



Project no.:
AAL-2009-2-137

PeerAssist

**A P2P platform supporting virtual communities to
assist independent living of senior citizens**

Deliverable 3.1 “Methods for capturing user intent”

Lead Participant/Editor	Warp / Rafael Ramos
Authors	Rafael Ramos, Michael Fried, Emilia Cimpian, Yiannis Karras

Table of Contents

1	Introduction.....	1
2	User interaction.....	2
2.1	Interaction modalities.....	2
2.1.1	GUI.....	3
2.1.2	VUI.....	4
2.2	Interaction styles.....	5
2.2.1	Command-line interfaces.....	5
2.2.2	Menus.....	6
2.2.3	Forms.....	7
2.2.4	Question-and-answer interfaces.....	7
2.2.5	Direct manipulation interfaces.....	7
2.2.6	Natural language.....	8
2.2.7	Guidelines.....	8
2.3	Interaction strategies.....	9
2.3.1	Browse.....	9
2.3.2	Find.....	9
2.3.3	Query.....	10
2.3.4	Structured.....	10
2.3.5	Guided.....	10
2.3.6	Guidelines.....	11
3	Query building.....	11
3.1	Formal language.....	12
3.2	Query templates.....	14
4	Sensing user status & context information.....	15
4.1	Indoor sensors.....	15
4.2	Outdoor sensors and systems.....	16

4.3	Healthcare devices.....	16
4.4	Remote control devices.....	16
4.5	Limitations and effectiveness of user devices.....	27
5	Platforms and services.....	18
5.1	Open platforms.....	18
5.2	OSGi platform.....	19
5.3	Voice platforms.....	21
6	Conclusions.....	22
7	References.....	22

1 Introduction

User intent is the task that the user wants to accomplish through the system. It is usually some goal related to human concepts, such as "talk with people" or "consult the doctor", and it constitutes a piece of data that the user knows and the system needs to find out. The process of expressing, receiving and processing data for that purpose is commonly referred to as "capturing the user intent", and it is a fundamental part of Human-Machine Interaction. Usually this data also involves context or status information about the user.

User intent, user status information and status updates for home environments may be detected using various ICT platforms and technologies. Even though standardization of platforms, software and equipment integration is in an early stage, there have been many efforts on providing advanced AAL platforms. The availability of cost effective technologies and components is assumed necessary in order to reach large-scale and is widely agreed that open solutions (and in particular open service platforms) would be a good starting point to foster the development, deployment and operation of applications.

The user intent is explicitly expressed by user actions (using gestures, voice, device operation) providing information regarding user intention or desire to perform a specific action. User status is related to the user's health, mood, activity or social status. User environment includes surrounding environmental conditions (location, time, weather, altitude, light, temperature), building/room conditions (doors status, number of people inside a room), nearby devices condition (working, battery low).

This data (intent, status and environment) may be updated by two mechanisms:

- User based: the user expresses directly an action or context parameters through some device, which is part of a Graphical User Interface or Voice User Interface.
- Sensor based: the context information parameters (user status or environmental conditions) are updated upon sensor events.

The figure below describes how sensors and user input contribute to context monitoring, processing and adaptation processes, as analysed in work package 3.

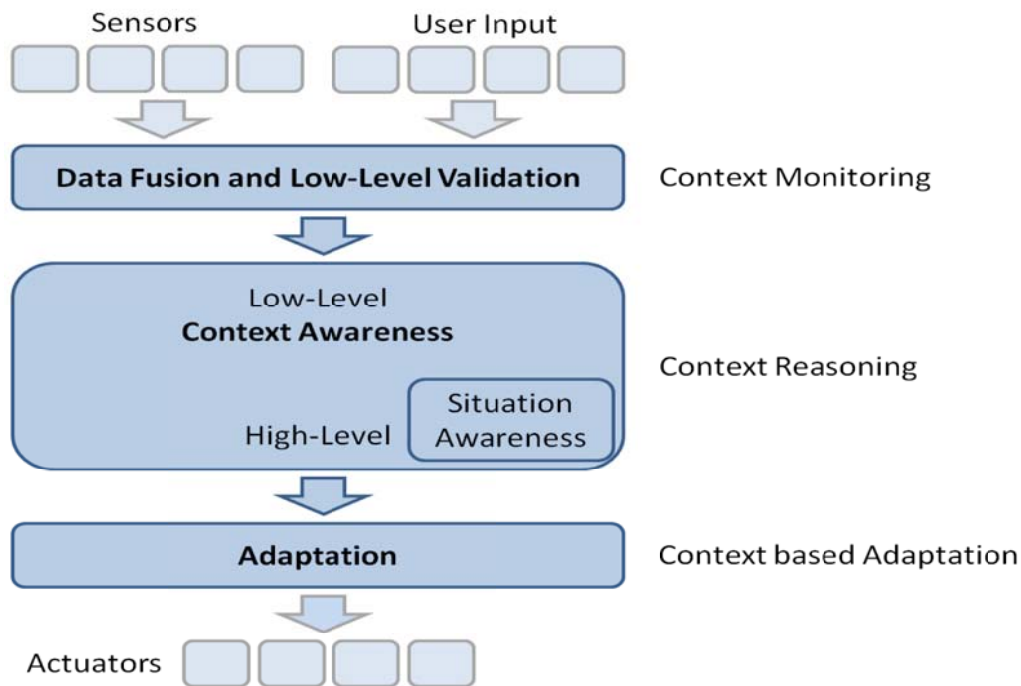


Figure 1: Context awareness

Throughout this document we discuss several issues related to intent capture techniques. Section 2 presents theory about methods for user interaction, and analyzes their suitability for the PeerAssist system in terms of usability and accessibility. Section 3 focuses on the concrete task of query building, which requires special methods to capture the user's target and constraints when searching for items. Section 4 deals with mechanisms and devices to acquire status and context information, providing examples and design considerations. Finally, section 5 presents existing platforms and technologies to support intent capture and context sensing.

2 User interaction

This chapter analyzes issues related to human-computer interaction. It is organized in several layers of the interaction process, arranged from the lowest to the highest level of abstraction: Modalities, Styles and Strategies. Each layer deals with different aspects that must be considered in order to achieve an optimum design for the target users.

Each section presents the available options for each layer, describes their characteristics and discusses their positive and negative points. All the evaluations are made from the user's point of view, rather than from a technical perspective.

2.1 Interaction modalities

Modalities are ways of using a system, concerning the sensory and actuator capabilities the user utilizes to interact with the machine (sight, hearing, etc.). This concept was discussed in D2.1. There is a wide variety of available modalities, but they can be grouped in some basic categories.

For this project, two main groups are considered: Graphical User Interfaces (GUI) and Voice User Interfaces (VUI).

2.1.1 GUI

Graphical User Interfaces are those which present their content on a visual space, to be perceived with the eyes. This takes advantage of the strengths of human sight: high bandwidth (many things can be seen at once) and focalization (direct the gaze to a specific point). These abilities are naturally used by people for communication: read texts, recognize colours and shapes, perceive motion, etc. Therefore, visual interfaces are the most powerful, and the most widely used.

Following is a list of common input devices for graphical UIs.

- **Keyboard:** keyboards have a set of keys to be pressed with the fingers, to enter letters, numbers, and other symbols or commands. It's the primary input device for PCs, along with the mouse. It can be very fast for expert users, but very slow and difficult for untrained users; in spite of that, it's the most effective device to enter arbitrary text.

Other button-based devices include TV remote controls, gamepads or phone keypads. Besides data-entry buttons (numbers, commands), they usually include two-dimensional cursor keys (up, down, right, left); these are used to move a cursor on the screen by discrete leaps, highlighting the actionable GUI elements, as opposed to continuous motion with pointing devices. This technique eliminates accuracy problems, but may be slower.

- **Pointing device:** these devices are used to control the movement of the cursor on the screen. Generally they allow to click (e.g. pressing a button) to order an action related to the cursor's current position (e.g. press a button, select an item). More complex operations may be allowed, depending on the physical properties of the device, like double-clicking or dragging. Some examples of pointing devices are mice, trackpads, trackballs, joysticks or drawing pads.

The mouse is the most common device for desk workstations (PCs). It is accurate, fast and versatile, requires little hand movement, and allows the user to keep the eyes on the screen. However, the need of a flat surface makes it unusable in mobile setups, where it is usually replaced by trackballs or touchscreens. Moreover, it requires a certain degree of eye-hand coordination, which makes it awkward and difficult for untrained users.

- **Touchscreen:** this device allows the user to press the screen where the application is displayed, thus interacting directly with the UI elements. The coordination between eyes and hand is direct, which makes the experience more natural and easier to learn. Besides, it is very suitable for mobile devices, as there are no separate moving parts.

On the negative side, touchscreens are less accurate, so the elements need to be bigger. The finger may obscure the screen, and if it is large, prolonged usage may lead to arm fatigue. For these reasons, touch monitors for desks are not widely accepted (mouse is still dominant), but they may be helpful if the target users are unskilled with the mouse.

The output devices to display graphical UIs are basically screens. Nowadays it is common to find full-color high-resolution displays of all sizes. Here are some of the most usual types of screens:

- **Monitor:** a medium sized screen, suitable to watch at short distance by a single user, as in a desktop PC. It covers a wide area of the field of vision, and is able to show high amounts of fine-grained information such as small font texts and detailed images.
- **TV:** a big screen intended to be watched by several people at a medium distance, normally in a living room. Its main purpose is to display video (e.g. TV shows and films), rather than computer applications, so it does not generally provide as much resolution as a monitor. Hence, a GUI for a TV should be adapted to display larger items and data.
- **Smartphone:** mobile devices carry a small screen, to be watched at short distance by a single person. Usually it is also a touchscreen that acts as the input device, which also affects the interaction design. Here the size and layout of the GUI elements must be carefully designed, probably involving heavy adaptations to enable an effective usage despite the device limitations.

2.1.2 VUI

Voice User Interfaces are based on speech communication. The goal is to provide a more natural way of interacting with computers by making them able to communicate by spoken language, as humans do. This would substantially reduce the amount of training required to use a system. It also enables hands-free operation, which makes the system available in multiple situations (a car, a kitchen), as well as enable usage to physically impaired users.

Regarding data input, the device consists of a microphone along with some speech recognition software. The main drawback is the low reliability, especially under noise conditions. The following are several modes of speech recognition:

- **Discrete word recognition:** identification of individual words, suitable for simple commands.
- **Continuous speech recognition:** identification of words within a stream of speech, it allows more expressivity but is more error prone.
- **Speaker-dependent recognition:** the system recognizes the speech of a specific speaker, who must train the system; it is used for data-intensive applications, such as dictation.
- **Speaker-independent recognition:** the system recognizes any speaker with no prior training, convenient for occasional uses such as automated phone information services.

Concerning data output, we can consider both spoken messages or simple sounds generated by the system.

- **Non-speech sound:** tones, beeps, alarms, etc. They can be used to notify events, such as an incoming message or a system failure. They should be optional for normal use, but in

some special cases they can be highly convenient, e.g. a phone ringtone or an emergency siren.

- **Digitized speech:** digital playback of pre-recorded messages spoken by humans. It sounds more natural, but the output is limited to the recorded words. This is used when a fixed set of messages is required, e.g. a talking clock or a driving assistant.
- **Text-to-speech synthesis:** automatic generation of synthetic voice from text data. It may sound awkward or "robotic", but the voice is generally understandable. It is used when arbitrary data must be spoken aloud, e.g. screen readers.

2.2 *Interaction styles*

Interaction styles are the kinds of actions the user must perform in order to use a system. Whereas modalities are about the physical medium to transfer information, styles deal with the logical tasks that the user is supposed to do as part of the human-computer interaction. These "actions" don't refer to the operations of the application domain (e.g. create a group), but to the manipulation of the UI elements (e.g. click a button, write somewhere).

The style determines both input and output way of communicating. The user must be aware and understand it to be able to use the system properly.

There is a variety of common interaction styles, each with its advantages and drawbacks. Sometimes they can be combined, and usually several alternative styles are suitable for a specific goal. The developer must choose the style that best fits the application, bearing in mind the features it must offer, and most importantly, the intended target users [17].

2.2.1 **Command-line interfaces**

Command-line interfaces consist of a text console. The user is presented with a prompt, where s/he must type commands; when Return is pressed, the system executes the command and shows the output on the next lines. This was the primary interaction style of early computer systems, but it is still common today (e.g. Unix, MS-DOS).

With this style the user must have a clear understanding of the intended operation, and know the available commands to do it. It's difficult for novice users to learn the command language, but once mastered it represents the quickest form of interaction with computers (i.e. with abbreviations, combined commands, scripts, etc.).

Since the commands are entered as sequences of keystrokes of any length, the number of available actions can be huge, as opposed to buttons or menus in a graphical UI. That gives CLI its power and flexibility. Besides, typing is often faster than moving a mouse. On the other hand, it requires a lot of training on the language, it heavily relies on the user's memory, and the rigid syntax makes it prone to mistakes. So, this style is certainly not appropriate for the elderly.

2.2.2 Menus

Menus are lists of options that are presented to the user so s/he can choose one of them. Usually the items are arranged in a hierarchical structure, based on topics, to make them easier to find. These items may be represented as text strings, icons or buttons, and they should be self-explanatory.

Menus may come in different forms:

- **Fixed menus:** they remain in one place until an option is selected.
- **Pull-down menus:** they appear below an element when it is clicked, and it hides back once an option is selected. They are commonly found as menu bars on desktop applications' windows.
- **Pop-up menus:** they appear in the cursor position when the user clicks (or right-clicks) on certain areas of the screen. The offered options are usually related to the clicked item.
- **Cascading menus:** they have sub-menus in a tree structure that appear next to each other, when selecting options at each level. This characteristic may be found on other menu types.

When designing menus, some aspects must be considered:

- Order of items: alphabetical, by category, by frequency...
- Selection of items:
 - Press numbers or letters assigned to options.
 - Point and click the options with a pointing device.
 - Highlight the item through arrow keys.
- Navigation through hierarchical menus:
 - Deep: few alternatives, many levels. It requires more navigation.
 - Broad: many alternatives, few levels. It is faster, but the choices become more complex.

The idea behind menu-based UIs is to save the user from having to remember all the available options, instead s/he must only recognize them in the list. For that purpose, the hierarchy of actions allows the user to browse sub-menus from general topics to concrete operations. Therefore, little memory or training is needed. Besides, the user has higher control because the actionable elements are visible, which also reduces errors.

As possible drawbacks, menus may be too complex to navigate, and item names may be unclear. Also, menus are impractical if the number of choices is high.

In conclusion, this style is appropriate for untrained users, since it is based on seeing and selecting an action. Therefore it will probably be suitable for PeerAssist, as long as menus remain simple, shallow and meaningful.

2.2.3 Forms

Fill-in forms are intended primarily to enter data, rather than performing actions. Forms consist of a set of fields or boxes where the user must write text. Each field has a label or caption that indicates what data must be entered (e.g. name, address...). The fields should be spatially arranged in a meaningful layout.

Forms are in general easy to use, because of the humans' prior knowledge of paper forms. They require little memory or training, as the requested fields are visible. Also, the placement and type of other fields provides context information for better understandability.

On the negative side, it assumes certain knowledge about valid inputs (e.g. phone numbers, postal codes). Forms also require typing skills, and errors can be made. However, these problems are inherent to the need of entering arbitrary textual data.

Concerning the PeerAssist application, forms seem a natural choice when arbitrary data is to be gathered, e.g. editing the user profile or building a query. Nevertheless, some assistance techniques may be applied.

2.2.4 Question-and-answer interfaces

Question-and-answer interfaces are another way of entering data that resembles a human dialog. The user is faced with one single question at a time; then s/he answers it either by typing text or selecting an option. Since the system controls the workflow, the user does not have to worry about navigation.

The advantages of this style are similar to those of fill-in forms: the fields are self-explanatory, and little memory is required. In addition, navigation is easier and the screen is less bloated. The most interesting feature is the ability of arranging questions in a complex dynamic workflow (i.e. branches, loops) where the next steps depend on previous user answers.

The drawbacks are its inefficiency, the loss of control by the user, and the lack of context from forward questions.

Overall, it is an appropriate approach for novice users.

2.2.5 Direct manipulation interfaces

Direct manipulation style is based on the idea of managing visible objects. First the user selects an object in the UI (e.g. an icon, window or text), then s/he selects an action to perform on it (e.g. move, close, underline). The common criteria is to use metaphors or representations of real-world objects in the UI, and make them behave like their physical counterparts. Some examples are the buttons, sliders, tabs, draggable items, etc.

This approach is highly intuitive, since the object reactions are observable and expected. That gives the user a sense of control of the environment. It also avoids the use of abstract command languages, most of the interaction is done by pointing and clicking.

Direct manipulation is the primary interaction style of common desktop GUIs, which use the WIMP model (window, icon, mouse, pull-down menu). For some specific tasks it is, in fact, the only practical approach (e.g. image editing).

This style is good both for novice and expert users, as it enables quick and powerful tasks as well as easy learning of functionalities. It is predictable, controllable, and actions are usually reversible. Besides, the user continuously sees the results of his actions, which helps him to direct his moves towards the intended goal. As a drawback, the pointer device may not be accurate enough to fully control the task. In PeerAssist this style could be applied whenever a non-textual action is required, taking into account the accuracy issues for each device (PC, TV, touchscreen smartphone, etc.).

2.2.6 Natural language

Natural language style aims to allow the user to control the system through messages formulated in natural language, as in human-human communication. This technique may be used either for selecting actions or entering arbitrary data, or both. The user experience should be similar to talking to a human listener.

This goal requires certain degree of artificial intelligence, as the system must be able to handle the vagueness, ambiguity and ungrammatical constructions of natural language. There are currently algorithms and tools for that purpose, but no complete solution exists yet. To overcome this problem, the set of acceptable input messages must be limited to a subset of natural language constructs, defined by a grammar.

Concerning interaction modalities, NL is best suited for speech input, since speaking is easier than writing. Speech recognition technology works at an acceptable level of reliability, but it might not always succeed. In case of entering text by typing, the task is slow, verbose and prone to spelling errors.

Nowadays, NL interaction is not very widely used. It will be considered for application in PeerAssist.

2.2.7 Guidelines

The criteria for choosing styles for the PeerAssist UI must take into account primarily the target users - the elderly. The following guidelines have been identified:

- Avoid the keyboard as much as possible. Use selection rather than typing (no command-line). When arbitrary text must be entered, try to allow speech input.
- When filling text fields it is good to offer suggestions to click (e.g. Google search), which may save the user from typing some text.

- UI should be self-explanatory and easy to learn, so Direct Manipulation is recommended (i.e. buttons, icons).
- Some form of help is appropriate to ease usage and learning to novice users, e.g. an assistant.
- Trained users are not the primary goal, but some facilities for them could be provided, i.e. shortcuts, disable help.

2.3 Interaction strategies

The strategies of interaction are mental models by which users approach a task towards a specific goal. These strategies endorse a set of actions that must be performed by the user to reach his objective. They are related to the user's behaviour rather than the UI features. However, the UI designer must provide certain elements to support these strategies.

This chapter describes strategies for the UI sub-field of Information Retrieval. That focuses on the act of consuming data from some information space (e.g. a database, the Web). Research works have identified five approaches for getting data in a IR system: Browse, Find, Query, Structured, and Guided [18].

2.3.1 Browse

The act of browsing consists of exploring an information space, without a clearly defined goal. The user just wanders through pages hoping to run across some interesting items, usually trying to direct his path towards the most promising zones. Sometimes users have an almost-defined target, and they want to have a look at some items that may fit it.

This searching pattern is well understood by users, as it resembles shopping in physical stores. However, graphical interfaces have difficulties at showing large amounts of data on a screen. For that purpose, they must enable navigation through links or more elaborate browsing methods.

A good Browsing interface design should fulfill some goals: present an "aerial" view of the available information, make it easy for the user to understand and make connections between items, and establish clear paths so the user knows where s/he is and where to go.

2.3.2 Find

In a Find activity the user knows exactly what s/he is looking for. S/he knows the unique identifier of the item (name, model, serial number), and only needs help to locate it within the information space. Then s/he will be presented with additional data and will be able to operate with that item.

The UIs that support Finding usually provide a way to enter the identifier, e.g. in a text box. They typically skip the presentation of a results list, since only one item should match, so this item is presented. An example of such interface would be an API documentation site, where the user would search a specific command or procedure by its name.

Some good practices for designers of Finder UIs are: tolerate misspellings and provide alternative terms; navigate to the target as directly as possible; facilitate repeated searches; and suggest related items that may interest the user.

2.3.3 Query

The Query approach is similar to Find, as both share the goal of reaching an item. The difference is that, in Query, the user does not know the identity of the item. The user has a mental model of an imaginary element that would satisfy his needs. Therefore s/he must enter some search criteria and let the system find the matching items.

The interface to build a query can be complex and confusing if it involves many options or fields. This can be alleviated by several techniques: provide examples of valid entries, hide syntax requirements, offer basic and advanced query interfaces, etc. Another interesting practice is to show the results list along with the query itself, so the user can refine his constraints while visually narrowing the results.

Common guidelines for query interfaces include: assist the user to build the query, allow multiple criteria at once, enable iterative refinement of the query, and clearly display the relevance of results.

2.3.4 Structured

With the Structured approach, the information is arranged by a defined hierarchy (e.g. categories, topics...). The UI can then allow the user to navigate through a series of successive choices, narrowing the search until a satisfactory item is found. The easiest implementation would be a multi-level tree-structured table of contents.

This technique is commonly used for shopping or help applications, where a large set of data can be effectively displayed as a series of small lists of items from which the user makes selections. This can fail, however, if the items may belong to more than one branch, or the logical organization in the UI is different from the user's mental model.

The Structured navigation can also be used to guide the user through the functionalities of a system, if the domain is unfamiliar. Application menus are an example of this strategy. Another interesting use is to have a dynamic structure, where the set of options at each step depends on previous selections. This can also be applied to a query building interface, by displaying dynamic text boxes or suggestions based on previous entered data.

Some important design issues are: present clear and understandable choices, not having empty or too large option lists, and allow easy back-tracking.

2.3.5 Guided

A Guided interface is intended to drive the user through a sequence of steps in a specific order. The navigation options are limited, since the system controls the workflow; that makes it appropriate for inexperienced users who want to be led through the application.

An example of a Guided interface would be the "previous" and "next" buttons on a multi-page content, such as an online book. Another common application is to perform a constrained task, like the ordering process in a e-commerce site, which involves a sequence of forms for item selection, shipping address, payment details, etc. These kind of interfaces are usually short in duration, and limited to a specific sub-task of the user experience, like the mentioned purchase process.

Guided UIs should clearly indicate the current location of the user, break material in properly sized chunks, and allow the user to easily exit or restart.

2.3.6 Guidelines

Typically, interfaces offer more than one of these approaches, as the user will probably change the strategy during the session. For example, the user may start browsing with no luck, and then make a query to try to find an interesting item. In other cases, the operation itself may impose a procedure, like a guided ordering process.

It is generally convenient to provide multiple strategies, and allow the user to easily combine them or alternate between them. Concerning the application in the PeerAssist system, different approaches may be appropriate for each action:

- The Find strategy is useful to look for a specific person by name (e.g. a relative or a friend).
- The Query strategy is the natural choice for most of the use cases: search matching people, create or search communities, search services, etc. The user will express some criteria and the system will find matching elements.
 - This task (build a query) can be enhanced with the Structured technique. Available options for query constraints may be hierarchically arranged (e.g. ontology classes), so this can make easier the selection by having several steps (e.g. choose Service, then MedicalService, then Cardiologist). This method for entering data can be used for filling query templates, as explained in section 3.2.
 - The Guided strategy can also help in building queries. Instead of displaying a baffling bunch of text boxes, the UI can request the user only one constraint at a time, in the Question/Answer style.
- The Guided strategy is also useful for UI navigation, i.e. helping the user to reach the part of the system where the intended action is available. This approach is common on personal assistants.

3 Query building

One important task related to intent capture is the act of building queries. Many use cases involve searching items (users, groups, services...) in a large information space, as in the aforementioned

"Find" and "Query" interaction strategies. The final query constitutes a representation of the user intent, since it contains a more or less accurate definition of the set of items s/he is looking for, in the form of search constraints.

However, the process of expressing these constraints can be difficult for the user. Furthermore, the query formation depends heavily on the underlying search technology (e.g. SQL, SPARQL...), which in turn is determined by the storage and processing tools and languages.

In order to support multiple actions within the PeerAssist project, like the retrieval of services and adaptation of the user interface based on user preferences, it was decided to use a state-of-the-art knowledge base as data storage. Within the Semantic Web, multiple storage containers have evolved over time. As stated in D3.3. OWLIM was chosen as RDF (which is the "language" of the Semantic Web) triple store, since it is one of the most common, performant solutions. However data naturally needs not only to be stored, but moreover to be queried in a reliable, meaningful way, which should also allow easy query formulation from the user interface.

This chapter will introduce the query technology that will be applied within the scope of the PeerAssist project. Then, query templates will be presented as the method to build queries enabling easy user interaction.

3.1 Formal language

The Resource Description Framework (RDF) [1] is one of the technological backbones of the Semantic Web. Within RDF knowledge is represented as resources connected with relationships, which means that a statement always consists of a tuple with a subject and object resource, that are related through a predicate. These so called triples <subject, predicate, object> can be expressed in various syntaxes. With evolving ontology languages like RDF(S) and OWL, which will be discussed in D 3.3., the question arises how to query data available in RDF format.

Analogous to relational databases, W3C specified an SQL like query language standard for Semantic Web data. The resulting SPARQL Protocol and RDF Query Language (or short SPARQL) [2] became an official W3C recommendation in the beginning of 2008. It offers the possibility to create complex queries on RDF data. Knowledge in RDF is represented as directed graphs of entities. Searching information in these graphed is based upon so called graph patterns. Four major types of queries have been standardized, which slightly differ in syntax and result set:

1. Select

This type is used to extract raw values from an RDF data set. The results are returned in a table format containing variable binding names and values.

2. Construct

Not only raw values are extracted, but rather complete information. Results are transformed into valid RDF, consisting of a graph with triples.

3. Ask

In order to put simple true/false questions, Ask returns a Boolean value. A positively answered Ask query can be easily transformed into a Select query to receive actual result values for the query.

4. Describe

This type of query returns a single result RDF graph containing RDF data about resources. It will however not be used within PeerAssist since Select, Ask and Construct are sufficient enough in order to express all possible queries that can occur within the project.

There is a multitude of other RDF query languages available. Those include DQL (queries and results expressed in DAML+OIL a predecessor of OWL), N3QL (based on [Notation 3](#)), R-DEVICE, RDFQ (XML-based), RDQ ([SQL](#)-like), RDQL (SQL-like), RQL/RVL (SQL-like), Versa (non-SQL-like compact syntax), XUL (matching with [template](#) elements) and Adenine. Most notably SeRQL, which is pretty similar to SPARQL, is the standard query language in Sesame (which builds the foundation of the OWLIM store used in PeerAssist). We however chose to use SPARQL, as it is a W3C standard, well adopted and most commonly used within the research community.

SPARQL is a very powerful query language. However it is still being intensely developed and the current recommendation has some major limitations, especially when compared to classical SQL. The most obvious one is the missing functionality to delete or update RDF data. There are some approaches like SPARQL/Update [3], which is a part of the current W3C SPARQL 1.1 working draft, but has not been standardized by the consortium so far. Support of negation is also missing, except for some framework specific SPARQL implementations like Virtuoso's SPARQL-BI. Aggregates like min, max and average are not present in the W3C recommendation, but have been implemented in several projects as SPARQL extensions. However, the current Big OWLIM distribution already supports SPARQL 1.1 Update. So we can potentially use the functionality within the PeerAssist project. SPARQL 1.1 Update introduces methods to:

- INSERT data
- DELETE data
- LOAD data
- CLEAR data
- CREATE graphs
- DROP graphs
- COPY graphs
- MOVE graphs
- ADD graphs

A sparql query could for example look as follows:


```

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person a foaf:Person.
  ?person foaf:name ?name.
  ?person foaf:mbox ?email.
  ?person foaf:interest <http://dbpedia.org/resource/The\_Beatles>.
}

```

The PREFIX clause defines namespaces, foaf which is the friend of a friend vocabulary in this case. Name and email of a person will be returned as stated in the SELECT clause. At last the WHERE part restricts the results to return only name and email of persons who are interested in “The Beatles” described by a DBpedia, which is the RDF representation of Wikipedia version, link.

The purpose of SPARQL within the PeerAssist project is to pose queries in a formal way to the knowledge base. These queries are constructed using input form the user interface level of the application and query templates that will be explained in the following section.

3.2 Query templates

A SPARQL query template is very similar to a SPARQL query, except that it has ??vars which need to be replaced by values from the form – in this case values obtained from the user interface. In the example presented in section 3.1, the value represents the interest of a user – i.e., a certain peer wants to find other peers with a certain interest. In this case the query template is defined as follows:

```

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person a foaf:Person.
  ?person foaf:name ?name.
  ?person foaf:mbox ?email.
  ?person foaf:interest ??interest
}

```

The user interface should be adapted to allow a peer first to select the search criteria, and to input the search term. It should also allow a peer to select what information it wants to see about the peers sharing the same interest, in this case name and email address.

The query templates developed for capturing this kind of user intent have a two-fold purpose. Firstly and most important, they have to take into consideration the usability perspective, i.e. the abilities of the users. The templates are meant to facilitate elders and non-computer experts to interact with the system and to pose queries. Secondly, the templates must be developed taking into consideration the functionality aspect: they should support the users to unambiguously express their queries, such that the platform would return appropriate and precise results.

The templates need to encapsulate all types of queries that are potentially posed by the users, in the same time reflecting the internal ontological structure of the repository. At this stage in the project the exact templates cannot be yet created. This activity will be resumed once the ontology reaches version 1.0, when the specific template files will be created.

4 Sensing user status & context information

This section analyses the available mechanisms, components and devices that may be used directly or transparently by the user in order to provide information about the user's status, environmental conditions and other user context information. Monitoring the user status and context information enables context awareness for the mechanisms used for user interaction (section 2) and services, fulfilling the requirements imposed in Figure 1: Context awareness. While section 2 describes how the user may physically manage devices in order to use systems, this section analyses the available mechanisms and systems that may capture the user status or information about its surrounding environment.

As described in the next paragraphs, sensing user status and context information may be enabled with either indoor devices (sensing user related information inside the house), outdoor devices (sensing user information outside the house), healthcare devices (more complex and critical indoor or even outdoor devices sensing information about the user's health) and remote control devices providing the user a direct mechanism to provide information about his status. The limitation and effectiveness of the devices above may however be hindered by the environmental characteristics and especially by the user's abilities to operate them; special considerations for the elderly users are introduced in the last paragraph.

4.1 *Indoor sensors*

Indoor sensors provide information on the environment of the user and consequently for the user. The devices are usually transparent to the users, requiring minimal maintenance or operations on them. Such devices are motion sensors, door sensors, bed sensors, appliance sensors, cameras, door access cards, comfort and safety sensors.

- Motion sensors: may be used to detect if someone is inside the house or in a specific room, thus providing location information. It may also be used to get the last time someone has been in the house/room. Motion detection is significant for ubiquitous computing, since many actions may be linked to presence in a specific area, and options for several services may be narrowed down.
- Door sensors: may be used to detect if a door is opened or closed, as well as the event of opening or closing a door. For example, opening the entrance door with no presence outside the entrance door may show that the user intend to leave the house, so a reminder to get the keys or that another door is open would be useful.
- Bed sensors: simple magnetic sensors like the door sensors, indicating if a user has lied on the bed or not. Even if installation is tricky it is very useful especially for elderly people.
- Appliance sensors: refrigerator sensors notifying for open door or oven sensors for heating cycle are useful information. However such sensors may only be installed externally since appliance manufacturers do not support such features commercially.

- Cameras: may be used by relatives to check on their parents leaving alone, especially when they do not answer the phone or when an alarm of any kind has been triggered. Even though cameras raise some privacy issues they have been proved really helpful.
- Door access cards: may be used to provide security as well the deterministic knowledge that someone has entered the house.
- Comfort sensors (temperature and illumination sensors): may provide information on the proper setting of the user's environment.
- Safety sensors (flood and smoke sensors) may identify cases of safety risk and provide information to authorised people.

4.2 Outdoor sensors and systems

Outdoor systems and sensors are devices that may be used to sense information about the user when the user is outside the house.

- Location systems (GPS): global positioning systems and geographical aware services may help capturing user intention (e.g. following the road to the supermarket) or help the user with guidelines to get back home if lost.
- Acceleration sensors: could be used to detect falling and send an alert if the generated alarm is not canceled by the user.

4.3 Healthcare devices

Advances in healthcare equipment provide connectivity over home networks (especially Bluetooth and Wi-fi). So it not uncommon in our days to have blood pressure or oxygen saturation measurements stored automatically on a PC or server and providing mechanisms for remote access. Doctors and authorised individuals may thus have access to health status information.

4.4 Remote control devices

Remote control devices are simple devices like a one button remote control, not using a graphical or voice user interface, but are able to perform simple functions associated with services provided to the user. For example:

- Panic buttons: may be used by the elderly themselves in order to call for help when they are not able to speak loud (heath problem or accident) but still need immediate help. The use of such device would in most cases be delivered with a professional monitoring service.
- Alarm controls: may be used by the elderly for turning on or off a security or safety alarm
- Home status (sleeping, away): may be used to set the status of the home, thus enabling the change of certain parameters (turn all lights off).

4.5 *Limitations and effectiveness of user devices*

The range and complexity of devices used in the home are increasing. Devices have to be used in an environment that affects devices operation: a) Space, obstacles, floor surfaces b) Lighting and noise levels c) Temperature and humidity d) Activity level and nature e) Cleanliness f) Other residents (including animals & vermin) g) Electromagnetic interference h) Electric power and back-up sources i) Device portability j) Device appearance (especially discreetness) k) Battery life l) Device durability and ruggedness. The parameters above should be taken into account for every device integrated in a home environment.

Furthermore, effective use of devices is directly depended by the user's abilities to operate them. These user abilities include the following [16]:

- ✓ Physical abilities: Size, strength, stamina; dexterity, flexibility, coordination
- ✓ Sensory abilities: Vision, hearing, tactile sensitivity
- ✓ Cognitive abilities: Memory, literacy and language skills, knowledge/experience
- ✓ General health
- ✓ Mental and emotional state
- ✓ Ability and willingness to learn and adapt

Particularly for elderly people, due to conditions for which they are receiving care, users may have:

- x Reduced physical strength or stamina
- x Diminished visual or hearing capabilities
- x Impaired cognitive abilities
- x Combinations of these conditions

Design of services and devices for elderly people should take into account the above possible cases. In more detail, the figure below presents the hierarchy of ergonomics and hedonomics in a house (Hancock, Pepe & Murphy, 2005). Safety is number one concern for every device operation, while functionality is the next important. After ensuring that the device is functional as required, the device usability shall be considered, as well as the experience it provides.

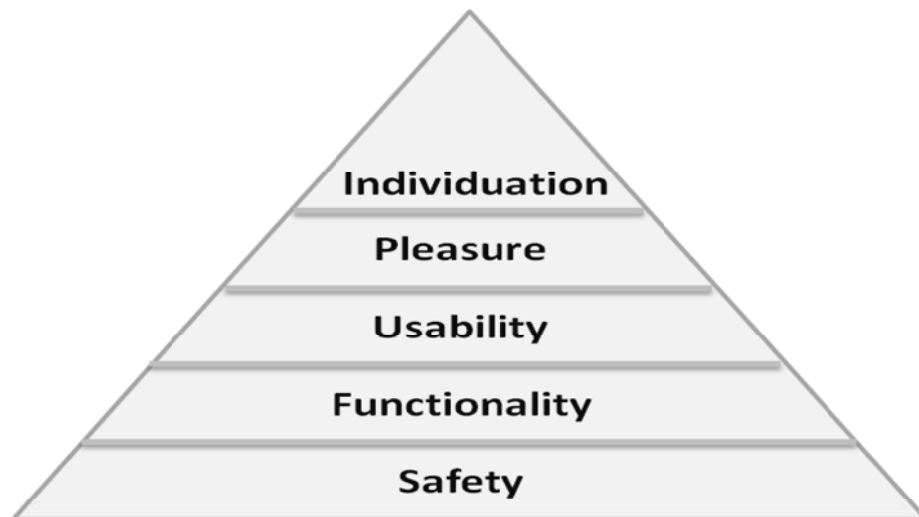


Figure 2: Hierarchy of ergonomics and hedonomics in a house

Universal Design approaches shall be considered in case of people receiving care. Universal design refers to broad-spectrum ideas meant to produce products and environments that are inherently accessible to both people without disabilities and people with disabilities. The principles of universal design are: Equitable Use, Flexibility in Use, Simple and Intuitive Use, Perceptible Information, Tolerance for Error, Low Physical Effort, and last Size and Space for Approach and Use.

5 Platforms and services

The chapters above have described the methods for capturing user intent and sensing context information in a way that enables a holistic approach towards discovering the actual user intent. It has also been described how specific domestic devices and peripherals may enable such monitoring. This section describes the need for a home platform connected with all peripherals, coordinating all user actions and services, capturing user intent and status.

5.1 Open platforms

Ambient Assisted Living (AAL) refers to ICT solutions that will *help elderly individuals to improve their quality of life, stay healthier, live independently for longer* [5]. This assumes the availability of cost effective technologies and it is widely agreed that open solutions and in particular open service platforms would be a good starting point to foster the development, deployment, and operation of applications. But ICT solutions become increasingly complex and it is very difficult to identify precisely what should be open as well as what should be standardised.[15]

The MonAMI IST project [6] addressed the problem and reached the conclusion that we should start with one single interface, the one that separates applications from the underlying platform. The definition of MonAMI approach was based on one criterion, namely to find the features that will make the business environment for developing AAL services simpler, which would in turn foster the creation of a business ecosystem. By business environment, we refer to the community

of stakeholders involved in the development and operations of AAL services, as well as the decision makers. Two features are subsequently identified:

- First of all, the *use of a home platform to run services*. By focusing on a home platform, the business environment for developing services is made simpler compared to approaches where a service is deployed at the web server level, or even worse when the service consist of pieces that need to be deployed both at the web server level or at the home device level.
- Secondly, the *use of standard interfaces in the home platform to create a separation between services and the rest* (e.g. web server and home devices). As a result, the business environment for developing services is made simpler. Programmers of services do not have to adapt their code to specific devices and technologies.

Several initiatives specific to AAL have been launched to address the issue of creating open service platforms. Soprano [9] is an on-going FP6 project. It is using an OSGi-based home platform as well as a sensor/actuators ontology and a context ontology. Persona [10] is an on-going FP6 project. It also uses OSGi technology. The service/platform interface is also based on ontologies, reusing contribution from a previous project. MPower [11] is a recently completed project. MPower is service oriented architecture (SOA) based. UniversAAL [12] is a recently started FP7 project which intends to consolidate current results and to promote open platforms. OASIS [13] is an FP7 IP project investigating the use of ontologies at all levels and is OSGi based. Finally I2Home [8] is a FP6 project which has developed an open accessibility framework based on the URC specification [7]. All the mentioned initiatives have worked on the definition of the service/platform interface. There is no discussion so far on the definition of a common overarching interface, even if this would enforce a clear separation between stakeholders developing applications and stakeholders developing platform elements.

5.2 OSGi platform

OSGi [14] is an initiative launched in 1999 by IBM, Ericsson, EDF and other organisations for a Java-based service platform that can be remotely managed. OSGi is a mainstream technology. The key reasons for proposing OSGi as the PeerAssist home platform are summarized below:

- **Reduced Complexity** - Developing with OSGi technology means developing bundles: the OSGi components. Bundles are modules. They hide their internals from other bundles and communicate through well defined services. Hiding internals means more freedom to change later. This not only reduces the number of bugs, it also makes bundles simpler to develop because correctly sized bundles implement a piece of functionality through well defined interfaces.
- **Reuse** - The OSGi component model makes it very easy to use many third party components in an application. An increasing number of open source projects provide their JARs ready made for OSGi. However, commercial libraries are also becoming available as ready made bundles.

- **Easy Deployment** - The OSGi technology is not just a standard for components. It also specifies how components are installed and managed. This API has been used by many bundles to provide a management agent. This management agent can be as simple as a command shell, a TR-69 management protocol driver, OMA DM protocol driver, a cloud computing interface for Amazon's EC2, or an IBM Tivoli management system. The standardized management API makes it very easy to integrate OSGi technology in existing and future systems.
- **Dynamic Updates** - The OSGi component model is a dynamic model. Bundles can be installed, started, stopped, updated, and uninstalled without bringing down the whole system. Many Java developers do not believe this can be done reliably and therefore initially do not use this in production. However, after using this in development for some time, most start to realize that it actually works and significantly reduces deployment times.
- **Adaptive** - The OSGi component model is designed from the ground up to allow the mixing and matching of components. This requires that the dependencies of components need to be specified and it requires components to live in an environment where their optional dependencies are not always available. The OSGi service registry is a dynamic registry where bundles can register, get, and listen to services. This dynamic service model allows bundles to find out what capabilities are available on the system and adapt the functionality they can provide. This makes code more flexible and resilient to changes.
- **Simple** - The OSGi API is surprisingly simple. The core API is only one package and less than 30 classes/interfaces. This core API is sufficient to write bundles, install them, start, stop, update, and uninstall them and includes all listener and security classes.
- **Secure** - Java has a very powerful fine grained security model at the bottom but it has turned out very hard to configure in practice. The result is that most secure Java applications are running with a binary choice: no security or very limited capabilities. The OSGi security model leverages the fine grained security model but improves the usability (as well as hardening the original model) by having the bundle developer specify the requested security details in an easily audited form while the operator of the environment remains fully in charge. Overall, OSGi likely provides a secure and usable application environment.
- **Runs Everywhere** – Well, that depends. The original goal of Java was to run anywhere. Obviously, it is not possible to run all code everywhere because the capabilities of the Java VMs differ. A VM in a mobile phone will likely not support the same libraries as an IBM mainframe running a banking application. There are two issues to take care of. First, the OSGi APIs should not use classes that are not available on all environments. Second, a bundle should not start if it contains code that is not available in the execution environment. Both of these issues have been taking care of in the OSGi specifications.
- **Widely Used** - The OSGi specifications started out in the embedded home automation market but so far OSGi has been extensively used in many industries: automotive, mobile

telephony, industrial automation, gateways & routers, private branch exchanges, fixed line telephony, and many more. Since 2003, the highly popular Eclipse Integrated Development Environment runs on OSGi technology and provides extensive support for bundle development. In the last few years, OSGi has been taken up by the enterprise developers. Eclipse developers discovered the power of OSGi technology but also the Spring Framework helped popularize this technology by creating a specific extension for OSGi. Today, you can find OSGi technology at the foundation of IBM Websphere, SpringSource Application Server, Oracle (formerly BEA) Weblogic, Sun's GlassFish, and Redhat's JBoss.

For the reasons above it has been proposed and accepted to use an OSGi home platform along with clear interfaces for internal or external services for the PeerAssist project.

5.3 Voice platforms

Elderly people and people with disabilities can benefit from voice interfacing and speech recognition programs. Speech recognition is useful for people who have difficulty using their hands, ranging from mild repetitive stress injuries to involved disabilities that preclude using conventional computer input devices. Individuals with learning disabilities who have problems with thought-to-paper communication (essentially they think of an idea but it is processed incorrectly causing it to end up differently on paper) can benefit from the speech recognition. Speech recognition is also very useful for individuals that are deaf or have hearing problems; it may be used to automatically generate a closed-captioning of conversations.

For the above categories of users, voice interfacing and platforms should integrate speech synthesis and speech recognition technologies for serving their application needs: home devices and appliances control and monitoring, search of information, simple data entry (e.g., entering a credit card number), voice dialing (e.g., "call home") and call routing (e.g., "I would like to make a collect call"), speech-to-text processing (e.g., word processors or emails), text to speech applications e.g. providing information to the user or asking the user for more details.

The most important requirements for speech recognition and synthesis that a voice platform should cover are:

- **Scalability:** allows the deployment of systems from a few simultaneous ports to many thousands, guaranteeing fault tolerance conditions and outstanding reliability.
- **Cost-effectiveness:** should avoid the need for expensive dedicated hardware.
- **Standards-based:** developing services based standard W3C XML languages (VoiceXML, CCXML), does not require any particular skills in terms of architecture or proprietary programming
- **Efficient network integration:** possible to interface to VoIP and SIP/RTP, enabling access to traditional voice networks via simple and cost-effective gateways

6 Conclusions

The research carried out on this task has identified several ways to approach the problem of capturing user intent and context. This has delivered some useful insights about the appropriateness of the different high-level models to address this feature.

Some intent capture techniques try to figure out what the user wants, based on some partial data the user enters, e.g. the keywords in a text box of a search engine such as Google. Then the system would apply probabilistic methods to this data, combined with other knowledge sources, to finally guess the user intention - what s/he is looking for - and deliver better results.

However, in the PeerAssist platform we can use a better approach for intent capture. The user's input is not limited to a few words as in Google, but a richer UI can be provided, so this greater expressivity can be exploited by the system. The most appropriate technique seems to be to proactively request the user information about his/her goals. By applying several assistance methods the system will ask the right questions so the user can fully express his/her desired actions (e.g. searching groups) and all necessary input data (e.g. search criteria). This procedure is less error-prone than probability-based guessing methods. In addition, the successful progress of the conversation is a positive feedback for the user, who at the same time learns about the application processes and the data he/she must provide.

On top of this underlying concept, concrete interaction techniques and devices must be used to implement a functional UI with an assistance approach. As discussed in this document, the most convenient interaction modalities, styles and strategies will be selected for an optimal user experience (concrete guidelines are given on each chapter). Regarding query building, an assisted template-filling process will be offered.

As for the acquisition of status and context data, a range of devices can be installed to increase the system's awareness of the user's situation. On the technical side, it has been discussed the need for a home platform that coordinates all sensor and user devices to provide a holistic solution for system interaction.

These ideas will be applied to design a Personal Assistant (see D3.2) that will help the user to express his/her intent and manage all the system functionalities.

7 References

- [1] Manola, F. & Miller, E. (Ed.) (2004). *RDF Primer*. W3C Recommendation, February 2004.
- [2] SPARQL language specification, <http://www.w3.org/sparql>, accessed 24.05.2011
- [3] SPARQL/Update, <http://openjena.org/~afs/SPARQL-Update/SPARQL-Update-v4.html>, accessed 26.05.2011

- [4] B. Morales and E. Christopoulos: *Peer Assist use scenarios definition 2*, PeerAssist D2.2, 2011.
- [5] AAliance roadmap, <http://www.aaliance.eu/public/documents/aaliance-roadmap/aaliance-aal-roadmap.pdf>.
- [6] MonAMI IST project. <http://www.monami.info>.
- [7] Universal Remote Console. <http://myurc.org/>.
- [8] I2Home IST project. <http://www.i2home.org>.
- [9] Soprano ISS project. www.soprano-ip.org.
- [10] Persona IST project. <http://www.aal-persona.org>.
- [11] MPower IST project. <http://www.sintef.no/Projectweb/MPOWER>.
- [12] UniversAAL IST project.
<http://www.sintef.no/project/Tradlospasient/091105%20Workshop/universAAL.pdf>.
- [13] Oasis IST project. <http://www.oasis-project.eu>.
- [14] OSGi <http://www.osgi.org>.
- [15] Alliance for an AAL Open Service Platform, Antonio Kung and Gunnar Fagerberg , AALIANCE conference - Malaga, Spain - 11 and 12 March 2010
- [16] Medical Devices in Home Healthcare, Molly Follette Story, PhD, Human Spectrum Design, LLC, October 1, 2009
- [17] F. Kurfess, "Interaction Styles", lecture notes for "CPE/CSC 486 Human-Computer Interaction Theory and Design" course at Cal Poly,
<http://users.csc.calpoly.edu/~fkurfess/Courses/486/S06/Slides/03-Interaction-Styles.ppt>, accessed 09/05/2011
- [18] W. Quesenbery, "On Beyond Help: User assistance and the user interface",
<http://www.wqusability.com/articles/on-beyond-help.html>, accessed 09/05/2011