Towards a framework that allows using a cognitive architecture to personalize recommendations in e-commerce

Jordi Sabater-Mir, Joan Cuadros, Pere Garcia

IIIA - CSIC, Campus UAB, 08193 - Bellaterra, Catalonia, Spain

Abstract. In this paper we propose the use of a cognitive architecture as a mechanism to improve the level of personalization in a recommendation process. The main idea is to use the cognitive architecture as an advisor that lies between the user and a set of specialized recommenders. The cognitive architecture should know about the strengths and weaknesses of the recommenders, when they can be used and their degree of reliability given the amount of information available, among other things. With this information and the detailed knowledge about the user needs, it should be able to provide much more reliable and trustful recommendations by combining the strengths of each recommender given the specific context. The flexibility of a cognitive architecture in this place of the recommendation process will allow also to consider a much more sophisticated interaction with the user by allowing for example argumentation processes between the user and the system.

1 Introduction and motivation

There is no doubt that the use of recommender mechanisms has become almost a must in e-commerce applications. Probably the most known example is the famous"Customers who bought this item also bought..." in Amazon but in fact, any e-commerce site is expected to have some kind of recommender that suggests to the user new products that ideally satisfy his/her particular preferences. The research in the area of recommender systems has developed a variety of technologies that exploit different sources of information and knowledge [9]. Some of these technologies, like collaborative filtering or content filtering, are already very mature and used in commercial applications.

These technologies are domain independent, something initially very desirable because this means that they can be applied in many different scenarios. However, this independence of the domain implies that in order to achieve a good degree of accuracy, someone has to analyze the specific scenario and decide which is the best technique to be used and how to tune it.

For example, imagine we have the list of products that users A, B, C and D have bought the last three months in a supermarket. A simple user-based collaborative filtering technique would use that list to establish a similarity between users, where two users are similar if they usually buy the same products.

Suppose that A buys almost the same things that B and D. However, B and D always buy whole milk and this product is one of the few products that does not appear in the list of things that user A usually buys. At this point the recommender infers that, because A, B and D are similar and that because B and D buy whole milk, whole milk should be of interest for user A. Unfortunately user A cannot tolerate milk.

Of course, a solution to this specific problem could be to consider families of products instead of all products to establish the similarity between users (for instance how similar are A, B, C and D regarding only the purchase of dairy products). Here for sure the purchase habits of A are completely different from the purchase habits of B and C and therefore the recommender never will suggest milk as a possible product for A. But what if A is looking for an alternative to a specific product (for example milk) that is similar from a nutritional point of view? In that case a collaborative filtering technique is probably not the best option and it would be better to use a content filtering technique that takes into account the nutritional properties of the products and that can suggest things like yogurts, soy milk, etc.

What we want to illustrate with this simple example is that, even in a very specific scenario (in this case buying in a supermarket), a single recommender technique cannot be enough to cover all the real necessities of the different users that are looking for a recommendation. When this fact is ignored, what we have is a recommender that gives reasonable recommendations but from time to time says stupid things. Even if only a few percentage of the recommendations are silly, it is enough to generate distrust on the recommender and, as a consequence, the user stops using it.

The solution to this problem seems obvious: let's take a full "palette" of recommender techniques and, like a painter, use at run time the right "color" for each situation (properly fine tuned to recapture the nuances of the context). Such an automatic decision maker could be designed and implemented ad hoc for the specific problem (you could even think on a simple if-then structure that considers all the possibilities). However, such a static decision maker is not reasonable/feasible for the great majority of applications.

What are we looking for is an automated decision maker that can decide: (i) which is the best recommendation technique (or techniques) to use, (ii) how to adapt the results to the specific needs of a given user and (iii) how to properly show the results to that user. Our proposal is to use a cognitive architecture as such a decision maker.

In this paper, we present a framework (still in a preliminary stage) that integrates a cognitive architecture into the workflow of a recommendation process. The purpose of the paper, apart from describing the framework itself from a technological perspective, is to show the potential we think can have the use of a cognitive architecture in the area of recommender systems.

After a related work section (section 2), in section 3 we describe the framework. We start presenting a general overview (section 3.1) and then we go through the different parts in detail: the data analysis block (section 3.2), the

user interface (section 3.3), the cognitive architecture (section 3.4) and the recommenders (section 3.5). In the second part of the paper (section 4) we present, with the help of a use case, a simple instantiation of the framework. Again we go through the different parts of the framework but this time presenting a specific implementation: Data analysis (section 4.1), user interface (section 4.2), cognitive architecture (section 4.3) and recommenders (section 4.4). We finish with the conclusions and the future work.

2 Related work

There are several studies where a cognitive architecture is used as a base for a recommender. In the case of the BDI architecture, a well known example is the work from Casali et al. [3]. They use a graded BDI agent model based on multi-context systems to specify an architecture for a Travel Assistant Agent that helps a tourist to choose holiday packages. This graded BDI agent is built using the graded BDI agent development framework introduced in [2]. This graded BDI agent, using a modal many-valued approach, allows to represent and reason about graded notions of beliefs, desires and intentions.

Another example of multi agent recommender system using a BDI architecture is presented in [1]. This system presents a multi-agent model that facilitates aspects of shopping mall management, as well as increasing the quality of leisure facilities and shopping on offer. The work focuses on the use of a multi-agent architecture with deliberative agents that incorporate case-based planning and BDI. The core of the multi-agent systems is a recommender agent in charge of the route generation in response to a clients request looking for the best shopping or leisure time alternatives, when he/she wants to spend his/her time in the mall. The recommender agent takes into account the client's profile, the maximum amount of money that the client wants to spend and the time available.

Regarding cognitive architectures different from BDI, in [6] the authors propose a new type of personalized recommendation agents called fuzzy cognitive agents. Fuzzy cognitive agents are designed to give personalized suggestions based on the users current personal preferences, other users common preferences, and experts domain knowledge. Fuzzy cognitive agents are able to represent knowledge via extended fuzzy cognitive maps, to learn users preferences from most recent cases and to help customers make inferences and decisions through numeric computation instead of symbolic and logic deduction.

In [5] it is presented a method for the personalization of information selection based on rational analysis and cognitive architectures. The authors developed an application called the Personal Publication Assistant. The user model underlying the Personal Publication Assistant is based on a rational analysis of memory, and takes the form of a model of declarative memory as developed for the cognitive architecture ACT-R, a theory for simulating and understanding human cognition.

Our approach deviates from all these attempts in that they use a cognitive architecture to create a recommender while in our proposal the cognitive ar-

chitecture is a middle layer between the users and different types of specialized recommenders.

3 The framework

3.1 A general overview

Figure 1 shows the different elements of the framework and the information workflow. There are four main blocks:

- The data analysis block, that deals with the processing of the information and prepares the data both for the cognitive architecture and for the recommenders (see section 3.2).
- An interface that allows the user to interact with the system (see section 3.3). It can be a web page, a mobile application, etc.
- The cognitive architecture, core of the system, that interacts with the user and the recommenders to satisfy the user needs (see section 3.4).
- A set of specialized recommenders, each one with its particularities (see section 3.5).

Finally there is a domain ontology and some general knowledge that are used in a transversal way by the other blocks.

A traditional recommendation flow connects directly the user with the recommender. As we commented in section 1, this does not allow to combine smartly several recommenders in order to exploit the strengths and avoid the weaknesses of each one. Ultimately, one could think that the user could do that final combination. The system can just provide the information coming from all the recommenders and is up to the user to use one source or another. This, however, is not realistic because a regular user does not know how the recommenders work internally and therefore ignores when you can trust a particular recommender and when you cannot in a specific situation. The role of the cognitive architecture therefore has several facets:

- It is the entity that knows about the user needs and his/her desires, goals, restrictions, beliefs, etc. This knowledge about the user comes on the one hand from the data analysis block and from the other from the interaction with the user itself through the interface.
- It knows about the recommenders that are available, their strengths and weaknesses, and how they can be used properly given a specific situation.
- Given the previous information, tries to satisfy the user needs in terms of recommendation.

One could argue that the role that in our framework plays the cognitive architecture could be played by a simpler decision maker like, for example, a finite state machine. That's true if we have well known and static scenarios where users have limited and specific needs. However, as we will explain in the following sections, we try to settle down the foundations for a much more

ambitious system with advanced capabilities like the capacity to argue with the user after a given recommendation or the capacity to infer user motivations that are not explicit.

The second reason of using cognitive architectures in the context of recommendation processes is more philosophical. We strongly believe that for certain tasks (recommendation is one of them) it is crucial that the behavior of the machine be similar to that of a human. A user can have difficulties to trust a recommender that appears as a black box. Trust has to be built on top of the mutual understanding and this will be possible only if the user is able to understand the reasoning process of the machine and vice versa. To achieve that, we think that the most natural way is to use cognitive architectures based on human cognitive theories and allow them to build the bridge between the user and the (sometimes obscure) techniques for recommendation.

As we said in the related work section (2), we are not proposing to use the cognitive architecture as a new kind of recommender but as an advisor that can use current recommendation techniques and make them much more trustful and accessible to the user.

In the following sections we will go through the different parts of the framework in more detail.

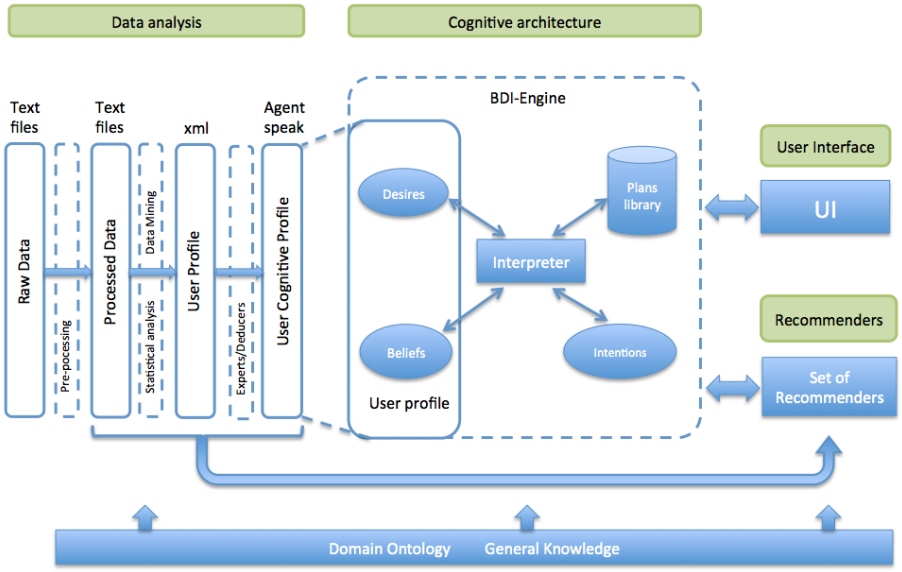

Fig. 1. The Cognitive Recommendation Framework.

3.2 Data analysis

This part of the framework is responsible for processing raw data so it can be used both by the cognitive engine and the recommenders. We distinguish different levels of processing, each one providing a greater level of abstraction.

Pre-processing: Initially we usually will start with data that, even in the case it is structured, probably will not be ready to be analyzed using automatic techniques. First, this raw data has to be cleaned, parsed, normalized, etc. before it can be used. At this level we are not extracting new information from that data, we are just preparing it to be analyzed using different techniques in the following steps.

Data processing: From simple statistical measures to advanced data mining algorithms, there is a plethora of techniques that allow to extract information from structured data. Of course, it is not the purpose of this paper to go through them. For example, in the context of supermarket purchases, we can try to discover shopping habits like frequency, shopping volume, or identify clusters of users that prefer some kind of products. In our framework, the result of applying these kind of techniques allows us to build the *User profile* that is the base for what we call the *User Cognitive profile*. In this step it is also performed the preparation of the data to be used by the recommenders.

Experts/Deducers: In this step we create a knowledge base from the *User profile* to be used by the cognitive engine (*User cognitive profile*). This implies to represent the information in terms of the cognitive architecture (for example in the case of a BDI architecture, in terms of beliefs and desires). Some of the information coming from the *User profile* will just go straight to the cognitive architecture without further processing. However, because we want the cognitive architecture to deal as much as possible with qualitative high level concepts, more than often it will be necessary further processing and this is the task of what we call the "experts/deducers".

For example, if the *User profile* has the number and volume of purchases performed by a client during the last month, after comparing those numbers with the ones coming from other users and after noticing that the shopping volume is one of the highest, one of the "experts/deducers" could transform that information into something like "this user is a very good client". This allows the reasoning in the cognitive architecture to be done in terms of qualitative and abstract concepts. It is important to notice that the system maintains the link between the data in the *User profile* and the information in the *User cognitive profile* so it can be updated if it changes and at the same time can be traced back to the origins if it is required for example during an argumentation process. This step has also to act as a filter, moving to the *User cognitive profile* only the data that is really relevant as decided by the "experts/deducers" modules.

3.3 User Interface

The user interface section allows the user to interact with the system. Although you can think about any kind of user interface, most of the time it will be either

a web interface or, as it happens in our proof of concept, a mobile application. What is important is that the user interface gives enough flexibility to establish a dialog with the user. As we have commented, one of the advantages of adding a cognitive architecture in the flow of a recommendation process is the possibility of having argumentation processes between the user and the system. For this, it is necessary to have a user interface that facilitates this kind of dialog.

3.4 Cognitive architecture

This is the core of the framework. As we already said, our proposal is not to use the cognitive architecture as a recommender but as an advisor that lies between a set of recommenders and the user. In our first approach, and as it is illustrated in section 4, we use a BDI architecture. The idea, however, is to explore in the future other cognitive architectures like SOAR [4], Clarion [10] or ACT-R [7] among others.

The cognitive architecture is fed by information coming from three sources:

- **The data analysis block.** As commented before, it is the result of using statistical and data mining techniques in the analysis of historical data associated to the user plus the inference/filtering process performed by what we call the "experts/deducers". For instance, following our supermarket example, the kind of knowledge that could come from the data analysis block toward the cognitive architecture could be things like: "The user goes to the supermarket the first week of each month", "The user buys cheap products", "The user is loyal to brands", etc. Of course, all of this has to be codified in a form that can be understood by the specific cognitive engine.
- **The interaction with the user.** Using the user interface, the user can explicitly state information about him/herself like for example: "I like soy milk", "I want to lose weight", "I prefer high quality products", "I don't care about price", etc. Also the analysis of this interaction can give information to the cognitive architecture to understand how the user wants to receive the recommendations.
- **The recommenders.** The information coming from the recommenders usually will be a list of recommendations as a result of a previous query coming from the cognitive architecture. Some more advanced recommenders can attach a justification to the recommendation (that the cognitive architecture has to be able to understand), a degree of certainty for the recommendation or they can even start an argumentation process with the cognitive architecture.

All this information has to be introduced in the cognitive architecture in a form that can be understood by its reasoning mechanism. This, of course, is architecture dependent and it will vary a lot from one architecture to another. In section 4 we will show an example using a BDI architecture.

Loaded with this information, the cognitive architecture has to be able to:

- Receive a notification from the user expressing the willingness to obtain a recommendation of a certain kind.

– Analyze if that request is coherent with what it knows about the user and decide what is the best action to perform.
– Query one or several recommenders that, given the knowledge the cognitive architecture has about them, can provide an answer that will satisfy the user's request.
– Adapt, combine, modify the recommendations received from the recommenders to personalize the final answer to the user.
– Show the processed answer to the user in a personalized way.

This is the basic behavior that we will illustrate (in its simplest form) in section 4. However, as we already said, the motivation to use a cognitive architecture goes far beyond this initial capabilities. We expect in the future that the cognitive architecture be able to:

– Establish a dialog with the user. For example, if the request from the user is not coherent with what the cognitive architecture knows about him/her, establish a dialog with the user to express the problem and be able to adapt the internal knowledge based on what the user explicitly states during that dialog.
– Justify the recommendations. The justification has to be adapted to each kind of user. Some users prefer simple and short justifications while others want all the details.
– Participate in an argumentation process with the user where both, the user and the cognitive architecture can express arguments and counterarguments to achieve a consensus about a recommendation.
– Incorporate automatically new recommenders to the palette of recommenders and be able to use them without external intervention. This ambitious capability requires the description of each recommender in terms of its strengths and weaknesses.

3.5 Recommenders

This part of the framework is a repository of recommenders connected to the cognitive architecture through a web service. There exist a big amount of recommendation techniques, but as we already said, it does not exist the recommender that can be used in any situation. Therefore the idea is to offer to the cognitive architecture a broad spectrum of recommenders to chose from, so it can select which is the best (or how to combine several of them) given a certain context. The simplest way to approach that problem is to have an expert that codifies that knowledge at design time into the cognitive architecture (for example codifying specific plans that encapsulate that knowledge, as it is the case in our *proof of concept*). This, although a good solution for a first approach, limits the number of recommenders that can be used to those already present at design time.

A much better (although much more complex) approach that we plan to explore in the future is the development of a description language that allows to

describe recommenders in terms of strengths and weaknesses and that can provide clues about when and how those recommenders can be used. The cognitive architecture should be able to understand this language and therefore be able to reason about these strengths and weaknesses. This "understanding" of the recommenders should give to the cognitive architecture the capacity to decide, given a certain context, which recommender or recommenders is better to use. Ideally, adding a new kind of recommender to the system would imply only to describe the new recommender using the description language. Automatically the cognitive architecture would be able to incorporate the new recommender in the reasoning process with no further intervention. A positive side effect of this advanced reasoning process is that the cognitive architecture should be able also to justify why it is using a specific recommender or combination of them.

4 Proof of concept

In this section we present a use-case to illustrate the basic functionality of an instantiation of the Cognitive Recommendation Framework. At this point, the goal is not to show the full potential of the framework (something that goes far beyond the scope of this article), but only to present a set of possible technologies, put them in place, and illustrate the basic data flow in a running example. Consider this proof of concept a first instantiation of the ideas presented in the previous sections. In spite of being a preliminary work, hopefully the potential of the framework should become apparent.

The domain selected for our proof of concept is that of supermarket shopping. Let's start describing the scenario of our use-case:

> *Mary always buys the same yogurts. She is a little bit tired of eating the same yogurt one day after another and today she wants to try something new. She goes to the supermarket and once in the dairy products aisle, takes the smartphone and scans the barcode of her favorite yogurt. She asks to the system for an alternative. The system processes her request and suggests an alternative that could satisfy her requirements.*

In the following subsections we will present an instantiation of the general elements of the framework described in section 3 that is able to provide the functionality described in the use-case. We will start describing the data analysis block, then we will describe the user interface, the cognitive architecture and finally the recommenders block.

4.1 Data analysis

Pre-processing: The raw data used in our use-case is composed by a set of text files coming directly from points of sale terminals (POS) from the Alimerka supermarket chain[1]. Specifically we have more than 900000 files coming from 176

[1] Alimerka S. A. (http://www.alimerka.es) is a supermarket chain located in the north of Spain with supermarkets in the regions of Asturias, Galicia and Castilla y Leon.

supermarkets covering a period of 18 months. The files are grouped by supermarket (one file per supermarket per day) and contain purchases, promotions, coupons, loyalty points, etc. Given the big amount of data, the first action is to clean all the non relevant information. For us, only the purchases are relevant so we have removed the rest of information, like for example the promotions or coupons, from each file. Then, each line is parsed and we maintain only those fields that are really relevant (product, client id, price, hour/date of purchase, supermarket). The resulting files contain the purchases of different clients during the day in a given supermarket. However we need the purchases organized by client instead of by supermarket so it is necessary to reorganize the lines and group them by client. At the end of the process what we have is a set of structured files, one per client, with all the purchases of that client in the last 18 months. In our use-case we have restricted to 500 the number of clients and the time period to 12 months.

Data processing: In this step we prepare the user files created previously so they can be used by a recommender based on collaborative filtering (the kind of recommender used in our proof of concept). Specifically, because Mary is looking for an alternative to a product, we have to perform a collaborative filtering limited to the products that belong to the same family. Our recommender compares Mary with the other 499 users, establishes a similarity among them and uses what the most similar clients have bought that Mary has not as a recommendation (see section 4.4 for a more detailed description of how the recommender works). Given that, in order to prepare the data for the recommender we have to use only those lines that refer to products that are in the same subtree that the reference product. Because we don't know the product that Mary will chose as a reference product, the process of filtering the client files has to be done at run time although our tests demonstrate that this is not a problem in terms of performance.

Experts/Deducers: Our simple use-case does not incorporate yet the use of experts/deducers.

The elements in the data analysis block have been programmed in *Python*.

4.2 User Interface

For the user interface we have developed a smartphone app (see figure 2). The app has been implemented using the *phonegap*[2] framework and *jquery mobile*[3]. *Phonegap* and *jquery mobile* are open source solutions for building cross-platform mobile apps with standards-based web technologies like *HTML5, JavaScript and CSS3*. We have chosen these technologies because they are device independent (the same code can be executed in iOS and Android) and are ideal to build prototypes quickly. We also have used the *Barcode Scanner* plugin for phonegap that allows to scan bar codes using the camera device. We developed this app using the *Xcode* environment and then it has been tested in an *iPod Touch*. The

[2] http://phonegap.com
[3] http://jquerymobile.com

connection between the app and the cognitive architecture is implemented using a *RESTful* web service.

The sequence of screenshots that are presented to Mary in the use-case is illustrated in figure 2. First Mary has to log into the system (Fig 2 (a)). This identifies Mary and will allow the system to instantiate a cognitive engine loaded with her profile and that will be in charge of her request. After that, Mary can select the kind of recommendation that she wants (Fig 2 (b)). In our example there is only one possibility: get an alternative product. The product of reference is scanned in the next screenshot (Fig 2 (c)) and all this information is sent to the cognitive architecture. Finally, once finished the internal process described in the following sections, Mary receives the recommendation. In the case illustrated in this sequence, she was asking for an alternative to "YOGUR DANONE[4] NATURAL C/AZUCAR 125P4" (plain yogurt) and the system recommends "YOGUR DANONE LIQ.DAN UP FRESA 600ML" (strawberry liquid yogurt).

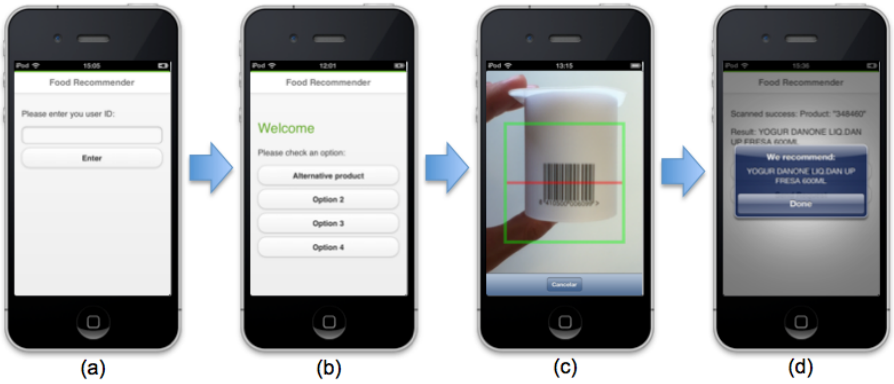

Fig. 2. Smartphone app

4.3 Cognitive architecture

The cognitive architecture selected for our proof of concept is a BDI architecture. Specifically we use *Jason* [8], an interpreter for an extended version of *AgentSpeak* and a platform for the development of multiagent systems. *AgentSpeak* is an agent-oriented programming language that is based on logic programming and the BDI architecture for (cognitive) autonomous agents. In our case we are using only the *AgentSpeak* interpreter, that is, the BDI engine, because we are not considering yet a multiagent environment. However, in the future we plan to work on cooperative recommendation and the inter agent communication

[4] DANONE is a registered trade mark of group Danone S.A.

facilities that provides Jason can be very helpful. Jason is developed in *Java* and is distributed under *GNU LGPL*.

Coming back to our use-case, when Mary uses the user interface to ask for an alternative product, the smartphone app sends the request to the server. A web-service that is waiting for the request, collects it and creates an instance of a Jason BDI engine that is loaded with Mary's cognitive profile. In our simple example, the cognitive profile is almost empty (it only contains a few simple plans that we will detail later) but in a real situation it would contain all sort of information about Mary expressed in AgentSpeak.

Also as a consequence of the request performed by the smartphone app, a new belief is introduced into the Jason BDI engine indicating that there is need for a recommendation. Specifically that Mary is looking for an alternative to the product `productSel`. The belief has the form:

```
get_alternative(id, productSel).
```

where `id` is a unique identifier for the request. As a consequence of the addition of a new belief, the Procedural Reasoning System (PRS) of the Jason BDI engine looks for a plan in the plan library that can be fired. In our example, Mary's plan library contains the following plan:

```
+get_alternative(ID, X) : true <- ask_collaborative_filtering(ID, X).
```

A plan in Jason has the form:

```
triggering_event : context <- body.
```

where the `triggering_event`, as the name indicates, is the event that triggers the plan (in our example the addition (`+`) of the belief `get_alternative(ID,X)`); the `context` is the set of preconditions of the plan (`true` meaning there are no pre conditions) and the `body` is a sequence of actions (among other elements) that conform the plan itself. The term `ask_collaborative_filtering(ID, X)` is what in Jason is called an action and basically it is the call to a java method. This particular method generates a web service request asking for a recommendation that is based on collaborative filtering.

After the interaction with the recommenders' web service, the Jason's web service receives the recommendation in XML format. The XML recommendation is translated to an AgentSpeak belief that is introduced in the Jason's belief data base with the form:

```
possible_rec(id, list_of_products).
```

This belief expresses that there is a possible recommendation for the request `id`. This step is important because we don't want the recommender to send the result directly to the user, we want the cognitive architecture to refine the recommendation taking into account all the information that it has about the user and the context. For example, if in the list of recommendations we have a banana yogurt and a strawberry yougurt in this order but we know the user likes a lot strawberries, it is better to recommend the strawberry yogurt even if it is not the first recommendation given by the recommender. Although this is

a simple example (we foresee a much more complex reasoning at this level), we think it illustrates how this fine degree personalization of the result can make a difference.

In our proof of concept, the plans that react to the addition of a "possible recommendation" are very simple and limited to take as a result the first product in the list returned by the recommender:

```
+possible_rec(ID, Y) : get_alternative(ID, X)
                         <- -possible_rec(ID, Y); +rec(ID, Y).
+rec(ID, [Head|Tail]) <- send_rec(Head).
```

Here `-possible_rec(ID,Y)` removes the "possible recommendation" and is substituted by a final recommendation (`+rec(ID,Y)`). When this new belief is added, the plan that reacts to the addition of a belief of type `rec(ID,[A|B])` is fired and, following a Prolog like approach, selects the first element in the list of answers and sends it to the user.

4.4 Recommenders

The request performed by the cognitive architecture asking for an alternative to a product arrives to the recommenders' web service. In our proof of concept there is only one single recommender available. Specifically the recommender we use in our use-case is a user-based collaborative filtering recommender from the machine learning library *Mahout* [5].

The functioning of these kind of recommenders is based on the idea that what is interesting for a user that is similar to me, probably will be interesting also for me. Following this idea the specific recommender we use works like this:

First the recommender prepares a matrix where the rows are the users (in our case 500 selected clients of the supermarket chain) and the columns the products. A '1' in a cell stands for that the user has bought that product while a '0' stands for the contrary. In our use-case we are interested in looking for alternatives to a certain reference product, therefore we have to limit the products to be considered by the recommender to those that belong to the same family that the reference product. This avoids that the recommender can propose a product that is too far from a semantic perspective. To do that we use the ontology that the supermarket chain is using at this moment to classify their products. As explained in section 4.1, the filtering of products for the recommender has to be done at runtime. For the specific use-case where Mary is asking for an alternative to a yogurt, the number of products considered by the recommender is 68 belonging to the sub family of 'normal yogurts'.

Using the user that has made the request as a target (Mary in our use-case), the recommender calculates the degree of similarity of that user with the rest. To calculate the similarity it uses the following formula (what is called the Tanimoto coefficient):

$$S = \frac{N_{a,b}}{N_a + N_b - N_{a,b}}$$

[5] http://mahout.apache.org/

where $N_{a,b}$ is the number of coincident products that have bought the users a and b, N_a the number of different products bought by user a and N_b the number of different products bought by user b.

Then, the recommender fills each cell that has a '0' in the row of the target user with the maximum of the similarities of those users that have bought that product. For example, if the target user has a '0' in the column of product X (meaning that the target user has not bought product X in the past), the recommender selects from all the users that have a '1' in that column the one that has the maximum similarity with the target user calculated using the Tanimoto coefficient. This similarity value is put in the cell.

After this process, the '0's in the row of the target user become a value that expresses the likelihood that the product in that column be interesting for the target user. With this values, the recommender creates a ranking of products that can be interesting for the target user.

This ranking is returned as the result of the request made by the web service and is the ranking that will be considered by the cognitive architecture to return the final recommendation to the user after considering the rest of information.

5 Conclusion and future work

What we have presented in this paper is a first approach to the use of a cognitive architecture in the context of a recommendation process. The idea, to the contrary of other previous works that use also cognitive architectures in this context, is not to use the cognitive architecture as a recommender but as a middle layer between the user and a set of state of the art recommenders.

We have presented also a first instantiation of the framework. This first prototype is still very primitive but we think it is illustrative of the potential that has the proposal.

As a future work we plan to explore the different improvements that have been enumerated in the paper. In particular we want to incorporate the argumentation mechanism and to evolve the idea of a language to describe recommenders so they can be incorporated and used automatically by the cognitive architecture at run time.

Acknowledgments

This work has been supported by the Ministerio de Economia y Competitividad of the Spanish government (SmartExFood project IPT-2012-0688-060000, CBIT project TIN2010-16306, AT project CONSOLIDER CSD2007-0022 INGENIO 2010); the Generalitat de Catalunya [2005-SGR-00093]; and the European Union (Sintelnet project FP7-ICT2009-C 286370).

References

1. Bajo J., Gonzlez A., D.L.A.S.A., J.M., C.: A Shopping Mall Multiagent System: Ambient Intelligence in Practice. University Castilla la mancha (2006)

2. Casali, A., Godo, L., Sierra, C.: A graded bdi agent model to represent and reason about preferences. Artif. Intell. 175(7-8), 1468–1478 (2011)
3. Casali, A., Godo, L., Sierra, C.: Modeling travel assistant agents: a graded bdi approach. In: Bramer, M. (ed.) Artificial Intelligence in Theory and Practice, IFIP International Federation for Information Processing, vol. 217, pp. 415–424. Springer US (2006)
4. Laird, J.E.: The Soar Cognitive Architecture (2012)
5. van Maanen, L., van Rijn, H., van Grootel, M., Kemna, S., Klomp, M., Scholtens, E.: Personal publication assistant: Abstract recommendations by a cognitive model. Cognitive Systems Research 11(1), 120–129 (2010)
6. Miao, C., Yang, Q., Fang, H., Goh, A.: A cognitive approach for agent-based personalized recommendation. Knowl.-Based Syst. 20(4), 397–405 (2007)
7. Niel Taatgen, C.L., Anderson, J.: Modeling paradigms in ACT-R (2006)
8. Rafael H. Bordini, Jomi Fred Hbner, M.W.: Programming Multi-Agent Systems in AgentSpeak using Jason (2007)
9. Ricci, F., Rokach, L., Shapira, B., Kantor, P.B.: Recommender Systems Handbook (2011)
10. Sun, R.: The CLARION Cognitive Architecture: Extending Cognitive Modeling to Social Simulation (2004)