Providing assistance by reusing episodes stored in traces: a case study with SAP Business Objects Explorer

Raafat ZARKA¹, Amélie CORDIER², Françoise CORVAISIER³, Alain MILLE²
¹SAP Business Objects, INSA de Lyon, LIRIS, raafat.zarka@sap.com
²Université Lyon 1, LIRIS, {prenom.nom}@liris.cnrs.fr
³SAP Business Objects, francoise.corvaisier@sap.com

Abstract

It is a big challenge in the information technology field to develop techniques to help users in their tasks. For that purpose, we need to develop assistants able to help people without disturbing them in their main task. This paper discusses building an assistant based on traces and making use of a trace-based system. Traces can be used as a knowledge source to discover other useful knowledge but also to reuse experience. The main idea of the approach is to find useful episodes (previous experiences) in interaction traces and to reuse them to provide users with contextualized help. The assistant can then adapt the retrieved episodes to provide assistance to the user by reusing previous experiences. This work has been done in partnership with SAP-BO. We have implemented our proposal in the SAP explorer project which aims to ensure that all business users have easy access to all the information they need to make confident decisions based on up-to-date, reliable information, so they need to help business users while doing their tasks. The main contribution described in this paper is the algorithm enabling us to retrieve past episodes corresponding to a “task signature” (a description of the main characteristics of the episode). For that purpose, we have implemented a solution based on Finite State Machines.

Keywords: Assistant agent, Trace-Based Reasoning, similarity measure, task signature, Finite State Machine, sequence pattern mining.

1 Introduction

Everyday new software products appear in different domains, and a major challenge is to make their usage as easy as possible. This challenge leads us to focus on various ways to develop intelligent applications more flexible and scalable. For example, there is a growing interest in developing auto-adaptable applications able to fit specific user needs, or able to provide them with a relevant assistance at the right time and the right place. This adaptability of application is a major concern for groups such as SAP. Indeed, for effective decision making, business users require quick, easy answers to off-the-cuff questions and a better understanding of the business, without extensive training and dependence on IT. This is the reason why they need reactive applications. It was also the main goal for SAP when developing SAP-BO Explorer software. SAP-BO Explorer is software offers users an intuitive path to quickly search and explore data for instant insight into their business [1]. But still, the application is hard to master for most users, and additional assistance would be useful. To help reaching this goal we work on developing assistant enabling users to explore the system easily, to reuse their previous experiences and to benefit from advance reasoning functionalities provided by the system.

This paper presents a preliminary research work, in the context of a Master Thesis. We propose to make use of the trace-based reasoning (TBR) paradigm in the development of an assistant for a given application. While using a system, a user leaves behind him “interaction traces”. Because interaction traces capture users’ experiences, they can be exploited to provide experience-based assistance thus helping the user to conduct his task efficiently. Hence, in TBR, interaction traces are used as a specific knowledge container. In this field, our contribution is to provide an algorithm to retrieve “episodes” (previous experiences) in a trace and to reuse them to provide assistance to users in a specific context. The episode is a sequence of interactions that allow solving a problem. The algorithm takes as input the “current” trace and makes use of a base of

¹This work was supported by SAP Business Objects, 157-159, rue Anatole France, 92309 Levallois-Perret Cedex, France
“task signatures” and a base of traces. A task signature is a formal description of a specific task for which a user might need assistance. Task signatures are given by experts in the form of rules that are transformed in automata. The proposed algorithm processes this automaton. This work is supported by SAP Business Objects and we are working on a specific application, SAP-BO Explorer. We have instrumented the application to be able to collect interaction traces. The examples provided in this example come from the demo dataset of the application.

This paper is organized as follows. In section 2, we describe the context of this work and we do a short survey on trace-based systems as well as assistance systems. Section 3 presents the general architecture of the assistant and gives definitions and models required to understand our retrieval algorithm. Then, the main algorithm is described and the assistance principles are illustrated. Section 4 discusses our approach and the issues it raises. The paper ends with a conclusion in section 5.

2 State of the art

In computer science, traces have many shapes and usages; log files, navigation history, versioning. That is why many researchers were interested to study the benefits of using these traces to develop more powerful applications with the trace of interactions. The SILEX team2 (Supporting Interaction and Learning by Experience) is interested in this topic and has proposed the notion of $M$-Trace (Modelled Trace) and of Trace Based System [2].

There are various researches on traces, aiming at proposing models and tools for representing, transforming, sharing and visualizing traces of users’ experiences. In [2] the authors present a new approach for modelling user’s experience while using a computer system, with the objective of reusing this experience as knowledge in context for helping users along their tasks. This work has evolved, and recent approaches propose to handle traces using a Trace-Based System (TBS). A TBS can be seen as a kind of Knowledge-Based Systems (KBS) whose main source of knowledge is the set of traces representing user-system interactions and evolving with users’ activities. In [3], the authors presented the general architecture of the TBS framework, and they specified a language to describe $M$-Traces (i.e. modelled traces), $M$-Traces based patterns, queries, and transformations. The work presented on this paper uses this formalization.

While working with traces, one big challenge is to define trace models, because models should be defined according to the target application. That is why the trace theory has defined the concept of $M$-Trace (modelled trace), that defines the vocabulary of the trace obsels (a trace consists of a set of observed elements which has been abbreviated in obsels), the properties of the obsels, the relationships that may exist between these elements, and the way to timestamp each obsel. According to $M$-Trace, an episode is a sequence of obsels correspondent to the interactions that allow solving a problem. With the availability of $M$-trace concept and by using the Trace Based Management Systems (see section 3.1) new trace-based applications can be dynamically designed. For example, [4] designed a model for addressing issues related to traces modelling and visualization in synchronous collaborative learning activities, to allow users to visualize and analyse their own experiences by observing their actions on the learning system. They proposed a generic trace model for synchronous collaborative activity based on the notion of interaction in two modes: whiteboard sharing and text chatting. Another example is [5], where the authors presented an approach to context-aware assisting systems relying on traces of user interactions. They used the notion of Explained Task Signatures (ExTaSi) to describe tasks performed by users. The system collects traces and transforms them. The assistance process is triggered when the current trace matches the beginning of an ExTaSi. Then, the assisting agent searches the TBS for all the occurrences of this ExTaSi in the interaction trace. These occurrences can be ranked according to a similarity measure defined for the current trace. However, we may note that, according to the authors, the assistant agent requires some actions from the user to validate the tasks that make sense to him.

Another related work, described in [6], is about building systems that can ease the sharing and re-using of experience between large communities of users. The authors proposed to use traces of interactions as a new way of performing CBR (Case-Based Reasoning) to enhance the traditional CBR by providing more flexible ways of reasoning from experience. They showed that the traces can be used to record experiences

2 http://liris.cnrs.fr/silex
“in context” and thus, bring more flexibility to traditional CBR. The agile CBR idea described by Susan Craw in [7] shares the same point of view, where the main idea of agile CBR is to transform traditional CBR into a “dynamic, knowledge-rich, self-organizing, cooperative problem-solving methodology”. [6] has raised a lot of research issues, especially for building Trace-Based Assistants (TBA). Some of them, such as “how to define a similarity measure”, will be discussed in our contribution.

Most CBR systems try to solve problems by using log files. For example [8] presents a new scheme for coding web server log data into sessions of behavioural sequences. Where [9] propose a Ceaseless CBR model that can enrich the CBR diagram that considers the CBR task as on-going rather than one-shot, to support the analysis of unsegmented sequences of observational data stemming from multiple coincidental sources. A hybrid neuro-symbolic system has been presented in [10], this model takes into account the temporal dynamics contained in the sequences and allows to avoid problems related to the comparison of different length sequences. Another example is [11], where the authors presented case-based system for cooperative information browsing, called “Broadway”. This system follows a group of users during their navigations on the WWW (proxy-based architecture) and advises them by displaying a list of potentially relevant documents to visit next. The advices are based mainly on similarity of ordered sequence of past accessed documents. While in our work we make use of traces on which we can do transformations to be able to have more abstract traces which is not possible with raw logs.

One major problem in trace-based reasoning is to retrieve episodes in traces. To tackle this issue, data mining techniques could be helpful. Sequential pattern mining was first introduced in [12] for sequential database that consists of sequence of ordered events. This algorithm aims at finding frequent occurrences of sequences allowing to describe the data or to predict future data. Time-related data mining is an active research area. In [13] we can find a comprehensive description of this domain. Existing methods only focus on the concept of frequency because of the assumption that sequences’ behaviours do not change over time. The environment from which the data is generated is often dynamic, however, so the sequences’ behaviours may change over time. To adapt the discovered patterns to these changes, new concepts are incorporated into traditional sequential pattern mining [14].

We aim to develop an assistant to help users doing their tasks effectively, but developing assistants is not an easy work; because sometimes they can have negative effects. For example, the Office Assistant drew a strongly negative response from many users; that is why Microsoft has removed the office assistant in the new versions of Microsoft Office and replaced it with a new online help system [15]. The Office Assistant was a feature to assist users by way of an interactive animated character, which was interfaced with the Office help content. It used technology initially from Microsoft Bob and later Microsoft Agent, offering advice based on Bayesian algorithms. It popped up when the program determined the user could be assisted with using Office wizards, searching help, or advising users on using Office features more effectively. It presented tips and keyboard shortcuts. For example, typing an address followed by "Dear" would cause Clip pit to pop up and say, "It looks like you're writing a letter. Would you like help?". According to some studies, even if the recommendation was sometimes helpful, in most of the situation, users considered it as annoying. Often, the problem with assistant is that they are designed to assist “prototypical users”. However, assistance is a very specific task which is efficient when it takes into account the needs of specific users. This probably is the reason why a lot of assistant agents have failed. In the context of this work, we aim at designing auto-adaptative assistants able to take into account specific needs of users. We are aware that this goal involves a lot of other issues (ergonomics, human-computer interactions, etc.), but they are out of the scope of this paper.

3 Trace-Based Assistant: an approach based on FSM principle

In our paper we are interested in reusing experience of the user to help him achieving his tasks effectively. Suppose that the user is interacting with the system for doing a specific task, the assistant will automatically observe what the user is doing, then apply the same transformations on the past traces using tasks signatures to the current traces, while task signatures are already defined and stored in the task signatures base. After that, the assistant will retrieve and present to the user the past episodes that correspond to his current task, these episodes will be ranked using similarity measures, and adapted to the user to be able to reuse them.
In order to build our Trace-Based Assistant (TBA) we need to collect the traces and manipulate them by using a Trace-Based Management System. The assistant will be connected with, executing queries to find the related episodes or even to add new experiences. Figure 1 shows the general architecture of our trace-based system, we can see that there are three main parts, the system for which we want to provide the user with help, the TBMS that collects the traces of the user while he is working on the application and support many functionalities like transformation, querying, and visualizing by using the task signatures in the tasks base, and the assistant that observes the way the user interacts with the system, to help him doing his task effectively, by retrieving the related episodes from the TBMS and adapting them.

3.1 Trace-Based Management System

We can see a TBMS as a Database Management System, by considering these mappings: Trace models / ERD model, obsels/data as the main objects in this system, transformation, querying, and visualization functionalities. [16] defined the notion of Trace Based Management System (TBMS) as systems devoted to the management of modelled traces. A modelled trace is a trace explicitly associated with its trace model. Each trace consists of a set of elements called obsels which results from the observation of the interaction between the user and the system, and each obsel has a set of attributes/values. A modelled trace formally defines the structure and the types of obsels, and the relationships between these obsels. It can be considered as a formal ontology that describes the vocabulary of the trace.

The various traces are managed by the Trace-Based Management System. The collecting process is the first function that builds the primary trace by observing and storing the interactions between the user (could be agent, user or external source) and the system. This primary trace can be transformed to generate a new trace of a new level. Transformation can be applying filters, rewriting and aggregating elements, computing elements attributes, etc. We can apply the transformation on the transformed traces and have many levels of abstraction, which could be more reusable and exploitable in a given context than the primary trace. A TBMS guarantees the possibility, at any time, to navigate between transformed traces. This ability provides an important flexibility. Figure 2 shows a primary trace $T1$ which is transformed into $T2$ and $T3$ (level 2), and $T3$ is transformed in its turn into $T4$ (level 3). Each trace has its trace model that contains different types of obsels that may have relations between them. The trace $T2$ is somewhat detailed where $M2$ is its trace model that contains three obsel types ($c1$, $c2$, $c3$) and one relation type $r1$. $T2$ contains three obsels ($o1$, $o2$, $o3$) and there is a relation between $o2$ and $o3$. 

Figure 1: Trace-Based Assistant: a general architecture
TBMS provides us also with a querying service that is dedicated to retrieving traces from the trace base or to write or update the traces.

A complete formalization of the meta-model proposal for traces models, traces, queries and transformations can be found with precise semantics in [3]. The core of our work is building the Trace-Based Assistant that implements such meta-model.

SAP-BO Explorer offers to the users an intuitive path to quickly search and explore data for instant insight into their business. Firstly we modified this software for being able to collect the obsels. This application is divided into two sides, the server side which is implemented in Java, and the client side which is a flex application. Many users could work on this application at the same time. That is why we made the collecting process on the client side. Every time the user tries to use the system, a new session is opened. Each session contains many obsels, so each action of the user will be collected as an obsel presented in a XML format specifying the obsel type, timestamps, and the values of this obsel. We suppose that the interface of SAP-BO Explorer is divided into task oriented blocks, where each block contains obsels with similar kind of tasks. For example when the user tries to choose a measure, we capture this action as an obsel of the type “Select measure” from the second block “The measures block” and with a value “Quantity Sold” and the current timestamps.

Each session is presented as M-Trace stored in the form of XML. Each session has a unique ID, the user who did this session, and the temporal list of the obsels that happened in this session. When a user connects to SAP-BO Explorer, a request to the server-side will be sent to open a new session and creating a new XML output file for this session. Every time a new obsel is collected, it is presented in XML format and sent to the server to be added to the session file. The management of the session is in the server side, where we keep all the opened sessions in a Hash Table. When a session is closed it is removed from the Hash Table. So we don’t need to close and open the file every time we collect a new obsel, we just keep the files of the current sessions opened. If the same user opens two sessions at the same time, they are stored in the same file.

3.2 Task Signatures Base

Task signature concept has been introduced in [2] as a set of event declarations, entity declarations, relations, and temporal constraints. With the trace model and task signatures based on this trace model, it is possible to define new obsel types in the trace model by using the task signatures to generate a new trace by replacing the obsels of the task with one more abstract obsel.

A Finite State Machine (FSM) is a model of behaviour composed of a finite number of states, transitions between those states, and actions. It is similar to a “flow graph” where we can inspect the way in which the logic runs when certain conditions are met [17]. FSMs are widely used in modelling of application behaviour, design of hardware digital systems, software engineering, compilers, network protocols, and the study of computation and languages. Many researchers worked on modelling software with Finite State Machines [18], but as far as we know, there is no work in using FSM to do episode retrieval.
We will extend the concept of task signature to be able to define more sophisticated signatures rules, especially by adding more time constraints. Our extension will be to use a Finite State Machine as a sequence detector to produce a binary output saying either ‘yes’ or ‘no’ to answer whether an episode is accepted by the machine or not. At the time when all obsels of the current episode is processed, if the final state is an accept state so this episode is accepted and matched the task signature, otherwise it is rejected. We will not explain more about how FSM are working, but we just want to explain how we use them in our model to define the task signatures.

By using FSM we can apply any rule we want, like recognizing series of obsels which can be directly linked or separated. In Figure 3 we can see an example of a task signature saying that an episode to be accepted it should start by an obsel of type c1 followed directly by at least one obsel of type c2 and then an obsel of type c3, where state1 is the entry state, state5 is the end success state, and state4 is the failure state. When we want to check if an episode has a task signature, we can pass it to the FSM task signature, and see in which state it will ends. If it is the refuse state, it means that it doesn’t has this signature, else if it is the accept state, it has this signature, else it is one of the intermediate states, here we can measure the distance between this state and the accept state to be used by the assistant for helping the user to reach the accept state and do his task. In addition to the FSM we could define temporal rules like the maximum duration of the task, the distance between two obsels, etc

![ FSM Diagram](https://via.placeholder.com/150)

*Figure 3: Example of a FSM Task Signature*

As an example in SAP-BO Explorer: the task signature in Figure 3 could be “Visualizing the quantity sold in New York of the year 2001 and changing the chart type to pie chart” So the obsels will be:
- C1: Obsel (Type='Select Measure’, value='Quantity Sold’)
- C2: Obsel (Type='Choose Dimension Value’, value='Year is 2010’)
- C2: Obsel (Type='Choose Dimension Value’, value='City is New York’)
- C3: Obsel (Type='Change Chart Type’, value='pie’)

In this example the task starts by selecting the measure and then we can choose many dimension values without any conditions in their orders, but it should be followed by changing the chart type.

Task signature base is application oriented that will be manually filled by system experts, where we suppose that the experts are capable of defining the task signatures as the form of rules. Every rule is converted to a FSM as a form of Graph that will be used as an input to our retrieving algorithm, so we need a conversion process that takes a rule defined by system experts, and returns a graph representing this rule, the nodes of this graph will be the states, where the links will be the transitions.

We do the transformations of the traces using the task signatures stored in the tasks base. Firstly we search for all episodes in the primary trace that has a task signature and generate the trace of level 1 by replacing the obsels of the task signature by one new obsel describing this task with setting the start point as the start point of the first happened obsel, the end point as the end point of the last happened obsel. In case that there are some episodes have more than one task signature, we generate many transformed traces at the same level, so we can have many traces at the same level with differences in the obsels types. We repeat the same operation for all the traces of level k to generate traces of the level k+1, until we don’t have anymore episodes with new task signatures. That will make the trace model in a form of a graph because we could have two childs have the same parent.
3.3 Trace-Based Assistant

The assistant is connected to a TBMS and has access to all its functionalities and knowledge, including traces and task signatures. During the interaction between the user and the system, the assistant generates the current online trace. This trace is transformed in the TBMS using the same task signatures that have been used for the past trace transformations. As the traces are time-series data, we can depend on the concept of sequence pattern mining [13] for developing the assistant to be able to retrieve the related episodes. (After the retrieving step we need to adapt the retrieved episodes to be reusable by the user.)

In our algorithm of retrieving the related episodes and ranking them (see Algorithm 1), either we consider that the task signature base is already filled by the signatures by the system experts manually, or we suppose that we are capable of generating task signatures automatically while the user is doing his task. The algorithm takes the current trace and its modelled trace, the task signature base as input, and it retrieves a ranked list of the suggested obsels. These obsels are the suggested tasks that the user can do after his current task. We can specify a maximum length of the sliding window for indicating the number of obsels in the current episode. The algorithm will start from the top level transformed traces, for each trace in the top level we take the current episode of the maximum sliding window length to find all episodes with the same length. If there are no suggested results, it will decrease the length and try again until the length of one obsel.

If the algorithm finishes without any results, it repeats all the process for each child trace (decreasing one level) until it arrives to the primary trace. In each iteration, it tries to retrieve all the episodes of the same level in the historical trace that have the same obsel types in the same order, and returns rules indicating their successor episode (the related episode) with two values (support and confidence). These values will be used to rank the results and avoid some results which are under specified thresholds. Support and confidence are similar to the same statistics measures of significance and interest used by Association Rules and Sequence Pattern Mining algorithms (See Definition 1, Definition 2, Definition 3).

After finishing the retrieving process, we need to adapt the retrieved obsels to fit the current task, because the values of the obsels could be different so we need to find the sufficient value of the solution obsel.

Definition 1: Rule

Let $M_{ij} = \{c_1, c_2, .., c_n\}$ be the modelled trace $i$ of the level $j$ that has a set of $n$ obsel type.

Let $T_{ij} = \{o_1, o_2, .., o_m\}$ be the trace $i$ of the level $j$ which is a sequence (time ordered list) of $m$ obsel(s).

Each obsel has a type $c_i \in M_{ij}$ corresponding attribute values, and the associated time stamps. An episode is a sequence of $T_{ij}$ satisfying a specific episode signature.

We can define a rule as an implication of the form $X \rightarrow Y$, where $X,Y$ are episodes of $M_{ij}$ called antecedent (left-hand-side or LHS) and consequent (right-hand-side or RHS) of the rule respectively.

In our assistant we will suppose that the length of the consequent episode is one obsel. To select interesting rules from the set of all possible rules and ranking them, constraints on various measures of significance, and interest can be used like minimum thresholds on support and confidence.

Definition 2: Support

The support $\text{Supp}(X)$ of an episode $X$ is its occurrence percentage of all episodes with the same signature in the trace $T_{ij}$.

$$\text{Supp}(X) = \frac{\text{Number of occurrence of } X \times \text{length}(X)}{\text{length}(T_{ij})}$$

Definition 3: Confidence

The confidence of a rule is defined $\text{Conf}(X \rightarrow Y)$ as the occurrence percentage of the episodes that have a signature of merging $X,Y$ to the occurrence of the episodes that have the signature of $X$.

$$\text{Conf}(X \rightarrow Y) = \frac{\text{supp}(X+Y)}{\text{supp}(X)}$$

Where the operator $+$ returns new episode by adding $Y$ to the end of $X$. 

**Algorithm 1: TBA retrieving algorithm**

```
Input: T, M (Where T is the current Trace, and M is the current modelled trace)
Output: Suggested (a ranked list of the adapted suggested obsels)

Suggested=Ø
For each trace T_i,j in TopLevelTraces(T)
    Results = Retrieve (T_i,j, M_i,j)
    For each rule in Results
        AdaptedRule = Adapt (rule)
        Suggested.Add(AdaptedRule)
    End For
End For
Return Suggested

Retrieve Function:
Input: T_i,j, M_i,j (Where T_i,j is the current Trace j of level i, and M_i,j is the current modelled trace j of level i)
Output: Results (a ranked list of the suggested obsels)

Results=Ø
Length = EpisodeLength;
While (Length > 0)
    Episode = TBMS.getCurrentEpisode(T_i,j, M_i,j, Length)
    Rules = TBMS.RetrieveRelatedEpisodes(T_i,j, M_i,j, Episode)
    For each rule in Rules
        Support = rule.getSupport()
        Confidence = rule.getConfidence()
        If (Support > suppThreshold and Confidence > confThreshold)
            Results.add(rule)
            Length = -1 // Break while loop for not keep decreasing the length
        End If
    End For
    Length= Length - 1
End While
If (Results = Ø) // If there is no results in this level, we check the lower level traces
    For each trace T_i-1,k in TBMS.getChilids(T_i,j, M_i,j)
        Results.add(Retrieve (T_i-1,k, M_i-1,k))
    End For
End If
Return Results
```

**GetCurrentEpisode**: Function that returns the last n obsels from the current trace \( T_i \).

**RetrieveRelatedEpisodes**: Retrieves all the historical episodes in which obsels are matched with the same temporal order.

**GetSupport, GetConfidence**: Calculates support and confidence depending on **Definition 2, Definition 3**.

**Add**: a procedure for adding a rule with its support and confidence values to the results list. Rules are ordered from the highest to lowest rank, by using a ranking function depending on the support and confidence values.

**GetChilids**: Returns the child traces of a specific trace (move down a level).

**Adapt**: is a function that adapts the solution by taking into account the differences between the current problem and the retrieved episodes. The adaptation process is well explained in [6].
Sometimes, obsels order shouldn’t be so strict, for example to do a specific task we may not care about which obsel should happen before another, but we care that both of them should happen before a third one. By considering this idea we can make an extension to Algorithm 1, to make it capable to accept episodes which don’t match exactly the current episode. This extension could be done by using a loose array, with \( n \) columns and \( n \) rows, where \( n \) is the number of the obsel types, both of columns and rows are the obsel types, where each cell contains integer value indicating the loosing amount for changing the order between these two obsels. These values will be predefined according to each application. When the algorithm tries to retrieve the related episodes it will try all the possible changes of obsel orders and calculate the loosing value, if it was under a specific threshold, this try will be ignored.

As an example of use of this algorithm, we can consider that we have a user using SAP-BO Explorer. His objective is to visualize the quantity sold of all the products in New York in the year 2001, and then export his visualization into Excel file. Then he can send the results to his manager as a result of his exploration. For doing this, he starts by choosing the measure ‘Quantity sold’ and choosing the year and the city, and then he chooses the chart type. Now the TBA will try to retrieve the similar episodes in the stored traces, by trying to recognize if the current episode has a signature from the task signature base. TBA will recognize that the user is doing a task with the same signature in Figure 3, so TBA will retrieve all the episodes that have this signature and rank them. So it could suggest to the user either to export his data, or to send it by email. As the user is usually exporting the results more than sending it by email, so support and confidence values of export will be greater than send by email. That is why the export will be ranked first. So the user will choose to export his visualizing into Excel file.

4 Discussion

Our assistant is extensible and accepts any kind of traces; we can see that the proposed extension on Algorithm 1 makes it sufficient because the matching is more flexible. This assistant is built over a TBMS, which means that the algorithm can use all TBMS functionalities. Thanks to task signatures extension, more tasks can be defined and more rules can be applied. By using these task signatures for the transformation, the user can understand what he is doing, by exploring the suggested episode and moving between the levels to see more details.

This paper is written in the context of a Master Thesis. We aim to develop a Trace-Based Assistant for SAP-BO Explorer, and try to make it as generic as we can. We are working to apply it to SAP-BO Explorer, to help business users make their analysis effectively, so we can support more experiments and examples to prove our approach. Some difficulties are facing us, because we need to have a huge amount of traces to be analysed, so we can build the task signature base and give the values to all the measures and thresholds that depend on the application.

Some of the issues and questions raised by the implementation in addition to defining some values are: How to adapt retrieved episodes? How to present the results to the user and be sure that he will be satisfied? What is the benefit of observing also the reaction of the user when the TBA suggests an episode to him? How to share the experience between the users? How to improve the accuracy of the ranking measures? All these questions are research issues that we intend to explore. In addition, we could consider as prospective some intelligent issues; the most important one is how to make the definition of task signatures automatically, because in our work we need system experts to define task signatures according to the observed application. We could also make the assistant able to understand the context of the work if the user opens many sessions of the same time. For example if the user opens two sessions to compare the sales between two cities in different periods, the assistant can recognize that the user is trying to do comparison, so a comparison table or chart could be generated automatically by the assistant in spite of the user.
5 Conclusion

In this paper, we have described an algorithm making use of interaction traces to provide an experience-based assistance to users of a specific application. This work is conducted in collaboration with SAP Business Object and the application we work on is called SAP-BO Explorer, a web application enabling users to load, explore, visualize and export data. The aim of the project is to develop a Trace-Based Assistant for this application, implementing state of the art results on this domain. The contribution of this paper focuses on one specific aspect of the assistant, the design of an experience retrieval algorithm. For that purpose, we have implemented a collect process in order to gather interaction traces, thus recording experiences of users in a reusable format. Then, with domain experts, we have defined “tasks signatures” to describe specific actions that users can perform on the application. Last, we have defined an algorithm able to retrieve in the stored interaction traces, previous episodes (i.e. previous users’ experiences) matching a given task signature. This algorithm makes use of FSM in order to retrieve patterns corresponding to the signature. We have also shown that the retrieved episodes can be reused, and possibly adapted, in order to provide assistances to users.

At the time being, the collect process is implemented, but the retrieval algorithm is not finished yet. This is an immediate prospect for this work. Once this is done, we will be able to address other issues related to the development of a trace-based agent such as “how to adapt retrieved episodes to specific situations?”.

6 References

[1] “SAP Business Objects Explorer: Explore your business at the speed of thought,” SAP.