Dynamic Case-Based Reasoning for Contextual reuse of Experience

Amélie Cordier, Bruno Mascret, Alain Mille

Université Lyon 1, LIRIS, UMR5205, F-69622, France
{firstname.lastname@liris.cnrs.fr}

Abstract. This paper reviews Trace-Based Reasoning (TBR), a reasoning paradigm based on interaction traces left by users in digital environments. Because interaction traces record the users' problem-solving experiences in context, TBR facilitates the reuse of such users' experiences. Moreover, interaction traces can be used as a knowledge source to discover other knowledge useful for the reasoning process. This paper describes TBR principles and proposes a common framework for TBR applications. Especially, we focus on the articulation between Trace-Based Reasoning and Trace-Based Systems in the context of user assistance. For this purpose, we describe knowledge and knowledge models involved in reasoning tasks. We discuss the advantages of using traces as knowledge containers for reusing of experience and we show how TBR is related to the research field of provenance. Finally, we report a brief state of the art and a work agenda for TBR. We also show why TBR can take advantage of the current dynamic of the work about traces while globally improving trace-based applications.

1 Introduction

This paper takes up the challenge of developing dynamic and reactive systems that can quickly and incrementally adapt to changes in users' needs and habits. It has been consistently argued that, in many cases, users' needs and habits cannot be fully anticipated *a priori*. The incomplete anticipation of user needs complicates the development of reasoning mechanisms that could make the system self-adaptable. Typically, in reasoning based on previous experiences, such as Case-Based Reasoning, the system's ability to adapt is limited by knowledge models and by the fact that reasoning mechanisms are defined at design stage.

We introduce here the *Trace-Based Reasoning* (TBR) paradigm and argue that this paradigm supports the development of more flexible and scalable systems. While interacting with a system, a user leaves behind *interaction traces* that constitute digital inscriptions of the users' experiences. We consider interaction traces as knowledge containers in which experiences are implicitly stored but are not really organized in advance. Previous experiences, called *episodes*, are only retrieved when a specific need appears. This mechanism ensures flexibility and adaptability of the process, but also raises complex issues (e.g. how to find a previous experience in a complex trace?). In TBR, episodes are always

linked to the trace that contains them. Consequently, at any time, it is possible to retrieve contextual elements related to the current episode and to use them to improve the reasoning process. Hence, TBR allows us to handle experiences in a much less constrained way than if they were caught in structured cases.

This paper is organized as follows. In section 2, we quickly describe recent work that seeks to develop more dynamic and scalable applications, in particular within the CBR field. Section 3 presents TBR, starting with a short description of the Trace-Based System concept. Then, we discuss the issues raised by the development of trace-based applications and we show the possible benefits of TBR in the field of user assistance. The paper concludes with a discussion of our research agenda for Trace-Based Reasoning.

2 Towards dynamic and evolving experience-based applications

Recently, several suggestions have been made to push the limits of Case-Based Reasoning. For example, the *Agile CBR* idea described by Susan Craw in [1] makes one step towards a more dynamic approach to CBR. The main idea of Agile CBR is to transform traditional CBR into a "dynamic, knowledge-rich, self-organizing, cooperative problem-solving methodology". Agile CBR draws from agile programming and focuses, amongst other aspects, on individuals and interactions as well as on quick responses to change. Hence, according to Susan Craw, "Agile CBR's opportunism, commitment and flexibility aims to providing the adaptability needed to design processes and workflows, and focusing first on enabling just-in-time information". TBR works the same way. Traces provide us with rich knowledge sources that should be helpful to address the knowledge management issues raised by the Agile CBR challenge. For example, combining several users' experiences to solve one specific problem can be efficiently addressed with a reasoning process based on fusion of individual interaction traces.

The notions of traces and provenance are also closely related. As it is shown in [2], there is a growing interest on the concept of knowledge provenance and on how to use provenance information for various tasks (capturing experience, improving similarity assessment, maintaining case bases, etc.). In this work, the authors show that CBR always records prior problem-solving experiences in the form of cases, but does not record the case provenance. The authors also show that provenance information is often valuable. With traces, we always keep provenance information because we find "cases" in the trace only when we need them. We believe that research on provenance in CBR and in TBR can be usefully combined,

The idea of taking context into account in the reasoning process is also explored. For example, [3] shows that combining Case-Based Reasoning methodology and context awareness is a new and powerful way of modeling and reasoning from contexts. This contribution shows how user behaviors can be learnt in a case-based way, which could be much more efficient with a TBR approach. Il-

lustrating this idea, [4] uses time-stamped activity logs called *paths*. Paths are collected by client-side tracking processes implemented in several tools (such as web browsers and text editors, etc.). These paths are then exploited to gather information used to feed recommender systems.

What we call *episodes* can be seen as historical cases as described by [5] in the Historical Case-Based Reasoning (HCBR) framework. This framework allows the expression of both relative and absolute temporal knowledge, representing case histories in the real world. In HCBR, a case base is formally defined as a collection of (time-independent) cases, together with its corresponding temporal reference. A TBR approach includes all these features and adds several other possibilities, such as context extension, abstraction level navigation, provenance assessment, dynamic elaboration, etc.

Other researches, linked to experience sharing, such as described in [6], can be related to the work described here. In HeyStaks (http://www.heystacks.com), users try to share their search experiences with others. The authors propose a classification approach to increase the relevance of a *stak* (a list of interesting search results) because a stak may have been noised for several reasons (spam, mistakes, out of date records, etc.). The most interesting point here is that even though experience is collected, the system still needs help to determine which real context is used. Instead of using only classifiers, we could combine them with TBR in a revision step based on interesting elements contained in the history of the stak construction. We think that traces can be helpful in such a situation where context is changing and not given directly by the system.

As an example of what could be an assistant using traces and a CBR approach, we can cite [7] who uses Case-Based Reasoning techniques to predict the user's goal(s) from a sequence of his workspace (inter-)actions needed to achieve this goal. A TBR approach generalizes this ability to assist the user for whatever could be found in its interaction traces as previous experiences.

3 Trace-Based Reasoning

A Trace-Based System (TBS) is made of three main components: a Trace-Bases Management System (TBMS), one or more observed applications and one or more trace-based applications. The general architecture of such a system is presented on figure 1 and detailed below.

The core object of this architecture is the trace. A trace reflects the result of the observation of a given situation. In figure 1, the observed situation is that of the use of the application by the user. A trace consists of a set of elements called *obsels* (for observed elements). A modeled trace is a trace to which is associated a model that formally defines the structure and the types of obsels that can be contained in the trace, as well as the relationships between these obsels . The obsels are temporally located within the trace. We distinguish two types of traces: primary traces, and transformed traces. Primary traces are the results of the collect process. By definition, they are intended to be untransformed.

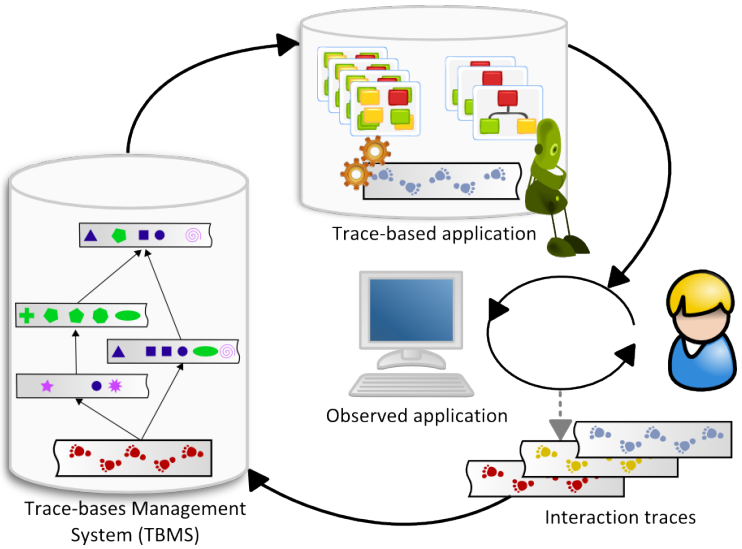

Fig. 1. Trace-Based System: a general architecture. User produces traces stored in the TBMS by interracting with an observed application. The trace-based application uses traces to solve problems and may ask the TBMS to apply transformations. The solution is proposed to the user and this process is also recorded on the trace.

Transformed traces are the results of transformation operations performed on the primary trace or on other existing transformed traces.

The Trace-Based Management System is the component that manages the various trace bases. Trace bases contain primary traces as well as the whole set of transformed traces, and the associated models. A TBMS provides input/output primitive functions to manipulate these traces (read, write, update, transform). Traces are stored in the TBMS during the collect process. This process exploits one or several collect sources that can have various natures. Either the target application is instrumented so that it automatically sends the elements that have to be collected to the TBMS, or an intermediate process builds the primary trace from available observation elements (such as log files of the observed application). Hence, an application, a tool, a set of log files, a trace from another provenance, etc. constitute as many collect sources. In brief, the collect process builds the primary trace that can subsequently be transformed using existing transformation operators.

The notion of trace transformation is of major importance. A transformation is an operation on one or several modeled traces. The result of a transformation is a new modeled trace, linked to the source traces. We can distinguish several types of transformation operators, among them: filtering, merging and reformulating operators. A filtering operator consists in applying a filter on a given trace in order to produce a new trace containing only the obsels matching the rules

of the filter. Here is a very simple example. If we consider a trace containing obsels of several colors, we can define a filtering operator that will keep only the "red" obsels. Applying such a filter will transform the initial trace in a transformed trace containing only red obsels. Obviously, transformations can be more complex and may combine several operators. Merging and reformulating operators work the same way but in addition, they can dynamically produce new types of obsels (for example, two obsels in the initial trace may be combined and represented by a third one, dynamically created) in the produced trace.

Transformations can be automatic, semi-automatic or manual. A TBMS guarantees the possibility, at any time, to navigate between transformed traces. Indeed, the TBMS always keeps a link between the transformed trace and the traces from which it comes from. Besides, it records the transformation operators that have been used to perform the transformation. In other words, the TBMS stores a lattice of transformed traces in order to guarantee the possibility to navigate between transformed traces. Transformed traces can be seen as various representations of the same situation, at different abstraction levels (this idea is illustrated in the example at the end of this section). Because traces at different abstraction levels are linked to eachother, it is possible to consider a trace at a specific level of abstraction, and, at some point, to consult another trace representing the same situation at a more specific abstraction level. This possibility to navigate between various modeled traces thus provides an important flexibility.

Another way to understand transformations could be to simply consider them as knowledge: a transformation is performed *because* some knowledge gets the system to activate it. So transformations may be both knowledge and processes. This new point of view raises open issues discussed later on this paper.

The last element of the architecture is the tool that uses traces. This kind of tool can be connected with the TBMS through input/output primitive functions provided by the TBMS. These tools can also connect with the observed application, but this is out of the scope of this paper. These tools have various natures: (interactive) visualization of traces, analysis of usages, user assistance, etc. Tools have to rely on specific knowledge to accomplish certain tasks (described below) with traces. We call this type of knowledge "reasoning knowledge".

Trace-Based Reasoning. While reasoning, the first tasks consist in retrieving previous episodes in the trace to reuse them. For episode retrieving, we introduce the concept of task signature that contains a set of distinctive elements characterizing an episode. With each task signature, we associate a set of similarity measures that allow us to retrieve the most similar episode in the trace, and a set of adaptation rules (see below). Another difficulty adds up to the traditional issue of finding similar episodes: the search has to be performed among the various transformation levels of the trace. Therefore, the search has to take into account the various possible representations of a single problem.

Once retrieved, the episode can be reused in various situations: visualization, interactive visualization, replay, or adaptation. In case of visualization task, we have to exploit the available presentation knowledge in order to display the episode to the user in a nice and understandable way. Interactive visualization

relies on the same principle, but additionally allows the user to navigate between the different abstraction levels of the trace. Interactive visualization allows us to handle various viewpoints on the same problem in an efficient way: for the user, it is possible to consider a problem at a high abstraction level and then to go to a lower level (to more specific transformation traces) if more specific details are needed. Replaying episodes consists in retrieving an episode and replaying it the same way. The possibility to replay an episode implies strong constraints on the collect phase. One of these constraints is to make sure that the whole set of observations has been collected in the form of obsels. Last, adapting an episode consists in reusing it and, possibly, transforming it to provide help to the user (assistance, recommendations, automatic task execution, etc.). Within the TBR context, it is necessary to rely on a specific (and possibly generic) structure for adaptation. This structure has to offer a way for the system to identify the elements that can be subject to an adaptation in the episode. We face the same problem than in CBR and, as we have showed it in a previous work [8], it is possible to reuse some of the solutions provided by CBR researches in this context. In TBR, however, an episode adaptation can be envisioned at several levels: content adaptation, interaction modality adaptation, presentation adaptation, navigation between different abstraction levels (i.e. adaptation by navigation between transformed traces). This diversity among the different types of adaptation introduces an additional complexity from the point of view of the knowledge representation necessary for the reasoning.

Figure 1 illustrates one specific issue, e.g. the issue of designing a trace-based assistant. The challenge here is to reuse experiences to provide assistance to the users. The system can reuse the same users' interaction traces or can browse interaction traces to reuse chunks of other users' experiences.

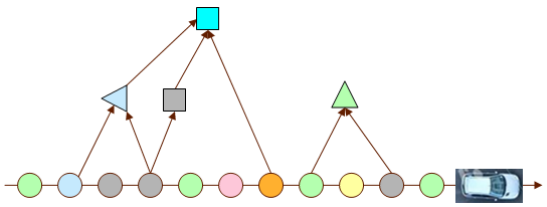

Fig. 2. Several transformations of a single trace. Transformation operators are combined to produce a more abstract trace. The more the trace is abstracted, the more it contains knowledge.

An example with Abstract. To give an example of how traces can be collected, transformed and used at several abstraction levels, we briefly describe the Abstract project [9]. Abstract is a cognitive science project intended to provide means for a better understanding of human activities using a com-

puter recording of these activities. From a computer science point of view, this project aims at discovering knowledge from traces of activity. The ABSTRACT application starts by observing the activity by various means (i.e. various kinds of sensors) and collects a primary trace. The set of data collected is also attached to a moment of the activity identified by a "time-code". The trace can then be manipulated and enriched with more abstract symbols through several transformations. Figure 2 and figure 3 illustrate the way traces are manipulated in ABSTRACT. Figure 2 shows three abstraction levels of the primary trace from a generic point of view. Figure 3 illustrates the same principle, but applied to a specific application domain: driver behavior analysis.

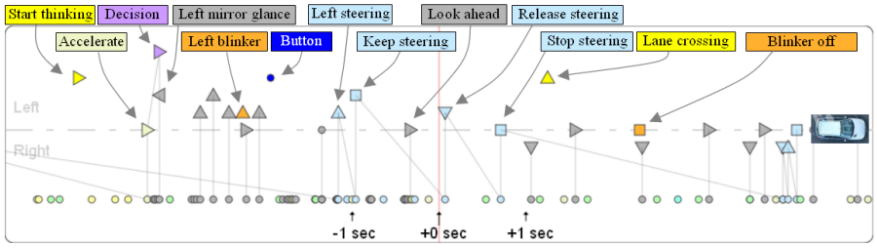

Fig. 3. Abstract: an example of the user interface. The primary trace has been transformed to produce traces "understandable" by users (e.g. analysts).

Summary. In a trace-based application, the Trace-Bases Management System (TBMS) has an operational role: it manages the traces and their transformations. It also provides primitive functions to manipulate traces and allows navigation between the different levels of representation. The trace-based application, meanwhile, relies on presentation and reasoning knowledge. Hence, the application can achieve various goals: visualizing an episode, reusing it, adapting it, etc. A major issue is that of knowledge evolution and of knowledge models. The implementation of mechanisms supporting knowledge evolution is an open problem. If the update of certain knowledge is immediate (by definition of the concept of trace), the update of knowledge models, particularly in the case of modeled traces, is more tedious and may require a process involving an expert. We consider the issue of updating a knowledge model as a major challenge for TBR.

4 Discussion and concluding remarks

Trace-Based Reasoning is a problem solving paradigm in which both the user and the system solve problems and learn new knowledge at the same time. It is through interactions that the user gives additional knowledge to the system and that the system provides the user with solutions to specific problems. In this

process, traces constitute a flexible support used at the same time by humans and machines. Traces keep a record of interactions, and thus of the problem solving process. In some extent, this is why we argue that they allow us to record problem solving experiences "in context" and that they constitute a rich knowledge container.

One way of introducing the concept of Trace-Bases Management System is through the analogy with a file system. The file system organizes, stores, and accesses the files, but the users can't directly exploit the files without any specific applications to read, modify or transform them. It is the same problem with traces: without any specific software to manipulate them, the traces stored in a TBMS usually have little interest for end users. We believe that TBR, because it offers a dynamic, responsive, and scalable approach for the collect and reuse of experience, is very promising. Moreover, the work about traces is currently very active and TBR could advantageously benefit from it.

Among researches on TBR, we can distinguish two categories. The first category includes work concerned with theoretical aspects of traces and their management (modeled trace theory, mining traces to find relevant patterns, interactive visualization, etc.). The second category includes researches that are intended to produce trace-based applications (Trace-Based System for behavior analysis, user assistance, learning purpose, etc).

Within the second category, issues related to transformation are challenging. Indeed, the role of transformations is of utmost importance. A transformation may be seen in several ways: historical, such as a TBR's process using operators to transform a trace in a higher abstraction level one; knowledge-oriented, which considers transformations as knowledge containers; combined, which mix ideas issued from these two approaches. This last question raises new issues: can all kind of knowledge be represented through transformations, and are transformations a good way to represent knowledge?

Traces and provenance address complementary issues. The common understanding of provenance in CBR is that a system may improve its process if it has knowledge on how cases have been elaborated. As shown previously, traces embed information on provenance and consequently, this provenance information can be used by the TBR approach. Provenance information may be exploited for various purposes. For example, in an experience sharing system, provenance is used for confidence purposes: provenance may help the system to determine what "kind" of experience should be reused in a given context. Let us consider the copy/paste example (illustrated figure 4. Imagine a blind user trying to use a sighted person's trace. Most of the sighted people use the mouse to perform a copy/paste with the contextual menu. However, blind people usually prefer to perform this action with the keyboard only. To support experience sharing between blind and sighted people, the first step is to consider the traces at the right abstraction level. In this example, if we consider the low-level obsels (such as "Ctrl+C" or "Right Click + Copy"), we cannot provide any help. However, if we consider obsels that belong to a higher abstraction level (such as "Copy"), we can then support experience sharing between different users. Once the right

level of abstraction is found, the second step consists of adapting the way of performing the action ("Copy") by taking into account the user preferences (e.g. "only with the keyboard"). This example suggests an interesting observation: while most users use the mouse to perform a Copy/Paste, blind users may prefer less common solutions. Hence, provenance information may help a user to find the best available episodes for him, or at least to give him relevant adaptation knowledge (how to present "paste": with mouse or with keyboard?). We think this issue should open new paths especially in adaptive systems and assistive technologies.

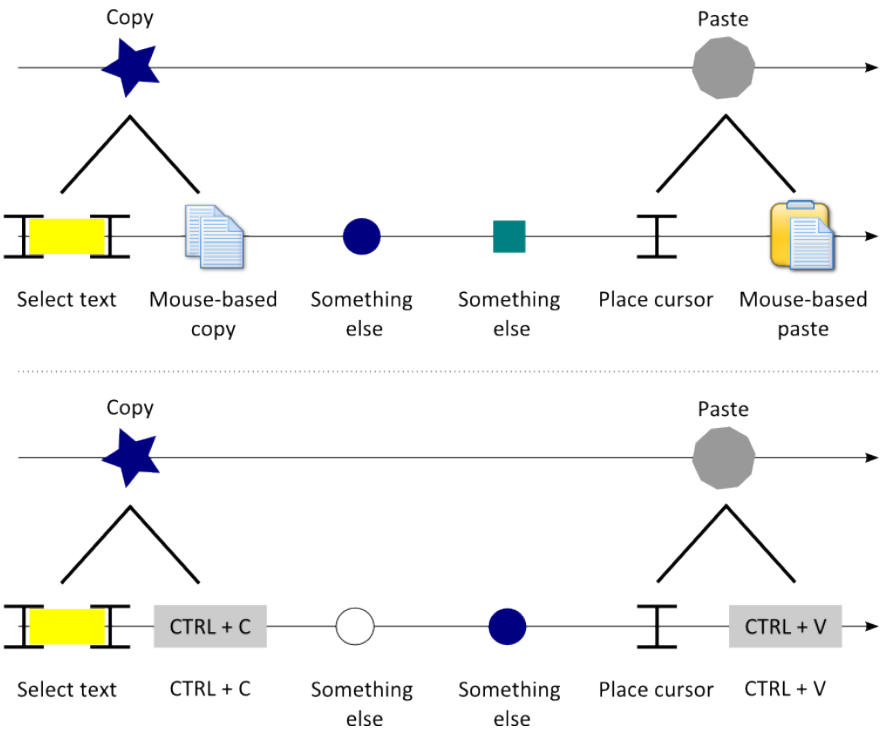

Fig. 4. Copy/paste example. The trace of the mouse-based copy/paste (above) is compared to the trace of the keyboard-based copy/paste (below). At the highest abstraction level, both traces are identical. However they represent two different ways of performing the same task. Note that in both cases, the bottom traces are not primary traces, they are already transformed from other traces (for example "Right Click + Copy Selection" has been transformed in "Mouse-Based Copy").

The issues raised by the implementation of TBR are various and many questions arise: how to gather experience through observing interactions? How to define transformation mechanisms for modeled traces and how to make them

evolve? How to adapt retrieved episodes and how to present the results of this adaptation to end users? How to make the system knowledge evolve, and from what sources? Are traces sufficient or should we rely on other sources of knowledge, and if so, how to integrate them in the system? All these questions are research issues that we intend to explore. We will focus our researches on assistance systems which are a challenging experimentation field.

Acknowledgments

Authors wish to thank the workshop organization team for providing this opportunity to discuss the notion of provenance from very different viewpoints, and the reviewers for their constructive and helpful comments.

References

1. Craw, S. Agile case-based reasoning: A grand challenge towards opportunistic reasoning from experiences. In *Proceedings of the IJCAI-09 Workshop on Grand Challenges in Reasoning from Experiences*, pp. 33-39, Pasadena, CA, 2009.
2. Leake, D.B and Kendall-Morwick, J. Four Heads Are Better than One: Combining Suggestions for Case Adaptation. In *Proceedings of ICCBR-09*, pp. 165-179.
3. Zimmermann, A. Context-awareness in user modelling: Requirements analysis for a case-based reasoning application. In *Case-Based Reasoning Research and Development*, pp. 1064–1064, 2003.
4. Chalmers, M. Abstract paths and contextually specific recommendations. In *Proceedings of DELOS/NSF Workshop on Personalisation and Recommender Systems in Digital Libraries*, Dublin, June 2001.
5. Ma, J. and Knight, B. A framework for historical case-based reasoning. In *Case-Based Reasoning Research and Development*, pp. 1067–1067, 2003.
6. Champin, P.A., Briggs, P., Coyle, M. and Smyth, B. Coping with Noisy Search Experiences. In *29th SGAI International Conference on Artificial Intelligence (AI-2009)*. Springer, pages 5-18, Cambridge, December 2009.
7. Schwarz, S. and Roth-Berghofer, T. Towards goal elicitation by user observation. In *Workshop on Knowledge and Experience Management* at GI FGWM, LLWA 2003.
8. Cordier, A., Mascret, B. and Mille, A. Extending Case-Based Reasoning with Traces. In *Grand Challenges for reasoning from experiences*, Workshop at IJCAI'09, Pasadena, CA. 2009.
9. Georgeon, O., Henning, M. J., Bellet, T., and Mille, A. (2007). Creating Cognitive Models from Activity Analysis: A Knowledge Engineering Approach to Car Driver Modeling. *International Conference on Cognitive Modeling*, Ann Arbor, MI: Taylor & Francis, pp. 43–48.