HAL
open science

Trace replay with change propagation impact in client/server applications

Raafat Zarka, Amélie Cordier, Elöd Egyed-Zsigmond, Alain Mille

To cite this version:

Raafat Zarka, Amélie Cordier, Elöd Egyed-Zsigmond, Alain Mille. Trace replay with change propagation impact in client/server applications. IC 2011, 22èmes Journées francophones d'Ingénierie des Connaissances, May 2012, Chambéry, France. pp.607-622. hal-00746727

HAL Id: hal-00746727

https://hal.science/hal-00746727v1

Submitted on 29 Oct 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Trace replay with change propagation impact in client/server applications

Raafat Zarka[1,2], Amélie Cordier[1,3], Elöd Egyed-Zsigmond[1,2], Alain Mille[1,3]

[1] Université de Lyon, CNRS
[2] INSA-Lyon, LIRIS, UMR5205, F-69621, France
[3] Université Lyon 1, LIRIS, UMR5205, F-69622, France
{raafat.zarka, amelie.cordier, elod.egyed-zsigmond, alain.mille}
@liris.cnrs.fr

Abstract: To help end-users mastering complex applications, it is often efficient to enable them to "replay" what they have done so far. In some cases, it is even more useful to enable them to modify some values of the actions they are replaying. However, while doing so, it very important to deal with the consequences of these changes on the remaining of the replay process. In this paper, we describe our models to enable replay of user's interactions and to manage impact propagation of changes during the replay process. These models are built upon traces, i.e. digital objects that enable us to record user interactions and to reuse them in different ways. We have implemented the replay process in a Web application called SAP-BO Explorer, an application helping business users to access large amounts of information. Our tool helps users to better understand the application.

Keywords: impact propagation, macro recording, bookmarks, replay traces, human computer interaction.

1. Introduction

With the multiplication and the rapid development of software systems and applications, we now have access to more and more tools, which are usually more and more complicated. While using these tools, we are often lost, usually because we lack time to understand applications, to get used to them and to exploit them efficiently. In response to this problem, some application designers came up with solutions for helping users either to discover the application or to learn how to be more efficient while using it. Providing a relevant assistance to users becomes a real challenge for application designers. Among the proposals for assistance strategies, we usually find tutorials, how-to, videos, assistants, training courses, etc. However, all these assistance strategies rest upon a static description of the application, hard-coded a priori. They are proposed to users in an identical way and thus, are not always well suited to specific needs of specific users. To overcome this issue, we have proposed, in a previous work (Zarka et al. 2010) to use interaction traces in order to provide user with a personalized and contextualized assistance based on previous experiences.

Interaction traces are relatively new digital objects. An interaction trace is a record of the actions performed by a user on a system. In other words, a trace is a story of the user's actions, step by step. Hence, traces enable us to capture users' experiences. Traces are recorded according to a pre-established model, so that they can be reused in different ways: replay, exploration, modification, modification plus replay, etc. Working with traces raises numerous research issues. How to collect, represent, store, and visualize traces? What mechanisms have to be implemented in order to allow user to browse their personal traces? How to implement a replay mechanism in a pre-existing system? How to take into account privacy issues when working with traces?

Recent researches provide us with solutions to some of these problems and enable us to work within an existing framework for manipulating traces (see (Champin et al. 2004), (Cordier et al. 2009) and (Settouti et al. 2009)). In this paper, we focus on a specific research question: how to replay a trace in a system and which issues are raised by the replay when the initial situation has been modified? To better understand this problem, let us consider the following example. A user makes a sequence of manipulations to improve a colored picture: transformation in gray-scale, selection of a scale of gray, luminosity attenuation for the selection, blur effect on the selection. Not satisfied with the result, he decides to go back to the initial state (the original picture) and to replay the whole set of actions, except from the transformation in gray-scale. The question is: "is the remaining of the actions still possible?"

The issue we address in this paper is then: how to enable a trace replay while monitoring the impact of a modification in the trace on the remaining of the process? In order to address this issue, we have firstly elaborated a mechanism enabling to do a simple replay of a trace (i.e. with no modification) from any point in the trace. Then, we have defined a model for impact analysis in order to manage impact propagation after a modification of the trace. Both models are described in this paper. The trace-replay mechanism has been implemented in the widely used SAP-BO Explorer application (SAP 2010), a web application enabling user to load, explore, visualize and export large quantities of data. SAP-BO needed a tool to help their users better understand the tool and this is the solution we designed for them. We have instrumented the initial application in order to collect interaction traces and we have developed a graphical interface in order to display the traces according to an ad-hoc representation. We have also instrumented the application in order to enable replay of recorded traces. The application is operational and a demo video is available[1].

[1] A demo video of trace replay and visualization is available at: https://liris.cnrs.fr/~rzarka/ReplayTraceDemo/

This paper is organized as follows. In section 2, we survey related work. Then, in section 3, we show how we use traces in order to enable replay of user's interactions. In section 4, we discuss the consequences of a change during the replay, and we propose an impact propagation model. Section 5 gives implementation details. Evaluation and discussion of our proposal are made in section 6. The paper ends with a conclusion and a description of future research issues.

2. Related work

In most of existing macro recording systems, users have to be proactive: they need to start and stop macro recording. Bookmark systems are one of the most common macro recording systems. They enable users to "replay" web pages. With Koala (Little et al. 2007), the user can record a sequence of actions and generate a script of keyword commands that can be replayed later. Recorded scripts are stored automatically on a wiki, which might be shared by a workgroup, allowing easy exchange and improvement of scripts. CoScripter (Leshed et al. 2008) is a Firefox plug-in created by IBM Research. It allows users to record and share interactions with websites. It records user actions and saves them in semi-natural language scripts. The scripts made are saved in a central wiki for sharing with other users.

WebVCR (Anupam et al. 2000) and WebMacros (Safonov et al. 2001) record web browser actions as a low-level internal representation, which is not editable by the user or displayed in the interface. All these systems require planning to enable recording while Smart Bookmarks (Hupp & Miller 2007) supports retroactive recording: it automatically captures users' interactions while they navigate the web and displays them through a graphical presentation. When users want to bookmark a webpage, the system automatically determines the sequence of commands needed to return to the page, and saves the sequence as a bookmark. While Smart Bookmarks lets users save or share actions from ongoing browsing sessions, ActionShot (Li et al. 2010) enables users to share actions they have performed before by providing them with a visual interface for browsing their entire history. ActionShot system is built on top of the CoScripter platform. History data is reused through the re-execution of recorded steps. Sharing also is supported through Facebook, Twitter or via email. Both ActionShot and Smart Bookmarks are generic, but they are implemented as Firefox extensions which is a limit. Besides, they cannot work with dynamic pages (e.g. Ajax or Flash based).

In Smart Bookmarks, users can modify parameters values before the bookmark starts running. However, these new values may affect commands and cause inconsistent states in the application. Hence, it seems relevant to

study impact propagation of these changes. Impact propagation analysis is widely studied in software engineering and database domains. In (Briand et al. 2003), the authors propose a UML model-based approach to impact analysis that can be applied before any implementation of the changes, thus allowing an early decision-making and change planning process. Most techniques to predict the effects of schema changes upon applications that use the database can be expensive and error-prone, making the change process expensive and difficult. In (Maule 2010), the authors present a novel analysis for extracting potential database queries from a program, called query analysis. The impacts of a schema change can be predicted by analyzing the results of query analysis, using a process they call impact calculation. Many systems also support impact analysis. One of them is Sybase Power Designer Modeling Tool that provides powerful methods for analyzing the dependencies between object models (Sybase 2010).

Table 1. Comparison table of related work

System	Representation	Simple Replay	Replay with change	Adaptation
WebMacros WebVCR	No	Proactive	No	No
Koala	Wiki Scripts	Proactive	No	No
CoScripter	Text, Firefox Extension	Proactive	No	No
Smart bookmarks	Graphical (screenshots), Firefox extension	Retroactive	Yes, without impact propagation	Classify buttons for side-effecting
ActionShot	Graphical text explanations, Firefox extension	Retroactive	Yes, without impact propagation	No
Photoshop	Actions list	Macro and undo command	Yes	Yes
Power Designer	Does not trace	Undo command	No	Impact rules
Trace Replay (Our approach)	*M-Trace* with text explanations	Retroactive	Yes	Impact rules and adapted values

Some applications allow users to replay their actions like Photoshop (Harrington 2009), by using undo or playback commands. In Photoshop, graphics designers and photographers have a number of processes they frequently perform on their images. By creating macros called “actions” they can automate many routine tasks using simple text files that are recorded in a macro-style. Whether is the goal is to convert an image for the Web or to transform a color photo into a black and white photo, designers can reduce several steps to a click on a single button. Users can create their own macro scripts which are mini recordings of commands. This is also what we would

like to provide, but in our case we need to apply macro recording for systems that do not support undo commands like most of client-server applications. In addition, we do not want to ask the user to start or stop recording his actions. **Table 1** shows a comparison between all the presented works according to the way they allow visualization of past actions and if they support the replay with or without change of values.

3. Simple trace replay (go back to a previous state)

In client-server applications, simple undo commands imply data interchange between client and server. This may take a lot of time (especially if the undo has to be repeated many times) and can cause server overload. Besides, such a problem may face loss of data issues. Last, it is not a scalable solution for situations where a lot of users access the server at the same time. For all these reasons, undo commands are hard to implement. Instead, to enable users to go back to a previous state, we propose to implement a "trace replay mechanism". This mechanism enables users to replay their interaction until they reach the expected state of the application. In order to implement this mechanism, we have defined a trace model (see **Fig. 1**).

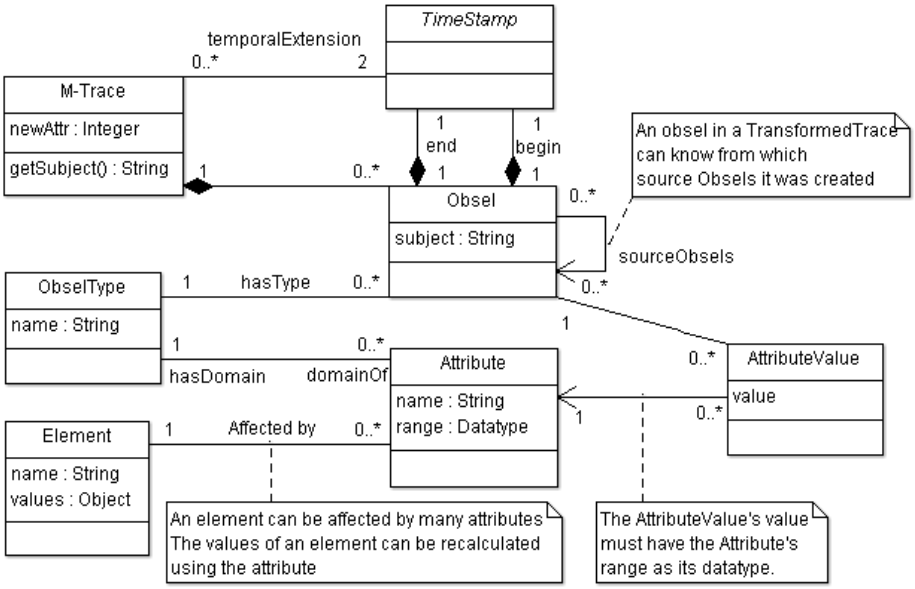

Fig. 1 Modified trace model to support trace replay

Each user's session is represented by a M-Trace which consists of a set of observed elements (obsels). Each obsel has a type and two timestamps representing its beginning and ending instants. Each obsel type has a domain

of attributes and indicates the values of its attributes respecting the range of the attribute type. An obsel can affect many elements at the same time. For example, pressing a “delete all” button can erase the values of many elements together. By using the obsel attribute values, we can calculate the new values for the related elements, where each obsel attribute concerns only one element. Using this model we can get all the obsels that can modify every element and all the elements that can be affected by an obsel. When capturing the traces we don’t need to store the values of elements at each time. We only store attributes and values of each obsel. For example if a user selects a chart, the value of the obsel will be the ID of the chart and not the whole information about the chart, so we need an element called “selected chart” that contains all the information about the selected chart.

3.1. Playback trace process

Our solution to go back to a previous state of the system is to playback users’ actions from a starting point (session start) and not by undoing last ones. When a user chooses to go back to a past state, he can choose the obsel that he wants to return to. The system will automatically go back to this state by replaying all the obsels that happened from the beginning of the session until the selected obsel; let’s call it the triggered obsel (the obsel where we want the system to play back to). **Fig. 2 [A]** shows a simple trace replay, a list of obsels starting from A to R, where R is the replay obsel and C is the triggered obsel. In R the user asked to replay traces to back with the system to its state when clicking on C. We can see that all the obsels that happened between C and R will be ignored (EDA). This replay will be done by one command which means one call from the client to the server. After replaying traces the system will go back to the past state and the user will continue his usage to the system, and new obsels will be collected. An Obsel R means that at this point a replay action happened.

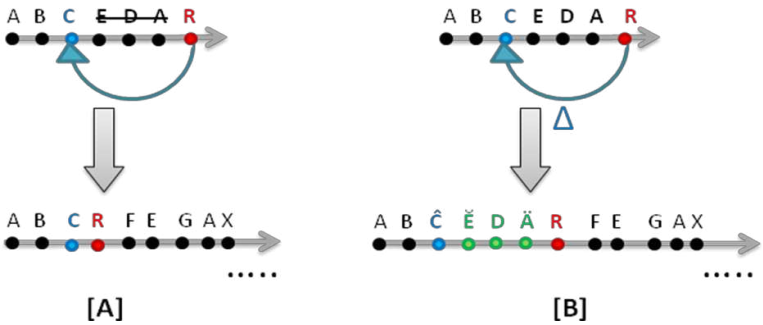

Fig. 2 [A] Simple Trace Replay, [B] Trace Replay with change

By replaying the obsels we can calculate the values of these elements at the replaying point. **SimpleReplay** algorithm gets M-Trace and the triggered obsel as input and goes back to a previous state. Firstly, it gets the subset of the trace that should be replayed starting from the first obsel to the triggered one by a chronological order. Then this trace will be optimized by using the optimization algorithm to delete extra obsels. Each element gets its default values and then a loop on all the obsels runs, where at each time the element values are updated according to the attributes of the current obsel. At the end, the new element values are updated making the system going back to this state. The replay event is also captured as a new obsel and taken in consideration during the analysis.

```
Program simpleReplay (M-Trace, TriggerredObsel)
    ReplayedTrace := getSubTrace(0, position(TriggeredObsel))
    optimize(ReplayedTrace)
    Elements := getDefaultValues()
    For pos := 0 to getObselCount(ReplayedTrace)-1
        Obsel := ReplayedTrace[pos]
        Attributes := getAttributes(Obsel.Type)
        For each attribute in Attributes
            Value := getAttributeValue(Obsel, attribute)
            Elem := getAffectedElement(attribute)
            Elements[elem] := GenerateElementValue(value)
        End For each
    End For
    update(Elements)
End Program
```

3.2. Optimized trace replay process

As not all the obsels play a role for changing the state of the system, the replay process can be optimized by reducing the number of replayed obsels. In addition, in some cases many obsels can be ignored, either because they have been canceled by other obsels or because of reset values. According to that, we don’t need to go through all the obsels in order to go back to the triggered one. Analyzing the previous obsels to get the right values of the elements enables us to optimize the replay process. We can get an optimized chronological list of obsels from the beginning of the session to the triggered obsel; this list will be used to generate the values for each element. Optimize algorithm tries to delete all unnecessary obsels that induce loops in the trace, For example, in the simple replay obsel, the subTrace from replay obsel to triggered obsel should be deleted. The same thing is also done for a reset obsel which means deleting all the obsels from the beginning to the reset obsel. So we consider that there is a list of unnecessary loop obsels in the trace, and in this algorithm all these loops will be deleted as shown in **Fig. 3**.

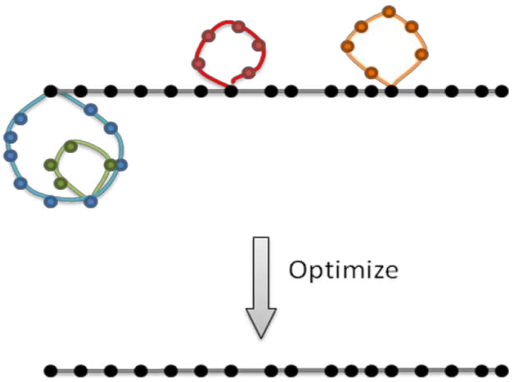

Fig. 3 Trace Replay Optimization

4. Replay traces with impact propagation

In this section we describe how we can replay traces after modifying an obsel by handling the consequences of changes on elements before actually performing these changes. This is illustrated on **Fig. 2 [B]**. R is a replay obsel that triggers a replay of the trace after doing a change on the values of the triggered obsel C. Because of a change in one of the attribute values of C, the values of some other obsels could be inconsistently modified, like E and A, while other obsels may remain consistent, like D. We need to calculate the new values, in order to take into account this modification. Then the trace can be replayed with these new values. After that the user can continue to use the system. We face many questions like: how can we determine the elements affected by a change? Can we be proactive and specify the appropriate new values, without asking the user to enter the new values? How can we replay the next obsels after applying this change? To answer these questions we propose to define impact rules of dependencies between the elements for manipulating the consequences of a change.

4.1. Impact rules for element dependencies

Impact rules define the dependencies between the elements in the system in order to be able to identify the elements that are affected by a change in another element, and to specify the modifications that could be done on the affected elements to stay consistent and valid. Each rule includes a source element and the condition on its values that specifies the dependence with a destination element and the condition on its values. A rule says that if specific conditions for the values of the source element are fulfilled then some of the values of the destination element determined by the destination condition cannot exist, which requires replacing these values by an adapted value.

Definition: Impact rule

Let E be a set of elements. Each element has a name and some values. Let O be a set of operations and F be a set of functions. We can define an impact rule R as an implication of the form:

$R = (E_S, C_S) \rightarrow (E_D, C_D) : Æ$, where E_S, E_D, $Æ \in E$, and E_S is the source element, C_S is the source condition, E_D is the destination element, C_D is the destination condition, and Æ is the adapted element. C_S and C_D are conditions based on operations and functions on the values of the elements. Conditions are composed of operations (O) and functions (F) on elements values. Operations can be logical (and, or, not, etc), mathematical (+, -, *, /, etc) or others. Functions can be grouping functions like (max, sum, min, count, avg) or custom functions like (isNumber, isHoliday, etc).

For each application, system's experts define impact rules for the dependencies between the elements, to determine the consequences of modifying a past obsel. We can get all the impacted obsels for each rule from the entity of the relations between elements and obsels. If we find impact rules having the elements of the modified obsel as source elements and their values satisfying the source conditions, then, for each destination element, if its value satisfies the destination condition, we need to replace the destination element by the adapted one. Adapted values can be specified manually as default values or can be generated automatically using past traces. For example, in SAP-BO Explorer, we consider an impact rule like: if the number of selected measures is greater than one, the element "Chart" cannot be of type "Pie". If a user asks to replay a trace after modifying the number of selected measures that activated this rule, and if there was a successor obsel for changing the chart type to "Pie", then this obsel will not be valid anymore because of this rule, and the chart type will be automatically changed according to the adapted value to be "Vertical Bars". The rule will be as following:

E_S = Selected Measures $\quad$ C_S = (Count () > 1)

E_D = Chart $\quad$ C_D = (type = "Pie")

Æ = (Type = "Vertical Bar")

The user can replay a part of his session after modifying some of the obsels values. These modifications can be of many types like shifting obsel by changing their timestamps, thus causing a change in the order between obsels, updating a value for an attribute of an obsel, or even deleting an obsel. By using impact rules we can determine the consequences of a change and the adapted values. In case of not finding an adapted value of an element or the absence of an impact rule, the corresponding obsels will be invalid. Then the user will have to select the suitable value manually; otherwise the replay process will fail.

4.2. Retrieving adapted value from past traces

When a user adds a new impact rule, the system asks him to choose the adapted value from a list of possible values, or to keep the system calculating it automatically using past traces. For this purpose, we propose to use a retrieval algorithm similar to the algorithm we presented in (Zarka et al. 2010). In the original algorithm, we tried to retrieve episodes similar to the current one without taking obsels values in consideration because we just wanted to know the next recommended obsels. So, in order to make this algorithm useful for finding the adapted values, we need to make a comparison between the values of the obsels. In addition, we want to retrieve the adapted value for the destination element and not the next recommended obsels

Get adapted value algorithm starts by selecting a subset of the trace from its beginning to the modified triggered obsel. Then it retrieves all the past similar episodes to the current one. Similarity includes values comparison. For each similar episode, it calculates the final value of the corresponding element (destination element in the impact rule) as we did in the simple replay, without updating the system. If there is more than one value, we take the one that occurs the most often and we consider it as the adapted one. If we are not able to retrieve any episode, we keep this element as an invalid element until another obsel modifies its value, otherwise the replay process will fail and the system will ask the user to choose the value manually.

5. Implementation

In the previous sections, we have described the models that we have defined to support replay of user interactions by exploiting traces. In this section, we show how we have implemented our trace replay model into the SAP-BO Explore application.

5.1. Trace collecting and visualization

Firstly we modified SAP-BO Explorer for being able to collect obsels. SAP-BO Explorer is divided into two parts. Server part is implemented in Java. The management of users' sessions is done in this part, thus enabling many users to work on the system at the same time. The client part is a Flex application; each user has a web application where he can do his exploration. The traces are collected in the client side. **Fig. 4** shows a snapshot of the user interface. Each time a user tries to use the system, a new session is opened. Each session contains many obsels, and each action of the user is collected as an obsel presented in a XML format specifying the obsel type, timestamps, and the values of this obsel. We consider that the interface of

SAP-BO Explorer is divided into task-oriented blocks, where each block contains obsels dedicated to similar kinds of tasks. The interface consists of blocks for measures, categories, visualization, export, search, etc. For example the measures block contains many types of obsels like select measure, add calculation, edit calculation, etc. For example, when a user tries to select a measure, we capture this action as an obsel of the type “Select measure” from the second block “Measures block”. The obsel has for value “Trade USD” and is time stamped with the current timestamp.

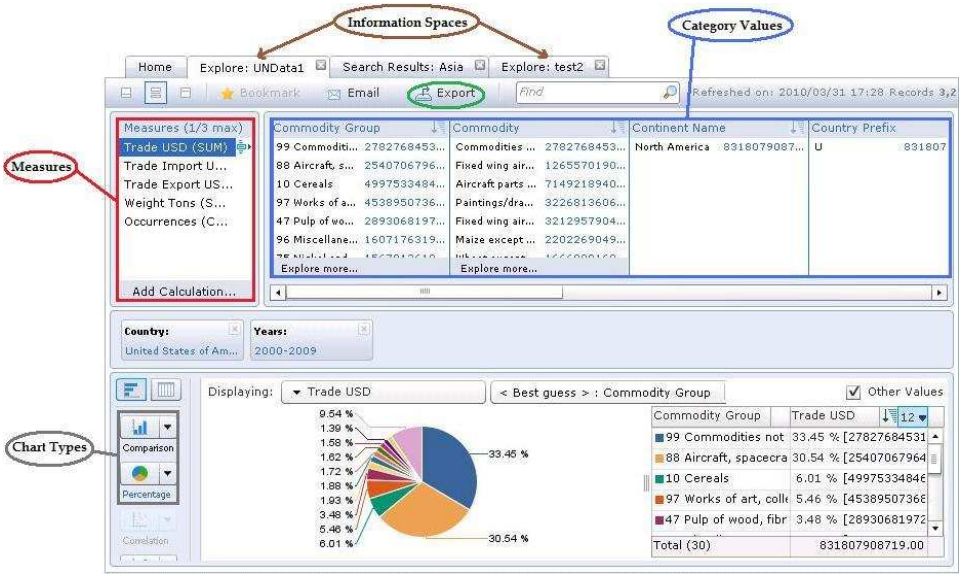

Fig. 4 SAP-BO Explorer user interface

Each session is presented as a M-Trace stored in XML and has a unique ID, contains the ID of the user who did this session, and the temporal list of obsels that happened in this session. When a user log himself in SAP-BO Explorer, a request to the server-side is sent in order to open a new session. This triggers the creation of a new XML output file for this session. Each time a new obsel is collected; it is formatted in XML format and sent to the server in order to be added to the session file. Each user can open and manipulate many Information Spaces at the same time. An Information Space is a collection of objects mapped to data for a specific business operations or activities. All the obsels of a session, whatever the Information Spaces they belong to, are stored in the same file.

We have developed a new interface to visualize users’ traces displaying a graphical representation of what they have done so far (see **Fig. 5**). Each obsel is captured according to our model classified according the available types and represented as colored bullets. Obsels appear on the left side of the

interface as a chronologically ordered list from the beginning of the session to the most recent obsel. By clicking on an obsel, we can see its description on the right side of the interface. Obsel's values are visualized in the form of a tree of attributes and their values.

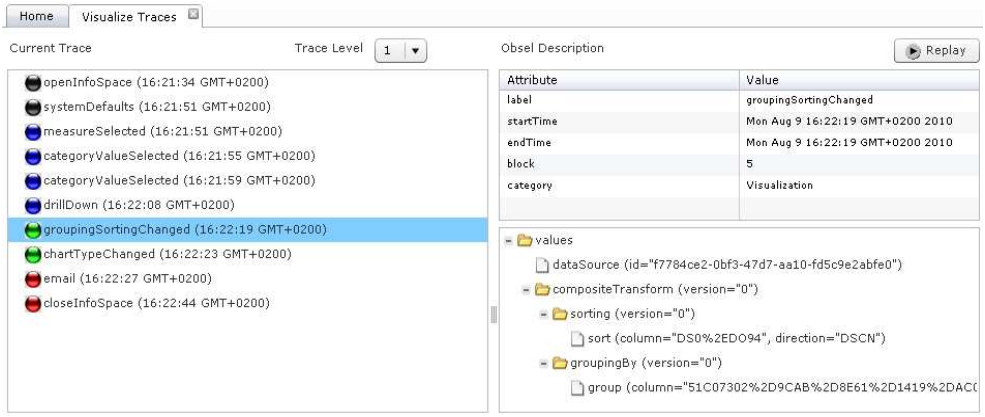

Fig. 5 Trace Visualization Interface

5.2. Trace replay implementation

If a user wants to go back to a previous state, he can at any time select the triggered obsel from the list of captured obsels and click on replay button (see **Fig. 5**). The system will automatically replay traces to go back to this state. A new obsel will be added to the obsels list of type 'Replay'. Its values are set according to the values of triggered obsel. This new obsel indicates that a replay action has occurred here and has triggered a previous obsel. As we explained before, the optimization algorithm uses replay obsels to minimize the number of the replayed obsels by deleting the obsels that are skipped in the replay action.

Each element has different type and number of values from other elements. For not analyzing each element in a different way, we need to make it more general. By using introspection we can determine the type of an object at runtime. Introspection refers to the ability to examine something to determine what it is, what it knows, and what it is capable of doing. Introspection gives us a great deal of flexibility and control. To do that we used Object as type of the values attribute of an obsel, which means that this attribute can have any type of values. We do introspection on this attribute in order to determine the content of it and then to manipulate it in a general way.

6. Evaluation and Discussion

We have implemented our replay method within the SAP-BO Explorer application. However, this method can be applied in any system. To enable trace replay, the first step is to collect traces. For this purpose, we use a model, the M-Trace, that enables us to collect all the traces according to the same abstract model. We have experimented with our system by using many types of datasets and by considering all obsels types, opening many sessions together and trying to go back to previous states many times in the same session. We even succeeded to go back to all sessions at the same time by one single go-back command. The execution time of the replay process is very fast, it is like any other action in the application, which means the time of message exchanging between the client and the server.

Systems like ours face number of challenges like replaying traces for already closed sessions, optimizing replay after modifying past obsels and rechecking impact rules after modifying elements values. But they also face more general problems, as mentioned in the discussion section. For example, in (Hupp & Miller 2007), the following issues are raise: privacy of the user and his permission to trace him, security of the system while collecting and visualizing traces, protection of users from undesirable side-effects triggered by the replay, and the robustness of the replay after doing some changes in the system.

When implementing our system, we also faced specific problems. For example, in SAP-BO Explorer the same user can open many sessions at the same time. We had to deal with the problem of replaying the trace of a closed session. Our replay process can handle this case by reopening the session, with default values and by applying all the replayed obsels until the triggered one. As we have not implemented yet the replay with changes, we have not faced the problem of optimizing the replay after these modifications. Application of impact rules can be recursive; a modification on an obsel value can have an impact on other obsel values if obsels are related. To deal with these problems we need to develop a graph of impact propagation to be able to solve loops problems and to know the dependencies between different obsels and elements. This will be one of our future works.

When the trace includes obsels that have secure and sensitive information like passwords and credit card numbers, our system detects and obscures the password when visualizing it. But it still needs a lot of enhancements and rules to detect this information and secure it, by notifying the user about it or even asking him to re-enter it again. Our system continuously collects and records user’s interactions which constitute a potential risk to privacy and security. This problem is share by all the systems that record rich history

traces (web browsers, recommendation systems, etc.). Dealing with this issue is out of the scope of our study. However we do notify our users that all their interactions are recorded. Side-effects are another issue we have to deal with. Indeed, replaying a trace may have unexpected consequences and can damage the system or cause deletion of. In our current implementation, we do not deal with this problem. However, we think that the proposition described in (Hupp & Miller 2007) is relevant to solve such a problem. The idea is to classify obsels into two classes: side-effecting and non side-effecting. This makes easier the annotation of critical obsels. Last, we have to face robustness issues. Indeed, we have to make sure that the trace system is still usable after major changes either on data or on processes of the system. This question is also out of the scope of our study because it is mainly related to the trace collecting phase. We make the assumption that robustness issues are handled by the trace-based system, responsible for traces management.

7. Conclusion and future work

In this paper we have described an approach using of interaction traces to allow users to return to a particular state of an application. This approach is an alternative way of undoing actions in applications where undo commands are not available (such as client-server applications). For this purpose, we use play-back of traces. Playback can be identical to the original trace or can introduce different action parameters. We analyze the impact propagation of changes performed on past actions. This work has been conducted in collaboration with SAP Business Objects and the application we used to implement our approach is SAP-BO Explorer. The aim of our contribution within this project was to support replay process in a client-server application, where classical undo commands cannot be implemented. The main contribution of this paper shows how we can playback interaction traces, in an optimized way, in order to go back to a particular state of the application. For that purpose, we have introduced the concept of predefined impact rules and we have built an algorithm that discovers adapted values of obsels affected by changes.

At the time being, the collect process and the simple replay process are implemented. In future work, we plan to address issues mentioned in the discussion concerning side-effects, robustness, and security. In addition, we are interested in studying how we can extract users' experiences in order to reuse them for assistance purpose.

ACKNOWLEDGMENTS

We thank Francoise Corvaisier, member of SAP-BO enterprise for her support, thoughts and for giving us the opportunity to do this work in SAP-BO. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the sponsors.

References

Anupam, V. et al., 2000. Automating Web navigation with the WebVCR. Computer Networks, 33(1-6), pp.503-517. Available at: citeseer.ist.psu.edu/anupam00automating.html.

Briand, L.C., Labiche, Y. & O'Sullivan, L., 2003. Impact analysis and change management of UML models. In International Conference on Software Maintenance 2003 ICSM 2003 Proceedings. IEEE Comput. Soc, pp. 256-265. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1235428.

Champin, P.-A., Prié, Y. & Mille, A., 2004. MUSETTE : a framework for Knowledge from Experience. In *EGC'04, RNTI*-E-2 (article court). Cepadues Edition, pp. 129-134. Available at: http://liris.cnrs.fr/publis/?id=1338.

Cordier, A., Mascret, B. & Mille, A., 2009. Extending Case-Based Reasoning with Traces. In Grand Challenges for reasoning from *experiences, Workshop at IJCAI'09*. Available at: http://liris.cnrs.fr/publis/?id=3862.

Harrington, R., 2009. Understanding Adobe Photoshop CS4 The Essential Techniquesfor Imaging Professionals, Peachpit Press.

Hupp, D. & Miller, R.C., 2007. Smart bookmarks: automatic retroactive macro recording on the web. Proceedings of the 20th annual ACM symposium on User interface software and technology, pp.81-90. Available at: http://portal.acm.org/citation.cfm?doid=1294211.1294226.

Leshed, G. et al., 2008. CoScripter: automating & sharing how-to knowledge in the enterprise. CHI 08 Proceeding of the twentysixth annual SIGCHI conference on Human factors in computing systems, pp.1719-1728. Available at: http://portal.acm.org/citation.cfm?id=1357054.1357323.

Li, I. et al., 2010. Here's what i did: sharing and reusing web activity with ActionShot. In Proceedings of the 28th international conference on Human factors in computing systems. ACM, p. 723–732. Available at: http://portal.acm.org/citation.cfm?id=1753326.1753432.

Little, G. et al., 2007. Koala: capture, share, automate, personalize business processes on the web. CHI 07 Proceedings of the SIGCHI conference on Human factors in computing systems, pp.2-5. Available at: http://portal.acm.org/citation.cfm?id=1240624.1240767.

Maule, A., 2010. Impact analysis of database schema changes. UCL (University College London). Available at: http://eprints.ucl.ac.uk/19497/.

Safonov, A., Konstan, J.A. & Carlis, J.V., 2001. Beyond Hard-to-Reach Pages : Interactive , Parametric Web Macros. Proc Human Factors and the Web, pp.1-14.

SAP, 2010. SAP Business Objects Explorer: Explore your business at the speed of thought. Available at: http://www.sap.com/solutions/sapbusinessobjects/large/business-intelligence/search-navigation/explorer/index.epx.

Settouti, L.S. et al., 2009. A Trace-Based Systems Framework : Models, Languages and Semantics. Available at: http://hal.archives-ouvertes.fr/inria-00363260/PDF/trace.pdf.

Sybase, 2010. Power Designer: Impact and Lineage Analysis. Available at: http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.dc38093.1520/doc/html/rad1232025100240.html.

Zarka, R. et al., 2010. Providing assistance by reusing episodes stored in traces: a case study with SAP Business Objects Explorer. In F. Le Ber & J. Renaud, eds. 18ème Atelier « Raisonnement à Partir de Cas ». Strasbourg: hal-00497210, p. 91--103. Available at: http://hal.archives-ouvertes.fr/docs/00/49/72/10/PDF/actesRAPC10.pdf.