Monitors and deduces if the device is indoors or outdoors using the GPS receiver.
- **WeatherContext:** Monitors and sends weather data for defined locations.
- **StepCounter:** Monitors the current number of steps the user walks within a given interval.
- **DistanceTravelledContext:** Monitors the total distance the device moves within a given interval.

The middleware is extensible, and both Context Observers and rules for the reasoner can be added, but this is outside the scope of this tutorial.

Usage

Connecting an app to the middleware requires an AIDL (Android Interface Definition Language) file, which defines methods for the app code to communicate with the middleware. This file is found in the source on GitHub, in the following location:

```
/reasoner/src/main/aidl/org/poseidon_project/context/IContextReasoner.aidl
```

Copy this into the app project src folder.

To use this interface, the app must first connect to the middleware by calling bindService on a context (this could be done in an Activity, or the Application object).

```
Intent serviceIntent = new Intent(IContextReasoner.class.getName());
boolean ok = bindService(serviceIntent, middlewareConnection,
Context.BIND_AUTO_CREATE);
```

middlewareConnection here is an object implementing the ServiceConnection interface:

```
IContextReasoner contextService;

ServiceConnection middlewareConnection = new ServiceConnection() {
    public void onServiceConnected(ComponentName componentName, IBinder iBinder) {
        contextService = IContextReasoner.Stub.asInterface(iBinder);
    }

    public void onServiceDisconnected(ComponentName componentName) {
        contextService = null;
    }
}
```

If the connection is successful, onServiceConnected is called, and we get a reference to an object implementing the AIDL interface. To subscribe to context data, we call its addContextRequirement method, or addContextRequirementWithParameters if the context requires parameters for configuration. Arguments are the package name of our app, and the name of the context to subscribe to, for instance:

```
contextService.addContextRequirement(getPackageName(), "indoorOutdoor");
```

The middleware will broadcast context updates using Android's Intent mechanism as long as there are subscribers to the context. Our app needs to register a BroadcastReceiver to catch the Intents. The context data is in the Intent extras:

```
IntentFilter filter = new
        IntentFilter("org.poseidon_project.context.CONTEXT_UPDATE");
BroadcastReceiver contextBR = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        Bundle bundle = intent.getExtras();
        String name = bundle.getString(ContextBridge.CONTEXT_NAME);
        String value = bundle.getString(ContextBridge.CONTEXT_VALUE);
    }
};
registerReceiver(contextBR, filter);
```

Finally, we must call removeContextRequirement when the subscription is no longer needed, so that the middleware can stop its observing and reasoning:

```
contextService.removeContextRequirement(getPackageName(), "indoorOutdoor");
```

Further reading

Here is a collection of references to other documentation relevant for developing POSEIDON Android applications. All is available from the POSEIDON website, in the developer documentation section unless otherwise noted.

User interfaces

- User interface section of these guidelines.
- **D4.2 Adaptive tablet interface**: Adaptive aspects in the implemented POSEIDON app interface.
- **D4.5 HCI user and developer manuals chapter 4**: Android User Interface Implementation – describes important aspects of the Android platform in relation to user interaction (summarised in the user interface guidelines).

SmartPlatform

- **SmartPlatform-developer-doc.pdf**: Developer documentation for the SmartPlatform part of the infrastructure, including API specification.

- **rest-lib-java**: Documentation for the SmartPlatform API wrapper library (Javadoc) is included in the rest-lib-java archive available from the Code section of the POSEIDON website.
- **edge-lib-java**: Documentation for the SmartPlatform edge wrapper library (Javadoc) is included in the edge-lib-java archive available from the Code section of the POSEIDON website.
- **D5.1 Development framework, chapter 5.1**: Description of the platform and its role in POSEIDON.

File server and data specifications

- **File-server-API.pdf**: Specification of the file server HTTP API.
- **Route-data-specification.pdf**: Developer documentation for route data for navigation.
- **Shopping-list-data-specification.pdf**: Specification of JSON data for shopping lists.
- **Video-list-data-specification.pdf**: Specification of JSON data for video lists.
- **D5.1 Development framework, chapter 5.2**: High-level description.

Context middleware

- **D5.1 Development framework, chapter 5.4**: Developer documentation for the middleware.
- **D3.2 Reasoning and Learning module**: Background documentation for the context reasoning (available on the Deliverables page of the website).

Calendar integration

The POSEIDON developer framework also specifies use of calendar data. The Google Calendar service is used to store the calendar events, but with POSEIDON-specific structured content. As with other instructional content, this can refer to media files on the POSEIDON file server.

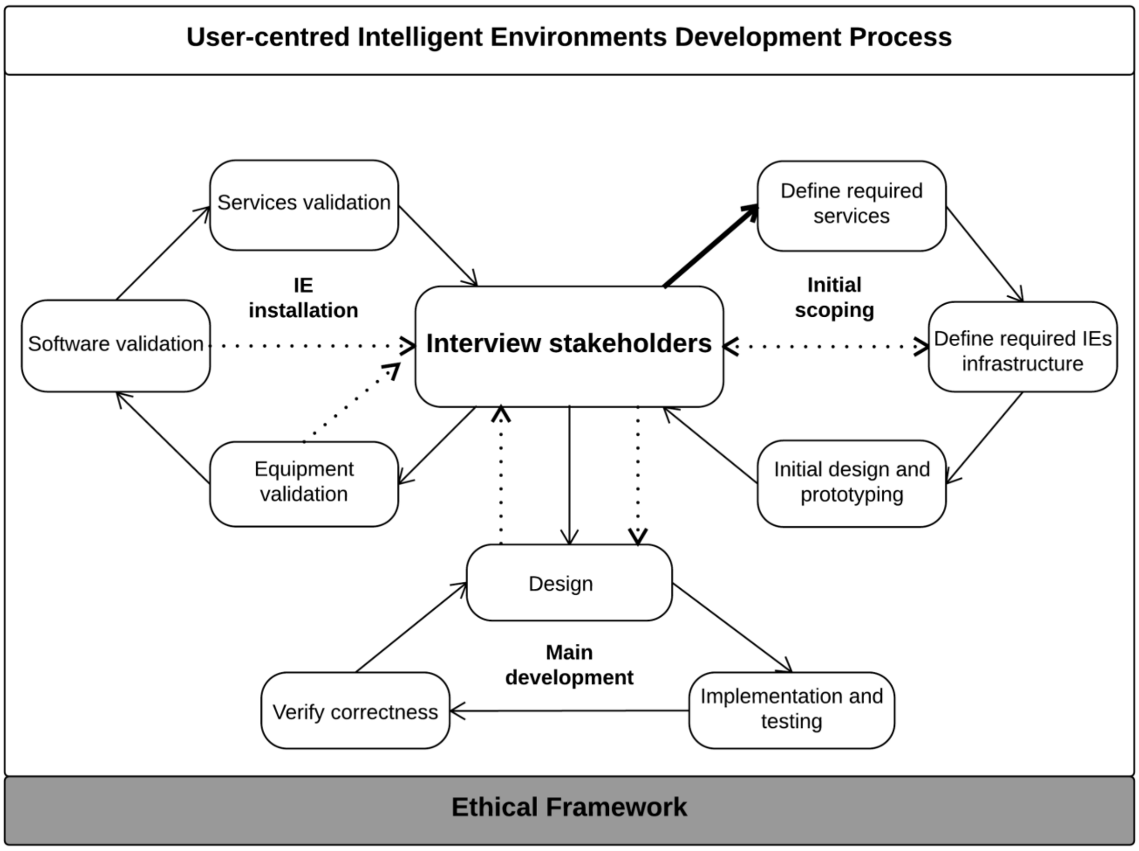
- **Calendar-data-specification.pdf**: Data specification and other developer information.
- **D5.1 Development framework, chapter 5.3**: Architecture and conceptual model.

4. Tutorial for context-aware systems

Context-awareness Development Framework

As part of the project activities we have refined and created a set of methods and tools to support the creation of context-aware systems. These are complex systems which so far do not have specific support, see (Alegre et al., 2016)[6]. Amongst the benefits of adopting these guidelines are that we can ease the creation of more reliable context-aware systems, reduce the implementation time of context-aware features in the long term and they cover most common development stages.

The overall strategy for developing systems which was applied to POSEIDON is the User-centred Intelligent Environments Development Process (pictured below). The generic process was explained in (Augusto, 2014)[7] and its application to POSEIDON was explained in (Augusto et al., 2017)[8].

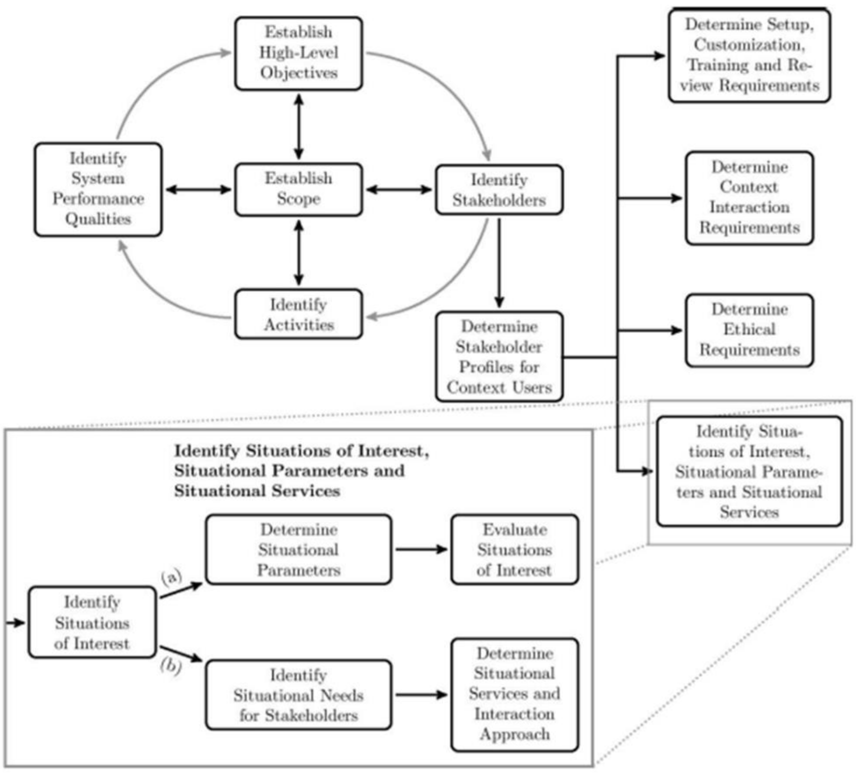


Each stage can have its own specific methodologies and tools which can be combined and complemented in any way. We also refined specific methods and tools for requirements and design explained below.

[6] Unai Alegre, Juan Carlos Augusto, Tony Clark (2016). Engineering Context-Aware Systems and Applications: A survey. Journal of Systems and Software Vol. 117, pp. 55-83. ISSN 0164-1212. Elsevier.
[7] Augusto, Juan Carlos (2014). A user-centric software development process. In: The 10th International Conference on Intelligent Environments (IE'14), 29 Jun – 04 Jul 2014, Shanghai, P.R. China.
[8] J. Augusto, D. Kramer, U. Alegre, A. Covaci and A. Santokhee (2017). The User-centred Intelligent Environments Development Process as a Guide to Co-create Smart Technology for People with Special Needs. To appear in Universal Access in the Information Society, Springer. 2017.

Requirements Gathering Approach

This helps understanding and agreeing what is expected from the final product. Our method follows several stages outlined in (Evans et al., 2014)[9] and then extended by (Alegre et al., 2016)[10]. The following picture captures the main steps of the process:


Establish High-Level Objectives
Identify System Performance Qualities
Establish Scope
Identify Stakeholders
Identify Activities
Determine Stakeholder Profiles for Context Users
Determine Setup, Customization, Training and Review Requirements
Determine Context Interaction Requirements
Determine Ethical Requirements
Identify Situations of Interest, Situational Parameters and Situational Services
Identify Situations of Interest, Situational Parameters and Situational Services
Identify Situations of Interest
(a)
(b)
Determine Situational Parameters
Evaluate Situations of Interest
Identify Situational Needs for Stakeholders
Determine Situational Services and Interaction Approach

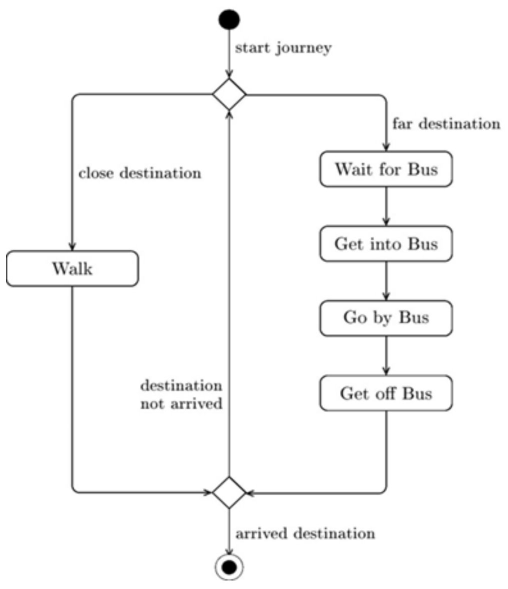
The forest of this section illustrates the process of identifying the contexts that matter and how to implement them. Stakeholders are identified and listed in the following table:

Stakeholder	Description
Primary Users	People with Down syndrome
Secondary Users	Parents or carers of people with Down syndrome
Tertiary Users	Teachers or supervisors of people with Down syndrome
Calls Provider	Company that provides phone calls and SMS to the mobile device
Internet Provider	Company that provides internet to the mobile device
Device Manufacturer	Company that manufactures the device
Operating System Developers	Group involved in the development of the operating system of the device
Maps Library Developers	Group involved in the development of maps libraries

[9] Carl Evans, Lindsey Brodie, Juan Carlos Augusto (2014). Requirements Engineering for Intelligent Environments. In: Proceedings The 10th International Conference on Intelligent Environments (IE'14), pp. 154-161. 29th June-4th July 2014, Shanghai. IEEE Press.

[10] Unai Alegre, Juan Carlos Augusto, Tony Clark (2016). Engineering Context-Aware Systems and Applications: A survey. Journal of Systems and Software Vol. 117, pp. 55-83. ISSN 0164-1212. Elsevier.

The following activity diagram shows typical activities involved in a bus journey:



The following table identifies different expected services by different stakeholders:

Stakeholder	Goals	Sub-Goals	Requirements
Primary User (PU)	Increase their independence from SUs when moving from one place to another (outdoors)	Guide the PUs through reminders of what they need to do next	Receive instructions that are able to understand, taking into account possible visual and auditory sensibilities
Secondary User (SU)	Reduce the attention that PU require on them when they have to do displacements outdoors	Show the SU that the PU is safe	The system has to enable the communication between the PU and the SU anytime
		Show the SU that the PU is doing what is meant to do	The SU can request the location of the PU
			The SU can know when a PU has finished a task

Then situation of interest are identified:

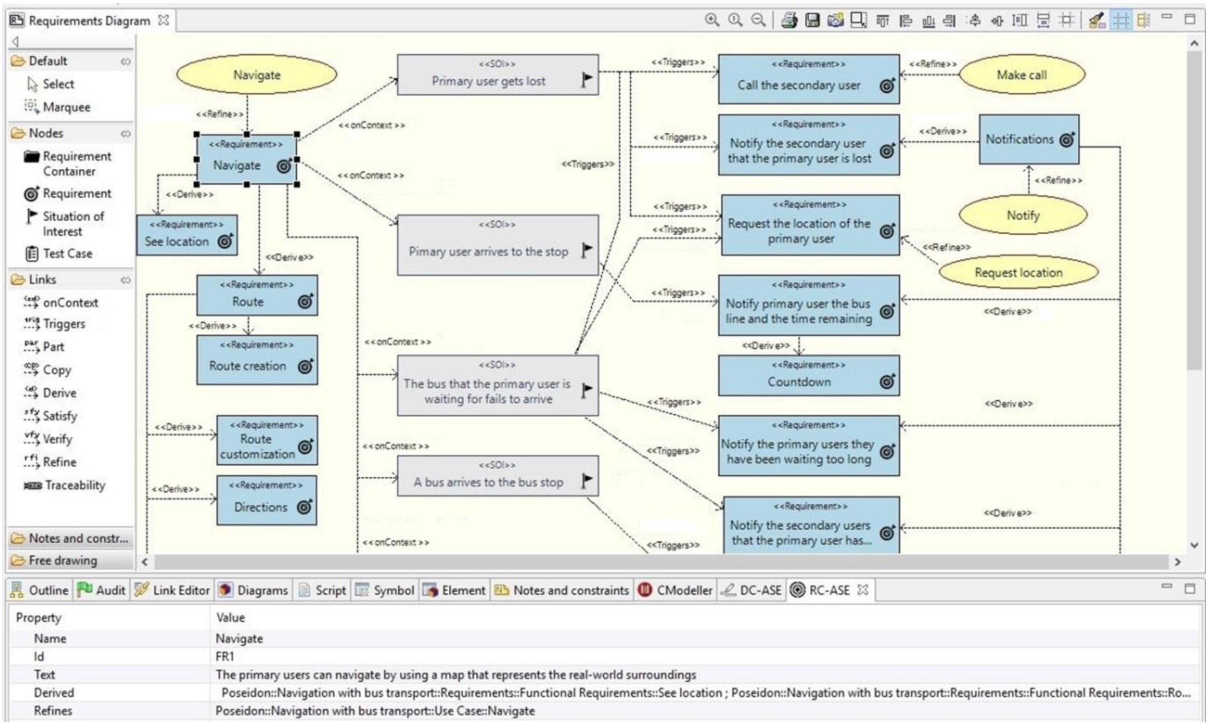
Activity	Situation of Interest
Waiting for bus	The user has just arrived to the bus stop
	The bus arrives to the bus stop
Going by bus	The bus arrives to the destination bus stop
	The bus arrives one stop before destination

Then we need to identify what is required to detect those Situations of Interest when the real system is deployed and operative:

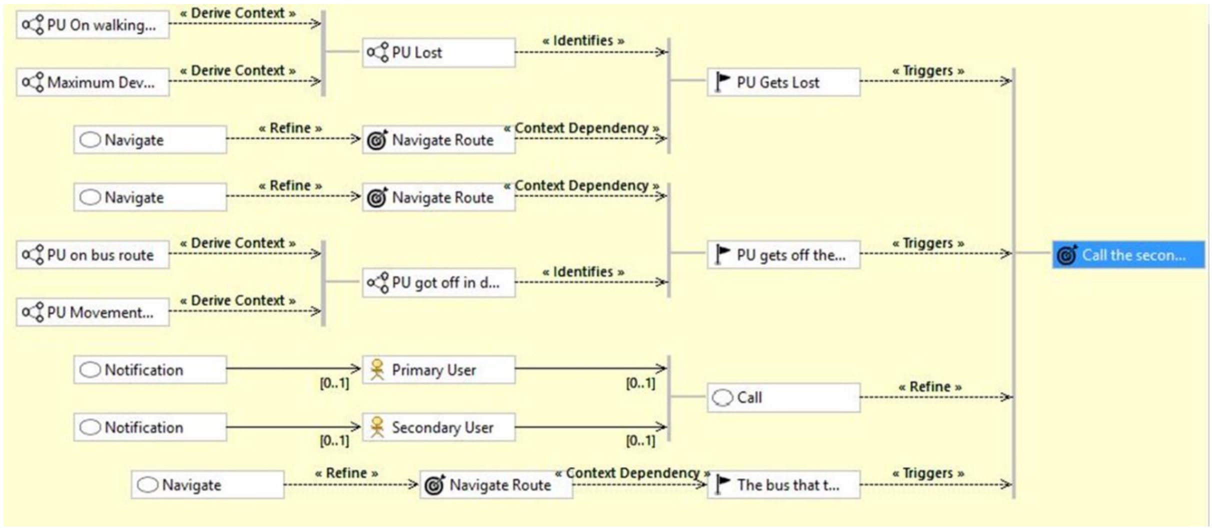
Activity	Situation of Interest	Situational Need	Situational Service
Waiting for bus	The user has just arrived to the bus stop	Know how much time they have to wait	Notification showing the time remaining for the next bus
	The bus arrives to the bus stop	Know if the bus is the one that they need to embark	Notification informing the user that the bus that has arrives is the one that needs to embark or not
Going by bus	The bus arrives to PU destination	Know when to get off bus	Notification reminding the users that they need to get off at this stop
	The bus arrives one stop before the desired one	Know when they need to press the stop button	Remind users that they need to press the stop button

Activity	Situation of Interest	Identification Description	Situational Parameter	Source
Waiting for bus	The PU arrives to the bus stop	The PU location coincides with the bus stop location	PU Location	PU Mobile device's coordinates
			Bus Stop Location	Bus stop coordinates
	The bus arrives to the bus stop	The time is approximately the time when the bus was scheduled to come	Bus Arrival Time	Bus line arrival time schedule
			Current Time	PU Mobile device's time

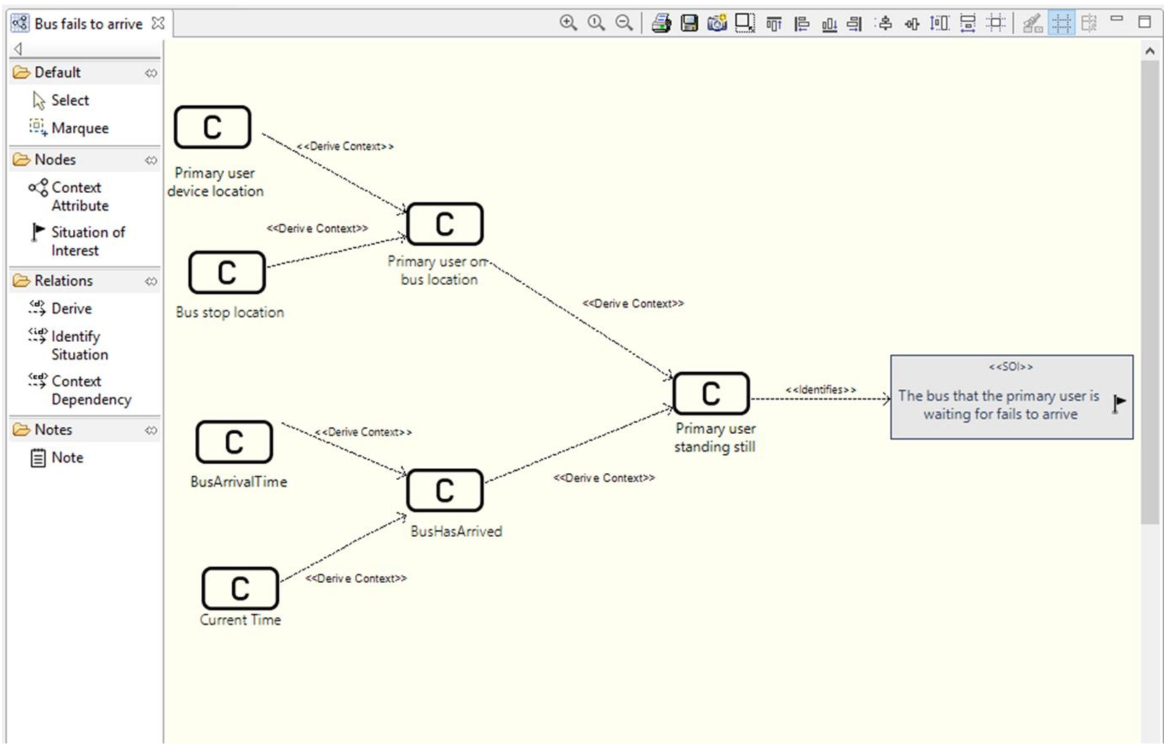
The methodological framework facilitates a more systematic analysis on all the contexts needed in a system, it support developers on thinking about contexts, encourages co-creation from early stages. This process is supported with tools which ease the information management and help tracing the relations between contexts at different stages of the development process:

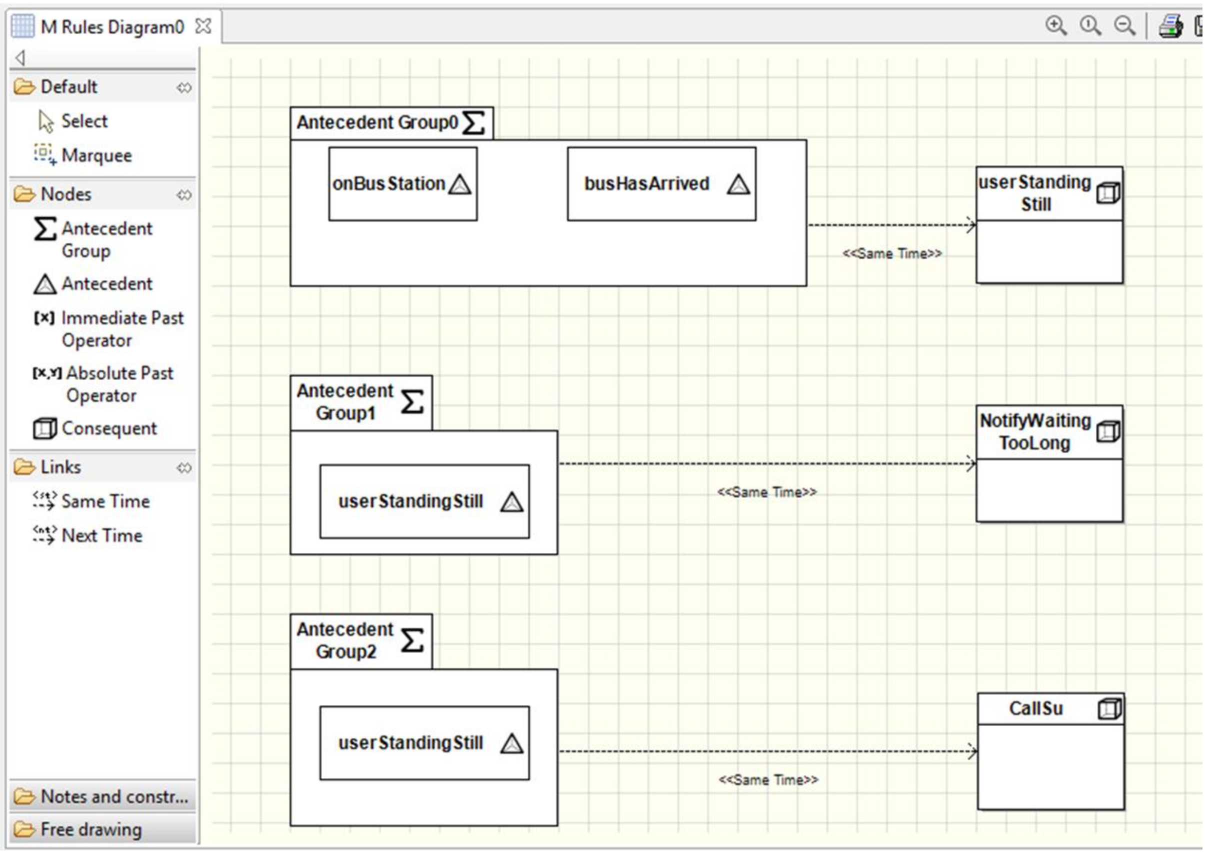

Requirements Diagram
Default
Select
Marquee
Nodes
Requirement Container
Requirement
Situation of Interest
Test Case
Links
onContext
Triggers
Part
Copy
Derive
Satisfy
Verify
Refine
Traceability
Notes and constr...
Free drawing
Navigate
<<Requirement>> Navigate
<<Requirement>> See location
<<Requirement>> Route
<<Requirement>> Route creation
<<Requirement>> Route customization
<<Requirement>> Directions
<<SOI>> Primary user gets lost
<<SOI>> Pimary user arrives to the stop
<<SOI>> The bus that the primary user is waiting for fails to arrive
<<SOI>> A bus arrives to the bus stop
<<Requirement>> Call the secondary user
<<Requirement>> Notify the secondary user that the primary user is lost
<<Requirement>> Request the location of the primary user
<<Requirement>> Notify primary user the bus line and the time remaining
<<Requirement>> Countdown
<<Requirement>> Notify the primary users they have been waiting too long
<<Requirement>> Notify the secondary users that the primary user has...
Make call
Notifications
Notify
Request location
<<Refine>>
<<Derive>>
<<onContext>>
<<Triggers>>
Outline
Audit
Link Editor
Diagrams
Script
Symbol
Element
Notes and constraints
CModeller
DC-ASE
RC-ASE

Property	Value
Name	Navigate
Id	FR1
Text	The primary users can navigate by using a map that represents the real-world surroundings
Derived	Poseidon::Navigation with bus transport::Requirements::Functional Requirements::See location ; Poseidon::Navigation with bus transport::Requirements::Functional Requirements::Ro...
Refines	Poseidon::Navigation with bus transport::Use Case::Navigate



We also used a methodological framework to guide developers into creating diagrammatic views of the system to help them visualize a solution and explore options. Again this was supported by tools which, as a side effect also facilitate the inclusion of other. The structural models created allows to define more specific elements like context variables, and to configure code generation:


Bus fails to arrive
Default
Select
Marquee
Nodes
Context Attribute
Situation of Interest
Relations
Derive
Identify Situation
Context Dependency
Notes
Note
C
Primary user device location
<<Derive Context>>
C
Bus stop location
<<Derive Context>>
C
Primary user on bus location
<<Derive Context>>
C
Primary user standing still
<<Identifies>>
<<SOI>>
The bus that the primary user is waiting for fails to arrive
C
BusArrivalTime
<<Derive Context>>
C
BusHasArrived
<<Derive Context>>
C
Current Time
<<Derive Context>>


M Rules Diagram0
Default
Select
Marquee
Nodes
Antecedent Group
Antecedent
Immediate Past Operator
Absolute Past Operator
Consequent
Links
Same Time
Next Time
Notes and constr...
Free drawing
Antecedent Group0
onBus Station
busHasArrived
userStanding Still
<<Same Time>>
Antecedent Group1
userStandingStill
NotifyWaiting TooLong
<<Same Time>>
Antecedent Group2
userStandingStill
CallSu
<<Same Time>>

5. Technology use of persons with DS

People with Down syndrome represent a unique target group since three major types of capabilities (first cognitive, but also to some extent motor and perceptual) are affected. The capabilities and disabilities often vary to a high extent from person to person with Down syndrome. Furthermore, each person with Down syndrome shows individual characteristics related to heredity, training and the social environment around him/her. Thus, only some generalisations are possible for persons with Down syndrome. The high individuality represents a major challenge for a technical development and underlines the importance of a user-centred design approach with many possibilities for individualisations. Despite the wide range of competencies and impairments, common problems for people with Down syndrome are:

- Limitations in vision and hearing, disturbed sensory skills, including hyposensitivity when touching something, problems with fine motor movements
- Weak muscles in arms and fingers
- Limitations in short term memory and cognition, including problems with verbal auditory memory, hence it is harder to recall information that is heard than what is read or seen
- Limitations in communication skills. Delayed development of receptive and expressive language. They often understand much more than they can express (receptive language is superior to expressive language). They have a relative strength in vocabulary and pragmatics and more difficulties with morphology and syntax. They also in general have reduced speech intelligibility. This lead to problems with complex conversational skills. (Overview: Lazar, Kumin & Feng, 2011)[11].

It is important to recognise both the strengths and difficulties a person with Down syndrome has. As McGuire and Chicoine (2006)[12] note, most children with Down syndrome enjoy and learn from social interaction with family and friends. As time goes by, they often have good social and emotional understanding, and most are able to develop age-appropriate behaviour, if this is encouraged and expected. People with Down syndrome generally learn visually; this is probably why reading can sometimes be a strength. This means that they learn best from watching and copying other people, and may find it easier to take in information if it is presented with the support of pictures, gestures, objects and written words. Using their hands, faces and bodies to communicate is another strength, and many people with Down syndrome enjoy drama and movement because of this strength. The approach taken within POSEIDON should always build on the strengths of people with Down syndrome while keeping the possible limitations in mind.

People with Down Syndrome as computer users

The possible limitations and characteristics of people with Down syndrome have a major impact on the usage of technology and underline the need for usability-focused and user centred design. The goal is not only to develop technology that is easy to use in general, the technology must also be useful to the user in spite of his/her possible limitations. The limitations in cognitive, language and motor skills all have a profound impact on computer usage for persons with Down syndrome. They have e.g.

[11] Lazar, J., Kumin, L., & Feng, J. H. (2011): Understanding the computer skills of adult expert users with Down syndrome: An exploratory study. In The proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility. ACM, 51-58.

[12] McGuire, D. & Chicoine, B. (2006): Mental Wellness in Adults with Down Syndrome: A Guide to Emotional and Behavioural Strengths and Challenges. Bethesda: Woodbine House.

difficulties in remembering, processing information, understanding abstract concepts, reading, writing, communicating, navigating, typing and using the mouse. These limitations present major challenges both for users, designers and technicians. Nonetheless the diversity of capabilities is a problem, but many persons with Down syndrome can use PC, tablets and smart phones surprisingly well.

In POSEIDON, we have looked at studies and extracted guidelines from the findings. The findings suggest that computers and computer devices are of great importance for people with Down syndrome because they can help to increase confidence and motivation through creative activities and web browsing. Using the computer has other benefits as well, including errorless learning, patient and immediate feedback, self-paced learning and independence of learning. It should be stated also that all those benefits and useful features are dependent on developing a technology which meets the heterogeneous demands of the target group.

The following table combines a study on preferences of people with Down syndrome with the more general "Information for all" standard. The first is an in-depth study of a small number of people with Down syndrome. The second is a standard for people with intellectual disabilities in general, and focuses on making information easy to read and to understand on different media. Both share the idea of using as simple wording as possible, and both strongly suggest the use of photographs and images to support the idea presented with words. The recommendations for the use of animations differ, but this is based on different usage of the animations. For example, if animations are always placed on the screen it can be distracting. On the other hand, some personalized animations used as reward system can be motivating. Furthermore, a general rule is to avoid everything unexpected, like e.g. pop-up windows.

Guideline subject	Preferences of people with Down syndrome	"Information for all" people with intellectual disabilities
Font	- Bright, adding depth - Large - Bold - Stylized - No font decoration	- Not serif fonts, rather Arial, Tahoma - Enough spacing between letters - Not italics and/or underlining - No shadows, especially in writing design - Large fonts, at least like Arial 14 - One font throughout text
Colour	- Darker: blue, purple, grey - Combinations of primary colours with high contrast - Tints and tones - Complimentary colours - No dull colours	- Avoid a background colour or a background with pattern that can make the contrast between text and background poorer
Graphics/Images	- Cleary identifiable - Naturally coloured not digitally manipulated - With people of similar age or older - Action images - Photographic better than illustrated - Fun and whimsical illustrations	- Photographs, drawings, symbols - Don't use images for younger people than the target age group - Clear, not too disturbing to look at - Fit to the text

Animations	- Bright colour - With motion - Animating colour - Personalized	- No animations on screen
Buttons	- Largest was clicked first - Dark background - Light text on top - Expected action clear - Framed - Arrows pointing to buttons - No spatial preference	- Large button to change size of the writing
General on websites		- No pop-ups - No large programs (hardware, internet speed restrictions) - Search tool - Easy-to-read in metadata
Homepage		- Clear what website is about - Phone number to contact person - Postal address to contact person - E-mail address to contact person - Easy-to-read symbol if the content is tested according to easy to read principles
Navigation		- Clearly show on which part of the website one currently is - One click to homepage - Same navigation bar on the same place throughout the website - Not more than 7-8 menu choices
Screen		- Whole text on one screen - No lateral scrolling - Menu of sections at top - Easy to return to top - Space between paragraph - No animations
Links		- Words, not pictures, should contain links - Underlining of links - Hiding long link behind word - Blue if not clicked - Purple if already clicked
Words		- Well known - Explain complex words - Use examples - No initials - Do not use metaphors - Do not use word from other languages

		- Use words in full, not initials - No percentages, no large numbers use “few”, “many” instead
Sentences		- Short - Direct speech - Address in 2nd person e.g. “you” - Positive - Active
Information order		- Group information about topic - Important information at the top - Repeat important information - Repeat explanation of difficult words if words are used several times

6. Ethical and privacy concerns

Ethical Framework

When developing AAL related applications, like POSEIDON , we recommend applications to adhere to the eFriend Framework (Jones *et al.*, 2015)[13]. This framework was informed by the *Intelligent Environments Manifesto* (Augusto *et al.*, 2013)[14] supporting the following principles:

- P3: Deliver help according to the needs and preferences of those who are being helped.
- P5: Preserve the privacy of the user/s.
- P6: Prioritise safety of the user/s at all times.
- P9: Adhere to the strict principle that the user is in command and the computer obeys.

The eFriend framework relies on the developer following these simple principles:

1. **Non-maleficence** and **beneficence:** systems should be created to not cause any harm, particularly to primary users. Systems should aim to bring social benefits to users, by the increase in their quality of life.
2. **User-centred** and **multiple user groups:** it is important to identify and accommodate the different preferences of various users and their potential conflicts and incompatibilities.
3. **Privacy:** The users of such systems should retain the ability to exercise control over monitoring, tracking, and recording activities in the systems. Users should be able to adjust privacy settings for different POSEIDON compatible services.
4. **Data Protection** and **Security:** All data collected in the process of running POSEIDON compatible services must comply with relevant data protection legislations in the territories in which they are consumed e.g. the UK Data Protection Act 1998. Users should also be capable to choose what personal information can be accessed, and how it can be used. Security is a responsibility you should take seriously, including maintaining safety and security of any collected, processed, or stored data.
5. **Autonomy:** Another important foundation for user trust. Primary users should be enabled to specify and adjust their level of autonomy. This can include the reconfiguration, customization, and overriding of components in the POSEIDON system, allowing the user to take control.
6. **Transparency:** It is important that primary users of POSEIDON compatible systems know and understand how different services can affect their lives in positive and negative aspects. This can be handled by making background tasks including surveillance more visible to the user.
7. **Equality** and **Dignity:** Developers should try to ensure the accessibility and affordability of devices, systems, and services to the primary user. Systems designed to be POSEIDON compatible should ensure social inclusiveness by accommodating different levels of cognition, competence and technical ability. These systems should under no except undermine user dignity, including stigmatising the user.

The most important step by the developing team is to embed these principles in the product, starting by having requirements which are related to each of the principles. In some systems, some principles will be more important than others so there will be represented with varying number of requirement, at least one per principle. In the design phase, this will include the embedding of these

[13] Jones, Simon and Hara, Sukhvinder and Augusto, Juan Carlos (2014) eFRIEND: an ethical framework for intelligent environments development. Ethics and Information Technology, 17 (1). pp. 11-27. ISSN 1388-1957. Springer Verlag.

[14] Augusto, J.C., Callaghan, V., Kameas, A., Cook, D., Satoh, I. (2013) Intelligent Environments: a manifesto. HumanCentric Computing and Information Sciences, 3:12, Springer. DOI: 10.1186/2192-1962-3-12 URL: http://www.hcisjournal.com/content/3/1/12

ethical principles in the system architecture and functional specifications. This will be followed by their incorporation into formal methods, behavioural properties, and system agents in the implementation phase. Following the installation of the system, formal verification and validation of the system will ensure that its behaviour is consistent with the key ethical principles. In the testing phase, pre-pilot studies will be conducted so that the capabilities of the hardware and software used are fit for purpose and fulfil the requirements. This should be accompanied by usability testing with different prototypes, involving field trials and pilot tests from which detailed feedback can be gathered from users.

7. User interface guidelines

Here is a collection of guidelines for user interface design and implementation. It is a good idea to read through these if you will be doing user interface design for a POSEIDON application.

POSEIDON guidelines for developing accessible user interfaces

This section provides guidance on usability and accessibility for the designers, developers and the ones responsible for testing and evaluating the POSEIDON system and its services. It provides a collection of practical advice of how to design the interaction so that it meets the requirements and capabilities of the target group of POSEIDON. The goal has been to try to keep these guidelines as simple as possible.

These guidelines are based on several existing guidelines. The most important references are:

- Principles of universal design[15]
- Information for all: European standards for making information easy to read and understand[16] (follows this document as separate attachment)
- Cognitive Accessibility User Research of W3C[17] Principles of universal design

Universal design is more than accessibility. With a focus on universal design we can have a more holistic approach to how we develop the POSEIDON system and services.

The concept of universal design is based on the design of products and environments to be usable by all people. Some of the principles are difficult to apply to ICT-based products and services. A subset of the principles for universal design will, however, apply for POSEIDON applications. In addition, we propose some implications these will have for such applications:

Principle 1: Equitable use

The design is useful and marketable to people with diverse abilities.

Principle 2: Flexibility in use

The design accommodates a wide range of individual preferences and abilities.

POSEIDON recommendations

- All users should be allowed to adjust their preferences for how the system should communicate with them (e.g., size of fonts, contrasts, colours, etc.). Enabling assistive technology such as synthetic speech (i.e., multimodality) is important.
- If a user has tremor, or problems with fine motor skills, and tapping a tablet/smartphone with her fingers could be problematic, the system should be set up so that it is more tolerant for user errors, timing for accepting that a button is pressed etc.

Principle 3: Simple and intuitive use

Use of the design is easy to understand, regardless of the user's experience, knowledge, language skills, or current concentration level.

POSEIDON recommendations

- Only relevant actions should be displayed in the user interface.

[15]http://www.ncsu.edu/project/design-projects/udi/center-for-universal-design/the-principles-of-universal-design/
[16] http://www.inclusion-europe.org/images/stories/documents/Project_Pathways1/Information_for_all.pdf
[17] https://w3c.github.io/coga/user-research/#down-syndrome

- All (most) action buttons should be located at the bottom of the screen, and they should always be visible.
- Help/information should always be located at the bottom-right corner of the screen, or to the right from the specific location where help or additional information is available.
- Search (if relevant) should always be located at the top-right corner of the screen.
- Action buttons should be a combination of icons and text. Exceptions may be made for the user interface, such as lists of choices created by the end-user and where suitable icons are not available, or when uploading of an icon is not provided.
- Focus points of a video service should always be at the centre of the screen (important for video content).

Principle 4: Perceptible information

The design communicates necessary information effectively to the user, regardless of ambient conditions or the user's sensory abilities.

POSEIDON recommendations

- Regardless of context of use, all information on the screen should be easily readable and understandable.
- Textual information presented to the end-user must be meaningful, and consider the end-users' capabilities and cultural frame of understanding.

Principle 5: Tolerance for error

The design minimises hazards and the adverse consequences of accidental or unintended actions.

POSEIDON recommendations

- When the end-user enters wrong data, or is using the system in a "non-predicted" way, the system should be "forgiving" and provide guidance to the end-user, and help to recover the error.
- If there are technical problems (disturbances in data communication, internet connection failure etc.), the system should still try to provide meaningful information to the end-users.

Principle 6: Low physical effort

The design can be used efficiently and comfortably and with a minimum of fatigue.

POSEIDON recommendations

- This principle is important when handling tools, opening doors etc. The POSEIDON system that will be operated on tablet PCs and/or smartphones will require very low physical effort.

Principle 7: Size and space for approach and use

Appropriate size and space is provided for approach, reach, manipulation, and use regardless of user's body size, posture, or mobility.

POSEIDON recommendations

All action buttons both on the web and on tablet PCs and smartphones must be large and easy to click/tap.

Design methodology for mock-up development

To achieve the goal of highly usable applications, it is important to start the interaction design by developing high-level mock-ups (paper prototypes). These could be pictures of the user interfaces. This helps the design team to figure out how the application should work, what input does it require, and what kinds of output are expected. When the users perform an action "using" the mock-up, they do have some expectations for what will happen.

The initial design cycle should work to identify the end-users expectations, and at the same time identify what information is needed from a developer's perspective to make the system work, and to provide the expected result.

When developing mock-ups (high-fidelity paper prototypes), and POSEIDON user interfaces in general, there are some very central design principles to follow[18]:

Principle 1: Learnability

The user interface should be easy to use from the first time a user interacts with it. There should be no need to learn new functionality or new ways of user interaction. The system should be based on recognition rather than the need to recall previous experiences.

POSEIDON recommendations

- All action buttons should be located at the bottom of the screen, and they should always be visible, common tasks should be located at the same place all the time, such as help, search, information etc.
- Based on the experience of the system, the first time a user uses the application some help and guidance should be provided, and after some times of use the help and guidance should be less intrusive.

Principle 2: Efficiency

The number of steps a user takes to complete a task should be as few as possible. The need for horizontal and vertical scrolling should be kept to a minimum. Wizards should be used to simplify complex interactions. Real world metaphors should be used where applicable. Less is more: most likely we need to leave stuff out.

POSEIDON recommendations

- Main device-orientation is horizontal in the case of tablets. For smartphones, the vertical orientation is preferable.
- All scrolling is vertical – no horizontal scrolling.
- All action buttons are located at the bottom of the screen, and are always visible. Exceptions to this may be buttons to handle the video content (e.g. Start-button in the middle of the screen and the like).
- Only relevant functionality should be visible.
- All information and help texts should be context sensitive, the information and help text should be relevant and to the point.

Principle 3: Error recovery

[18] Adapted from: http://www.slideshare.net/OpenRoad/mobile-ui-design-user-centered-design-and-ui-best-practices, http://www.google.com/about/company/philosophy/, http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/mobilehig/UEBestPractices/UEBestPractices.html, http://designfestival.com/5-principles-of-user-centered-interface-design/, http://www.netmagazine.com/features/10-principles-mobile-interface-design, http://uxdesign.smashingmagazine.com/2011/07/18/seven-guidelines-for-designing-high-performance-mobile-user-experiences/

The system should be designed so that it is hard or even impossible for a user to make mistakes. However, when a user mistake occurs, this should be clearly communicated with information on which actions to take to continue the use of the system.

If there is a system error, this should also be communicated in a clear way, with simple and under-standable information to the end-user. All error messages should be useful. The system should provide guidance on how the user should recover from the error.

POSEIDON recommendations

- There are three different types of errors that need to be addressed and have a consistent error recovery methodology.
    - User error
    - Device application error
    - Server application error (including error on services invoked from the POSEIDON system)
- It should not be necessary to re-enter any data when an error situation appears.
- When restarting the app or service, it should launch at the same state as it was when the error occurred.
- The system should be "forgiving" on user errors and provide mechanism for graceful degradation of functionality when an error situation occurs.
- E.g., if the video service is not responding, the system should continue to work (just not display the video), and the "space" used for the video should not be left blank.

Principle 4: Simplicity

Tasks frequently performed should be easy to do, and less common tasks should be possible to do. Unnecessary functionality should be avoided. The layout and design should be uncluttered.

The navigation should be ***narrow and shallow***, providing only necessary functionality. For this, we need to understand profoundly the context of when and where our users will use the system.

POSEIDON recommendations

- There should not be more than maximum three levels of navigation, ideally there should only be one level of navigation in the POSEIDON system.
- The design and design elements should be clean and simple.
- Where applicable, well-known principles and best practice should be applied.

Principle 5: Mapping

What the user expects to happen is what should happen. There should be a mapping between the conceptual model the user has of the system, and how the system actually works.

POSEIDON recommendations

- The conceptual model behind the POSEIDON system should be well documented and de-scribed.

Principle 6: Visibility

The most important information should be most visible, and less important information should be less visible. When using a touch interface, no button should be smaller than the user's fingertips plus "necessary margin" for users with tremor or problems with fine motor skills.

POSEIDON recommendations

- Only relevant actions should be displayed.

- All buttons should be of an easily press able size, with clear boundaries.
- Buttons should not be smaller than the size of the thumb.

Principle 7: Feedback

The user should be in control of the interface and not the other way around. The system should provide quick responses. If the response will take some time a progress bar or some other useful information should be provided. Speed and responsiveness are crucial for the user experience.

In today's computing environment one second is an "eternity" to wait for response from the system or application. If a system does not respond within a reasonable time frame, the users will assume there is an error and try again, or press other buttons that will null-out existing action causing confusion and a bad user experience.

POSEIDON recommendations

- If an action takes more than one second, a progress bar or other relevant information should be displayed to the end-user.

Principle 8: Consistency

Identical items, and identical functionality should always be displayed and behave the same way across the entire system/application.

POSEIDON recommendations

- All agreed common user actions should be tested and documented.
- All confirmation, information, help and error "pages" should have the same look-and-feel throughout the system.
- The system should be as predictable as possible.

Principle 9: Satisfaction

The users should enjoy using the POSEIDON system/software. The software should perform its expected tasks well and nothing more. If she would like to perform another task, she would most likely use another application (or system).

POSEIDON recommendations

- Only core-functionality should be provided by the POSEIDON system.
- For secondary functionality, we should defer the end-users to other applications that solve those tasks well.

Principle 10: Predictability

When a system follows the principle of predictability, the user would know what to expect from the system: The behaviour is consistent throughout the application/system/service. With a consistent user interface the user will not experience surprises. When a user presses a button, or invokes a service, it should be evident for the user what to expect, and it should also be evident how the results will be presented.

To ensure a predictable user experience, it is important to understand the targeted users' expectations and the conceptual models[19] they have for the system they are using. If we design a system based on a different conceptual model than the one of the end-users', the user interaction and how they use the system will never match the anticipations of the developers, and the system will score

[19] In this context, a conceptual model is the mental model an end-user has of how the systems works, and the end-users understanding of how different services and functions provided by the system works. An end-user will use the system based on the mental model she has on how the system works.

low on usability and expectations of the users. If the system is designed following the conceptual model of the end-users, we will get a high score on usability, because the behaviour of the system is what the end-users predict. The system and the user interaction follow the users' expectations.

Ideally there should not be any surprises for the end-user when using the system. If something unexpected happens, the methods for solving the unexpected should be predictable and well known by all users.

POSEIDON interface design guidelines

The POSEIDON project and pilot applications have tried to implement a common POSEIDON theme. Important elements here are choice of colour palettes and the idea behind the icon design. The idea is to provide a "family resemblance" between different POSEIDON applications, so that the user always understands that she in inside the POSEIDON system. If you design an application to be used alongside other POSEIDON applications, it is a good idea to try to stick to this POSEIDON style, at least if the application is targeted at primary users. These are the guidelines for implementing the POSEIDON style.

The colour schema

- The main requirement is good contrasts everywhere, in all elements of the user interfaces.
- Make the colour "skins" available in an easily adjustable way, not hard-coded in all SW.
- For (at least) first prototypes, start with light backgrounds and dark text, buttons etc.

POSEIDON colours:

System: RGB code		**System: Hex code**		
Black	**R**: 0 **G:** 0 **B:** 0	Black	#000000	⬛
White	**R:** 255 **G:** 255 **B:** 255	White	#FFFFFF	⬜
(POSEIDON) Turquoise	**R:** 53 **G:** 132 **B:** 140	(POSEIDON) Turquoise	#008080	🟩
(POSEIDON) Orange	**R:** 241 **G:** 165 **B:** 50	(POSEIDON) Orange	#F1A532	🟧
Red	**R:** 255 **G:** 0 **B:** 0	Red	#FF0000	🟥
Blue	**R:** 28 **G:** 120 **B:** 204	Blue	#1C78CC	🟦
Grey	**R:** 174 **G:** 167 **B:** 159	Grey	#AEA79F	▨

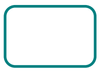
For visually impaired users, a POSEIDON primary user application should provide high contrast alternatives, such as black-and white palettes, in addition to the "branded" POSEIDON palette. This is very important for the readability of the text elements.

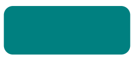
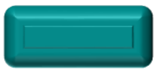
Text elements

- Font: Use only sans-serif screen fonts – no exceptions. **CALIBRI/Calibri**, or if not available, ARIAL/Arial. (Not Times New Roman or the like).
- High contrast choices everywhere, even if "neighbouring elements" would not follow the same principle, e.g.: two buttons beside each other where text is in different colour Black text on grey background White text on blue background.
- Always as large text as possible – even larger than what might feel "normal".
- All text must be implemented as character-based text, not images of text (for enabling synthetic speech later).

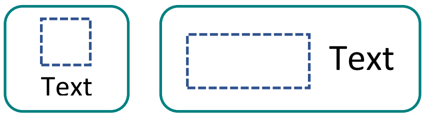
- No ALL CAPS text, no capital letters in the middle of expressions when not necessary (e.g. 'Start here', not 'Start Here') – this to harmonise the design and make it "calm". This can be changed later if necessary (NB. Conventions in different countries are different here!).
- No *Italics – at least long texts*. It is difficult to read.
- Normal, easy to understand language. No unfamiliar abbreviations or computer science jargon.

Action button design

Rounded edges: 

"Flat design" (no outline):  No 3D-effects:  (to harmonise initial design)

Text and place(holder) for transparent icons:



Icon library

A set of icons developed in the POSEIDON project is available as part of the framework: http://www.poseidon-project.org/product/symbols/

Here are some guidelines for creating new icons:

- All icons must be designed as transparent images, with high contrast against all possible backgrounds.
- All icons must follow "similar design". E.g., not a mixture of photo icons, pencil drawings and high quality graphics.
- Icons should when possible be combined with text.
- Symbols and terminology should be used consistently throughout the applications and systems in POSEIDON.

Decorative elements

- Do not use any unnecessary decorative elements such as boards, background images or the like.
- Do not use any animations (moving, blinking items) if not necessary for explaining something, e.g. a screen-cast connected to user support sections.

Navigation

- Allow the user to go back or "regret".
- Allow the user to start from beginning (Home, Start page).
- Show the user where s/he is "just now".

Scrolling

- Avoid scrolling.
- Avoid horizontal scrolling.

Actions and feedback to end user

- Let the user initiate actions, e.g. by clicking on buttons – even if not technically necessary.
- Acknowledge actions: show that a button is clicked on, that something may take time etc.

Language, national data formats

- Create a language library which allows alternative expressions and new languages be deployed without re-programming all software.
- Dedicate a work force to create an as compact and easy-to-read language as possible, especially for buttons.
- Make it possible to use familiar (diverse) formats for date, time, monetary units etc.

Help and support

- Provide help. Enable help to be added to the software "anywhere".
- Use commonly understandable, clickable symbols for more information, e.g. ⓘ
- Do not require the user to type information that the system already knows.

Look and feel

- We want the POSEIDON software to be as familiar and self-explanatory as possible.
- Strive for realisation of look and feel. Look and feel is a term used in respect of a graphical user interface and comprises aspects of its design, including elements such as colours, shapes, layout, and typefaces (the "look"), as well as the behaviour of dynamic elements such as buttons, boxes, and menus (the "feel")[20].
- Create family resemblance between all parts of the POSEIDON software by sticking to the rules in this document – even if you would personally disagree with something.
- Brand all apps with the POSEIDON app logos or banners. However, do not let these dominate the interface.

Responsive design

- We want the POSEIDON software to be usable across a variety of devices, depending on the end user's preferences.
- Strive for realisation of responsive design. Responsive web design is an approach to web design aimed at crafting sites to provide an optimal viewing experience—easy reading and navigation with a minimum of resizing, panning, and scrolling—across a wide range of devices (from desktop computer monitors to mobile phones)[21].

Accessibility

- We want the POSEIDON software to be usable, useful and easy to use for a variety of end users with different levels of abilities.
- Strive for realisation of accessible solutions on all POSEIDON platforms. Accessibility can be viewed as the "ability to access" and benefit from some system or entity. The concept focuses on enabling access for people with disabilities, or special needs, or enabling access using assistive technology[22].
- This document is just a short "start package" for the software team. For all more advanced accessibility issues, se accessibility guidelines such as[23].

[20] http://en.wikipedia.org/wiki/Look_and_feel
[21] http://en.wikipedia.org/wiki/Responsive_web_design
[22] http://en.wikipedia.org/wiki/Accessibility
[23] http://www.w3.org/standards/webdesign/accessibility

Android user interaction implementation

We have collected guidelines and tips for user interface implementation on the Android platform. A more extensive text on the subject can be found in deliverable D4.5, chapter 4. The following points are extracted from that text.

- The Android system usually have three navigation buttons/icons along the bottom of the screen, such as back and home buttons. They are the intended way to navigate between functions in the device, and some of them are completely outside the control of the app. Make sure to take both their functions and positions into consideration when designing a user interface.
- Using the back button, the user should be able to backtrack in our app, and leave the app when backtracking beyond the first screen.
- The Android navigation and system bars can be hidden when not needed, to maximize the available screen space, but keep in mind that the user can always bring them back. Since we cannot completely hide or control them, it may be best to leave them on screen, to avoid confusion.
- Try to minimize the need for keyboard input, as the on-screen keyboards on Android devices can be quite small and cumbersome to use.
- When keyboard input is available, make sure that the layout can handle the keyboard taking over much of the screen space. Make sure the keyboard does not hide the most essential information on the screen.
- Keep in mind that an application can lose focus or be shut down at any time (such as when the user presses the home button, or a phone call comes in). Make sure that the state of the application is preserved when this happens, and restored when it is shown again.
- Save changes made by the user when the user leaves a view. Show a little message to notify the user that changes were saved.
- Try to keep a clear separation between the user interaction logic and the concrete implementation in graphical elements, so that it is possible to change this implementation or provide alternatives based on user abilities and preferences.
- Use the Android resource system with qualifiers for alternative versions of resources. This makes it easy to handle adaptations to different languages, screen properties, colours and styles.
- Do not define text, font size, dimensions or colours directly in a layout file. Such properties are best defined one time in their own resource files, and referred to in layouts. This way they can be varied and changed independently of each other.
- To follow our user interface guidelines, we want to minimize the need for scrolling. However, designing a good layout without scrolling can be very difficult, depending on how much variability we want to support in terms of different screen sizes, font sizes and languages (text may be much longer or shorter when translated to a different language). We therefore need to make trade-offs. When scrolling is necessary, it is good to show the most important information first, and make it clear that there is more information.
- A mobile device screen can be used in either horizontal (landscape) or vertical (portrait) orientation. For phones and smaller tablets, the portrait orientation tends to be the natural way to hold the device, except for when looking at film or photos, while a larger tablet is seldom held this way. Either design the layouts to work well with both orientations, or design for only one of the orientations and restrict the user interface to this orientation.
- Android devices have different Android versions and manufacturer customisations. Keep this in mind and try to target as broad a scope as possible.

8. Lessons learned

In the following we present some of the most important lessons learned. They are classified according to the software development phases of requirement gathering, app design and implementation and the final phase of app testing.

Requirement analysis:

- During requirement analysis talk to the user group and to their carers. The schedules and needs of persons with Down’s syndrome are different as the one of the general public.
- Persons with Down’s syndrome like to please. Whenever you ask them if they like something or not, their answer will most likely be “yes”.
- If you want to ask the person with Down’s syndrome if she prefers one thing or another you need to be prepared with hands-on mock-ups or nearly ready versions of the software. This is due to the fact that most persons with Down’s syndrome have difficulties grasping abstract things.

App design:

- Even wearing glasses, persons with Down’s syndrome might have very poor vision. Hence text has to be adjustable to the text size settings of the device.
- When you design the app flow don’t assume that persons with Down’s syndrome will know where to navigate next. Let the flow of the app do that for them. Highlight fields where the next interaction is needed.
- Use symbols wherever you can. The symbols might be supported by additional text.
- Usually the content of an app needs to be highly personalisable. Hence, there is a lot of configuration, preparation work load which usually the carer has to do. Please make sure it is as easy as possible for the carer to input this data. Using a dedicated app to create the content has proven successful.
- Do not give negative feedback to the person with Down’s syndrome. Always formulate in a positive way. If the user has done something wrong, help him, suggest to repeat the action.
- Be aware that the frustration limit of persons with Down’s syndrome is very low. This means that for example there should be no long loading times (when nothing happens on the screen). If this happens they will lose focus and interest and it will be very hard for the carer to redirect the attention of the person with Down’s syndrome to the task.

App testing:

- *Pre-testing*: Before testing your app with the target user group (persons with Down’s syndrome), test software first with general population. Testing feedback can be in written form from the test participants.
- *Testing*: After testing with the general population, then intensely test in a small target user group for a short time. Observe the usage closely. Testing results are derived from the observations and from talking to the test participants or to the carer, who can usually explain what additional support would be needed.
- *Piloting*: After testing with a small user group, extend the number of participants of the target user group as well as the testing period. It is very important to assign first tasks which have to be completed with the help of the app.