A Knowledge-Based Approach to Augment Applications with Interaction Traces

Olivier Curé[1], Yannick Prié[2], and Pierre-Antoine Champin[2]

[1] Université Paris-Est, LIGM, CNRS UMR 8049, France
ocure@univ-mlv.fr
[2] Université Lyon 1, LIRIS, CNRS UMR 5205, F-69622, France
{yannick.prie,pierre-antoine.champin}@liris.cnrs.fr

Abstract. This paper presents a trace-based framework for assisting personalization and enrichment of end-user experience in an application. We propose a modular ontology-based architecture, to provide semantics for interaction traces, observed elements and their associated objects, and we extend existing inference services, with a declarative and generic approach, in order to reason with those interaction traces. We present the architecture of our framework and its reasoning levels, provide a proof of concept on a medical Web application, and emphasize that different kinds of actors can benefit from the supported inferences.

1 Introduction

We present an assistance framework for personalizing and enriching end-user application interactions. These features rely on user profiles generated from interaction traces that provide information on how data is used and created. Our declarative approach is based on explicitly modeled interaction traces consisting of recordings of *observed elements* (henceforth *obsels*) that are collected during the use of the application.

A first contribution is to provide a Description Logics (DL) [4] based approach to modeling traces by constructing a global knowledge base of entities and actions that can be reasoned upon. The semantics of interaction traces, obsels and their associated objects are provided through mappings to DL concepts or roles. DL formalism has been selected because it enables to reason in a sound and complete manner, and enables interoperability by underlying Semantic Web technologies. Another contribution is to define means of reasoning over various levels of interaction knowledge. Standard (*i.e.* subsumption) and non-standard DL inferences support these services. In order to represent uncertainty, a probabilistic approach is used. Various levels of reasoning are presented that support various kinds of user assistance based on the manipulated objects, traces, and user profiles. Moreover, we have implemented this approach in an existing medical application, and have shown its interest by adding several functionalities that have proven valuable to various actors of this system, from admin to end-users.

This paper is organized as follows. Sec. 2 is dedicated to related works. Sec. 3 details how an application can be augmented with our framework for managing interaction traces, while Sec. 4 describes the different reasoning approaches. In Sec. 5, we provide an evaluation on our running example, emphasize on the system's inferences adequacy.

A. ten Teije et al. (Eds.): EKAW 2012, LNAI 7603, pp. 317–326, 2012.

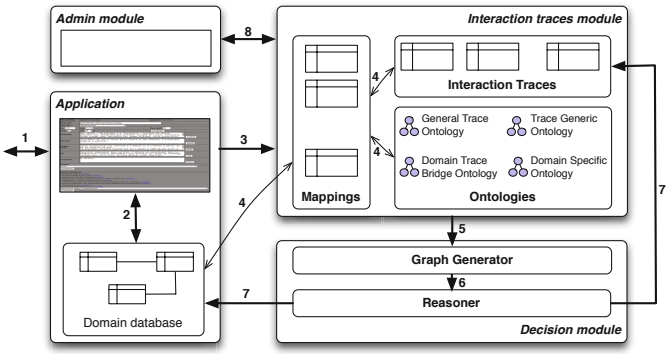

Fig. 1. Architecture overview of an application enriched with a trace-based system for assistance

2 Related Works

Interaction traces have long been considered a valuable source of information, either for an *a posteriori* analysis of the activity, or during the traced activity in order to provide some kind of user-assistance [13]. The latter is based on the assumption that the traces can help to automatically build a user's profile, instead of requiring the user's to manually provide their personal information and preferences [11]. Different paradigms have been explored to provide trace-based user-assistance, such as case-based reasoning (traces can be used to dynamically generate cases corresponding to a problem at hand [10]), or machine learning (traces are used as a training set for further recommendations [11]). Traces can also be modeled using ontologies, allowing complex inferences to be performed in order to assist the user [7]. Like our system, this work uses Semantic Web technologies to represent ontologies. There is indeed a growing interest in modeling temporal information related to users' activity on the Semantic Web [6]. [1] combine statistical analysis with a semantic model and rules in order to measure the health of on-line communities by analyzing the behavior of their users. There has been other work aiming at integrating time in DL [3] and Semantic Web technologies [2,12]; however those approaches consider time as a dimension orthogonal to the data. As such, they are not adapted to our approach where time is merely a property of every obsel.

3 A Framework for Integrating Interaction Traces in Applications

Fig. 1 describes the different information flows between the components of our architecture. The user interacts *(1)* with the application, causing updates to the domain database *(2)* and the interaction traces component *(3)*. In the latter, a subset of the end-user interactions is stored as temporally situated interaction traces, *e.g.* start of a session or description of an item. This subset of interactions is defined through a set of mapping assertions (*Mappings* component) between interaction traces and elements of the ontologies. Note that some mapping assertions also relate the database with the ontology *(4)*. A set of interaction traces, mapping assertions and ontologies is used by the *Graph generator* to define a global graph *(5)*, containing the overall information involved in the process of describing interactions. Moreover, it integrates metadata such

as object specific information coming from the ontologies and/or the domain database. The global graph is passed to the reasoner *(6)* to generalize on the represented information, computing subsumption relationships between ontology concepts. The generalizations are both stored as (transformed) interaction traces and in the domain database *(7)*, *e.g.* to store end-user models which will be used to personalize the application together with information directly coming from the traces. Finally, the *Admin* module enables to manage and query information stored as raw or transformed interaction traces *(8)* to detect assets and drawbacks of the application, e.g. objects that are rarely accessed, *etc*.

Representation of Interaction Traces. We use the meta-model proposed in [9] for managing interaction traces: a trace is defined as a list of timestamped *obsels*, holding a set of attributes and relations. Each trace is related to a *trace model*, defining the different types of obsel the trace can contain, and which attributes and relations obsels of each type can have. A trace-based management system (TBMS) collects obsels from the application in so-called *primary traces*. A TBMS also manages *transformed traces*, which are computed based on the content of other traces, providing higher levels of interpretation. Hence we have three knowledge containers for each trace: its metadata , its model, and the list of contained obsels. In our framework, the set of obsels recorded in an interaction trace is the representation of an end-user's interactions with a given application. They mainly correspond to CRUD (*Create*, *Read*, *Update*, *Delete*) operations, but can be extended to higher level operations (*e.g. Copy*, *Paste*).

Each end-user session is represented by a single trace, satisfying the schema (*idT, idU, ssDate, seDate*) where attributes respectively identify an interaction trace, the subject of the trace (*i.e.* the end-user whose activity is recorded), the session start and end dates. Traces then contain a number of obsels describing the data operations performed within the application during the session. Obsels satisfy the schema (*idT, obselDate, idObj, op, objField, oldValue, newValue*) where *idT* references the containing trace, *obselDate* is the timestamp of the obsel (and must be comprised between *ssDate* and *seDate* of the containing trace), *idObj* identifies the object impacted by the interaction, *op* identifies the data operation performed on that object, *objField* identifies the object's field impacted by the operation, if appropriate, and *oldValue* and *newValue* contain, if appropriate, the field value before and after the operation respectively. The last three attributes are optional, depending on the kind of operation. That is a *Retrieve* operation does not impact a specific field of an object and has none of those three attributes; a *Delete* operation only has an old value, an *Insert* operation only has a new value and an *Update* operation has the three attributes filled in.

Ontologies for traces. A particular attention has been given to the ease of plugging in and out ontologies and to provide a comprehensive decoupled organization. The default setting of our framework consists of the following 4 ontologies. The *general trace ontology* provides a model for a high-level view of interaction traces. It contains 3 concepts: *Trace*, *Obsel* and *Subject* (the user whose interaction was traced); 2 object properties: *describes* and *composedOf*, respectively relating an interaction trace to its subject and its obsels; 4 datatype properties, supporting subject identification and temporal information. The *trace generic ontology* supports the definition of a set of obsels and corresponds to a hierarchy of concepts some of which subsume concepts of the *general trace ontology*. It aims to specify data operations needed to reason with interaction traces.

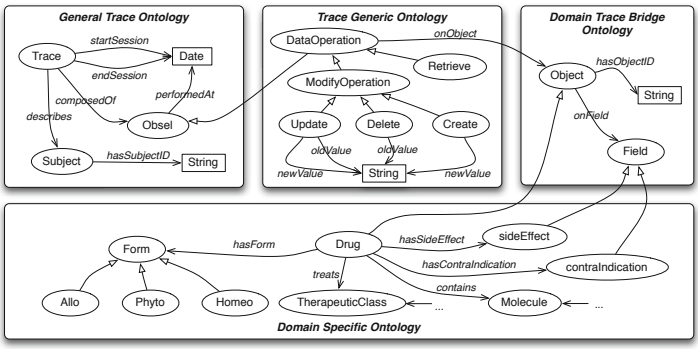

Fig. 2. Ontology-based framework for considering both traces and domain objets

The *domain specific ontology* describes the semantics of the objects observed in the interaction traces. In opposition to the other ontologies of the architecture, this component may integrate an ABox.Obviously, the more information is provided on observed objects (either in a TBox or an ABox), the more accurate the results of our inferences. The *domain trace bridge ontology* links the *trace generic ontology* to the *domain specific ontology*. It contains 2 concepts: *Object* and *Field*, and 2 properties to identify an object and relate an object to its fields. These 2 concepts subsume some of the concepts of the *domain specific ontology* and are related to some of the *trace generic ontology* concepts via object properties, *e.g. onObject* and *hasField*.

Example 1. Fig .2 provides a comprehensive view of a framework instance for our medical application. The *domain trace bridge ontology* provides information on drug objects which have a certain form (*i.e.* allopathy or homeopathy), treat some therapeutic classes and contain some molecules. Two other concepts of this ontology, namely *ContraIndication* and *SideEffect* are subsumed by the *Field* concept of the *domain trace bridge ontology*. Note that only *Drug* is subsumed by the *Object* concept from the bridge ontology, so only operations on drugs will be recorded in the trace. However, *Molecule* and *TherapeuticClass* can be used as external knowledge to support inferences on drugs and the related obsels.

If our framework was to be used in a completely different domain, the *domain specific ontology* would have to be replaced. The modular structure would however make it easy to reuse an existing Semantic Web ontology for the new application domain, and link it to the other ontologies through mappings to the *domain trace bridge ontology*.

4 Reasoning over Traces

We model interaction traces using a set of ontologies, using DL to provide them with model-theoretic semantics [4]. A DL knowledge base (KB) is composed of a TBox and an ABox which respectively correspond to a set of terminological axioms and concept/property assertions. A key feature of DLs is to integrate in the system the following set of standard inferences: concept satisfaction and subsumption, instance

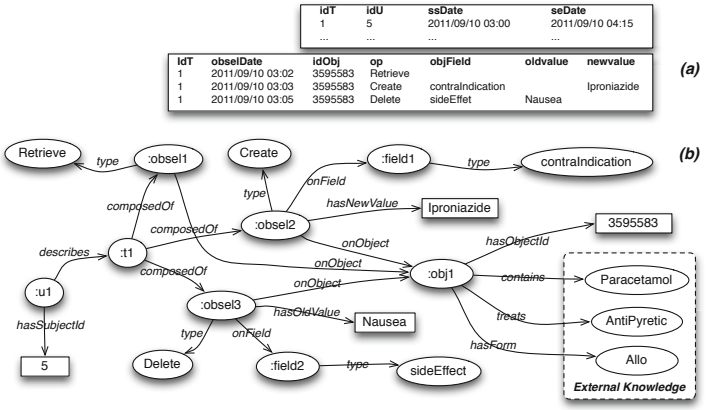

Fig. 3. A trace extract and its graph instance (with some data omitted for readability)

checking, realization and retrieval. Although we use some of these inferences, we also adapt two non standard DL inferences named Most Specific Concept (MSC) and Least Common Subsumer (LCS) [4]. The MSC of an individual α wrt a KB is the concept description C such that (i) α is an instance of C and (ii) if there is another concept C' of which α is an instance of, then C' is more general w.r.t. subsumption than C. The LCS of a given set S of concept descriptions is the most specific concept (wrt the subsumption relationship) subsuming all concepts descriptions in S. In general, they are computed with either a structural subsumption algorithm or via the specification of a normal form. Within our framework, MSC is used to generate DL concepts (hence models) from individuals corresponding to instances of interaction traces, obsels or their objects. Given these instances, the system generates concept description in the $\mathcal{EL}$ DL [5] which underpins the OWL2 EL fragment. This DL presents good properties in terms of reasoning: subsumption is polynomial even if one allows for cyclic terminologies and the MSC of an ABox individual always exists. Moreover, users' goals and intentions are too complex to be entirely accounted for by formal inferences. To handle this uncertainty, we use Probabilistic $\mathcal{EL}$[8], an extension of $\mathcal{EL}$ that associates probabilities to concept descriptions.

As seen in Fig. 1(5), the reasonner works on a global graph generated from different knowledge sources. Fig. 3(a) presents some interaction traces performed over our medical application while Fig. 3(b) displays its associated instance graph. Note that the *contains*, *treats* and *hasForm* properties associated to the *:obj1* node can not be generated from the interaction traces. In fact they correspond to external knowledge retrieved from the domain database. This is a difference with the original TBMS approach [9]. A major advantage of using ontologies is the ability to generalize patterns discovered with type inference, *i.e.* the most common concept that satisfies a given situation.

Inferencing over various levels of interaction knowledge. The analysis of the end-user's behavior can be performed at different levels. We propose 5 of them which cover a wide range of relevant features and still support a generic approach. They correspond to the main DL concept abstractions found in our ontologies: *Field*, *Object*, *Obsel*, *Trace* and

Subject. These levels require different forms of processing: a finite model checking approach handled with a simple query answering interface; or a more elaborate logical reasoning. Inference services provided by each level build on the previous ones.

The *Field level* is the most specialized level and little information can be retrieved from it. Intuitively, it enables to retrieve which kind of fields specified in the *Domain specific ontology* have been observed. Hence a simple query answering interface is sufficient. For instance, the question: "Which fields have been observed?" may be answered with an enumeration of subconcepts of the Field concept that have been effectively observed. This query may help to detect which fields are never operated upon.

The *Object level* is central as it is the first one to require logical reasoning and all remaining levels build on its inferences. Both query answering and logical reasoning are needed at this level. For instance, query answering can be used to identify objects operated upon. The main logical reasoning of this level enables to generalize on a given object description (detailed in Sec. 4).

The *Obsel level* enables to study the operation associated to an observed object. Typical queries are CRUD-like and reasoning amounts to generalize obsel description.

The *Trace level* is an interesting level due to the non-functionality of the *composedOf* property, *i.e.* a trace can be composed of several obsels. Moreover, an interaction trace is associated to some temporal values representing its begin and end session dates. Thus, a query answering approach can answer to questions such as "When was a trace recorded? How long did it last?". Logical reasoning is mainly concerned with generalizing a given interaction trace. An interesting aspect of reasoning at this level involves considering ordered sequences of generalized traces. By storing the result of that generalization as *transformed* traces, we can then query sequences of similar sessions.

Finally, the *Subject level* enables to infer over a set of interactions performed by an end-user. Query answering can be used to reply to the following questions for a given end-user: "When did this user use the application for the last time?", "How often does this user use the application?". We can also consider queries involving a group of end-users, *i.e.* queries aggregating over end-users. Logical reasoning amounts essentially in defining an end-user *profile* (i.e. concept descriptions) given her set of interaction traces. In our running example, this user profile aims to define her pharmacology expertise, *i.e.* in terms of molecules, therapeutic classes and drug forms she is an expert in.

Reasoning method. The reasoning methods needed by our levels of analysis are based on the following 3 algorithms. The first algorithm, *simpMSC*, corresponds to a simplified version of MSC and aims to generate a DL concept description for any individual present in the global graph. These DL concepts are specified by the following normal form: $\bigsqcap A_i \sqcap \bigsqcap \exists r_j A_k$ where A_i and A_k are atomic concepts and r_j is an object property. The algorithm takes as input an individual α and acts as follows. For all assertions of the form $\Gamma(\alpha)$ (*i.e.* defining the type of α), it creates a conjunction AC of Γs. Moreover, for all assertions of the form $r(\alpha, \beta)$ with r an object property assertion, it creates a conjunction EQ of $\exists r.\beta$. The algorithm returns as output $AC \sqcap EQ$. This algorithm is used at both the *Object* and *Obsel* levels respectively to produce a DL concept for an object in terms of elements of the *Domain specific ontology* and an obsel in terms of elements of the *Trace generic ontology* and *Domain specific ontology*.

Example 2. Consider the *:obj1* individual in Fig. 3. Its assertion set contains: a type definition *Drug*(:obj1), 3 object property assertions (*contains*(:obj1, Paracetamol), *hasForm*(:obj1, Allo) and *treats*(:obj1,AntiPyretic)) and a data type property assertion: (*hasObjectID*(:obj1,3595583)). The execution of *simpMSC* over this individual returns: $Drug \sqcap \exists contains.Paracetamol \sqcap \exists treats.AntiPyretic \sqcap \exists hasForm.Allo$.

The second algorithm, *probLCS*, corresponds to an approximation of LCS extended with probabilities on concepts. We use the notation of [8]: a concept $Trace \sqcap \exists composedOf(P_{=1/2}Retrieve \sqcap \exists onObject.O1)$ describes an interaction trace composed of an obsel on object O1 and a *Retrieve* data operation with probability 0.5. We restrict probabilities to occur in front of DL concepts and allow disjunction in the form $A \sqcup B \sqsubseteq C$ since it can be eliminated modulo the introduction of the new concept C.

The input of this algorithm is a set of concept descriptions produced by *simpMSC*. In a first step, a disjunction of atomic concepts is computed. That is, it counts the number of occurrences of each non quantified atomic concepts over the set of descriptions and produces a probabilistic formula for each of them where the probability is the number of occurrences of this concept divided by the total number of occurrences of the set. In the second step, for each distinct existentially quantified property, the first step is applied over its set of concepts. The *probLCS* algorithm serves at the *Trace* level to specify a probabilistic view of the obsels present in an interaction trace.

Example 3. Consider the obsels of Fig. 3 (*i.e.* :obsels$_i$ with i ∈ [1,3]). Applying *simpMSC* to these obsel individuals yields the respective concepts (where O1 is the concept description from Ex. 2): $Retrieve \sqcap \exists onObject.O1$, $Create \sqcap \exists onField.ContraIndication \sqcap \exists onObject.O1$, $Delete \sqcap \exists onField.SideEffect \sqcap \exists onObject.O1$. Computing *probLCS* over this set yields the following concept: $(P_{=1/3}Retrieve \sqcup P_{=1/3}Create \sqcup P_{=1/3}Delete) \sqcap \exists onField.(P_{=1/3}$ ContraIndication$\sqcup P_{=1/3}$ SideEffect$) \sqcap \exists onObject.P_{=3/3}O1$

The last algorithm, *setProb*, operates on a set of probabilistic concepts to create a generalized concept, using the notion of concept comparability.

Definition 1. The comparability property, defined over the normal form of a probabilistic concept, *i.e.* $\bigsqcap(SC) \sqcap \bigsqcap(\exists rSC)$ where SC, standing for Simple Concept, is a disjunction of probabilistic atomic concept, is stated as follows: non quantified simple concepts are comparable and simple concepts existentially quantified by the same property are also comparable. No other concepts are comparable.

Given this comparability property, the generalization of probabilistic concepts is defined as summing the probabilities of comparable simple concepts. To compute a relevant sum, all probabilities are expressed over the total number of occurrences found locally. This approach enables to sum properties and to have a coherent global view of the distribution of probabilities. The *setProb* algorithm is used at the *Subject* level when a generalization of interaction traces described by a given end-user is required.

Example 4. Consider the composition of the 2 following interaction traces:
- $(P_{=1/3}Retrieve \sqcup P_{=1/3}Create \sqcup P_{=1/3}Delete) \sqcap \exists onField.(P_{=1/3}$ ContraIndication $\sqcup P_{=1/3}$ SideEffect$) \sqcap \exists onObject.P_{=3/3}O1$
- $(P_{=2/4}Retrieve \sqcup P_{=2/4}Update) \sqcap \exists onField.(P_{=2/4}$ SideEffect$) \sqcap \exists onObject.P_{=4/4}O2$

The execution of *setProb* over these interaction traces yields: $(P_{=3/7}Retrieve \sqcup P_{=1/7}Create \sqcup P_{=1/7}Delete \sqcup P_{2/7}Update) \sqcap \exists onField.(P_{=1/7}$ ContraIndication $\sqcup P_{=3/7}$ SideEffect) $\sqcap \exists onObject.(P_{=3/7}$ O1 $\sqcup P_{4/7}$ O2)

The probabilities of the concept description computed by *setProb* represent the uncertainty of our conclusions about the user's activity at a certain level. In the context of the *Subject* level, this concept description serves to create an approximation of a user model. A threshold θ is used to cope with the uncertainty of our approximation.

Definition 2. Given a probabilistic DL concept, a certain rewriting of this concept wrt a threshold θ is computed by retaining only the inner concepts whose probability is superior or equal to θ.

The certain rewriting of a probabilistic concept wrt to a threshold θ serves to generate a model for a given subject. That is the remaining concepts of the description specify the expertise/point of interest of the end-user.

Example 5. Consider the probabilistic concept generated in Ex. 4 and a threshold θ=1/3. The certain version corresponds to: (*Retrieve*) $\sqcap \exists$ onField.(SideEffect) $\sqcap \exists$ onObject.(O1 $\sqcup$ O2) where the drug O1 (resp. O2) has form Allo, treats Cough and contains Dextromethorphan (resp. has form Phyto and treats the respiratory system). This enables to define a user model corresponding to a domain expert involved in *Retrieve* data operation and side effect field over allopathic and phytotherapeutic drugs of both the Cough and respiratory system, containing the Dextromethorphan molecule.

Finally, testing subsumption of concepts filling the *onObject* property is performed. Intuitively, using the Domain specific ontology, the system checks for concepts covering wrt. to subsumption. Thus a set of subsumed concepts are replaced with the super concept. This approach limits the size of generated user models and improves their relevance by relating them to named concepts rather than complex concepts expressions.

5 Trace-Based Assistance in a Self-prescription Application

Reasoning occurs at the different levels of interaction knowledge. Several kinds of actors (end-user, staff manager and developer) can benefit from various inference services.

At the *Object* level, the system can efficiently compute statistics on frequently modified objects and fields and hence provide up-to-date information on market evolution.

The *Obsel* reasoning level enables to detect repetitive operations performed on same category objects. The system then proposes the end-user to execute them automatically on the remaining objects in this category, as soon as it has been repeated more than a given threshold (defaulting to 3). This kind of information also supports the discovery of some integrity constraints, e.g. which fields or objects are most frequently updated for the drug category of homeopathy or respiratory system. For example, it is relatively frequent in the medical domain that some molecules are being attributed new properties. This may cause drugs containing that molecule to change from OTC to requiring a prescription. After the end-user has changed the state of 3 such drugs, the system will offer to make the change to all remaining drugs. Before the adoption of our trace reasoning level, all modifications had to performed manually.

In our medical application, all modifications are double-checked by an expert of this product's category, *i.e.* either form, molecule or therapeutic class. For instance, a homeopathy expert usually does not have the expertise to check information on the allopathy drug category. The *Subject* level automatically generates this expertise profile for each user based on their interaction traces. Based on these user models, we are able to personalize the home page of each end-user in the following way: (i) she is only asked to check drugs in her domain of expertise, (ii) she is provided with a list of contacts in her domain of expertise and (iii) health news displayed in the application are prioritized according to her domain of expertise. This approach has simplified the tasks of our team of health care professionals and has significantly lowered time delivery of data updates.

The administration tool is also very useful to improve our system. Due to different forms of reasoning, it enables to identify drug categories not frequently operated upon (obsel level), to control the cleaning/checking activity of a given domain expert (trace level), to detect domains of expertise missing in a team of end-users, to discover who checks and/or overrules the updates of a given expert (subject level), *etc*.

Evaluation. We have conducted an evaluation to highlight our framework's assets and weaknesses on certain criteria related to *Subject* level reasoning and user profiles: a) productivity gain of end-users, b) correctness of the models generated for each end-user. The experimentations have been conducted on our medical application over a real dataset involving 12 health care professionals over 3 months and resulting in the recording of 420 interaction traces and over 23000 obsels.

The first experiment tackles the gain of productivity of end-users. The experimentation took place after a period of 2 months of recording interaction traces.None of them were aware of the existence of the framework. Given the user models generated after the recording of 2 months of interaction traces, we divided the group of 12 end-users into 3 homogeneous (*i.e.* based on the precision of their models) groups of 4 persons. Then over a period of 3 weeks, we conducted a survey over the evolution of the precision of the *check box*. This box is displayed on the home page of each end-user and contains a list of drugs that have been recently modified and which need a checking by another health care professionals before being put into production. Hence, it is an invitation for end-users to control the information associated to a given drug. For group #1, the *check box* did not benefit from the inferences of the framework. The end-users hence had to browse the list of drugs to find by themselves the ones in which they have some expertise. For group #2, the *check box* was progressively taking benefit from the 2 months of analysis. That is in the first week, the box benefited from the user model deduced after 3 weeks of analysis, in the second week, it benefited from 6 weeks of analysis and in the last week it used the models generated over the 2 months of analysis. In the case of group #3, the *check box* benefited from the 2 months of analysis right from day one of the experimentation. Participants in each group rated the adequacy of the drugs presented in the *check box* according to their medical expertise. A Likert scale with 5 ordered responses was used, ranging from '5=strongly adapted' to '1= not adapted'. Fig. 4(a) emphasizes that group #1 does not see any improvement in adequacy of the box list. As expected, through the period of the evaluation, participants of group #2 sensed an improvement in the adaptability of the proposed list of drugs. Finally, the improvement was felt right away for members of group #3.

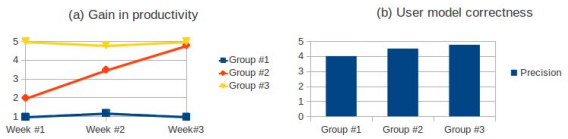

Fig. 4. Results of experimentations

The second experiment was conducted right after the first one and once all end-users were informed about the existence and aim of the framework. We presented a comprehensive view (*i.e.* presented in natural language) of the user model generated for each participants (out of the 2 months plus 3 weeks of storing interaction traces on experiment #1) and asked them to rate (using the same Likert scale as in (1)) the precision of their profile. Fig. 4(b) presents the average of individual results over members of the 3 groups. The averages range from 4.5 for group #1 to 4.75 for group #3. We consider these results to be satisfactory as it rewards the work invested on this framework. Anyhow, the quality of these results is related to the well-defined expertise of each health care professional participating to the project.

References

1. Angeletou, S., Rowe, M., Alani, H.: Modelling and Analysis of User Behaviour in Online Communities. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 35–50. Springer, Heidelberg (2011)
2. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: EP-SPARQL: a unified language for event processing and stream reasoning. In: WWW, New York, NY, USA, pp. 635–644 (2011)
3. Artale, A., Franconi, E.: Temporal description logics. In: Fisher, M., et al. (eds.) Handbook of Temporal Reasoning in Artificial Intelligence, vol. 1, pp. 375–388. Elsevier (2005)
4. Baader, F., et al. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
5. Baader, F., Brandt, S., Lutz, C.: Pushing the el envelope. In: IJCAI, pp. 364–369 (2005)
6. Champin, P.-A., Passant, A.: SIOC in Action – Representing the Dynamics of Online Communities. In: 6th Int. Conf. on Semantic Systems, I-SEMANTICS 2010, Graz, Austria (2010)
7. Groza, T., Handschuh, S., Müller, K.: The NEPOMUK project - on the way to the social semantic desktop, Graz, Austria (2007) (peer-reviewed)
8. Gutiérrez-Basulto, V., Jung, J.C., Lutz, C., Schröder, L.: A closer look at the probabilistic description logic prob-el. In: AAAI (2011)
9. Laflaquière, J., Settouti, L.S., Prié, Y., Mille, A.: A trace-based System Framework for Experience Management and Engineering. In: EME 2006 in Conjunction with KES 2006 (2006)
10. Smyth, B., Briggs, P., Coyle, M., O'Mahony, M.P.: A Case-Based Perspective on Social Web Search. In: McGinty, L., Wilson, D.C. (eds.) ICCBR 2009. LNCS, vol. 5650, pp. 494–508. Springer, Heidelberg (2009)
11. Sugiyama, K., Hatano, K., Yoshikawa, M.: Adaptive web search based on user profile constructed without any effort from users. In: WWW, New York, USA, pp. 675–684 (2004)
12. Tappolet, J., Bernstein, A.: Applied Temporal RDF: Efficient Temporal Querying of RDF Data with SPARQL. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 308–322. Springer, Heidelberg (2009)
13. Wexelblat, A.: History-rich tools for social navigation. In: CHI 1998 Conference Summary on Human Factors in Computing Systems, CHI 1998, New York, NY, USA, pp. 359–360 (1998)