EASYREACH is a Project of the AAL Program (Call 2009-2)

FOSTERING SOCIAL INTERACTION OF HOME-BOUND AND *LESS EDUCATED ELDERLY PEOPLE*

Complete Agent Implementation and Documentation

Deliverable No.		**D3.3**	
Work Package No.	**WP3**	Workpackage Title	**Interaction Support Agent Prototype**
Authors (per company, if more than one company provide it together)		**Orkunt Sabuncu (UNIPOTS)** **Riccardo Rasconi (CNR)**	
Status (F: final; D: draft; RD: revised draft):		**RD**	
File Name:		**EASYREACH Deliverable D3.3_v1.5.doc**	
Project start date and duration		**01 November 2010, 28 Months**	

DOCUMENT HISTORY

Version	Date	Comments	Author
1.0	07/08/13	Draft: Document creation	Orkunt Sabuncu
1.1	24/08/13	Draft	Orkunt Sabuncu
1.2	13/09/13	Draft	Riccardo Rasconi
1.3	13/09/13	Revised Draft	Orkunt Sabuncu
1.4	15/09/13	Revised Draft	Riccardo Rasconi
1.5	16/09/13	Revised Draft	Orkunt Sabuncu

List of abbreviations

ASP	Answer Set Programming
SAT	Satisfiability Checking
KRR	Knowledge Representation and Reasoning
API	Application Programming Interface
STB	Set-Top-Box
IR	Infrared
RFI	Radio Frequency Identification
ICT	Information and Communication Technology
IT	Information Technology
AI	Artificial Intelligence
DB	Database
PA	Personal Assistant
GOAC	Goal Oriented Autonomous Controller
T-REX	Teleo-Reactive Executive
OMPS	Open Multi-Components Planning and Scheduling
P&S	Planning and Scheduling

Table of Contents

EXECUTIVE SUMMARY

This document aims at presenting the complete implementation and documentation of the interaction agent of the EasyReach system. The agent, which is called Personal Assistant (PA), suggests new interactions to the user by considering his profile, history of interactions he has performed within the EasyReach network. Moreover, the agent can provide reminder feedbacks and can provide an agenda functionality. In general, PA constitutes the intelligent component of the whole system.

The suggestions are in the form of new people or groups shown in the EasyReach client. In this way EasyReach system fosters new social interactions for the elderly user. This implicit suggestion method is less confusing and annoying for the elderly than asking explicitly each time and making him feel forced to interact. These items (people or groups) are selected by the PA in a way that these items have common properties with the user. For instance, the selected items may share common interests or they may have occurred in previous communications/interactions with the user.

The main connections among the different components/applications/services considered in the EasyReach architecture and the Personal Assistant (PA) module will be presented and discussed. In particular, the PA's reasoning engine for selecting items is based on Answer Set Programming (ASP). For reminding notifications and feedback, PA uses timeline based reasoning.

1. Introduction

The Personal Assistant (PA) is the software component that suggests new interactions through the social network, and is integrated as part of the EasyReach client (see Figure 1 for the architecture of the whole EasyReach system). The PA not only analyses the profile of a user, but also monitors his activities and interactions within the social network, reasoning on these data to suggest new interactions. Basically, the PA suggests which items should be shown in the user's lists, where an item can be a person or a group; the suggestion may lead to a new interaction depending on the user's will to engage in an interaction regarding the selected person/group. A person who is not followed by the user might be shown in the contact list because they have common interests. Similar reasoning can also apply to a group the user is not a member of. The user can check the suggested items and can send the person a message or check the group message board. This implicit suggestion method is less confusing and annoying for the elderly than asking explicitly each time and making him feel forced to interact. Moreover, the PA is capable of sending feedback and notifications to the user. Since these notifications are time critical (consider that it gets a new message from a person), it uses timeline-based reasoning to correctly notify the user. It also supports the feature of agenda, where the user can enter reminders.

The PA module communicates with 2 modules through their respective interfaces: the presentation layer and the social engine DB (see the architectural diagram in Figure 1). Note that the interface for the social DB features a one way communication (i.e., the PA fetches data from the social engine to monitor interactions of the user), while the interface between the PA and the presentation layer manages two way communication. For instance, the presentation layer can invoke the PA when it needs to show the user's list items, and the selected items are returned to the representation layer. Additionally, the interface is designed so that the PA can get updates related to activities and interactions of the user within the social network; in this way some of inefficient polling of the social engine database are avoided. The detailed information about the architecture of PA is explained in the next section.

We implemented a framework for the PA to suggest new interactions. Answer Set Programming (ASP [1]), a popular declarative problem solving approach in the field of knowledge representation and reasoning, is utilized to implement the reasoning capabilities needed by the PA to intelligently select items to be shown in the user's lists. In this work we use tools from the Potassco answer set collection, namely, *gringo* for grounding and *clasp* for solving answer set programs.

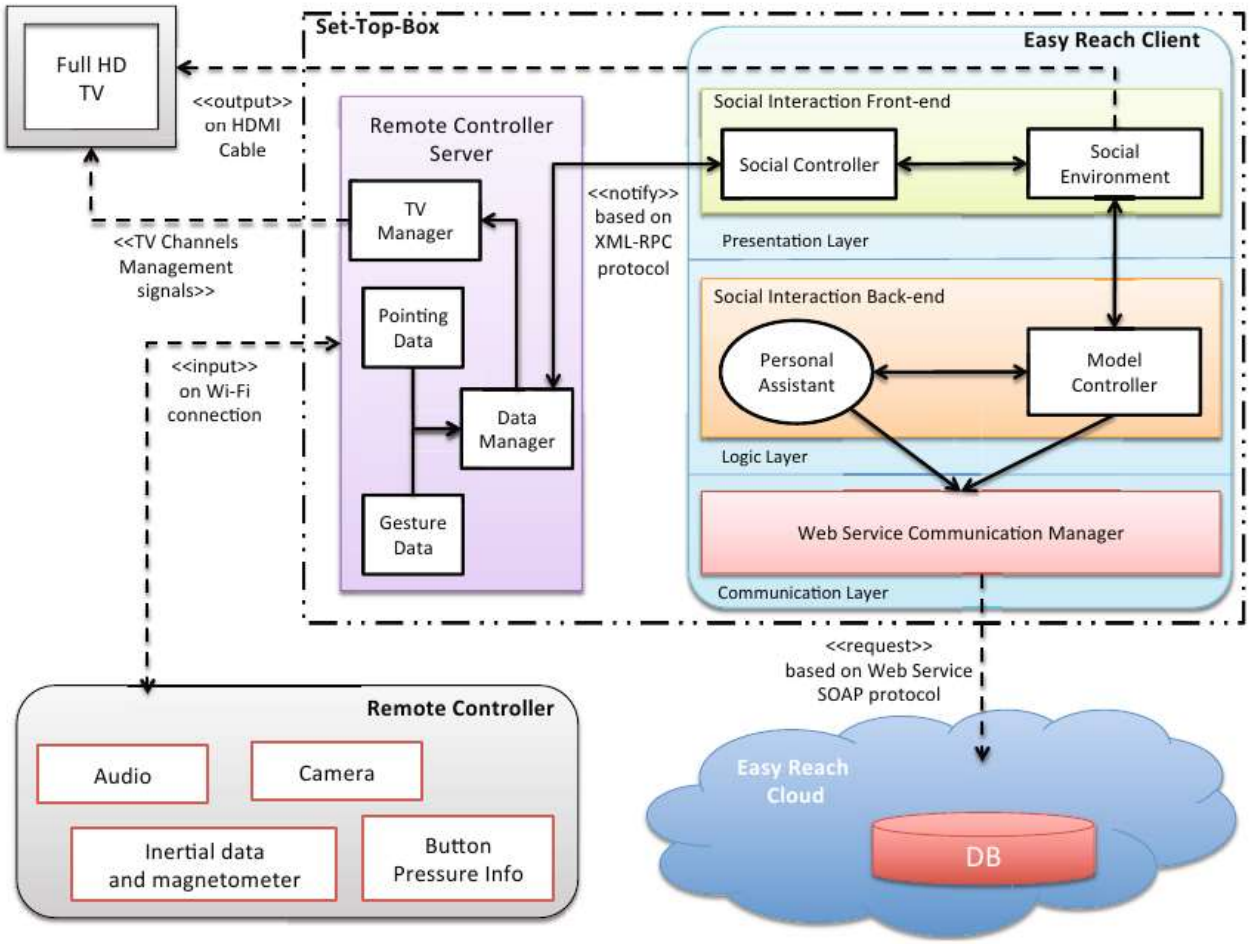

Figure 1: The EasyReach architecture

2. General Architecture of the Personal Assistant

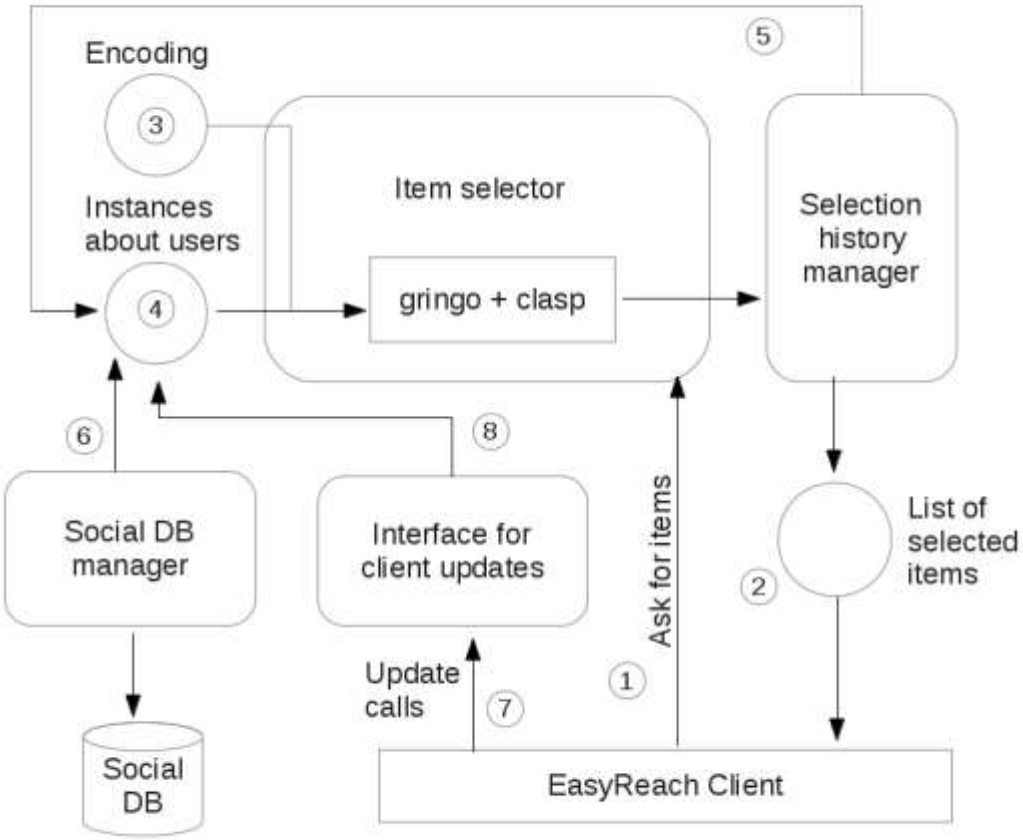

Figure 2: The general architecture of the personal assistant

The personal assistant (PA) suggests items (people and groups) to the EasyReach client. It is composed of four main modules; *Item selector*, *Selection history manager*, *Social DB manager*, and *Interface for client updates*. Figure 2 depicts these modules within the main architecture of PA.

The suggestion process of PA starts with a call of the EasyReach client. The client may anytime ask for people or groups to be suggested for the current user of the STB (marked by 1 in Figure 2). The module responsible from selecting items for selection is the Item selector. The selected items are returned back to the client (marked by 2).

The Item selector uses Answer Set Programming (ASP) for suggesting items. The logic program is composed of a general encoding of selecting items using weights (marked by 3). Instance files (marked by 4) are needed for each user using the STB. These instance files hold data about the user, for instance, his friends, registered groups or his profile.

The logic program composed of the encoding and instances is fed to gringo for grounding and then to clasp for solving. Gringo is a grounder for ASP while clasp is an answer set solver. Both tools are from the Potassco toolset for ASP developed by University of Potsdam [3].

The found answer set represents the people and groups selected for suggestion. The Item selector does not directly return this answer set to the client. Instead it is processed by the Selection history manager, which is responsible for maintaining a history for selected items to be utilized for performing fair selections. Unfair suggestions formed by selecting the same person or group all the time should be avoided. Note that one of the main objectives of EasyReach is to improve social life of the user. New suggestions increase the chance of fulfilling this objective. The Selection history manager saves the current selection as an instance file (marked by 5). At the next call to Item selector this history is considered since the encoding applies some penalty to previously selected items. In this way, it is possible that some items with not high weight values can have chance to be selected. The history has also a capacity (e.g., holding only last 10 suggestions). In this way, a previously selected item may later on be selected again.

The Social DB manager module is responsible from populating instance files (marked by 6) for the current user by connecting to the central social DB. When a user logins to the STB, this module is called to update data in instance files related to the user. These data include people the user follows, his groups, and profile by means of his interests. Additionally, after analysing the user's interests within the taxonomy of interests, other people who the user might be interested for contacting and groups which the user might be interested in registering are fetched from the central social DB.

Consider that a new group about some interest topics similar to the ones of the current user have been created in the EasyReach network. This is definitely a potential for suggestion. In order to make such recent changes in the EasyReach social network available to the Item selector, Social DB manager updates instance files periodically.

The updates in the social network are important for better and complete suggestions. Although the Social DB Manager periodically fetches recent updates, its period is not too frequent considering the costly operation of connecting and fetching data from the central Social DB. Some updates, however, are more crucial for suggestions and should be made available to Item selector quickly. Actions performed by the current active user of the STB cause such updates in the social network.

For instance, when the user sends a message to another person, it causes an update which should be considered fast enough so that the user gets better suggestions. The Interface for client updates module's task is to utilize such kind of updates just in time when they occur. It provides several interface procedures, which are called by the EasyReach client when an action is performed by the user (marked by 7). The Interface for client update module writes these updates to an instance file (marked by 8). In this way some of inefficient polling of the social DB are avoided. As a result, at the next call of Item selector these updates are considered without the need for waiting the next update of the Social DB manager module.

3. Item Selection Framework Using Answer Set Programming

We implemented a framework for the PA to select items for suggesting to the EasyReach user. There can be several reasons for the PA to suggest a user or a group. The framework analyses these reasons and chooses the most viable set of items. It is formally encoded in Answer Set Programming (ASP). All reasons have an associated weight. The encoding enumerates possible reasons for selecting items and uses their weights to cast the suggestion problem as a quantitative optimization problem. The answer set solver tries to maximize the total weight of a selected set of items (we have already used a similar approach to implement suggesting interactions related to events in the social network is [2]). Here we list the main parts of the logic program encoding which is responsible from suggesting items.

The possible reasons for selecting an item are interests mentioned in a user's or group's profile, whether the user follows the person or not, whether the user is registered to the group or not, whether a group is created by a person who the user follows and recent communication between the user and a user or a group. The interests are actually nodes from a taxonomy of interest keywords. How a taxonomy is used to reason on interests and suggestions will be explained in the next section. The following shows ASP facts encoding the reasons with fixed weights.

```
weight(friendship,1). % The user follows the person
weight(regisgroup,1). % The user is registered to the group
weight(friendowns,2). % The group is created by a friend of the
user

Recent communication is represented by the comm/4 predicate.

% user 100 (id) has sent a msg to user 201 at 15:00:01 on
19/12/2012
comm(sendmsg,100,201,"20121219150001").

% user 100 has received a msg from user 103 at 13:42:42 on
22/5/2013
comm(recvmsg,100,103,"20130522134242").
```

```
% user 100 has sent a msg to the group 910 at 16:00:00 on
22/5/2013
comm(sendmsg,100,910,"20130522160000").
```

The weight of a recent communication is calculated by a fixed time window based algorithm. In the current setting we consider past communications in the last one month. The time window (1 month) is also divided into 4 periods where the first one is the shortest and the length gradually increases. The intuition is we want to give more importance to the recent communications than the older ones. Generating and calculating weights of possible items with which the user had a recent communication are encoded by the following logic program part. CommCount predicate gives how many number of communication instances occurred during a specific period. Then, commScore gives weight scores for each period and item (with which the user had communication). Note that the rules generating commScore give more weight to earlier periods than later ones (corresponds to giving more importance to more recent communication instances). At the end possible(I,comm,S) predicate encodes that item with id I is possible to be selected for suggesting because the user has communicated with it and it has a overall weight score of S.

```
commWith(U,P)       :- user(U), comm(_,U,P,D).
commCount(Per,P,C) :- user(U), commWith(U,P), commPeriod(Per,S,E),
                      C := #count{ comm(T,U,P,D):D>S:D<=E }.

periodCount(C)      :- C:= #count { period(P) }.

commScore(I,Per,C*W) :- commCount(Per,I,C), C>0, periodCount(N),
Per <= (N/2), W:=N-Per+1.
commScore(I,Per,W)   :- commCount(Per,I,C), C>0, periodCount(N),
Per > (N/2), W:=N-Per+1.

possible(I,comm,S) :- user(U), commWith(U,I),
                      S:=#sum[ commScore(I,Per,W):period(Per)=W ].
```

The following rules are related to the reasons in which the user follows the person, the user is registered to the group, and the group is created by a person who the user follows.

```
possible(I,friendship) :- user(U), follows(U,I).
possible(G,regisgroup) :- user(U), registered(U,G).
possible(G,friendowns) :- user(U), follows(U,F), groupowner(G,F),
                          not registered(U,G).
```

Up to now the encoding generated instances of predicates possible(I,R) and possible(I,comm,S)

corresponding to possible items for selection for various reasons. The other possible predicate representing items with common interests with the user will be explained in the next section. The rules below encode the total score for an item. Note that an item may have more than one reason to be selected. The #sum literal in bodies of rules generating itemscore predicate aggregates all the weights of reasons for each possible item.

```
% general reasons
possibleitem(P,person) :- possible(P,_), personid(P).
possibleitem(P,group)  :- possible(P,_), groupid(P).

% interest related
possibleitem(P,person) :- possible(P,_,_,_), personid(P).
possibleitem(P,group)  :- possible(P,_,_,_), groupid(P).

% recent communication events
possibleitem(P,person) :- possible(P,_,_), personid(P).
possibleitem(P,group)  :- possible(P,_,_), groupid(P).


itemscore(I,followed,S)  :- possibleitem(I,person), follows(U,I),
  user(U),
  S:=#sum[ possible(I,R):weight(R,W)=W, possible(I,K,Kn,WW)=WW,
                                        possible(I,_,W3)=W3 ].

itemscore(I,notfollowed,S) :- possibleitem(I,person), not
  follows(U,I), user(U),
  S:=#sum[ possible(I,R):weight(R,W)=W, possible(I,K,Kn,WW)=WW,
                                        possible(I,_,W3)=W3 ].

itemscore(I,registered,S)  :- possibleitem(I,group),
  registered(U,I), user(U),
  S:=#sum[ possible(I,R):weight(R,W)=W, possible(I,K,Kn,WW)=WW,
                                        possible(I,_,W3)=W3 ].

itemscore(I,notregistered,S)  :- possibleitem(I,group),
  not registered(U,I), user(U),
  S:=#sum[ possible(I,R):weight(R,W)=W, possible(I,K,Kn,WW)=WW,
                                        possible(I,_,W3)=W3 ].
```

The choice rules below generate the whole search space for the item selection problem. There are 4 groups of items PA selects; people followed, people not followed, registered groups, and not registered groups. Instances of item predicate in an answer set shows the items selected by the PA. The second argumnet of the item predicate represents which group the item is from, i.e., whether it is followed, notfollowed, registered, or notregistered. The upper bounds for selections can be set

when first setting up the PA. For instance, if max_num_not_followed is set to 5, then at most 5 people, whom the user is not following, will be selected by the answer set solver. Note that while the current EasyReach client asks the PA for 2 groups which are people not followed and groups not registered to, PA is flexible enough and supports the whole 4 selection groups. The maximize statements tries to maximize the number of items selected (of course it is up to the limit given by the upper bound). This is not given as a lower bound in the choice rule since there may be not enough items from the respective group to fulfil the desired number of selected items.

The last optimization statement ensure that items with highest scores are selected. The intuition is that whenever an item has a greater weight score than another item's score, it has more reasons to be selected. This, in turn, means that it is better item for suggesting to the user since it leads to a social interaction more probably than the other one with a lower score. The optscore predicate gives the total score for the selected item. Optscore is calculated using the respective itemscore predicate instance with an additional consideration of penalties. Penalties are applied for already selected items. In this way selecting always the same set of items is eliminated and fair selections become possible. Recall that the Selection history manager module of PA takes record of selected items in the past as explained in Section 1.

```
{ item(I,followed):possibleitem(I,person):
      follows(U,I):user(U) } max_num_followed.

{ item(I,notfollowed):possibleitem(I,person):
      not follows(U,I):user(U) } max_num_notfollowed.

{ item(I,registered):possibleitem(I,group):
      registered(U,I):user(U) } max_num_registered.

{ item(I,notregistered):possibleitem(I,group):
      not registered(U,I):user(U) } max_num_notregistered.


#maximize { item(I,followed)      @2 }.
#maximize { item(I,notfollowed)   @2 }.
#maximize { item(I,registered)    @2 }.
#maximize { item(I,notregistered) @2 }.

#maximize [ item(I,C):optscore(I,C,S)=S @1 ].
```

4. Usage of the Taxonomy of Interests for Selecting Items

In order to reason on user interests or group topics we need a model of interests; we utilize a taxonomy of users' interests for this purpose. The selection of interests is based on preliminary

investigations about elderly needs, hobbies and expectations. Formally, we modeled our taxonomy as a forest of keywords of interests where the edges represent the subsumption relation among keywords. Figure 3 depicts a subset of the taxonomy used in EasyReach. Note that the keyword *Documentary* is subsumed by *Programs*, and *Programs* is subsumed by *TV*. Thus, a keyword at a deeper level of a tree in the taxonomy represents more specialized interest than one at a shallower level.

A user creates a profile by specifying keywords corresponding to his interests. The EasyReach client lists the taxonomy in an easy to select way at the user creation stage. The taxonomy allows the PA to exploit the semantic information inherent in a user profile. For example, when a user specifies *Formula 1* as his interest, PA can use not only *Formula 1* but also *Auto racing* or

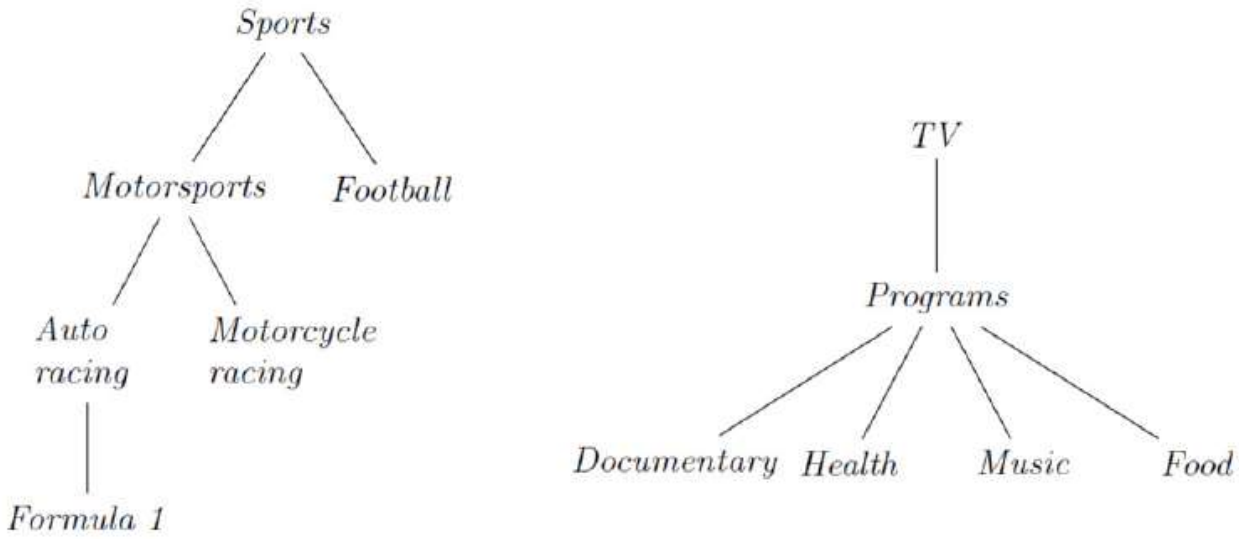

Figure 3: A subset of the taxonomy of user's interests

Motorcycle racing for reasoning to suggest new items in his list.

In the ASP program of the PA, the taxonomy is encoded by logic program facts for keyword nodes and subsumption relation.

```
keywordid(67).
keywordname(67,"TV Programs").

keywordid(82).
keywordname(82,"Documentary").

subset(82,67).
```

When PA is asked for selecting items, it considers the interests mentioned in the user's profile.

Starting from interest nodes in the taxonomy, it traverses to connecting nodes using the subsumption relation. A connecting node must be reachable with a path whose length is expressed as a parametric value denoting the maximum allowed path length. For instance, let {*Motorcycle racing*, *Formula 1*} be the set of interests mentioned in a user's profile and the taxonomy used be the one shown in Figure 3. Assuming that the maximum allowed path length is 2, the interest nodes reachable from *Motorcycle racing* are {*Motorsports*, *Auto racing*, *Sports*}. Considering all the interests in the user's profile, the PA takes the set {*Motorcycle racing*, *Formula 1*, *Motorsports*, *Auto racing*, *Sports*} into account when checking other users with common interests. Additionally, the PA assigns weight to the interests dynamically according to the depth of its node in the taxonomy tree. For our example the weight of *Formula 1* is greater then *Motorsports*. The intuition is that the more specialized a common interest is, the better the suggestion. We encoded all the taxonomy traversal and weighting in ASP and total weight scores of item related to interests are represented by instance of the possible(P,K,Kn,W) predicate (as mentioned in Section 3), i.e., item P (a person or a group id) is suitable for selecting with a weight score W regarding the common interest K (with name Kn).

5. Taxonomy of User Interests

The following list shows the taxonomy of user interests used by the PA. These interests are populated after preliminary user studies. Hence, they are more geared towards the elderly people considering the main user group in the EasyReach project. The subsumption relation between taxonomy nodes is shown by proper indentation.

- Sports
  - Events
  - Basketball
  - Fishing
  - Football
  - Motorsports
    - Auto Racing
      - Formula One
    - Motorcycle Racing
  - Olympics
  - Tennis
  - Winter Sports
- Television
  - Regional
  - News
  - History
  - Programs
    - Comedy
    - Documentaries
    - Educational
    - Food
    - Health
    - Talk Shows
    - Music
    - Films
    - TV Series
    - Entertainment

Reality Shows

Arts
- Literature
- Books
- Photography
- Performing Arts
- Theatre
- Dancing
- Painting
- Music
- Singing
- Movies / Cinema

Health
- Medicine
- Fitness
- Disabilities

Society
- Religion
  - Parish Church
- Organizations
  - Charities
  - Voluntary Association
  - Cultural Association

Games
- Yard/Outdoor Games
  - Bocce
- Card Games
  - Bridge
- Board Games
  - Chess
  - Backgammon

Home
- Gardening
- Do-It-Yourself
- Cooking
- Recipes
- Pets
- Collecting
- Stamps
- Coins
- Postcards
- Records

News
- Newspapers
- Radio

Tourism
- Culture
- Travel
  - Outdoor Trips
  - Health Tourism
- Festivals

6. Social Database Interface

The PA fetches instance data from the social DB. The Social DB manager module of PA is responsible from this (as shown in Figure 2). When a user logs in to the EasyReach client, Social DB manager fetches/updates instances for the user. Additionally, it updates instance data of the current user in regular intervals (e.g., every 2 hours) which can be set parametrically. The regular updates are needed since there might be changes related to the current user in the whole EasyReach social network. For instance, a new group may be created and it may be about some topics the user is interested in.

Below is a list of queries that are needed by the Social DB manager to fetch data from the Social DB. The queries have input parameters which puts constraints on the data to be fetched. In this way we do not fetch unnecessary data. This is important optimization for the efficiency of the whole EasyReach system.

These functionalities are implemented as WSDL services like remote procedure calls. The communication protocol used is SOAP. (I) and (O) denotes input and output parameters respectively.

1. **Fetch taxonomy keywords (fetchTaxonomyKeywords)**
   Returns a list of tuples composed of
   (O) keyword_id:
   (O) keyword_name:

2. **Fetch taxonomy relation (fetchTaxonomyRelations)**
   Returns a list of tuples composed of
   (O) keyword_id1:
   (O) keyword_id2:
   The keyword_id1 is a immediate subset of keyword_id2. (e.g., <cooking,home>).

3. **Fetch people who are interested in an input set of interests (fetchPeopleInterestedTo)**
   (I) target_interests: List of keyword_ids
   Returns a list of tuples composed of
   (O) person_id:
   (O) person_name:
   (O) interests: List of keyword_ids where the person person_id is interested in.
   An output tuple should satisfy the condition that there exists a keyword $k \in$ *interests and* targetinterests.

4. **Fetch people who a user follows (fetchUserContacts)**
   (I) user_id: Id of the user
   Returns a list of tuples composed of
   (O) person_id: Id of the person who is followed by user_id
   (O) person_name:

5. **Fetch groups which are about an input set of interests (fetchGroupsRelatedToInterests)**
   (I) target_interests: List of keyword_ids
   Returns a list of tuples composed of
   (O) group_id:
   (O) group_name:
   (O) group_owner: Id of the person owning the group
   (O) interests: List of keyword_ids where the group is about.
   An output tuple should satisfy the condition that there exists a keyword $k \in$ *interests and targetinterests*.

6. **Fetch groups which are owned by people followed by a user (fetchGroupsOwnedByPeopleFollowedByUser)**
   (I) user_id: Id of the user
   Returns a list of tuples composed of
   (O) group_id:
   (O) group_name:
   (O) group_owner: Id of the person owning the group
   An output tuple should satisfy the condition that user_id is following group_owner.

7. **Fetch groups which a user has registered to (fetchGroupsSubscribedByUser)**
   (I) user_id: Id of the user
   Returns a list of tuples composed of
   (O) group_id:
   (O) group_name:
   An output tuple should satisfy the condition that user_id is registered to the group group_id.

8. **Fetch communication instances which a user sent after some date (fetchSentMessagesAfterDateByUser)**
   (I) user_id: Id of the user
   (I) lastdate: String. (e.g., 20121217132240 for 13:22:40 17/12/2012)
   Returns a list of tuples composed of
   (O) obj_id: Id of the person or group to which the user sent a message
   (O) type: ‘P’ if obj_id is a person id, ‘G’ if obj_id is a group id

(O) date: String. Date of the message (e.g., 20121217132240 for 13:22:40 17/12/2)
An output tuple should satisfy the condition that date > lastdate.

9. **Fetch communication instances which a user received (only from other people) after some date (fetchReceivedMessagesAfterDateByUser)**
   (I) user_id: Id of the user
   (I) lastdate: String. (e.g., 20121217132240 for 13:22:40 17/12/2012)
   Returns a list of tuples composed of
   (O) person_id: Id of the person from whom the user received a message
   (O) date: String. Date of the message (e.g., 20121217132240 for 13:22:40 17/12/2)
   An output tuple should satisfy the condition that date > lastdate.

10. **Fetch interests of a user (fetchUserInterests)**
    (I) user_id: Id of the user
    Returns a list of tuples composed of
    (O) keyword_id: Id of a keyword which the user is interested in

7. Accessing the Personal Assistant from the EasyReach Client

The main functionality of PA is to select items for suggestion. The PA provides an interface for the EasyReach client whenever it requests this functionality. The interface is the *callSelector* script. The EasyReach client calls *callSelector* after a user logs in to the system. In principle it can be called whenever the client needs to populate its lists of suggestions.

The return code for successful operation is 0. Anything other than that is an unsuccessful call. The output is composed of 4 parts, 2 of which are list of followed people (FOLLOWED) and registered groups (REGISTERED) sorted according to their scores. The other 2 parts are selected new items; not followed people (NOTFOLLOWED) and not registered groups (NOTREGISTERED). As mentioned earlier, the EasyReach client asks for only new items currently. The following example shows a call and its output.

Example:

```
callSelector
FOLLOWED:
itemscore(109,followed,10).
itemscore(103,followed,5).
itemscore(104,followed,4).
```

```
itemscore(101,followed,1).
itemscore(105,followed,1).
itemscore(107,followed,1).
REGISTERED:
itemscore(777,registered,1).
itemscore(999,registered,1).
NOTFOLLOWED:
item(201,notfollowed).
item(333,notfollowed).
NOTREGISTERED:
item(902,notregistered).
item(901,notregistered).
```

8. Handling Actions of the Users

Although PA periodically fetches instance data from the Social DB, it is not frequent considering the costly operation of connecting and fetching data from the social DB. Hence, there might be some frequent changes related to actions of the user that may not be fetched by the Social DB manager. These changes are important for better and complete suggestions. Consider that the user has just sent a message to another person or a group. These actions lead to changes in the Social DB which are not yet fetched. The PA has a mechanism to handle these changes fast so that it can give better suggestions. This is the responsibility of the Interface for client updates module.

The PA provides several procedures, which are called by the EasyReach client whenever the user makes an action generating updates in the Social DB. These procedures updates related instance data to be used by the next call of item selection.

The procedures for handling user actions are bundled in the *changeFromUI* script. The client calls this script when the related interaction action occurs (for instance sending a message). Below is a list of these procedures with respective example actions. (I) and (O) denotes input and output parameters respectively. The return code for the script is 0 for successful operations. Any other number means that the operation was unsuccessful.

1. **Follow a person**
   (I) user_id: Id of the user
   (I) person_id: Id of the person whom the user follows

   Example: The user with id 100 follows the person with id 108.

```
changeFromUI followspers --user 100 --pers 108
```

2. **Group creation**
   (I) user_id: Id of the user
   (I) group_id: Id of the group
   (I) group_name: String. Name of the group
   (I) keyword_id list: List of interest keywords for the group

   Example: The user with id 100 creats the group with id 999 whose name is "NEX Camera Group". The groups is about keywords 202 and 231. Note that the keywords are comma separated list.
   ```
   changeFromUI creategrp --user 100 --grp 999 --name "Nex Camera Group" --keyw 202,231
   ```

3. **Registering to a group**
   (I) user_id: Id of the user
   (I) group_id: Id of the group

   Example: The user with id 100 registers the group with id 999.
   ```
   changeFromUI registergrp --user 100 --grp 999
   ```

4. **Addition of an interest to the profile**
   Currently not supported.
   (I) user_id: Id of the user
   (I) keyword_id: Id of the added keyword

5. **Send a message to a person (followed or non-followed one)**
   (I) user_id: Id of the user
   (I) person_id: Id of the object person
   (I) date: Date of the communication. String. 20121217132240 for 13:22:40 17/12/2012. This date should be the same date as the DB entry for the corresponding communication.

   Example: The user with id 100 sent a message to the person with id 108 at 12:00:12 in 21/12/2012.
   ```
   changeFromUI sendmsgpers --user 100 --to 108 --date 20121221120012
   ```

6. **Send a message to a registered group**
   (I) user_id: Id of the user
   (I) group_id: Id of the group
   (I) date: Date of the communication. String. 20121217132240 for 13:22:40 17/12/2012. This date should be the same date as the DB entry for the corresponding communication.

   Example: The user with id 100 sent a message to the group with id 999 at 12:00:12 in 21/12/2012.

```
changeFromUI sendmsggrp --user 100 --to 999 --date
20121221120012
```

7. **Receive a message from a person (followed or non-followed one)**
   (I) user_id: Id of the user
   (I) person_id: Id of the person who sent the message
   (I) date: Date of the communication. String. 20121217132240 for 13:22:40 17/12/2012. This date should be the same date as the DB entry for the corresponding communication.

   Example: The user with id 100 received a message from the person with id 108 at 12:00:12 in 21/12/2012.
   ```
   changeFromUI recvmsgpers --user 100 --frm 108 --date
   20121221120012
   ```

8. **User login**
   (I): user_id: Id of the user

   Example: The user with id 100 logs in. UI should call this API function before asking for item selections.
   ```
   changeFromUI userlogin --user 100
   ```

9. The EasyReach Agenda

Among the services that EasyReach offers to the elderly users, the Agenda service is particularly useful as it provides proactive support for the user's daily life, by generating on-line notifications relatively to the user's personal commitments. The main objective of the EasyReach Agenda is to remind important deadlines and/or managing appointments by providing to the user a comfortable and direct environment through which to access and provide all the necessary information about the reminder messages she/he wishes to be notified in the future. In other words, through the agenda service, the user has the possibility to send *timed messages* to herself, i.e., by deciding *what* to communicate (the contents of the message) and *when* to communicate it (i.e., the time of the notification). Informally, the Agenda service works by providing a continuous monitoring of all the data input by the user, constantly checking their "expiration date" against the system's clock time, and "firing" the proper notification event when the two coincide.

Given that the EasyReach system is designed to target an elderly users basin (possibly pre digital divide users) not familiar with technological devices and procedures, many efforts have been put to simplify the utilization of the agenda service as much as possible, mainly by keeping the technological aspects as much as possible in the background, by limiting the quantity of information that the user must provide, and by re-utilizing the same system-user interaction methodologies used elsewhere throughout the system.

The EasyReach Agenda Timeline-based representation

The technology used to implement the agenda service and to represent all necessary components is the same technology exploited for timeline-based planning, an approach to temporal planning which has been mostly applied to the solution of several space planning problems [4].

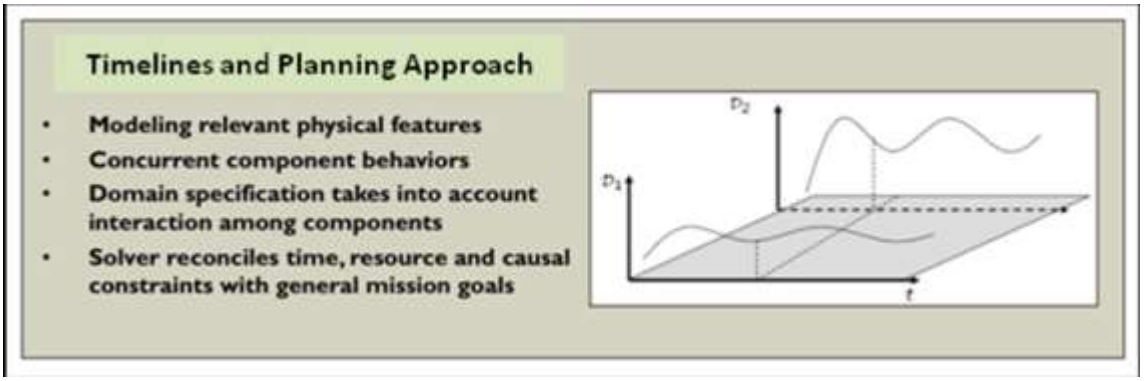

Figure 4: The broad idea of the TRF Domain Modeling and Solving

The philosophy underlying Timeline-based Planning and Scheduling is inspired by classical Control Theory, in that the planning and scheduling problem is modelled by identifying a set of relevant features whose temporal evolutions need to be controlled to obtain a desired behavior (see Figure 4). Within the Timeline-based Representation Frameworks (TRF), such problem features are called *components* and are the primitive entities for knowledge modelling. They represent logical or physical subsystems whose properties may vary in time. An intrinsic property of components is that they evolve over time, and that control decisions can be taken on components to define their evolution. In other words, components model different temporal behaviors over time based on a set of constraints while modeling the physical world. The problem-solving task consists of finding a temporal behavior from the set of component behaviors such that all requirements, including the current planning goals, are satisfied.

In timeline-based planning, the main data structure is the *timeline* which, in generic terms, is a function of time over a finite domain. Events on timelines are called *tokens* and are represented through a predicate extended with extra arguments belonging to the Time domain **T** (real or discrete). For example, a predicate *At(l)*, denoting the fact that an agent is at a certain location *l*, can be extended with two temporal arguments $s \in \mathbf{T}$ and $e \in \mathbf{T}$, with $s < e$, representing its starting and ending times, respectively; the *At(l, s, e)* formula would be true only if the agent is at location *l* from time *s* to time *e*. Tokens can be linked to each other through *relations* in order to reduce allowed values for their constituting parameters and thus decreasing allowed system behaviors.

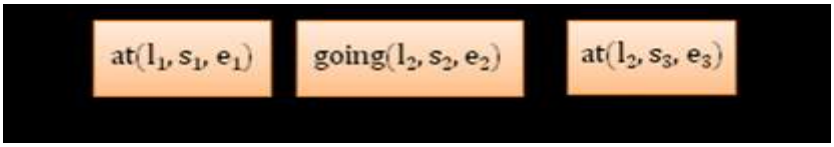

Figure 5: An example of Timeline

Figure 5 presents a simple example of a Timeline containing three tokens, each characterized by a start time *s*, an end time *e*, and an argument *l*. The timeline in this particular example may be used to describe the evolution of a moving vehicle over time, i.e., the vehicle being at location l_1 from $t=s_1$ to $t=e_1$, then moving to location l_2 from $t=s_2$ to $t=e_2$, and finally staying at location l_2 from $t=s_3$ to $t=e_3$.

In order to express planning domain/causal rules in the current internal representation we make use of the concept of *compatibility*. We describe compatibilities by means of logic implications *reference* → *requirement*. Within this context, the task of the planner is to find a legal sequence of tokens that bring the timelines into a final configuration that verifies both the *domain theory* (i.e., the set of compatibilities that model the domain's dynamic behaviour) and a determined set of desired conditions called *goals*. Starting from an initial state, the planner moves in the search space performing a complete refinement search by adding or removing tokens and/or relations (i.e., changing the current state) until all goals are satisfied.

As can be seen, the timeline-based technology allows for representing and reasoning on planning domains that may exhibit rather complex characteristics. As explained below, within the EasyReach current agenda service only a part of the timeline-based functionalities are used, i.e., those mostly related to the components representation issues, and to some basic scheduling capabilities (i.e., used to separate temporally overlapping tokens after the insertion in the timeline, operation executed transparently to the user). However, a full utilization of both the representation and reasoning the potential of timeline-based technology can surely allow for significant enhancements in future versions of the system. In particular, the exploitation of the planning capabilities on top of planning domains synthesized out of the some of the user's activities performed within the system, as well as of the temporal relations possibly existing among them, might significantly increase the overall system's proactive features.

The basic objective of our agenda environment is to create and dynamically adapt a timeline in which each token represents a reminder message to be acquired and duly notified to the user as time passes. The pursued idea is to represent the agenda as a *walking plan* composed of different "messages" to be sent to the users, characterized by both temporal and non-temporal features. Such plan, composed by time-tagged activities that trigger the presentation of multimedia events, is first synthesized starting from an abstract specification provided by the user (see next subsection), and then executed.

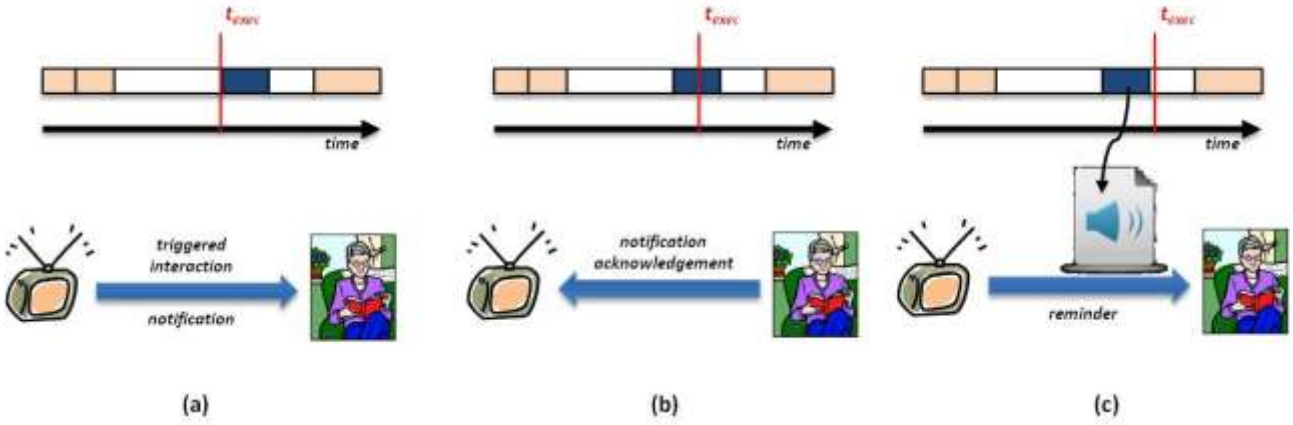

Figure 6: The EasyReach agenda interaction steps

During the plan execution, the value of the current system clock is continuously compared with the data associated to each token (see Figure 6); in particular, the information relatively to its start time will be particularly relevant as it determines the time at which the reminder is to be notified to the user. In all cases where the two previous values match, the proper notification will be fired, as shown in Figure 6 (a). Finally, as soon as the user decides to acknowledge the notification (Figure 6 (b)), the audio file associated to the link contained in the notification token will be played, thus delivering the reminder message (Figure 6 (c)).

The Agenda from the user's perspective

One of the main guidelines that have been followed for the design of the agenda has been the simplicity of use; for this reason, all possible efforts have been dedicated to make the utilization of this instrument as natural and comfortable as possible for the users, mainly by (i) minimizing the quantity of data that must be provided to exploit the service, and (ii) utilizing the same information input methodology that is used throughout the system, and that the user is supposed to be already familiar with.

Figure 7: The EasyReach Main screen

Figure 7 depicts the main screen which is presented to the user as she/he logs in. As Figure 7 shows, the agenda service is accessed by activating the "Open Agenda" command button (right side, 2nd button from the top). By pressing the button, the information area of the screen changes as shown in Figure 8.

The environment presented to the user does not offer many differences relatively to the ordinary contact or group message exchange screen. The whole EasyReach GUI is built making extensive use of recurring graphical elements, possibly arranged according to slightly varying layouts, in order not to confuse the user. In this particular case, the list of all received notifications (i.e., the messages) is still present at the top-left of the view in the "Agenda" box, and all messages are presented in the familiar fashion. The same holds for the "Reminder preview" box (bottom-right), which can be used to enjoy a preview of the newly created notification before it is added to the agenda. The "Past Activities" box (top-right) contains all the past notifications that have already been notified to the user, and represents a slightly new element with respect to the contact or group message reading page.

Figure 8: The EasyReach Agenda screen

From the "utilization familiarity" standpoint, the only really new element is therefore represented by the "New reminder" box (bottom-left). In this section, the user is requested to provide two types of information: (i) the data (i.e., date and time) about when the reminding will have to be notified to the user, and (ii) the contents of the reminding itself. The date and the time of the notification can be provided in a straightforward fashion by acting on the "Date" and "Time" button arrows, (day-month-year, and hour-minutes, respectively), while the contents of the reminder can be recorded by the user as an ordinary audio file, by pressing the "Record a reminder" button. It should be underscored that, in order to facilitate the utilization of the agenda, the creation of a reminder audio file follows exactly the same procedure as the creation of any other audio message to be sent to one of the user's contacts or groups. Once the reminder is created, its contents can be listened back by clicking on the "Reminder preview" box, before definitely adding the message to the agenda by means of the "Create new reminder" button.

From the user's utilization perspective, the management of all the created messages, as well as their fruition mechanism, is organized according to the following logic:

- as previously stated, all the newly created messages will appear in the "Agenda" box, together with all the previously created messages not yet notified to the user
- as each reminder is notified to the user (i.e., the user positively acquires the message), it is moved to the list in the "Past Activities" box. Caveat: all the messages in the "Past Activities" box will remain visualized for the entire duration of the current session only; as the user logs out, all the past messages will be deleted as no longer useful and will not be

visualized again, in order not to overload the visual representation

- all the messages in the "Agenda" box that could not be delivered to the user (e.g., the user never logged in during the day due for the notification) are however kept in the system, and will be notified to the user immediately after her/his next login to EasyReach. In this way, the user is kept informed of all the appointments she/he may have possibly missed.

All the agenda reminders are received by the user through exactly the same mechanism used for any other type of message. In particular, any new incoming reminder message is accompanied by a blinking of the EasyReach icon at the bottom-left of the GUI (see Figure 8), thus warning the user that there are messages pending to be read, and that her/his attention is required.

10. References

[1] Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)

[2] Jost, H., Sabuncu, O., Schaub, T.: Suggesting new interactions related to events in a social network for elderly. In: Proceedings of the Second International Workshop on Design and Implementation of Independent and Assisted Living Technology (2012)

[3] Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: The Potsdam answer set solving collection. AI Communications 24(2), 105–124 (2011)

[4] Muscettola, N.: HSTS: Integrating Planning and Scheduling. Technical report, Robotic Institute (1994)