An approach to User-Centric Context-Aware Assistance based on Interaction Traces

Damien Cram, Béatrice Fuchs, Yannick Prié, and Alain Mille

LIRIS UMR CNRS 5205, University of Lyon
42, bd du 11 Novembre 1918
69622 Villeurbanne, France

Abstract. This paper presents an approach to context-aware assisting systems that reuse user's personal experience as an alternative to traditional systems having an explicit context model associated to reasoning capabilities on this model. This approach proposes to model the use of the environment through interaction traces representing user's experience and different processes to manage user's traces intended to be exploited for later reuse. A user's assistance is based on the reuse of past traces that are semantically similar to the current sequence of interactions representing the current context of the user. The assistance is based on the identification of recurrent task signatures, which are sequential structures representing typical tasks. These signatures are *user-centric* since they have been interactively elaborated in accord with the user himself. This paper mentions some possible applications in terms of experience-based user assistance and makes a comparison of this approach to the case-based reasoning paradigm (CBR).[1]

1 Introduction

To better assist a user in his tasks, a good knowledge of what the user is doing and what constitutes his environment is required. A system that fulfills these requirements is called a "context-aware" system. Dey and Abowd [1] give the following definition of a context:

> *A context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*

To be context-aware, it then makes sense to first define a context model that expresses what contextual information represents for the best a situation, secondly to instanciate this context model for each situation, and thirdly to perform reasoning tasks about this context instance. This paper proposes a different approach to provide a contextual assistance to the user, where the context

[1] This research is supported by the french National Research Agency (ANR) as a contribution to the project PROCOGEC (see www.procogec.com)

is modelled on the fly, i.e. only when the situation requiring assistance appears. It consists in modelling user's interaction traces and in reusing them to gradually build a knowledge base of typical and abstract situations that could require an assistance. The user is involved in this building process, which enables a high *user-centricity* level of contextual information, and then a more relevant assistance system.

In Section 2, we describe some issues in context modelling and reasoning techniques, and explain how our approach brings new solutions to these issues. Section 3 gives definitions and models for concepts related to interaction traces reuse. The assistance process that collects interaction traces, abtracts them with the user's help, and reuses them to provide a contextual assistance is developped in Section 4. Section 5 is a discussion on our approach, and Section 6 concludes and draws some perspectives.

2 Limitations of usual techniques in context-aware assistance

2.1 The issue of modelling the context *a priori*

Considering the definition of a context given in [1], a natural approach to the conception of context-aware systems is to elaborate an expressive *context model* that takes into account every element of this definition. Reasoning techniques are then employed to infer knowledge that can be usefull for the assistance system. The task of modelling the context is usually performed by an expert of the user's activity. An ideal context model would have its instances (called *context objects*) containing every relevant details of each real situation of the activity. In a way, the expert temporally takes the place of the user and tries to imagine what entities and what actions of the activity are considered significant by the user.

This approach is quite paradoxal, as it freezes the definition of a context before the activity starts. We think that the context's definition cannot entirely be known *a priori*, mainly because it can evolve from one situation to another. Even if learning techniques can be used to infer a new context model from new situations, there is still another annoying issue : there is a loss of information in applying the same model to every situation. Indeed, each situation is unique, and observing a situation from the point of view of the context model potentially approximates some details that could have been usefull for assisting the user in a particular situation. This refers to the more general well-known issue of approximating the real world with models, and we think it is the reason why applying these methods to context awareness is not satisfying enough.

2.2 The issue of infering user-centric information

In [2], the authors propose to define the context model so as it answers the six contextual questions “who”, “what”, “when”, “where”, “why”, and “how” (5W1H). It is then argued that the obtained context model is a *user-centric* *context*, where *user-centric* means containing contextual pieces of information as the user perceives them. A context that is user-centric potentially improves the relevancy of the context-based user assistance system. If filling the context object with information about "who", "what", "when" and "where" from sensor data seems quite direct, it gets more complicated with *how* the user is doing his current activity and even more with *why* he is doing it. Indeed, ongoing processes are only sensorable at a very factual level and user's intentions are not sensorable at all. All the information we can really obtain from sensor data are clues of what the user intends to do, but never the real intentions. The expert can define rules to infer user's intentions from sensor data, but these rules are not very precise and can sometimes infer wrong. Knowing at a hight level of abstraction *how* the user is currently doing his activity and *why* he is doing it this way would be a precious information to build a relevant assistance, but this is still an important research issue.

2.3 The issue of abstracting information from sensored data

In a survey on context-aware systems [3], Baldauf and Dustdar point out that most context-aware systems have the same multi-layer architecture. Sensors are set up in the environment and the first layer consists in sensoring the environment and user actions to provide context information for upper layers. The second layer gives a uniform access to raw data coming from heterogeneous sensors to upper levels, mainly for modularity purpose and convenience. For example, information coming from a GPS and another position sensor can be accessed via a single `getPosition()` method. In the third layer, sensor data, which are often too technical and too verbose, are transformed into information at a higher abstraction level. The reason is that context-aware systems will need contextual information in a more meaningful form to enable contextual reasoning. For example, the position coordinates of a user and his surrounding objects can be transformed into nominal attributes with possible values like `closeTo`, or `farFrom`, which would give more semantics about the current context to upper levels. The fourth and the fifth layers reuse the context information produced by level three to govern the behaviour of the context-aware system. A typical kind of behavior is to provide an assistance to the user like displaying information about surrounding objects, recommending an action, automating the current task, *etc.*

The degree of relevancy of a context-aware system strongly depends on the third layer: the *abstraction layer*. The critical question is "how to get high level information from low level data?". Hilbert and Redmiles state that there are six levels of abstraction in interactions between a user and its environment [4]. Lowest levels describe very factual events like `gazingDirectionChanged`, `fingerMoved`, *etc.*, while highest levels describe tasks and goals (e.g. `buyingDVD`, `readingNews`). Interactions of a given level are composed of interactions of the level below. For example, `givingAddress` is composed of events `selectingField`, `typingAddress` and `clickingOK`. Similarly, `clickingOK` is composed of events `movingMouseOnOKButton` and `pressMouseButton`, *etc.* The only interaction levels that can be sensed are the lowest ones, but the highest levels are the most useful for context-aware decision-making. A context-aware system realizes this abstraction by instanciating the context model into context objects from sensor data, following certain predefined transformation rules. Sometimes these transformation rules are not reliable and the abstracted information infered by these rules might not be totally precise.

2.4 Our interaction-based approach of context-aware assistance

In our approach of context-aware assistance, we aim at proposing a solution to the three issues stated above. The idea is, instead of modelling *a priori* the context, to model the user's interactions with the environment. In other words, the system has no context object to reason about, but only *interaction traces* of the user's current and past activity. An *interaction trace* describes the user's activity: what actions he has done, what objects have been impacted, the date and time of actions, the roles of potential other users, etc. Thus, contextual information (who, what, when, where) is embedded in interactions traces, and reasoning on interaction traces comes to implicitly reasoning on context. But with this approach, the "how" is also taken into account since all user's actions and their relative order are considered. To provide a context-aware assistance to the user, the system compares the sequence of current interactions (the current trace) to past sequences of interactions (past episodes), and reuses these past episodes. Many types of assistance based on interaction traces are possible. For example, in [5], they are interactively navigable by the user in order to improve his reflexivity. We clarify in Section 4.3 what kind of assistance can be provided by reusing interaction traces.

This approach is qualified as *interaction-based* since we put the focus on modelling interactions rather than on modelling the context directly, even if the current context is implicitly contained in current interactions. With this approach, the uniqueness of the context of every situation can potentially be taken into account when traces are reused, and there is no information loss by trying to fit data coming from sensors to the context model. But we still face the abstraction problem; traces obtained from sensors are low-level. To get a more abstracted interaction trace, sequence mining algorithms like [6] search for recurrent episodes in low level traces and expose them to the user. The user then tags the episodes he can recognize as meaningful to him. Observations that compose each episode are transformed in a single, more abstracted observation that makes sense to the user. With this interactive approach of abstracting from sensor data, we also address the issue of user-centricity, since the user is involved in the definition of abstracted traces.

A system that handles interaction traces is called a *Trace-Based System* (*TBS*) [7]. A TBS provides tools to model interaction traces, to collect interaction traces in raw format from sensors and to transform them into the TBS format. *Trace transformations* are also handled in a TBS: filtering, rewriting, merging, etc. A *request system* enables an user to query the TBS and to extract transformed traces, that can be visualized through a *visualization system* or processed out of the TBS (cf. Figure 4). In TBS terms, the focus of this paper is to explain an interactive approach that defines "good" trace transformations, where "good" means that resulting traces are abstracted and user-centric, which enables a more relevant context-aware assistance.

3 Interaction-based context modelling

This section describes different concepts that need to be introduced before going into further details in the interactive abstraction process. Examples and figures are taken from a user working on the Content Management System (CMS) *Collaborative ECM*[2]. So, our environment is a computer system, which has been set up with sensors to track interactions using, among other technics, Javascript and auditing[3].

3.1 Modelling interaction traces

An *interaction trace* is a sequence of *events*. Each event can have many *relations* to some *entities*. The events represent actions performed by an user or the system, while entities represent objects that exist in the environment. Relations make the bridge between actions and impacted entities. Each event has a timestamp of the date when the event occured. The set of events of an interaction trace that occur between two dates $\mathtt{d}_1$ and $\mathtt{d}_2$ is called the *section* $(\mathtt{d}_1,\mathtt{d}_2)$.

A *trace model* is a set of concepts and relation types that expresses knowledge about elements in the interaction trace. Each action, entity, and relation is an instance of an element in the trace model. Typically, OWL can be used to build the trace model. Figure 1 shows a trace model example. A trace section example that is an instance of this trace model, is shown on Figure 2.

For the sake of simplicity, very few possible user actions are reified by event classes in the trace model of Figure 1. The class `Navigation` is instanciated when the user navigates from a page of the CMS to another one; `CreateContent` is instanciated each time a new content is created in the CMS. The example trace of Figure 2 can be read as follows. The user `damiencram` navigated through pages `index.jsp`, `importContent.jsp` and `setContentProperties.jsp`, then created the content `myContent`, and finally navigated to page `contentList.jsp`. Circles on Figure 2 are other events that occured during this process but not taken into account in this simple modelling example.

3.2 The task signature

The *task signature* concept has been introduced in [8]. It is a structure that represents a typical task in which a user can be engaged. It specifies what events

[2] Collaborative ECM is a collaborative CMS developped by our research partner *Knowings* (www.knowings.com)

[3] See the page "Audit" on Alfresco's wiki (http://wiki.alfresco.com/wiki/Audit)

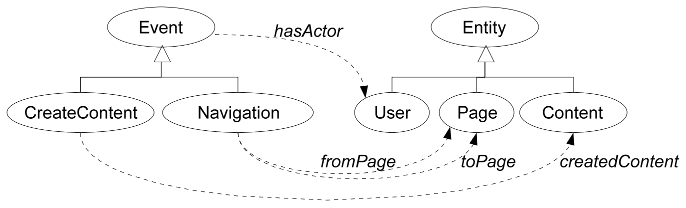

Fig. 1. An example trace model. `Navigation` and `CreateContent` are event classes (subclasses of the general event class `Event`); `User`, `Content` and `Page` are entity classes (subclasses of the general entity class `Entity`); `hasActor`, `fromPage`, `toPage` et `createdContent` are relation types.

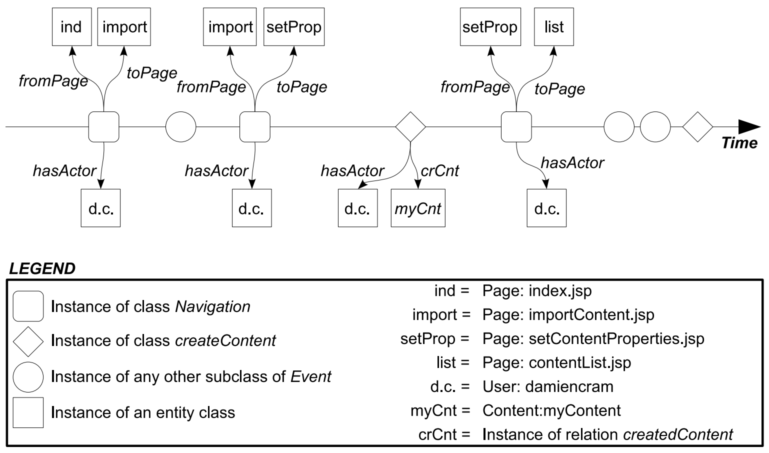
ind
import
import
setProp
setProp
list
fromPage
toPage
fromPage
toPage
fromPage
toPage
Time
hasActor
hasActor
hasActor
crCnt
hasActor
d.c.
d.c.
d.c.
myCnt
d.c.
LEGEND
Instance of class Navigation
Instance of class createContent
Instance of any other subclass of Event
Instance of an entity class
ind = Page: index.jsp
import = Page: importContent.jsp
setProp = Page: setContentProperties.jsp
list = Page: contentList.jsp
d.c. = User: damiencram
myCnt = Content:myContent
crCnt = Instance of relation createdContent

Fig. 2. A section of an interaction trace instanciating the trace model of Figure 1

(or event types) and entities (or entity types) are involved in a task, and how they are all temporally and structurally related to one another.

In this paper, a *task signature* is a set of *event declarations*, *entity declarations*, *relations* as previously defined, and *temporal constraints*. An *event declaration* is either an event or an event class; an *entity declaration* is either an entity or an entity class. A temporal constraint is a pair of event declarations (A, B) meaning that A must occur before B. In the case an event/entity declaration is a class, it acts as a structural constraint. An entity (resp. event) `e` *satisfies* an entity declaration (resp. event declaration) `E` if `E` is an entity instance (resp. event instance) and `e` equals `E`, or if `E` is a class and `e` is an instance of `E`. A section of an interaction trace is said to *match* a task signature when events, entities and relations of this section statisfy structural and temporal constraints. A section that matches a task signature is called an *occurrence* of this task signature.

Figure 3 gives an example of a task signature for the task "adding a content to the CMS". It defines a task signature in which the same user `?x` (whoever the user is) navigates from `import` to `setProp`, creates any content `?y`, and navagiates to `list`. The section of Figure 2 matches the task signature of Figure 3. In other words, it contains an occurence of this task signature. This task signature can be interpreted as "user `?x` creates the content `?y`". For example, an occurence of this signature having `?x=User:damien` and `?y=Content:mrc2008.pdf` would be read as "`damien` adds the content `mrc2008.pdf` to the CMS", another occurence having `?x=User:béatrice` and `?y=Content:mrc2008-rev.pdf` would be read as "`béatrice` adds the content `mrc2008-rev.pdf` to the CMS".

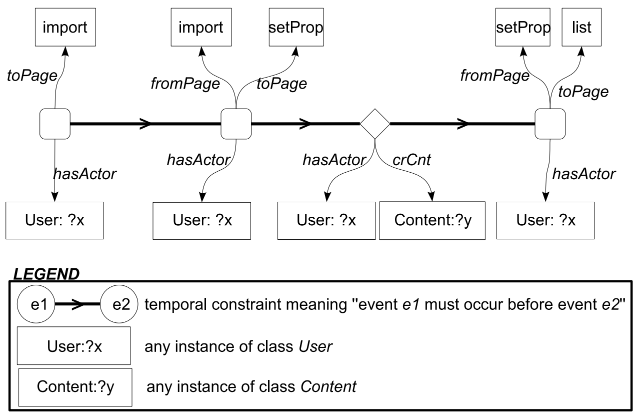

Fig. 3. Task signature of the task "adding a content". The entity declaration `User:?x` contraints occurences to have entities of class `User` as target for relations `hasActor`. The entity declarations `import`, `setProp` and `list` contraint occurences to have respectively `importContent.jsp`, `setContentProperties.jsp`, and `contentList.jsp` as relations `fromPage` and `toPage`.

3.3 Abstract classes and explained task signatures

In a way, a task signature represents an event that is more abstract than the ones it is composed of. Having a trace model and a task signature based on this trace model, it is possible to define a new event class in the trace model. For example, the task signature of Figure 3 could be reified in the trace model with the event class `addContent` having a relation type `hasActor` to the class `User` and a new relation type `newContent` to the class `Content`. This operation comes to reformulating the events of the task signature into a single more abstracted event class. Each time the signature occurs in the interaction trace, the occurrence could potentially be replaced with a single instance of the more abstracted class. In this paper, we reuse the term *ExTaSi* (Explained Task Signature), introduced in [8], to refer to a task signature that has been transformed into a class by the user. In [8], such a signature is qualified as "explained" because the user can also add an annotation to it in natural language to explain its meaning.

4 Interactive trace abstraction for context-aware assistance

Figure 4 shows how the context-aware assistance is made from interaction traces. The primary trace is built by the TBS from raw data that are collected from sensors (1). The primary trace is then abstracted under the control of the user interactively in the transformation system (2), an assistant system helps the user in his tasks by reusing abstracted traces (3). Following subsections describe these three steps into details.

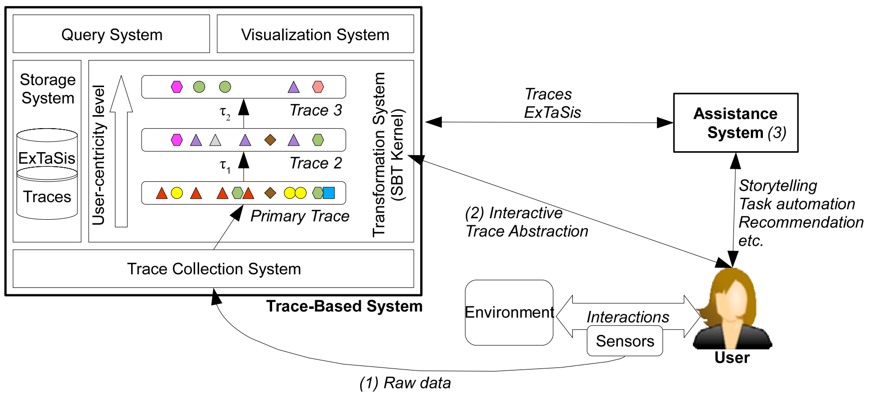

Fig. 4. Context-aware assistance based on interaction traces reuse

4.1 Trace collection

The trace collection (cf. Step *1* on Figure 4) consists in collecting interaction traces from all the sensors and integrating these data into a single interaction trace called the *primary trace*. This is the job of the *Trace Collection System* (TCS), which has the knowledge of the *primary trace model* and instanciates concepts and relations of this trace model with observations made from sensor data. Each sensored interaction in the environment triggers the TCS, which in real-time updates the primary trace with new observations. The primary trace is a single sequence of observations, containing both current (or contextual) information about what happens in the environment and information about what has happened in the past.

This level of information of the primary trace is like Baldauf and Dustdar's level two (cf. Section 2.3) in the sense that the primary trace from now constitutes the unique access to sensored data from the environment. It is very important to note that the primary trace model is usually not abstract at all, collecting very fine grain events, and letting the job of aggregating and abstracting to the transformation system.

4.2 Interactive trace abstraction

Abstracting traces (cf. Step *2* on Figure 4) is a process that is actually both interactive and iterative. It is *interactive* in the sense that, as argued in Section 2, the user must be involved in it so as to be user-centric. It is also *iterative*, because several transformations can be applied to get a satisfying abtraction level. On Figure 4, two trace transformations are represented: transformation τ_1 abstracts the primary trace into `Trace 2`, which in turn is abstracted into `Trace 3` by τ_2. Figure 5 zooms on a single iteration τ_k. An iteration is performed thanks to two complementary separate phases: the *trace analysis* and the *trace transformation*.

Trace analysis. The analyze step extracts from `Trace k` recurrent task signatures, using techniques of "Frequent Episode Discovery" in an event sequence [6], and suggests them to the user. Then, the user looks at suggested candidate signatures and validates the ones that make sense to him as *ExTaSis*. If the user is not satisfied with the suggested candidates, he can rerun the analysis with new entry parameters. New entry parameters can be new constraints guiding the extraction process, like "search for signatures with an instance of class `X`", or "search for signatures with at least two different users" (which is the task signature of a collaborative task), *etc.*

Trace transformation. Once the user has found ExtaSis from `Trace k`, the *tranformation* step reformulates each occurence of an ExTaSi in `Trace k` into an abstract event in `Trace k+1` (cf. $\mathtt{Ext}_1$ and $\mathtt{Ext}_2$), while every other non-matching observations in `Trace k` stay unchanged. In terms of trace models,

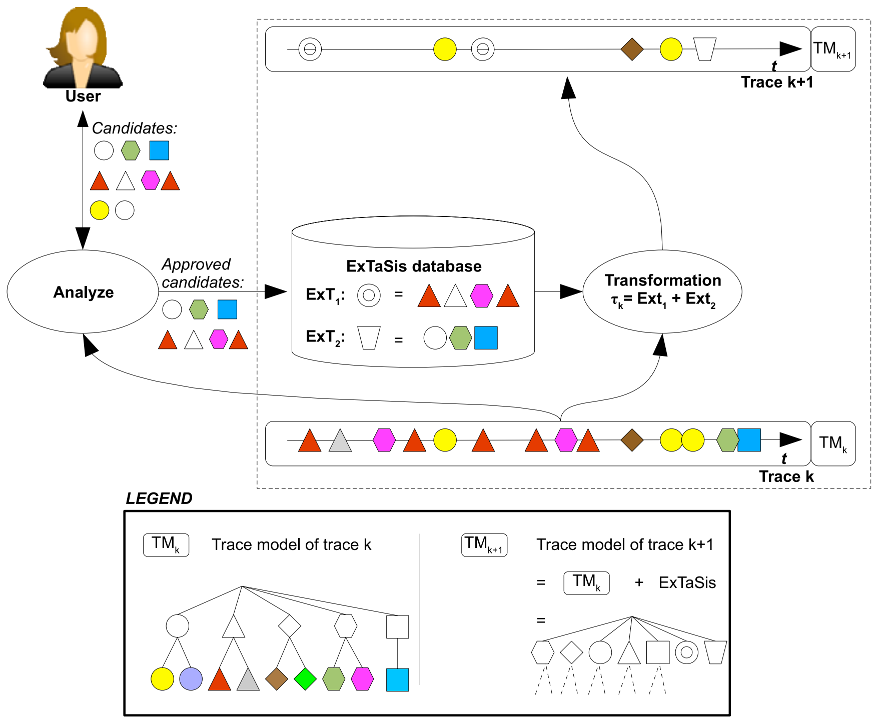

Fig. 5. An iteration of the interactive trace abstraction process.

$\texttt{TM}_{\texttt{k+1}}$ is built upon $\texttt{TM}_{\texttt{k}}$ by adding abstract classes that are each made from an ExTaSi, as explained in Section 3.3 (like $\texttt{Ext}_1$ and $\texttt{Ext}_2$ in Figure 5). Within the TBS framework [7], this operation consists of defining a transformation rule τ_k by specifying the pattern to transform (the task signature) and the result type (the abstract class). The TBS then does the transformation job for us. This transformation phase results in a new trace `Trace k+1` that is more user-centric, in the sense that it describes the user activity from the user's point of view. It is then a more readable and understandable trace to the user.

4.3 Signature-based context-aware assistance

Back to Figure 4, the last part of the context-aware assitance is the *Assistance System* itself (cf. Step *3*). It is composed of an assisting agent that observes the current interaction trace through the TBS's request system. When the current trace matches with the beginning of an ExTaSi, the assisting agent requests the TBS for all occurences of this ExTaSi in the interaction trace. These occurences can be ranked according to a similarity measure to the current trace. An assistance is provided to the user based on the past most similar occurences of his current task.

This assistance can be of many types. For example, a task automating system could propose to the user to finish the current task. Of course, this kind of system can only be set up in environments in which possible actions can be easily automated, typically a computer. Another type of usual assistance is to inform the user that some actions, like "look at this document" or "contact this person", could help him to better achieve his goals. These types of assistance systems are based on the current task recognition as the beginning of an ExTasi. The assistance then consists in recommending to the user what is coming next in this ExTaSi. The problem is that an ExTaSi is a general form of a task, which means that some of its components are concepts and not instances, and consequently a strong adaptation effort to the current context has to be made by the system.

There is another approach to assistance that would fit better our system: it consists in making visible to the user in real-time past occurrences of the recognized ExTaSi, enabling the user to visualize complete stories similar to what he is currently doing. This could give an access to every entities that were involved in these past stories, and then potentially provides additional task-contextual information to the user. The importance of storytelling in human reasoning is well-known and advantages it could bring to the human activity is getting aknowledged. That is why systems trying to facilitate storytelling, like [9], have started to appear. In [10], the authors argue that humans have a narrative indexation of documents, and that a narrative description of documents is more helpful to the user than a usual classification. Our system can provide such a storytelling assistance based on ExTaSi recognition.

5 Discussion

We have seen that the relevancy of an assistance system depends on the user-centricity of input traces, but it also strongly depends on the adaptation that the assisting agent is able to perform from past occurences to current context. This *adaptation* issue is already known in Case-Based Reasoning systems to be both quite difficult to formalize and critical for the usability of the system [11].

Another interesting issue is organizing ExTaSis. The problem is that an ExTaXi stands for several similar occurences in the trace, but more precisely each of these occurences tells actually a different story about the user's activity. We expect there will be a need for expressing that an ExTaSi is more specific than another one, e.g. expressing that "adding the minutes of a meeting" and "adding a project plan" are both more specific ExTaSis of "adding a content". These concerns have been addressed by Roger Schank in his theory of *dynamic memory* [12] to describe the human memory, where *Memory Organization Packages* (MOP) are structures that hold both concrete experiences (*scripts*) and more general ones in a single hiearchy that is constantly updated. Historically, this theory gave birth to the paradigm of *Case-Based Reasoning* (CBR) [11], which raised a great interest in the community of AI.

Existing literature gives some examples on CBR being a convenient paradigm to context-awareness, which is mainly due to the fact that contexts and CBR cases are similar in their definitions [13]; CBR then offers the possibility of comparing and reusing contexts. However, CBR systems can only focus on well-defined problems. In many environments, we don't have the knowledge of what the user will precisely intend to do *a priori*. We are then unable to define what will be the user problems, and consequently unable to build a CBR system. The system we propose does not need a problem or a case to be modelled beforehand. In a way, it is more a method that supports the emergence of problem definitions from usage. Indeed, each ExTaSi can be seen as a problem definition as it describes at a high level of abstraction *what* a task consists in. It also contains information about *why* a user realizes this task in terms of *user intentions* (cf. Section 2), as an ExTaSi has been labelled in natural language by the user.

6 Conclusion

In this paper we presented an approach to user's assistance based on the exploitation of interaction traces of the user's activity. We described a model of interaction traces expressing user's interaction with his surrounding entities, and proposed a three-steps process that reuses traces for a contextual assistance: collecting traces from sensors, abstracting traces, and assisting. Two steps are performed in interaction with the user : the discovery of typical task signatures that are interpreted and validated by the user and the context based assistance like automatization of tasks, or action recommendation under the control of the user.

Further work has to be done to precise what kind of transformations are performed and what granularity level has to be chosen for a better reusability and understandability of traces. Other works will consist in defining different kinds of assistance scenario and in studying how a CBR cycle would reason about traces. In a CBR system, a problem part and a solution part have to be identified as well as a request that are generally not clear in a current context. Such a CBR system has to associate knowledge to each kind of signature and learn from usage to enhance its reasoning capabilities by interactively acquire knowledge about how to reuse better traces.

References

1. Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M., Steggles, P.: Towards a better understanding of context and context-awareness. In: HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, London, UK, Springer-Verlag (1999) 304–307
2. Hong, D., Schmidtke, H.R., Woo, W.: Linking context modelling and contextual reasoning. In: 4th International Workshop on Modelling and Reasoning in Context, Roskilde, Denmark (2007)
3. Baldauf, M., Dustdar, S., Rosenberg, F.: A survey on context-aware systems. International Journal of Ad Hoc and Ubiquitous Computing (2004)
4. Hilbert, D.M., Redmiles, D.F.: Extracting usability information from user interface events. ACM Comput. Surv. **32**(4) (2000) 384–421
5. Cram, D., Jouvin, D., Mille, A.: Visualizing Interaction Traces to improve Reflexivity in Synchronous Collaborative e-Learning Activities. In Limited, A.C., ed.: 6th European Conference on e-Learning. (October 2007) 147–158
6. Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of frequent episodes in event sequences. Data Mining and Knowledge Discovery **1**(3) (1997) 259–289
7. Laflaquière, J., Settouti, L.S., Prié, Y., Mille, A.: A trace-based System Framework for Experience Management and Engineering. In: Second International Workshop on Experience Management and Engineering (EME 2006). (2006)
8. Champin, P.A., Prié, Y., Mille, A.: MUSETTE: Modeling USEs and Tasks for Tracing Experience. In: Workshop 5 'From Structured Cases to Unstructured Problem Solving Episodes For Experience-Based Assistance', ICCBR'03. (2003)
9. Lin, N., Mase, K., Sumi, Y.: An object-centric storytelling framework using ubiquitous sensor technology. In: Proceedings of Pervasive 2004 Workshop on Memory and Sharing of Experiences, Vienna, Austria (2004)
10. Goncalves, D.J., Jorge, J.A.: Ubiquitous access to documents: Using storytelling to alleviate cognitive problems. In: Proceedings of the Tenth International Conference on Human-Computer Interaction, Lawrence Erlbaum Associates (2003) 374–378
11. Mantaras, R.L.D., McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., Faltings, B., Maher, M.L., Cox, M.T., Forbus, K., Keane, M., Aamodt, A., Watson, I.: Retrieval, reuse, revision and retention in case-based reasoning. Knowl. Eng. Rev. **20**(3) (2005) 215–240
12. Schank, R.C.: Dynamic memory revisited. Cambridge University Press, NY (1999)
13. Zimmermann, A.: Context-awareness in user modelling: Requirements analysis for a case-based reasoning application. In: ICCBR. (2003) 718–732