Trace-based reasoning for user assistance and recommendations

Raafat Zarka



Order Number: Year: 2013

Université de Lyon

Institut National des Sciences Appliquées de Lyon

Laboratoire d'InfoRmatique en Image et Systèmes d'information

PhD Thesis

Trace-Based Reasoning for User Assistance and Recommendations

A dissertation submitted for the degree of

Doctor of Philosophy (PhD) at INSA de Lyon

Discipline: Computer Science
Ecole Doctorale: InfoMaths

Presented and publicly defended by

Raafat Zarka

on 04 *December*, 2013

in front of a jury composed of:

Directors:	
Alain Mille	Professor, Université Claude Bernard Lyon I, France
Amélie Cordier	Associate Professor, Université Claude Bernard Lyon I, France
Elöd Egyed-Zsigmond	Associate Professor, INSA de Lyon, France
Reviewers:	
Agnar Aamodt	Professor, Norwegian University of Science and Technology, Norway
Serge Garlatti	Professor, Telecom Bretagne, France
Examiners:	
Sylvie Calabretto	Professor, INSA de Lyon, France
Sylvie Després	Professor, Université Paris 13, France
Miltos Petridis	Professor, University of Brighton, United Kingdom
Invited:	
Luc Lamontagne	Professor, Laval University, Québec, Canada

This work was supported by the Rhône-Alpes Region through the Explo'ra Doc grant for an international mobility to Laval University in Quebec, Canada during a period of six months from March 2012 to August 2012.

To Syrian people.. Wish they find peace..

Acknowledgements

The best and worst moments of my doctoral journey have been shared with many people. It would not have been possible to write this doctoral thesis without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here.

First of all, I would like to express my sincere gratitude to my advisers, Prof. Alain Mille, Dr. Amélie Cordier and Dr. Elöd Egyed-Zsigmond, for supporting me during my PhD. The thoughts, guidance, enthusiasm, and immense knowledge of Alain helped me in all the time of research and writing of this thesis. Alain is a great human with a big heart, you never forget once you meet him. The good advice, support and motivation of Amélie, have been invaluable to achieve my PhD, for which I am extremely grateful. I appreciate all her contributions of time, ideas, and thoughts to make my PhD experience productive and stimulating. It has been an honor to be her first PhD student. Elöd patiently provided the vision, encouragement and advise necessary for me to proceed through my PhD. Special thanks to him for the insightful discussions and collaborations that enriched my work. My sincere thanks also go to Prof. Luc Lamontagne for offering me the summer internship opportunity in his group in Quebec. Working with him was pleasant and we exchanged a lot of ideas. He has an adorable personality and he is always ready to help.

I wish to thank the reviewers of my PhD thesis, Professors Agnar Aamodt and Serge Garlatti for their detailed report and their insightful comments. I am also thankful to the other members of my PhD committee, Professors Miltos Petridis, Sylvie Calabretto, Sylvie Després and Luc Lamontagne for their valuable suggestions while evaluating my research work.

Members of LIRIS Lab and staff of INSA de Lyon and Université Lyon 1 also deserve my sincerest thanks, the discussions and cooperation with all of them have contributed substantially to this work as well. In addition, I had the chance to meet great people there and friend them. I will never forget the many wonderful lunches, rich discussions and fun activities we have done together, specially playing billiards with Sarah and Houssam.

I have to thank the French government for financially supporting this research work. I also thank the Rhône-Alpes Region for providing me with the Explo'ra Doc grant for an international mobility to Laval University in Quebec, Canada. I would like to thank all my colleagues in Quebec. I am happy that I have made great friends there such as Wejden and Sinda.

I also thank Webcastor for their collaboration with us. I am particularly thankful to Jean-Charles Betrancourt and Larbi Fettah for giving me the opportunity to validate my research results using the Wanaclip application. Special thanks to Thomas Piccolo for his work in implementing the visualization module of the recommendations during his graduation project.

My friends in Syria, France, Canada and other parts of the world (too many to list here but you know who you are!) were sources of laughter, joy, and support. Special thanks go to Tareq for all the great moments we spend together in Syria and

Canada and for the discussion we had in planning our futures. I would like also to thank Raafat Hantoush for his support and for helping me with some administrative issues in Syria. My life in Lyon was made enjoyable in large part due to many friends I have met here. I am especially grateful for time spent with my friends Joudy and Merza enjoying different activities, travels and training together in gym club. I would also like to thank Rana, Hammam, Zeina, Georges, Ghezlan, Houssam, Salam, Mazen, Fatima, Abdullatif, Hasan, Mahmoud, Sarah, Melhem, Sarra, Simon and Adriana, for all the great times that we have shared and caring they provided. Special thanks also go to Abir, Tareq, Merza, Soleiman and Benoît for helping me to correct language mistakes before submitting my PhD thesis.

I owe my deepest gratitude to my parents Nada Talab and Mohamad Maky Zarka, my sisters and brothers: Rania, Hania, Nashaat, Raneem and Raoof for their endless love and support. Their love provided my inspiration and was my driving force. I owe them everything and wish I could show them just how much I love them. Last but not least, I want to express my appreciation and love to my fiancée Bayane for her patience, encouragement, and love especially in these moments.

Finally, I would like to dedicate this work to the Syrian people. I wish they find peace and obtain their rights.

Abstract

In the field of digital environments, a particular challenge is to build systems that enable users to share and reuse their experiences. In this thesis, we are interested in the general problem of contextual recommendations for specific web applications in a particular context: complex tasks, huge amount of data, various types of users (from novice to professional), *etc.* Contextual recommendations are a form of user assistance. More precisely, we focus on providing user assistance which takes into account the context and the dynamics of users' tasks. We seek to provide dynamic recommendations that are enriched by new experiences over time. To provide these dynamic recommendations, we make use of Trace-Based Reasoning (TBR). TBR is a recent artificial intelligence paradigm that draws its inspiration from Case-Based Reasoning. In TBR, interaction traces act as an important knowledge container. They help to understand users' behaviors and their activities. Therefore, they reflect the context of the activity. Traces can feed an experience-based assistant with the adequate and appropriate knowledge.

In this thesis, we introduce a state of the art about dynamic assistance systems and the general concepts of Trace-Based Systems. In order to provide experience-based assistance, we have made several contributions. First, we propose a formal representation of modeled traces and a description of the processes involved in their manipulation. Notably, we define a method for computing similarity measures for comparing modeled traces. These proposals have been implemented in a framework named $\mathcal{T}$Store for the storage, transformation, management, and reuse of modeled traces. Next, we describe a trace replay mechanism enabling users to go back to a particular state of the application. This mechanism supports impact propagation of changes during the replay process. Last, we define a recommendation approach based on interaction traces. The recommendation engine is fed by interaction traces left by previous users of the application and stored in a manager, such as $\mathcal{T}$Store. This approach facilitates knowledge sharing between communities of users and relies, among other things, on the similarity measures mentioned above.

We have validated our theoretical contributions on two different web applications: SAP BusinessObjects Explorer[1] for data reporting and Wanaclip[2] for generating video clips. The trace replay mechanism is demonstrated in SAP BusinessObjects Explorer. Trace-Based Reasoning recommendations are illustrated with Wanaclip to guide users in both video selection, and the actions to perform in order to make quality video clips. In the last part of this manuscript, we measure the performances of $\mathcal{T}$Store and the quality of recommendations and similarity measures implemented in $\mathcal{T}$Store. We also discuss the results of the survey that the users of Wanaclip answered in order to measure their satisfaction. Our evaluations show that our approach offers satisfactory recommendations and good response time.

Keywords: Dynamic Assistance, Contextual Recommendation, Similarity Measures, Replay Traces, Modeled Traces, Trace-Based System, Case-Based Reasoning, Trace-Based Reasoning, Human Computer Interaction.

[1]SAP BusinessObjects Explorer: http://help.sap.com/boexpl

[2]Wanaclip: http://www.wanaclip.eu

Résumé

Dans le domaine des environnements numériques, un enjeu particulier consiste à construire des systèmes permettant aux utilisateurs de partager et de réutiliser leurs expériences. Cette thèse s'intéresse à la problématique générale des recommandations contextuelles pour des applications web dans un contexte particulier : tâches complexes, beaucoup de données, différents types d'utilisateurs (du débutant au professionnel), *etc.* Les recommandations sont une forme d'assistance à l'utilisateur. Plus précisément, nous cherchons à fournir une assistance à l'utilisateur en prenant en compte le contexte et la dynamique des tâches que l'utilisateur effectue. On cherche à fournir des recommandations dynamques qui sont enrichies au fur et à mesure des expériences. Pour fournir ces recommandations dynamiques, nous nous appuyons sur le Raisonnement à Partir de l'Expérience Tracée (RàPET). Le RàPET est un paradigme d'intelligence artificielle relativement récent, qui tire son inspiration du Raisonnement à Partir de Cas (RàPC). Dans le RàPET, les traces d'interaction constituent d'importants conteneurs de connaissances. Ces traces permettent de mieux comprendre le comportement des utilisateurs et leurs activités. Par conséquent, elles représentent également le contexte de l'activité. Les traces peuvent donc venir nourrir un assistant à partir d'expérience en lui fournissant des connaissances appropriées.

Dans cette thèse, nous présentons un état de l'art sur les systèmes d'assistances dynamiques et nous rappelons les concepts généraux des systèmes à base de traces. Afin de proposer une assistance à base d'expérience, nous avons effectué plusieurs contributions. Tout d'abord, nous avons proposé une formalisation des traces modélisées et des processus qui permettent de manipuler ces traces. Nous avons notamment défini une méthode pour établir des mesures de similarité afin de comparer des traces modélisées. Nous avons implémenté ces propositions dans un outil appelé $\mathcal{T}$Store. Cet outil permet le stockage, la transformation, la gestion et la réutilisation des traces modélisées. Ensuite, nous avons proposé un mécanisme de rejouage de traces pour permettre aux utilisateurs de revenir à un état précédent de l'application. Ce mécanisme gère les conséquences de la propagation des changements lors du processus de rejouage. Enfin, nous avons décrit une approche de recommandations à partir de traces. Le moteur de recommandations est alimenté par les traces d'interactions laissée par les précédents utilisateurs de l'application. Ces traces sont stockées dans un gestionnaire de traces tel que $\mathcal{T}$Store. Cette approche facilite le partage de connaissances entre des communautés d'utilisateurs, et s'appuie, entre autres choses, sur les mesures de similarité proposées plus haut.

Nous avons validé nos contributions théoriques à l'aide de deux applications web : SAP BusinessObjects Explorer[3] pour l'analyse de données, et Wanaclip[4] pour la génération semi-automatique de clips vidéos. Le mécanisme de rejouage de traces est démontré dans SAP BusinessObjects Explorer. Les recommandations à base de traces sont illustrées dans l'application Wanaclip. Elles guident l'utilisateur à la fois dans la sélection des vidéos, et dans les actions à effectuer pour réaliser des clips vidéo de bonne qualité. Dans la dernière partie du manuscrit, nous mesurons les performances de $\mathcal{T}$Store et la qualité des recommandations et des mesures de similarité qu'il implémente. Nous discutons aussi des résultats du sondage que nous avons appliqué aux utilisateurs de Wanaclip pour mesurer leur satisfaction. Nos évaluations montrent que notre approche offre des recommandations satisfaisantes et un bon temps de réponse.

Mots-clés: Assistance Interactive, Recommandations Contextuelles, Mesures de Similarité, Rejouage de Traces, Traces Modélisées, Système à Base de Traces, Raisonnement à Partir de Cas, Raisonnement à Partir de Trace, Interaction Homme-Machine

[3]SAP BusinessObjects Explorer : `http://help.sap.com/boexpl`

[4]Wanaclip : `http://www.wanaclip.eu`

Contents

List of Figures

List of Tables

List of Definitions

Chapter 1

Introduction

Contents

While working on a computer application, have you ever wanted to do a task that you did before but cannot remember how you had done it previously? No doubt this task has been done by other people before and you want to benefit from their expertise when you are trying to do it yourself. Do you want to get help to accomplish your tasks easily? These three different issues raise the need of having a computer system that is capable of helping us perform our tasks more easily. Dynamic experience-based assistance is a general problem that has been studied and theorized by different researchers such as [Minsky 1975, Schank 1983]. We need to provide an assistant that offers advises based on our past experiences or those of others. For example, you want to design slides to present your work. You do not remember how to use a predefined template even though you had already done that before. You saw your friend adding transitions and animations to his presentation and you want to do the same. In such a scenario, it would be ideal if an assistant guided you through the different steps of design process to create a stunning professional presentation. Even if you are experience in preparing presentations, you may still need help to produce better presentations or to discover new functionalities.

What web pages offer you the information that you want? Want some recommendations of recent music that is similar to your favorite playlist that you were playing this morning? Need assistance on finding some good papers to enrich the state of the art you are doing for a research? The huge amount and the diversity of resources makes data selection, exploration and exploitation difficult. Recommendation Systems help filtering data by offering suggestions related to various decision-making processes. Recommendations should be related to the context in order to better understand the performed activity and to find the suitable suggestions. Contextual recommendations play an important role in personalizing and reusing pre-existing data to help performing tasks easily.

Current context and modern usage of computer tools, especially on the web, offer new ways to help users accomplishing their tasks more easily and efficiently. Taking into account that users are often faced with complex applications and environments generating data, information, resources, *etc.* Among the many possible solutions to help these users, the re-mobilization of their

own experiences and those of others, when shared, seems an interesting approach. The importance of this solution lies in the possibility of leveraging previous experiences and expertise to provide solutions to the current problems. Thus, this solution reduces the time and effort required to solve the problem and mitigates the obstacles and difficulties that have been resolved in the past. In addition, there are some atypical cases that were previously processed, these solutions may be appropriate to the current situation. For example, while adding a figure to your document, it may appear crossing the margins or misplaced. This problem is unusual and the reason behind it is often hard to guess. However, given the past experiences, a similar case may be found where the solution was to change the text wrapping.

1.1 Context and Motivations

In this thesis, we are interested in the general problem of user assistance applications in a particular context: web, complex tasks, huge data, various types of users (from novice to professional). The role of assistance applications is to facilitate learning and knowledge sharing in a dynamic environment. Being able to offer assistance to various computer applications needs explicit representation of the knowledge needed to perform a software process. Moreover, to support the rapid developing nature of software process and the constantly changing needs of users, an assistance system should have mechanisms of capturing dynamic knowledge in such a way that it can be used for further action. Therefore, two problems are raised: A. identify assistance providing mechanisms and B. discover knowledge to feed these mechanisms. Obviously, these two problems work in a virtuous circle where mechanisms may both provide assistance to user, and also enrich knowledge either by the feedback, or simply by experiencing new situations.

1.1.1 Mechanisms for Assistance Offering

The main issue is to achieve general assistance systems that take context into account. Providing a relevant assistance to users becomes a real challenge for researchers and application designers. Some application designers came up with solutions for helping users learn the application or to become more

efficient using it. [Paquette et al. 1994] proposes to classify assistance systems as they are proactive (if the system detects a need for assistance and provides assistance to user), reactive (if the system is at the request of user), or mixed. [Ginon et al. 2013d] identifies three main needs for assistance: the discovery of the system, achieving a task and improving practice.

There are various proposals for assistance strategies, we usually find:

- **Instruction manuals** in form of text or videos such as [Microsoft_Office 2010] online videos.
- **Context-sensitive help** is obtained from a specific point in the state of the software, providing help in situations that are associated with that state, such as in [Capobianco and Carbonell 2001, Matthews et al. 2000].
- **Advisor systems** *e.g.* [Richard and Tchounikine 2004] display messages and advice in the form of popup windows.
- **Conversational agents**, textual [Matthews et al. 2000] or graphical [Dufresne and PromTep 2006].
- **Recommendation systems** using a model built from the characteristics of an item (content-based approaches) [Pazzani and Billsus 2007] or the user's social environment (collaborative filtering approaches) [Su and Khoshgoftaar 2009].
- **Tutorials** *e.g.* Vismod which provides learners with a tutorial to help them understand the representation of their profils [Zapata-Rivera and Greer 2004].
- **Adaptive interfaces** like in Microsoft Office 2003 which is customized depending on the needs and preferences of the user.
- **Reuse of experience** allows a user to reuse the past actions [Mille et al. 2006].
- **Communities of practice** that designate a group of people who share common practices and allow them to help each other [Lave and Wenger 1991] *e.g.* forums and Facebook groups.

However, all these assistance strategies rest upon a static description of the application, hard-coded *a priori*. Static assistance is provided to all the users. Assistance is not always well suited to particular needs of specific users. Among the proposals for assistance strategies, we are interested in reusing experience and the contextual help via recommendation systems.

Recommendation systems are a particular form of information filtering systems. It provides assistance to users by helping them discover items they might not have found by themselves. They are software tools and techniques providing suggestions for items to be of use to a user [Burke 2007, Resnick and Varian 1997]. Most recommendation systems deal with applications having only two types of entities, users and items, and do not put them into a context when providing recommendations. Context is providing information that can influence the perception of the usefulness of an item for a user. For this reason recommendation systems must take into account this information to deliver more useful (perceived) recommendations [Ricci 2010]. The fact that user's interact with systems within a particular "context" and ratings for items within one context may be completely different from the rating for the item within another context [Anand and Mobasher 2007]. It is therefore not surprising that stories of inappropriate recommendations abound, for example: A student looking for scholarship offers but after finishing his studies he stills obtain recommendations about scholarships.

1.1.2 Dynamic Knowledge Capture for Assistance

Capturing knowledge is the basis of building an assistant. This knowledge is not limited to a single user or a single domain. Assistance systems must be able to increase their knowledge over time [Cordier et al. 2010]. This point makes it imperative for capturing knowledge to be dynamic and comprehensive. Nowadays, many recent applications retain traces of their usage by collecting user information. These traces help to understand users' behaviors and their activities thus reflect the context. Using the concept of "trace" offers a solution to this problem of loss of context. The traces capture the interactions between users and systems in their context and thus contain elements of information on user activity, and thus on their experiences.

We believe that traces are good source of knowledge that contains users'

experiences. Therefore, traces can feed an experience-based assistant with the adequate and appropriate knowledge, to insight users finding their way to achieve their tasks efficiently. In Trace-Based Reasoning (TBR), interaction traces are used as a specific knowledge container. TBR is a new paradigm of Artificial Intelligence in which inferences are performed on specific objects called modeled traces [Cordier et al. 2013]. Modeled traces are defined as a temporally situated record of events observed during an interactive process. Our work is part of the work of Silex team[1] in LIRIS laboratory[2]. One of the topics of our team is "Knowledge dynamics and traced experience", where the goal is to propose a dynamic engineering of knowledge, using the records left by individual and collective activity in the digital environment.

In this thesis, our main contribution is to provide assistance to users in the form of recommendations, using past experiences captured in traces. This assistance is able to reproduce solutions by taking into account the context and dynamics of assisted tasks. We are studying the dynamics of the knowledge involved in the TBR cycle by considering activity traces as the knowledge container that represents the context. This contribution has been built, developed and validated working successively on two applications, which are described later in this chapter. To study the nature of such an assistant, Different research questions presented in the next section need to be answered.

1.2 Research Question

Reusing of traces for a given activity requires an analysis phase and transformation of the trace collected by the system to propose an interpretation "that makes sense" for user. Analysis of traces is not a simple process. Imagine for a simple trace *e.g.* "a copy-paste trace" at first glance, it seems very easy. But in fact, there is a lot of information that must be identified. We need to determine the actor (user), the time and the source and the destination of the copying process. It also varies according to the object type from textual to graphical, *etc.*. Consequently, for a complex activity, it is more complicated

[1]SILEX Team: Supporting Interaction and Learning by Experience: https://liris.cnrs.fr/silex

[2]LIRIS Lab: Laboratoire d'InfoRmatique en Image et Systèmes d'information: https://liris.cnrs.fr

and needs a deep manipulation.

This thesis focuses on a particular aspect of the use of traces: to learn from interaction traces what could be relevant to a particular and changing analysis context, in order to develop an assistant that can understand the context, reuse past experience of the users and be able to adapt to their needs. This assistant is responsive and effective to user in a given application situation. Therefore, the general goal is the study of an assistant (knowledge and reasoning methods) based on experience, and to investigate the ability of modeling traces to achieve the assistance. Designing an assistant raises a number of research questions that should be addressed in this thesis:

1. How to build a general framework of trace experience management for assistance. It implies the collection process of relevant and reusable traces in the applications, modeling these traces for their exploitation and developing mechanisms for the storage, management and transformation of interaction traces.

2. What are the conditions necessary for the experience to be "replayed" in a new context. *i.e,* how to enable users to replay their interactions to return to a particular state of an application? This includes managing impact propagation of changes during the replay process.

3. How to define general similarity measures allowing us to compare the episodes of traces. An episode is a temporal pattern composed of an ordered set of events corresponding to a specific task.

4. By using the defined similarity measures, how to find episodes of traces that can be reused to provide recommendation of the sequence of actions related to a task. We need to realize a module able to adapt the recommendations to fit the current context and guide users through their activities.

Each research question has specific contributions with its particular method of validation. These questions are studied and validated using two different applications, which are described in the next section.

1.3 Illustration Applications

In order to test, validate our models and apply our theoretical ideas discussed above in real application domains, we have collaborated with industrial companies (SAP-BO[3] and Webcastor[4]). We have worked with two different web applications: SAP BusinessObjects Explorer[5] and Wanaclip[6]. The issue related to SAP BusinessObjects Explorer is to support the task of data analysis and reporting. For that, a trace replay solution has been implemented. Concerning Wanaclip, the aim is to facilitate the task of generating video clips. Thus, a recommendation engine associated with the task has been built in Wanaclip by reusing traces to help their users achieve their tasks better. Both of these applications have been instrumented to allow collecting interaction traces.

1.3.1 SAP BusinessObjects Explorer

SAP BusinessObjects Explorer is a data discovery application developed by SAP-BO that allows users to retrieve answers to their business questions from corporate data quickly and directly (see Figure 1.1). It offers the users an intuitive path to quickly search and explore data for instant insight into their business. They can get quick, easy answers to on-the-fly questions without training or IT involvement by simply entering a few keywords to search for relevant information and then intuitively exploring large volumes of data, without prior knowledge about what data exists or where to find it.

We have started our collaboration with SAP-BO during the Master thesis [Zarka and Mille 2010], where we worked on providing assistance by reusing episodes of traces. SAP-BO needs a tool to help their users better understand their application (SAP BusinessObjects Explorer) so we designed a trace replay solution for them. We are interested, particularly, in SAP BusinessObjects Explorer because of its clients who operate in different domains especially business users. This application accepts variant data sources, which makes it appropriate for the purpose of our research and makes the need to provide

[3]SAP: http://www.sap.com
[4]Webcastor: http://www.webcastor.fr
[5]SAP BusinessObjects Explorer: http://help.sap.com/boexpl
[6]Wanaclip: http://www.wanaclip.eu

assistance more important.

The initial application has been instrumented in order to collect interaction traces and a graphical interface has been developed to display the traces according to an *ad-hoc* knowledge representation. The application is operational and a demo video is available[7].

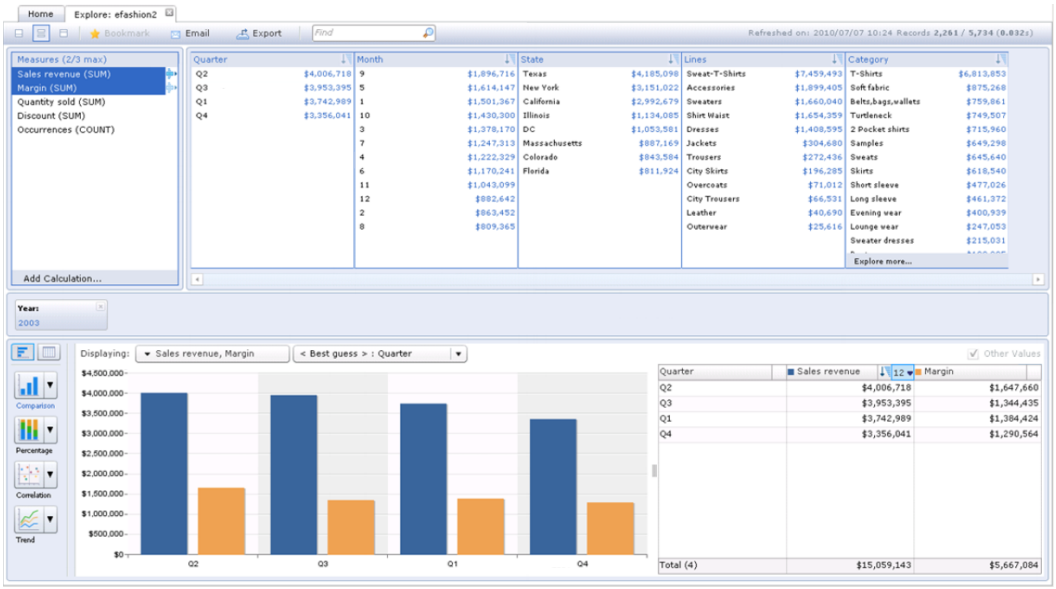

Figure 1.1: SAP BusinessObjects Explorer user interface (before).

1.3.2 Wanaclip

In collaboration with the company Webcastor, we are working on the Wanaclip application that allows users to compose video clips from different audio-visual sources (see Figure 1.2). Wanaclip allows users to import and annotate their own media in the media base. User content is added to the public content of the application. Users can organize media, adjust the duration, and customize content by adding comments, choose their fonts and colors and adjust the display time. In Wanaclip, users enter keywords, the system searches video sequences (rushes) annotated with these keywords and lets the users drag them into a timeline in order to compose a video clip. Several search cycles can refer

[7] A demo video of trace replay in SAP BusinessObjects Explorer: https://liris.cnrs.fr/~rzarka/ReplayTraceDemo/

to the same result clip. Wanaclip offers publishing and sharing functionalities and is plugged to social networks.

To validate our research ideas we wanted to find an industrial partner who needs to provide assistance and recommendations in their applications. We have visited different companies in France and exchanged our ideas. Most of them were interested in collaborating with us. We were looking for an application that targeted the general public so we could have a large number of users from various domains. Our interest is to benefit from users' feedback to validate our results and enhance them. For Webcastor, they wanted to provide recommendations to the users of Wanaclip since it is difficult to find the right video rush in the huge database. We proposed to apply a context-based recommendation based on recorded interaction traces to provide efficient help to users.

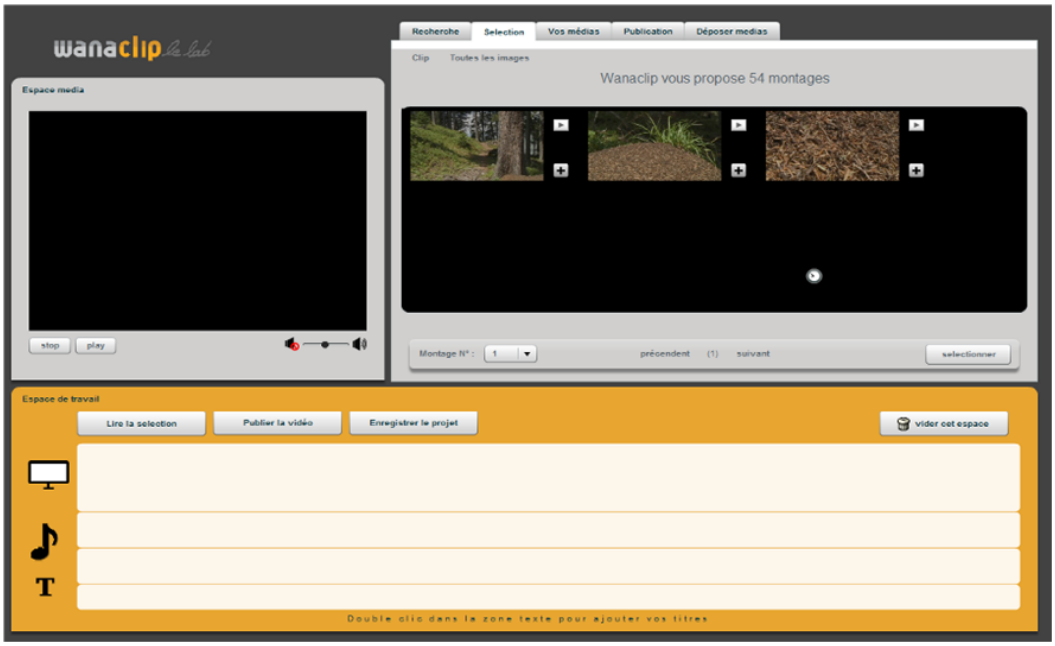

Figure 1.2: Wanaclip user interface (before).

In order to collect interaction traces, the Wanaclip application has been instrumented with a trace collector and an assistance system. Our recommendation system has been integrated to Wanaclip to guide users in both the selection of videos, and the actions to perform in order to make quality clips. The application is operational and available here: `http://www.wanaclip.eu`, Anyone can simply create an account, activate it using his email and log in.

Creating a video is easy: a user start by searching with keywords and the system will display the related clips. After that, he can select the clips that interest him, add subtitles, upload his own media, *etc.* During his usage, recommendations will appear to guide the user.

1.4 Contributions

To answer our research question and the needs of our applications, we made four main contribution parts:

1. **Defining and building a general framework of trace experience management for assistance.** It includes defining a general trace transformation approach based on the Finite-State Transducer (FST). This framework takes into account the performance and the safety of interaction traces. The system achieved ($\mathcal{T}$Store) serves as a demonstrator of these concepts. It is a web service tool that manages the storage, transformation and exploiting of traces. We also compare $\mathcal{T}$Store with other Trace Base Management Systems and show that $\mathcal{T}$Store offers good services and response time. It is available as an open-source software for further research[8].

2. **A mechanism for replaying traces derived from experience.** The necessary elements have been presented to allow restarting the application to a particular state. We make the hypothesis that the interactions change only the possibility of interacting with the system. These interactions do not change the state of the permanent objects of the application. We define models that allows to replay user's interactions and to manage impact propagation of changes during the replay process. In addition, we study the possible approaches to go to a past state and compare it with our replay mechanism. This mechanism has been implemented and demonstrated by applying it to the case of SAP BusinessObjects Explorer application.

3. **Defining similarity measures for comparing episodes belonging to modeled traces.** Our method relies on the definition of a

[8] $\mathcal{T}$Store: https://liris.cnrs.fr/tstore

similarity measure for comparing elements of episodes, combined with the implementation of the Smith-Waterman algorithm for comparison of episodes. The similarity measures have been implemented as a feature of the framework $\mathcal{T}$Store. These measures are used by the recommendation engine in the Wanaclip application.

4. **Study of a contextual recommendation mechanism for the selection of actions' sequences related to a task.** Our recommendation mechanism relies on a specific Trace-Based Reasoning paradigm. It uses the defined similarity measures for comparing episodes belonging to modeled traces. We also introduce a theoretical study comparing the different kinds of recommendation systems. We apply our approach and its models to the Wanaclip application to guide users in both video selection, and the actions to perform in order to make quality clips. The recommendation engine is fed by interaction traces left by previous users of the application and stored in $\mathcal{T}$Store. In addition, an interactive visualization of the recommendations has been provided to adapts the suggested actions to the current context. It allows users to execute the suggested actions and videos by a one click and it obtains a feedback of users' satisfaction. The obtained results for this demonstration are good, which offers an innovative recommendation mechanism.

These four theoretical contributions are introduced in the following chapters. Each contribution has been validated and implemented in a demonstration application. Some of them have been done in SAP BusinessObjects Explorer, the others in Wanaclip (see section 1.3 for more details). These illustrations demonstrate good expected properties of the approaches proposed. Both of performance and quality of result have been evaluated. The evaluation show that our approach offers quite satisfactory recommendations, comparison quality and response time. Next section describes the organization of the chapters of this manuscript.

1.5 Document Organization

In this chapter, we have presented the motivations of our work and we have positioned its context. The illustration applications SAP BusinessObjects

Explorer and Wanaclip have been presented. A brief introduction about our research work and contributions has been given. More research work details are discussed in the next chapters:

- In the next chapter (chapter 2) of this thesis, we introduce a state of the art on the dynamic assistance systems and the general concepts of Trace-Based Systems. We also present a global overview of Case-Based Reasoning and Trace-Based Reasoning.

- Chapter 3 introduces a formal representation of modeled traces. We demonstrate our concept by presenting $\mathcal{T}$Store as a Trace Base Management System that handles the storage, transformation and reuse of Traces. We describe its structure and functionalities and we detail its modules.

- Chapter 4 corresponds to the second contribution and shows how to use traces to enable replay of user interactions and how to manage impact propagation of changes during the replay process. This chapter also describes its application to SAP BusinessObjects Explorer and give its implementation details.

- Chapter 5 reports on a similarity measure to compare episodes in modeled traces. We describe related work in the field of similarity measures for sequential data. We define similarity measures between observed elements and their integration with the Smith-Waterman algorithm in order to compare episodes of modeled traces.

- Our contextual recommendation mechanism is described in chapter 6. We define a Trace-Based Reasoning mechanism that applies a retrieval algorithm using the defined similarity measures to get similar episodes of traces stored in $\mathcal{T}$Store. This mechanism reuses the retrieved episodes to provide recommendations of actions related to the task. A state of the art about recommendation systems is discussed in this chapter. In addition, we demonstrate our evaluation methodology and discuss the obtained results. We study the response time and the quality of the recommendation mechanism and the similarity measures. We also

discuss the results of the survey that we asked the users of Wanaclip to answer in order to measure their satisfaction.

- A summary of the contributions is presented and discussed in Chapter 7. We finally conclude by addressing perspectives about this work.

Writing convention. In this manuscript, we often refer to "the user" by the pronoun he; it is in no way an offense neither a hasty hypothesis on the expert's gender. We have decided not to try with she and keep using he. We only apply a common English rule, as explained, for example, in the book The elements of style, pp. 60-61, [Jr. and White 1979]:

"The use of he as pronoun for nouns embracing both genders is a simple, practical convention rooted in the beginnings of the English language. He has lost all suggestion of maleness in these circumstances. The word was unquestionably biased to begin with (the dominant male), but after hundreds of years it has become seemingly indispensable. [...] No one need fear to use he if common sense supports it. The furor recently raised about he would be more impressive if there were a handy substitute for the word. Unfortunately, there isn't or at least, no one has come up with one yet. If you think she is a handy substitute for he, try it and see what happens."

Chapter 2

Traces and Assistance

Contents

Abstract In this chapter we present a state of the art about dynamic assistance systems. Dynamic assistance systems are systems able to interactively learn how to assist users achieving their tasks. We are interested in systems that acquire knowledge interactively using interaction traces. An interaction trace is a rich record of the actions performed by a user on a system. This chapter contains three main parts: Firstly, we present the notion of traces and Trace-Based Systems. The second part discusses dynamic assistance systems and presents the structure of a Trace-Based Assistant. After that, we present Trace-Based Reasoning, an artificial intelligence paradigm that draws its inspiration from Case-Based Reasoning. In Trace-Based Reasoning, modeled traces act as the main knowledge container.

Keywords: Assistance Systems, Dynamic Assistance, Contextual Assistance, Trace-Based System, Interaction Traces, Modeled Traces, Case-Based Reasoning, Trace-Based Reasoning.

Related publications: [Mathern et al. 2012] and [Zarka et al. 2010]

2.1 Introduction

With the development of new information technologies, applications and tools have become very complex. As a result, providing users with context relevant assistance is critical to assist their use of applications and tools. Classical approaches of assistance often provide pre-calculated assistance. However, context should be taken into account in order to provide users with useful information at the right time and in the right situation. From a usability perspective, it would be much more efficient for the user if context-relevant information were dynamically presented, immediately available and easily visible while the user continues to work. According to [Pesot et al. 2008], if information could be presented in this fashion, users would be able to focus on their tasks without being interrupted.

Capturing knowledge is the basis of building a dynamic assistant. Nowadays, many applications collect interaction traces of users as well as many other logs (statistics, user information, *etc.*). These traces are of different types. For example, log files are the simplest ones. They contain raw data collected for a specific purpose. A typical example is a web server log which maintains a history of page requests. These files are usually not accessible to end users, but only to webmasters or any other administrative person. With log file analysis tools, it is possible to get a good idea of where visitors are coming from, how often they return, and how they navigate through a site. Interaction traces are a more complex form of trace. An interaction trace is a rich record of the actions performed by a user on a system. These interaction traces enable capturing users' experiences in digital environments. A statistical analysis of the traces may be used to understand users' behaviors and their activities. It can also be used to detect useful patterns or even error patterns.

This chapter will begin by introducing a state of the art about the dynamic assistance systems, discussing their concepts, types and dimensions. Next, the notions of trace and activity are presented. We also describe Trace-Based Systems (TBS) which use interaction traces as knowledge sources. These interaction traces represent user-system interactions and evolve with users' activities. Finally, two related reasoning paradigms are presented: Trace-Based Reasoning (TBR) and Case-Based Reasoning (CBR). We show that

the knowledge sources in TBR are traces instead of cases in CBR and we discuss the differences that this entails. We also show examples of Trace-Based Systems for providing user assistance.

2.2 Assistance Systems

"Helping" someone is different from "assisting" him, there is a very important distinction between these two expressions. When we help someone to do something, it means to step in and to do it ourselves. We generally assume that the person in question is unable to resolve his own problem and he is already in trouble. We offer him the solution to this problem ourselves. For example, when we see someone has a car accident, we help that person by insuring the security of the person, we call the police and we may take him to the hospital, *etc.* To assist someone means to guide him but not step over the line and do the job instead of him. For example, a PhD director assists his PhD students to achieve their research plans. He guides the students, giving them some references, assure that they are in the good direction, review their publication, *etc.*

In the context of computing environments, we aim to assist users to better exploit software capabilities to fit their needs. According to [Gapenne 2006], an assistance system is considered as a set of software tools aiming to accompany the assisted in the ownership and use of computer objects. User assistance provides information to assist a person to interact with the software. When we talk about assistance tools, unfortunately, the first thing that comes to our minds is the pop-up wizard that appears usually in the wrong time. However, there are a lot of different ways to provide assistance even without waiting for the users to ask for help. The less we bother the user, except when he needs it, the better. This assistance can take many forms described in the past chapter.

According to [Egyed-Zsigmond 2003], the artifacts performing assistance can be distinguished according to several criterias:

- Advisor systems vs. assistant systems [Selker 1994]. In this distinction, the advisors provide information, offers solutions, but are not directly involved in the task. Conversely, the assistants are dedicated to the

execution of repetitive tasks.

- Conversational systems vs. autonomous systems [Lieberman 1997, Selker 1994]. Conversational systems require the user to express a question or query, which is associated with a logical expression. Autonomous systems operate in the background and can proactively provide suggestions.

- The ability of the system to improve itself [Lieberman and Maulsby 1996, Verillon and Rabardel 1995].

2.2.1 Dimensions of Assistance

[Rech et al. 2007] organize the general data flow in an intelligent assistance system in three layers as shown in Figure 2.1. A data collection process that gathers data from different resources. This data is used by the core assistance algorithms to produce context specific information that is offered to the user.

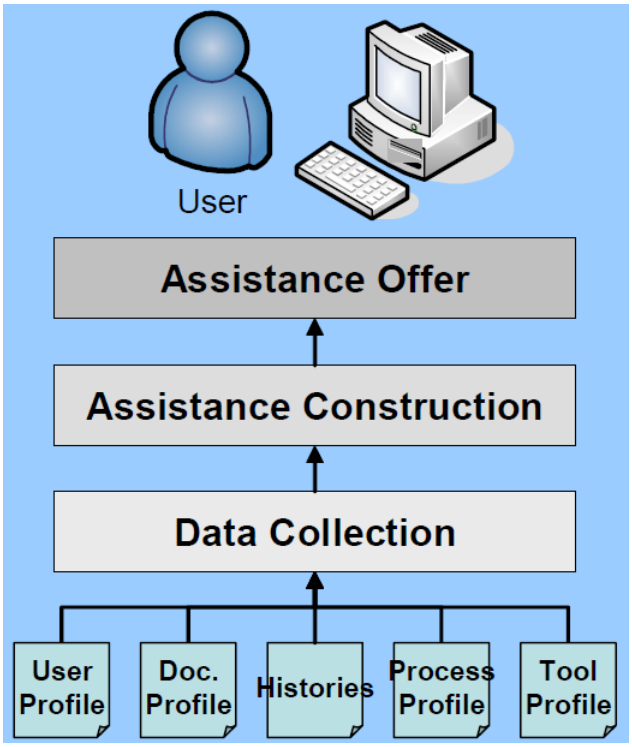

Figure 2.1: The general data flow in an intelligent assistance system (according to [Rech et al. 2007].

Data from several sources has to be processed in the construction phase in order to generate information to assist the user. The characteristics that distinguish the approaches are: where to extract and preprocess data and how to extract and preprocess data. [Rech et al. 2007] support software engineers during development, thus they categorize the assistance algorithms as follows:

- *Assistance for whom?* Depending on the user profile, the results need to be personalized or adapted to the users' expertise level.

- *Assistance about what?* Specifying the kind of object that should be enriched with assisting information.

- *Assistance in which process?* The process or activity the user is currently involved in might induce a special need for assistance.

- *Assistance in which tool environment?* If the information about the process is not available, the tool environment context might be used as well in order to optimize the assistance.

The presentation of the assistance may be differentiated according to several characteristics. It is important that the user is satisfied with the provided assistance, thus the following characteristics need to be identified:

- *When to assist?* Is it for each action of the user, or only for certain actions that change the context?

- *How to assist?* To determine the media used for the assistance; textual, graphical, video, *etc.*

- *Where to assist?* The assistance information can be presented in tooltips, popups, sidebars. Furthermore, within the active tool, a specific third-party tool, or in the operating system itself.

- *Why to assist?* Assistance may be proposed for many reasons: when the user is asking for help, during the identification of a similar sequence of actions, when the task is complex, *etc.*

2.2.2 Context-Sensitive Assistance

Context-sensitive assistance is obtained from a specific point in the state of the software, providing help for the situation that is associated with that state. [Abowd et al. 1999] 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"*.

Contextual assistance allows users to receive help in the current interface they are interacting with, rather than in another help interface. [Yeh et al. 2011] present a creation tool for contextual help that allows users to apply common computer skills-taking screenshots and writing simple scripts. Contextual help has been applied to initial guidance to new features in the context of a map interface [Kang et al. 2003], stencil-based tutorials for young school children [Kelleher and Pausch 2005], programming tasks in the context of an Eclipse editor [Bergman et al. 2005], and practical tasks such as booking flights in the context of a web browser [Bigham et al. 2009]. The importance of user opinions and reviews has highlighted the need to help users to produce better reviews. [Dong et al. 2012] describe the development and evaluation of a reviewer's assistant in form of a browser plugin that is designed to work with major sites like Amazon. This assistant provides users with suggestions that automatically adapt as the user writes their review (to fit the current context).

Contextual assistance is also important in the electronic games, especially when the players start a new level. For example, in the social game "Candy Crush Saga" [Gamewise 2012], each level has a game board filled with differently colored candies, and might contain obstacles. The basic move of this game is horizontally or vertically swapping the positions of two adjacent candies, to create sets of three (or more) candies of the same color. When new requirements happen, an assistance (in the form of a "hand") arise to show the player how to eliminate certain elements. This is a way to help him understand how to succeed in this level. In addition, if the user stays idle for few seconds, the assistant of the game makes some candies blink so as to give an advice on one of the possible next move. Advises are given according to

the context and thus, change often.

In order to provide context-sensitive assistance, it is necessary to have at least information about the tool and the task, skills and preferences of the user. Usually, a context model is defined to describe these contextual information and can be instantiated for different situations. Reasoning techniques are then employed to infer knowledge that can be useful to the assistance system, as it is the case in ontology-based approaches [Wang et al. 2004]. [Cram et al. 2008] propose a rather different approach to provide contextual assistance to the user. Instead of using knowledge-intensive techniques, they use past experiences to build assistance. In their approach, the context is modeled on the fly using a knowledge base formed by interaction traces. Such approaches are discussed more deeply later in this chapter.

2.2.3 Why do we Need Experience-Based Assistance?

Several studies (see [Capobianco and Carbonell 2007]) have shown that assistance systems are often ignored or rarely consulted. According to these studies, novices prefer the training courses to acquire the necessary knowledge to use the software, and casual users rely on their colleagues to find solutions to their difficulties in using the software. There are several reasons for this, including the difference between the perspective of the user and the vision of the designer. In addition, it depends on the background of the user and his experience. Usually, it is difficult to determine easily the needs of all kinds of users while designing assistance systems. This is a major problem not only with the assistance systems, but also for any computer software. Companies withdraw some products for different reasons: promote new products, the users were not satisfied enough, or these products do not achieve the goal they where created for. For example, Google has recently retired its system “Google Buzz” and replaced it by “Google Plus”, a product that learned from Buzz’s slip-ups and became a lot more popular than Buzz especially considering privacy issues [Greene 2011]. If designers of these systems had implemented sophisticated assistance mechanism, maybe it would have been easier for users to learn how to use the system and thus, those systems would have met a big success. As highlighted by [Leplat 1998], the origin of the difference between the visions of designers and users is that it is difficult

to predict the skills of the target users, especially for generic assistance, and therefore, these skills can be assumed unnecessary or underestimated.

As mentioned previously, considering context while providing assistance makes the solution more efficient. The task of modeling the context is usually performed by an expert of the user's 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. [Cram et al. 2008] showed that context can evolve from a situation to another. 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 useful for assisting the user in a particular situation. Therefore, to improve the assistance to users and thus to adapt it to their changing needs, assistance systems must be able to increase their knowledge [Cordier et al. 2010]. Different researchers aim at defining intelligent assistants by using interaction traces as the source of knowledge. When defining an intelligent assistant, Inspiration can be drawn from many principles such as Trace-Based Reasoning, as shown in the next sections.

2.3 Traces and Trace-Based Systems

2.3.1 The Notions of Trace and Activity

According to the Oxford Dictionary, the term "trace" originally means "A path that someone or something takes". The term trace has several significations: "A mark, object, or other indication of the existence or passing of something" [Oxford Dictionary]. "A trace is a surviving mark, sign, or evidence of the former existence, influence, or action of some agent or event; vestige" [The Free Dictionary]. Generally, a trace is defined as "the influence of an event on its environment" [Wikipedia]. In this thesis, our interest is in the idea of "What remains of the past" that define the trace as "A mark left by an action" [Serres 2002, Champin et al. 2013].

The term activity comes from "action" and means the condition in which things are happening or being done. Indeed, an activity is a sequence of actions. An interaction trace can be defined as "a mark left by an activity". An activity relates to a process; to a series of actions. Traces are related to what

remains, what can be observed in this process or these actions. The purpose of an activity is to change the state of something. A trace enables the observation of changing actions within this process. In his thesis, Mathern [2012] defined the activity as a set of processes more or less observable, immersed in a situation and aims to achieve a unified global goal.

Traces are the consequence of a set of contextual elements. Activity traces are therefore strongly connected to the context in which the activity occurs. For example, because it snowed, it is possible to see the footprints of animals in the snow. In this case, the meteorological context is closely related to the trace [Mathern 2012]. In other cases, the contextual elements are necessary for the interpretation of traces. The context should be taken into account while collecting traces in order to interpret them correctly. In this thesis, we build on the idea that traces reflect the context and therefore the traces can be analyzed to provide contextual assistance.

2.3.2 Digital Traces

A digital trace (or digital footprints) is a sequence of temporally observed elements, which is either human interactions mediated and inscribed in the digital environment by itself on the base of the user activity, or a sequence of actions and reactions between a human and a computer [Lund and Mille 2009]. If we use the general definition of the trace and we specialize it for digital traces, the corresponding definition would be: digital trace is made from digital imprints left voluntarily (or not) by the digital environment in the digital environment itself during the digital process [Champin et al. 2012b]. Digital traces provide data on what has been performed in the digital environment (*e.g.* what you clicked on, searched for, liked, where you went, your location, your IP address, what you said, what was said about you, *etc.*).

2.3.3 Modeled Traces

Most of the applications produce their own traces. Some of these traces have an explicit model, some do not, some have an *ad-hoc* model. One of the objectives of SILEX team[1], is to define a meta model for guiding the modeling

[1]SILEX Team: Supporting Interaction and Learning by Experience: https://liris.cnrs.fr/silex

of $\mathcal{M}$-Traces. The aim is to be able to exploit traces coming from different heterogeneous sources and integrate them so that they can be reused later by different applications. A generic representation of traces is required to provide a common view of the heterogeneous sources. Modeled traces were proposed in [Champin et al. 2004] and a detailed formalization has been explained, later, in [Settouti 2011, Champin et al. 2013]. **$\mathcal{M}$-Trace** (short for Modeled Trace) differ from logs in the sense that they come with a model and they contain **obsels**. An obsel (short for "OBServed ELement") represents an interaction and its properties (*e.g.* push the play button on a video player). A trace model defines the structure and the obsel-types that are contained in a trace, as well as the relationships between these obsels. A formal description of modeled traces is presented in the next chapter.

2.3.4 Trace-Based Systems (TBS) and Trace Base Management Systems (TBMS)

A Trace-Based System (TBS) is a system using $\mathcal{M}$-Traces as knowledge sources representing user-system interactions. A Trace Base Management System (TBMS) is a system for managing, transforming and reusing $\mathcal{M}$-Traces. The difference between TBMS and TBS is similar to the difference between a Database Management System (DBMS) and a software using data stored in the DBMS. Such as a DBMS does with data, the TBMS stores, manages, and provides querying services on traces. As shown in Figure 2.2, the TBMS provides $\mathcal{M}$-Trace storage and allows $\mathcal{M}$-Traces transformations and retrieval while the TBS is the application layer using $\mathcal{M}$-Traces. TBS can visualize $\mathcal{M}$-Traces and provide tools that reuse them or share them with other systems or users. In the illustration applications introduced in the previous chapter, both Wanaclip and SAP BusinessObjects Explorer are TBS. Wanaclip aims at providing recommendations based on $\mathcal{M}$-Traces. SAP BusinessObjects Explorer provides a mechanism of $\mathcal{M}$-Traces replay to return to a previous state of the application. In both applications, $\mathcal{M}$-Traces are collected and stored in $\mathcal{T}$Store, a web-based TBMS developed during this thesis (see Chapter 3).

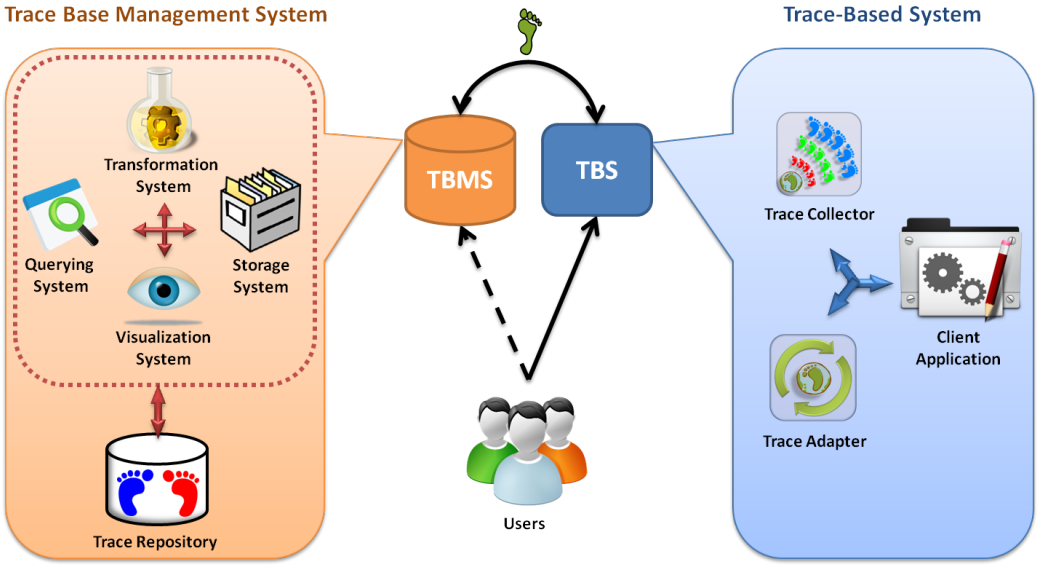

Figure 2.2: Trace-Based System and Trace Base Management System Architectures.

2.3.5 Trace Base Management System Architecture

The left part of Figure 2.2 shows the basic architecture of a TBMS. The main modules of this system are:

- **Trace Repository:** This is where the traces are stored. It contains all the traces and their models. It is usually a database or storage files such as XML files.
- **Storage System:** It is responsible of storing $\mathcal{M}$-Traces and creating the trace models.
- **Transformation System:** It can perform operations on traces like applying filters, rewriting and aggregating elements, computing elements attributes, *etc.* So as to produce so called transformed modeled $\mathcal{M}$-Traces that can be more easily reusable and exploitable in a given context than primary $\mathcal{M}$-Traces. Transformations can also be considered as semantic abstractions when associated with models (ontologies) like task model.
- **Visualization System:** It allows to visualize $\mathcal{M}$-Traces and navigate between the different transformations. It enables to extract any trace to

get its metadata, obsels and model. This system can be integrated in a more specific visualization system in the TBS.

- **Querying System:** It enables the extraction of episodes and patterns from the $\mathcal{M}$-Traces.

In Chapter 3, we propose to add new modules to the TBMS architecture. A Similarity Measure to compare between the obsels and the episodes of $\mathcal{M}$-Traces. The second module is the Security System to ensure the protection of $\mathcal{M}$-Traces.

2.3.6 Trace-Based System Architecture

Figure 2.2 shows the basic architecture of a TBS (right part) and how it is connected to the TBMS. This figure presents the basic components that any TBS system must has.

- **Client Application:** It can be any application of any kind (web, mobile, desktop, *etc.*) for different purposes (accounting, graphics, games, media, office, *etc.*). It is usually instrumented to be able to manipulate interaction traces.

- **Trace Adapter:** It is the connection layer between the Trace Base Management System and the client application. It is responsible of reusing and managing $\mathcal{M}$-Traces. This ability provides an important flexibility.

- **Trace Collector:** It is responsible of capturing users' interactions and sending them to the TBMS to be stored in the trace repository. It collects the observed data from different input sources (log files, streamed actions, video records, interface events, *etc.*). The collection process includes the creation of trace models and generating obsels related to this model.

2.4 Towards a Trace-Based Assistant

In this section, we show how Trace-Based Reasoning (TBR) can be used to provide dynamic user assistance. TBR draws its inspiration from Case-Based

Reasoning (CBR), an artificial intelligence paradigm which is briefly described in the next subsection. This assistant called a Trace-Based Assistant (TBA).

2.4.1 Case-Based Reasoning

Case-Based Reasoning (CBR) is the process of solving new problems based on the solutions of similar past problems. A new problem is solved by finding a similar past case, and reusing it in the new problem situation. A car mechanic who fixes an engine by recalling another car that exhibited similar symptoms is using CBR. Knowledge in a CBR system is composed of past examples, or cases, stored in a database or by some other convenient means, known as a case base. Of course CBR does not always give the ideal solution to the problem but, if it has experienced this problem, it always offers a solution [Cordier 2008]. The basic CBR principle, "to solve a target problem, retrieve a source case and adapt the solution to fit the target problem requirements.", can be summarized as in Figure 2.3

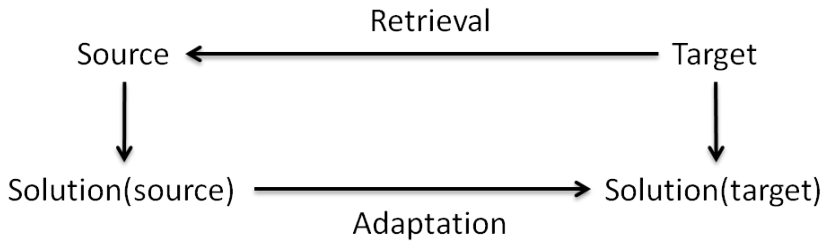

Figure 2.3: The classical CBR principle according to [Cordier 2008].

The roots of CBR can be traced back to Schank's dynamic memory model [Schank 1983]. He developed a theory of learning and reminding based on retaining of experience in a dynamic, evolving memory. CBR first appeared in commercial tools in the early 1990's and since then has been used to create numerous applications in a wide range of domains (diagnosis, help desk, assessment, decision support, design). The idea of taking context into account in the reasoning process is also explored. For example, [Zimmermann 2003] shows that combining Case-Based Reasoning methodology and context awareness is a new and powerful way of modeling and reasoning from contexts. The classical CBR cycle proposed in [Aamodt and Plaza 1994] has been described in 4 steps known as the four "Re": Retrieve, Reuse, Revise and Retain (see

Figure 2.4). Each step of the cycle exploits the knowledge base central to the system, which contains previously solved cases as well as general knowledge. This cycle now serves as reference for most studies in this field.

- **Retrieval** is the process of finding the cases in the case base that most closely match the information known about the current case. The current information is represented as a new case with some of the missing information. This step of matching cases is one of the most crucial steps in the CBR methodology.

- **Reuse** is the step where matched cases are used to compute a suggested solution.

- **Revision** is the testing of the suggested solution to make sure it is suitable and accurate. The result is a solution that has been confirmed by testing (either by human or machine).

- **Retention** is the storage of new cases for future reuse. A strong advantage of CBR over other reasoning technologies is the fact that new knowledge is continuously and easily added to the store of experience. This step is usually as simple as storing the solved case into the case base. Advanced CBR systems even use this step to enrich the other knowledge bases of the system (domain knowledge, adaptation knowledge, similarity knowledge, *etc.*).

Case-Based Reasoning has proven to be valuable and successful in many applications. In this thesis, we are interested in temporal CBR systems. In the following, related work in this area is discussed. Some CBR systems try to solve problems by using log files. [Martín and Plaza 2004] propose a Ceaseless CBR model that is an evolution of the CBR diagram. It considers the CBR task as on-going rather than one-shot, to support the analysis of unsegmented sequences of observational data stemming from multiple coincidental sources. [Jaczynski and Trousse 1999] present a Case-Based System for cooperative information browsing, called "Broadway". This system follows a group of users during their navigation on the WWW (proxy-based architecture) and advises them by displaying a list of potentially relevant documents to visit next. The advises are based mainly on similarity of ordered sequence of past accessed

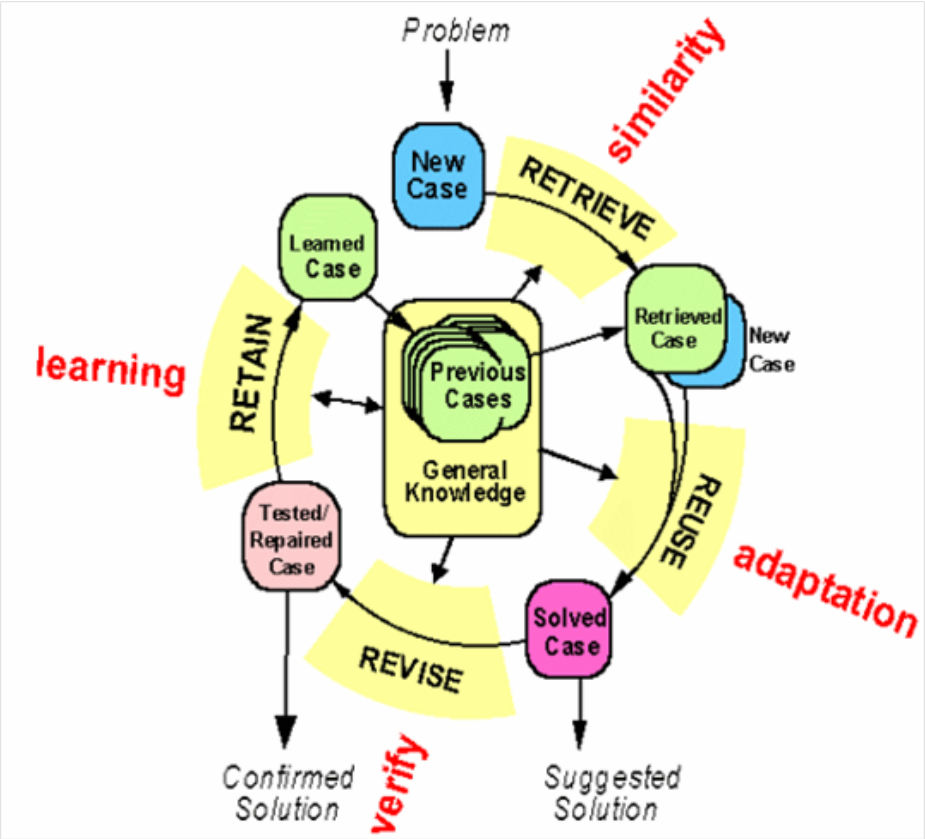

Figure 2.4: The classical CBR cycle from [Aamodt and Plaza 1994].

documents. In order to match the dynamic nature of web sites, [Krzysztof Dembczynski 2008] proposed a new scheme for coding web server log data into sessions of behavioral sequences. A behavioral sequence is defined as an ordered list of elementary behaviors. Each elementary behavior describes a visited Web page associated with a set of hypothetical actions that may explain how the user acts to reach the next visited page. In [Corvaisier et al. 1997], the authors recommend a path to find a document in the web. In our work, we make use of traces on which transformations can be applied to be able to have more abstract traces. This gives us much more flexibility than with classical raw logs.

2.4.2 From CBR to TBR

The term Trace-Based Reasoning was introduced by [Mille 2006]. TBR was then described as an extension of the traditional Case-Based Reasoning (CBR)

cycle proposed by [Aamodt and Plaza 1994]. The main motivation of such an extension was to overcome the limitations of predefined structures used to represent cases in CBR [Cordier et al. 2013]. TBR draws its inspiration from Case-Based Reasoning and reuses past experiences to solve new problems. The main difference is that, in TBR, modeled traces are the main source of knowledge that can be advantageously used in a dynamic reasoning process.

The most important difference between CBR and TBR is the temporal aspect. In CBR, cases are usually static descriptions of problems (attribute/-value list, hierarchical object, text, *etc.*). In TBR, cases are to be found in traces; which are temporal sequences. Thus, cases are not presented *apriori* in traces. We need to define a mechanism to recognize and extract cases from traces. In TBR, we use the concept of episode, each episode represent a case. An episode is a temporal pattern composed of an ordered set of events corresponding to a specific task. Each episode must satisfy an episode signature that describes some temporal constraints and conditions that describe a specific task. In addition, the adaptation mechanism between episodes is different. In [Cordier et al. 2009], the authors proposed to use traces of interactions as a new way of performing CBR to enhance the traditional CBR by providing more flexible ways of reasoning from experience. This mechanism is about building systems that can ease sharing and re-using of experiences between large communities of users. They showed that traces can be used to record experiences "in context" and thus, bring more flexibility to traditional CBR. However, when building TBR for user assistance, a lot of research issues are raised, such as "how to define a similarity measure between episodes?", which is one of our main contributions that will be presented in Chapter 5.

2.4.3 The TBR Process

[Cordier et al. 2006] propose an immediate evolution of the CBR cycle as presented in Figure 2.5. Two differences appear: the first one concerns the addition of the elaborate step to the traditional cycle. This step allows the system to evolve from a hill-defined problem to a problem which is "intelligible" by the system. The second evolution concerns the explicit representation of the expert in the center of the cycle. This cycle illustrates the requirement for interaction between the system and the "outside world" to permit the

acquisition of knowledge during the problem-solving process. Interactions may take different forms and the "interlocutors" may vary: user, expert, other system, *etc.* This work tackled some major issues of CBR that are better handled by TBR, as shown in the next paragraph.

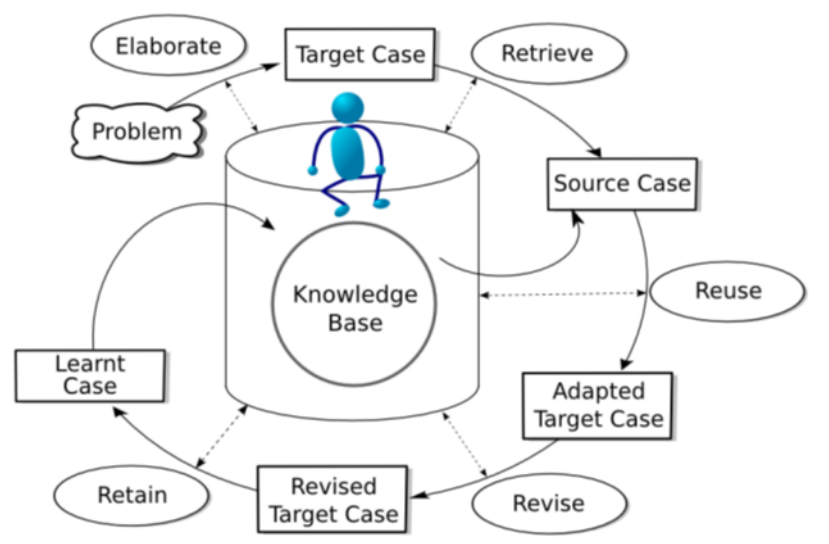

Figure 2.5: A CBR cycle centered on interactions with elaboration step as presented in [Cordier et al. 2006].

A detailed description about TBR is presented in [Cordier et al. 2013]. According to the authors, the TBR process usually follows three steps that they name: elaboration, retrieve and reuse. During all the reasoning processes, TBR exploits traces and transformations stored in the TBMS, but also additional knowledge represented in the TBR knowledge base (similarity measures, adaptation strategies, *etc.*). The three reasoning steps of TBR are strongly interrelated. Moreover, users are in the center of the reasoning loop. They can interact with the reasoning process at any time and get immediate feedback through traces. They can go back and forth in the reasoning process if adjustments are needed. In the following, the main definitions given by [Cordier et al. 2013] are recalled.

2.4.3.1 Elaboration Step

In this step, episode signatures are produced. An episode signature is a specification of the constraints that an episode must satisfy (pattern, duration,

etc.). There are many ways of expressing an episode signature (rules, sets of constraints, $\mathcal{M}$-Trace and constraints, automata, finite states machines, *etc.*).

Building episode signatures can be manual, semi-automatic or automatic depending on the application. It is manual in case that an expert in the application has to define the task signatures. Episode signatures are defined using trace transformations by defining the first and last obsel and the type of obsels between them. When the system recognizes fragments of traces as episodes and asks the user if it is significant, it is semi-automatic. However, the best method is the automatic one but it is not possible all the times because of ambiguity problems. This happens, when an episode is recognized by two different signatures. In this method, the system generates task signatures automatically when similar episodes are repeated many times. If these episodes do not have any signature, thus, the system considers that they are related to the same task. Task signatures can be reviewed by the users.

2.4.3.2 Retrieve Step

This step consists in finding a set of episodes that can be used to solve the current problem. In this step, a search algorithm query the TBMS to retrieve episodes of $\mathcal{M}$-Traces that match the signatures generated in the elaboration step. However, to compare the episodes, similarity measures between the elements of $\mathcal{M}$-Traces must be defined, as shown in Chapter 5. The search algorithm is based on the similarity measure to return all the episodes for which their similarity scores are similar to the current episode.

2.4.3.3 Reuse Step

It uses the retrieved episodes to provide a solution for the current situation. During this step, the user uses query operations on the retrieved traces to exploit their content. Ranking functions are defined in this step to select the most related episodes to the current context. There are many ways to reuse the past episodes depending on the task addressed in the application. The system can simply display the episode to the user. It can also replay the episode as form of macro, or even possibly adapt the episode to replay it differently, as shown in Chapter 4. Reusing episodes can be made through recommendations of actions to perform to achieve the task, as shown in Chapter 6.

2.4.4 Using Interaction Traces for Providing User Assistance

As shown in [Cordier et al. 2013], in the CBR field, several approaches make use of sequential information during the reasoning process. [Leake 2010] used provenance information to improve reasoning and explanation in CBR. [Weber and Ontañón 2010] used annotated traces recorded when a human user played video games in order to feed a case-based planner. In [Minor et al. 2012a], authors reported on their work on adaptation of workflow which can be considered as specific sequences. [Gundersen 2012] studied the particular properties of sequences in order to design more efficient similarity measures. All these approaches implements forms of reasoning on traces.

One of the advantages of developping a Trace-Based System is to facilitate activity analysis and modeling. See [Fayyad et al. 1996, Van Der Aalst 2009] for fundamentals on knowledge mining and discovery from activity traces. Following these principles, several studies have implemented TBR in real-world application. Georgeon et al. [2012] implemented a Trace-Based System to model the car-driving activity from traces collected with an instrumented vehicle. [Mathern et al. 2010] have applied TBR to Stream Mining. Stream Mining is the process of extracting knowledge from continuous records. They have shown that continuous records can be transformed into $\mathcal{M}$-Traces. The outcome of a TBR process applied on $\mathcal{M}$-Traces may be used to improve the results of the mining process.

TBR proved to be efficient for user assistance [Cordier et al. 2009, Ginon et al. 2013c]. [Cram et al. 2008] proposed a three-steps process that reuses traces for a contextual assistance: collecting traces from sensors, abstracting traces, and assisting. Their approach reuse user's personal experience as an alternative to traditional systems having an explicit context model associated to reasoning capabilities on this model. Ginon et al. [2013c] proposed a generic assistance model for the specification and the implementation of customized assistance systems concerning an existing target application, without having to change it. In the context of the Kolflow project [Champin et al. 2012b], the authors built an assistance tool that could help users better manage the merging of resources in a distributed context. More precisely, they focus on the activity of collaborative construction of knowledge, supported by a network

of distributed semantic wikis.

TBR has been specifically studied in the case of human learning systems [Ginon et al. 2013b, Settouti et al. 2009]. Intelligent tutoring systems and collaborative learning tools are, by nature, designed to provide assistance to learners. These tools often use different forms of traces as an input for the assistance [Champin et al. 2012a]. In these tools, records of past experiences, as well as explanations, play an important role. The Visu application provides a good example of the use of traces in a collaborative learning space [Bétrancourt et al. 2011]. The objective of [Sehaba 2012] is to develop an adaptive help system based on interaction traces, to enable tutors and learners to help each other through the sharing of traces.

2.5 Conclusion

In this chapter, we have presented a state of the art about assistance systems and we have shown how interaction traces can be used as the knowledge source in these systems. Firstly, we have shown the value of interaction traces and the structure of Trace-Based Systems. We have highlighted the importance of experience-based assistants, the dimensions of assistance and the need to provide contextual assistance. Finally, Trace-Based Reasoning and its connection with Case-Based Reasoning have been described. We have shown the benefits of using traces as a knowledge container in a reasoning process and some systems who use interaction traces to provide assistance have been presented. In the next chapter, we will present our system, $\mathcal{T}$Store, a web service tool for the storage, transformation and exploiting of traces. We also introduce a formal representation of modeled traces.

Chapter 3

$\mathcal{T}$Store: A Web Trace Base Management System

Contents

Abstract In this chapter, we present a formalization of a generic trace metamodel called $\mathcal{M}$-Trace. Each trace is described by a trace model that represents the structure of the trace elements and their relations. This chapter also presents $\mathcal{T}$Store, a Trace Base Management System (TBMS) handling the storage, transformation and exploitation of $\mathcal{M}$-Traces. $\mathcal{T}$Store brings a solution to performances and storage issues usually encountered by TBMS. To exploit traces, transformations are used. $\mathcal{T}$Store provides predefined transformation functions as well as a customized transformation based on Finite-State Transducers (FST), which are also presented in this chapter.

Keywords: Trace Base Management System, Finite-State Transducer, Trace Transformation, Human Computer Interaction, Trace Model.

Related publications: [Zarka et al. 2013a]

3.1 Introduction

Nowadays, many applications collect traces of their users' interactions. These traces can be used in many ways: analysis of users' behaviors, visualization of interactions, debugging, mining, *etc.* Storing these traces, and developing efficient mechanisms to exploit them, represent a considerable challenge. In this chapter, we report on $\mathcal{T}$Store, a Trace Base Management System that addresses this challenge.

In our work, we want to use interaction traces to provide assistance and recommendations. We have identified some limitations in the existing meta-models of traces according to our needs. Therefore, we propose a theoretical contributions to overcome some of these limitations, *e.g.* security, storage, generic transformations and similarity measures. Existing TBMS presented performance and security issues, notably due to the fact that they were not efficient enough to process large flows of events sent by web applications such as Wanaclip. For example, kTBS [Champin et al. 2013] is one of the best existing TBMS that has different functionalities for storing and managing interaction traces. However, we tried to use it, but it was not possible at the time. We wanted to be able to collect and send multiple obsels in the same time to avoid network problems. Moreover, kTBS cannot compare traces and retrieve similar episodes. To do so, we had to ask kTBS to retrieve all the traces in a specific period of time. Then, it is the client application which is responsible for comparing and manipulating the retrieved traces, which wastes time. After identifying the main problems, we decided to develop $\mathcal{T}$Store.

In this chapter, we present the architecture of $\mathcal{T}$Store. $\mathcal{T}$Store is implemented as a web tool allowing agents (software systems or human users) to concurrently access, store and reuse traces issued from various applications. In $\mathcal{T}$Store, we propose to add new modules to the Trace Base Management System architecture presented in Chapter 2. $\mathcal{T}$Store is composed of six modules: Storage Manager, Querying System, Transformer Module, Visualization System, Similarity Measure and Security Manager. It manages $\mathcal{M}$-Traces in a relational database and benefits from its storage and querying facilities. An experimentation was conducted to demonstrate $\mathcal{T}$Store efficiency (see section 3.5).

The main contributions described in this chapter are the security mod-

ule and the generic transformations. $\mathcal{T}$Store uses a role-based access control (RBAC) approach [see Ferraiolo et al. 2003] to ensure the security of interaction traces. $\mathcal{T}$Store uses the Finite-State Transducer (FST) principle [see Roche and Schabes 1997] to implement specific trace transformations. $\mathcal{M}$-Trace transformation is a process allowing the transformation of existing $\mathcal{M}$-Traces in order to better exploit them. Transformations are variant (filtering, segmentation, reformulation, *etc.*). Here, we show how to apply FST transformations to the recommendation problems in Wanaclip.

The remaining of this chapter is organized as follows. Next section introduces a formal representation of modeled traces. Section 3.3 presents the structure, modules and functionalities of $\mathcal{T}$Store. We report the implementation, experiments and performance study in Sections 3.4 and 3.5. We discuss the related work in Section 3.6, and conclude our work in Section 3.7.

3.2 $\mathcal{M}$-Traces Formal Representation

An **$\mathcal{M}$-Trace** includes both the sequence of temporally situated obsels, which is the instantiated trace and the model of the trace, which gives the semantics of obsels and the relations between them. $\mathcal{M}$-Traces differ from logs in the sense that they come with a model. Each obsel has, at least, a type and two time-stamps (begin and end). An obsel is performed on an object (button, list, image, video, *etc.*). For example, an obsel of type "*playvideo*" is performed on an object, a video, and has two time-stamps: the start and end time of watching the video. Obsels can also have an arbitrary number of attributes and relations with other obsels. Each obsel-type has a domain of attributes and indicates the values of its attributes in the range of the attribute type. Each $\mathcal{M}$-Trace has a trace model that describes the structure of the trace and its obsels. Each trace model contains different types of obsels that may have relations between them. The obsels are generated from the observation of the interaction between the user and the system.

We call a ***primary $\mathcal{M}$-Trace*** the trace issued directly from the interactions between the user and the system (including all mouse and keyboard actions). It contains the collected obsels associated with their models. Specific operations on $\mathcal{M}$-Traces allow producing ***transformed $\mathcal{M}$-Traces*** for

several purposes (filtering, segmentation, merging, *etc.*). A primary $\mathcal{M}$-Trace with all its transformed $\mathcal{M}$-Traces constitute a ***tracebase***. A tracebase is a collection of related $\mathcal{M}$-Traces structured for a specific purpose. The notations and definitions of an $\mathcal{M}$-Trace and its model are defined in detail in [Settouti 2011]. In the following, we provide a simplified formalization fitting our needs. Furthermore, the initial definition is extended by adding properties related to security of $\mathcal{M}$-Traces. We consider that each tracebase is owned by one user and belongs to him (the client who executed the actions). Different roles can be granted to the users to give them the appropriate privileges on $\mathcal{M}$-Traces.

3.2.1 Trace Model Definition ($\mathcal{M}_\mathcal{T}$)

Definition 1 (Trace Model)**.** *A Trace model is defined as a tuple* $\mathcal{M}_\mathcal{T}$= (T, C, R, A, P)

- T*: temporal domain representing the period during which the* $\mathcal{M}$*-Traces of this model occurred. Usually, it is simplified and represented by two time-stamps: start and end.*
- C*: finite set of obsel-types.*
- R*: finite set of relation types between the obsel-types in* (C)*, including (but not limited to) inheritance relation.*
- A*: a set of hierarchical attributes for each obsel-type in* (C)*.* $A(c)$ *is the set of attributes for an obsel-type* (c)*.*
- P*: it is used to determine the transformation relations. It points to the set of parent trace models that are transformed to generate this trace model. For the primary trace it is null.*

Intuitively, a trace model defines a vocabulary for describing traces: how time is represented (T), how obsels are categorized (C), what relations may exist between obsels (R), what attributes further describe each obsel (A), and the transformations using (p).

Example 1. In Wanaclip, we define a trace model ($\mathcal{M}_{\mathcal{T}1}$) that contains all the obsel-types such as "seach media", "add video", "play video", "add subtitle", "navigate", "add user", *etc.* Each obsel-type has a set of hierarchical

attributes. For example, the obsel-type "add user" has different attributes such as id, name, age, email, address, *etc.*. However, "address" attribute has sub attributes like building, street, city and country. This trace model is the primary model, thus $P(\mathcal{M}_{\mathcal{T}1})$=null. We also define a filtering transformation model ($\mathcal{M}_{\mathcal{T}2}$). This model has the same obsel-types of ($\mathcal{M}_{\mathcal{T}1}$) except the filtered ones such as "navigate" and "play video". As this model is transformed from ($\mathcal{M}_{\mathcal{T}1}$) so $P(\mathcal{M}_{\mathcal{T}2}) = [\mathcal{M}_{\mathcal{T}1}]$. ◊

3.2.2 $\mathcal{M}$-Trace Definition

Definition 2 ($\mathcal{M}$-Trace)**.** *An $\mathcal{M}$-Trace is a tuple*
$\tau = (\mathcal{M}_{\mathcal{T}}, O, \leq_O, u, s_t, e_t, v, \lambda_R, \lambda_C)$

- $\mathcal{M}_{\mathcal{T}}$*: trace model* $\mathcal{M}_{\mathcal{T}} = (T, C, R, A, p)$.
- O*: sequence of obsels of this trace, where* $|O|$ *is the number of obsels in the trace.*
- $\leq_O$*: partial order defined on the obsels, it represents their temporal order.*
- u*: the "id" of the user executing the actions associated with this trace.*
- s_t *and* e_t*: starting and ending time-stamps of the trace (simplified from the original definition).*
- v*: visibility of the trace. It can be: public, private or custom. It is used for defining security properties of a trace.*
- λ_R*: function describing the relation between two obsels* $\lambda_R : O \times O \rightarrow R$*, where* R *is the set of relation types defined in* $\mathcal{M}_{\mathcal{T}}$.
- λ_C*: total injective function that associates each obsel with its type* $\lambda_C : O \rightarrow C$.

Intuitively, an $\mathcal{M}$-Trace represents the actions of a user u collected according to a trace model $\mathcal{M}_{\mathcal{T}}$ over a period of observation time defined by s_t and e_t. An $\mathcal{M}$-Trace can be visible or hidden for other users (v). An

$\mathcal{M}$-Trace consists of an obsel sequence (O), has types (C). Obsels in O are represented in a temporal order $\leq_O$. The obsels can have different relations between each other defined by λ_R. Each obsel (o) has exactly one type (c) such as $\lambda_C(o) = c$, λ_C is a total function. Note that R includes inheritance relations between obsel-types. There may be none, one, or several relation(s) between two given obsels defined by (λ_R).

Example 2. Following to the previous example, we can have an $\mathcal{M}$-Trace (τ) associated with its model ($\mathcal{M}_{\mathcal{T}1}$) as $\tau_1 = (S_8 P_1 A_1 A_2 X_1 A_3 N_5 G_7 S_9 P_3 A_3 A_5)$. It represents a user searching for videos about "Lyon" (S_8: S is the obsel-type, and 8 is the value). Then, he starts playing (P_1) and adding videos (A_1 and A_2) to his selection. After that, he adds the text X_1 to the first selected video and adds another video A_3. He navigates to another tab (N_5) to generate the clip G_7. Finally, he searches for a different keyword "Paris". He plays and adds other videos. This $\mathcal{M}$-Trace has been executed by a specific user (u_1). All the obsel-types of (τ_1) must be defined in ($\mathcal{M}_{\mathcal{T}1}$). The obsels are temporally ordered and each obsel has two time-stamps (begin and end) corresponding to its execution time. A filtering transformation can be applied to (τ_1) to produce a transformed $\mathcal{M}$-Trace with the trace model ($\mathcal{M}_{\mathcal{T}2}$) as: ($\tau_2 = (S_8 A_1 A_2 X_1 A_3 G_7 S_9 P_3 A_3 A_5)$). Any $\mathcal{M}$-Trace associated with the same trace model is automatically filtered in the same way. ◊

3.2.3 Obsel Definition

Definition 3 (Obsel)**.** *We define an obsel as* $o = \{c, A_o, s_t, e_t\}$ *where:*

- *c: type of the obsel. Each obsel-type has a predefined set of attributes.*
- *A_o: set of attributes of the obsel o and their values.* $A_o = \{(a_i, v_i)\}_{i=1,|A_c|}$. *Note that a_i is an obsel attribute type, v_i is an obsel attribute value, and A_c is the set of attributes of an obsel-type c.*
- *s_t and e_t: starting and ending time-stamps of the obsel.*

An obsel represents the information about the action that the user u has performed on the system. Each obsel belongs to one $\mathcal{M}$-Trace and has only one direct obsel-type c that may have other parents defined in the trace model

$\mathcal{M}_\mathcal{T}$. The main information about the obsel is represented by its attributes A_o and indicates the values of its attributes in the range of the attribute type c. Note that obsel attributes are hierarchical so only the leafs can have values. These values are not mandatory. The duration of an obsel is defined by its time-stamps (s_t, e_t). Duration represents the execution time of the action that the obsel describes. When the two time-stamps are identical, the duration equals 0.

Example 3. The obsel (S_8) presented in the previous example has an obsel-type "Search Media". This obsel has 3 attributes ($tags$ = "*Lyon*", $kind$ = "*right*" and $duration$ = "30"). It has two time-stamps: start="2012-09-21 17:31:26" and end="2012-09-21 17:31:26". ◊

3.2.4 $\mathcal{M}$-Trace Transformation

Specific operations on $\mathcal{M}$-Traces allow producing transformed $\mathcal{M}$-Traces for several purposes (filtering, segmentation, reformulation, *etc.*). A transformation defines means to produce a new $\mathcal{M}$-Trace from one or several others, or to transform the original $\mathcal{M}$-Trace into a new one. Note that the original traces are not actually modified. Transformed $\mathcal{M}$-Traces can be easier to reuse in a given context than the primary $\mathcal{M}$-Traces.

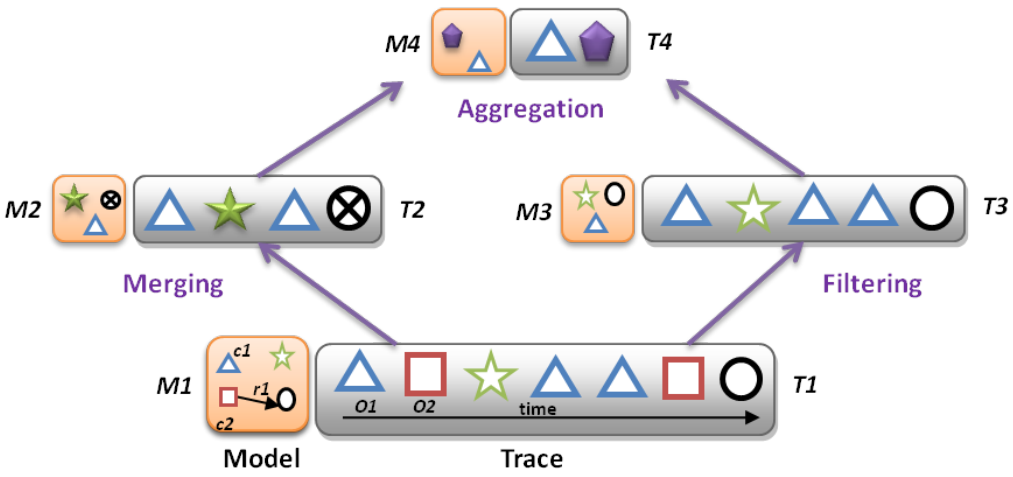

Figure 3.1: An example of $\mathcal{M}$-Trace transformations.

Example 4. Figure 3.1 shows an $\mathcal{M}$-Trace T_1 which is transformed into T_2 and T_3 (level 2). T_2 is transformed from T_1 by merging some of its obsels together to produce new obsels. T_3 is transformed from T_1 by filtering some of its obsels. This type of transformation does not generate new obsel-types in the transformed $\mathcal{M}$-Trace. This figure also shows that T_2 and T_3 are transformed together into T_4 (level 3). Each trace model contains different types of obsels that may have relations between them. For example, in the $\mathcal{M}$-Trace T_1, its trace model M_1 contains four obsel-types (c_1, c_2, c_3, c_4) and one relation type r_1. T_1 contains seven obsels $(o_1, o_2 \ldots)$. Connected $\mathcal{M}$-Traces represent a tracebase $([M_1, T_1], [M_2, T_2], [M_3, T_3], [M_4, T_4])$. ◊

3.2.5 What is an Episode?

An episode $(\mathcal{E})$ is a temporal pattern composed of an ordered set of events corresponding to a specific task defined by a task signature. The concept of task signature has been introduced in [Champin et al. 2004] as a set of event declarations, entity declarations, relations, and temporal constraints. They assume that episodes related to a particular task usually share some common features: involving the same kind of entities or events, co-occurring or occurring in a given order, *etc.* Once these common features have been identified for a particular task, they can be considered as a signature of this task. This supposes that an episode can be annotated, by information coming from the fact that it has been recognized as an occurrence of a particular task.

Definition 4 (Episode)**.** *Given an $\mathcal{M}$-Trace (τ), O is the sequence of its obsels. We define an episode $\mathcal{E}_k = O_{n_k}$ as a sub-set of O where $(n_1 < n_2 < \ldots)$ is a temporally coherent sequence of obsels of O having the same temporal order in (τ). An episode $\mathcal{E}$ can be derived from O by filtering some obsels without changing the order of the remaining obsels.*

Example 5. After having filtered the $\mathcal{M}$-Trace (τ_1) presented in Example 2, we obtained the transformed $\mathcal{M}$-Trace $\tau_2 = (S_8 A_1 A_2 X_1 A_3 G_7 S_9 P_3 A_3 A_5)$. Another transformation can be applied to (τ_2) to segment it into two transformed $\mathcal{M}$-Traces $\tau_3 = (S_8 P_1 A_1 A_2 X_1 A_3 N_5 G_7)$ and $\tau_4 = (S_9 P_3 A_3 A_5)$. Both of τ_3 and τ_4 represents an episode. ◊

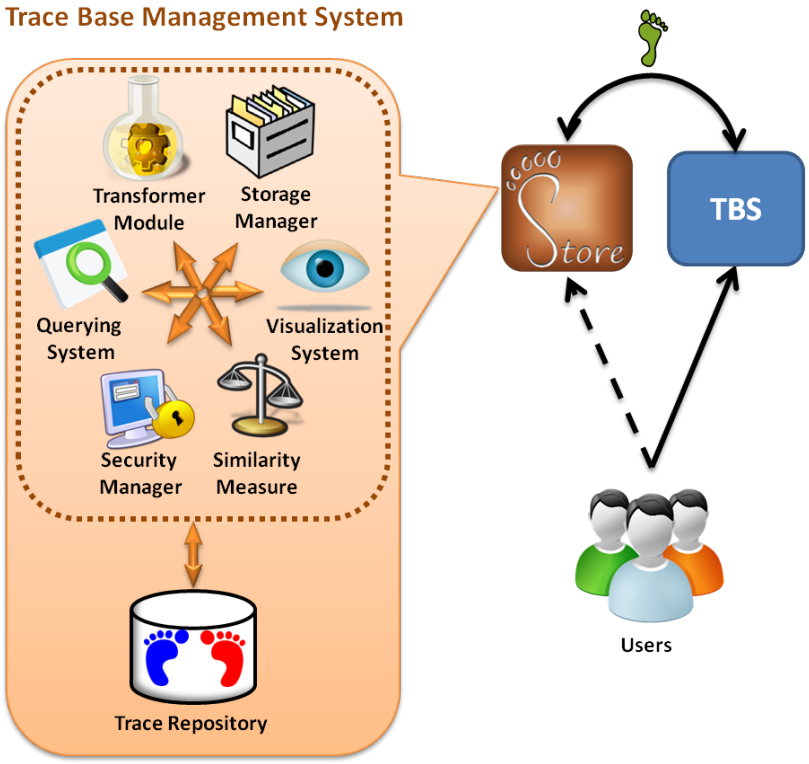

Figure 3.2: $\mathcal{T}$Store general structure.

3.3 $\mathcal{T}$Store Architecture and specifications

In this section, we describe the structure of $\mathcal{T}$Store and its functionalities (see Figure 3.2). $\mathcal{T}$Store is a TBMS for storing $\mathcal{M}$-Traces collected by the Trace Collectors of different client applications. In addition, $\mathcal{T}$Store answers the queries of the assistant that observes the way the user interacts with the system, in order to help the user to do his task effectively. $\mathcal{T}$Store relies on a Database Management System (DBMS) and therefore benefits from performance and storage facilities. $\mathcal{T}$Store contains six modules. The **Storage Manager** receives messages from external clients and stores them in the database in the form of $\mathcal{M}$-Traces. The **Querying System** retrieves $\mathcal{M}$-Traces from the database to answer queries of agents. The **Transformer Module** contains different functions to perform operations on $\mathcal{M}$-Traces to

produce transformed $\mathcal{M}$-Traces. The **Visualization System** allows to visualize $\mathcal{M}$-Traces of all users and to navigate between the different transformations. The **Similarity Measure** provides similarity measures to compare episodes of $\mathcal{M}$-Traces. Finally, the **Security Manager** ensures $\mathcal{M}$-Traces protection and the distribution of roles and privileges. Both similarity measures and security manager are new modules that we propose to be added to the Trace Base Management System architecture as presented in Chapter 2.

3.3.1 Storage Management

The Storage Manager is a module responsible for the communication between the $\mathcal{M}$-Traces stored in the database and the client application connected to $\mathcal{T}$Store. It contains services for creating models, storing $\mathcal{M}$-Traces, obsels and their attribute values. External trace collectors send messages containing the collected obsels to the Storage Manager. The storage manager, in its turn, store these obsels in the database. The Storage Manager allows concurrent access, so multiple users can interact with the system and store or retrieve their $\mathcal{M}$-Traces simultaneously.

Some objects cannot be created until all related objects have been created. $\mathcal{T}$Store schedules a sequence of executions to ensure the dependencies between objects. For example, storing an obsel requires finishing the storage of all its attributes and their values. $\mathcal{T}$Store benefits from XML structure to do so.

The Storage Manager can also automatically update a trace model if it has some missing items like an obsel-type or an attribute type. If some attributes do not have an attribute type in the predefined model, the Storage Manager automatically creates a new attribute type. This feature is useful because, unlike in web logs, it allows $\mathcal{M}$-Trace collectors to store their $\mathcal{M}$-Traces without defining all the details of their models.

3.3.2 Transformer Module

The transformation of $\mathcal{M}$-Traces helps to move from a first simple interpretation (almost raw data coming from sensors) to the defined knowledge level of abstraction. The Transformer Module has predefined functions for frequently used transformations such as filtering, aggregation and segmentation.

Filtering transformation is only based on obsel-types. It takes as input an array of obsel-types to generate a transformed $\mathcal{M}$-Trace containing only these obsel-types. Aggregation allows merging several $\mathcal{M}$-Traces in one transformed $\mathcal{M}$-Trace. Obsel-types of an aggregated $\mathcal{M}$-Trace contain a union of all obsel-types of the original $\mathcal{M}$-Traces. Unlike aggregation, segmentation cuts $\mathcal{M}$-Traces into smaller chunks to be more useful for understanding the behavior of client applications.

In these transformations, the transformed $\mathcal{M}$-Trace preserves the same obsel-types as the original $\mathcal{M}$-Trace and it does not produce new ones. However, simple rules are not suitable for generating outputs. Therefore, we need a mechanism able to recognize and generate transformed $\mathcal{M}$-Traces. Traditional rules are usually used for such transformations. [Mathern 2012] proposes to build Petri Nets from traces using Workflow Mining techniques. In this thesis, we propose to use a Finite-State Transducer (FST) transformation approach to define customized transformations. Transformations are performed using FST task signatures. The task signature concept has been introduced in [Champin et al. 2004] as a set of event declarations, entity declarations, relations, and temporal constraints. A task signature is a formal description of a specific task for which a user might need assistance. Each task is represented by an episode. Thus, task signatures describe the structure of episodes.

As shown in Definition 5, transducers are automata that have transitions labeled with two symbols. One of the symbols represents the input, the other is the output [Roche and Schabes 1997]. On this view, a transducer is said to transduce (*i.e,*, translate) the contents of its input symbols to its output symbols, by accepting a string on its input tape and generating another string on its output tape. A non deterministic transducer may produce more than one output for each input string. A transducer may also produce no output for a given input string, in which case it is said to reject the input.

Definition 5 (Finite-State Transducer)**.** *A Deterministic Finite-State Transducer (DFST) is described as a 7-tuple* $(\mathcal{Q}, i, \mathcal{F}, \Sigma, \Delta, \delta, \sigma)$ *where:*

- $\mathcal{Q}$*: is the set of states,*
- $i \in \mathcal{Q}$*: is the initial state,*
- $\mathcal{F} \subseteq \mathcal{Q}$*: is the set of final states,*

- Σ, Δ*: are finite sets corresponding respectively to the input and output alphabets (obsel-types) of the transducer,*
- δ*: is the state transition function which maps* $\mathcal{Q} \times \Sigma$ *to* $\mathcal{Q}$*,*
- σ*: is the output emission function which maps* $\mathcal{Q} \times \Sigma$ *to* Δ*,*

A transducer is said to be deterministic if both the transition function and the emission function lead to sets consisting of at most one element. Usually, the transition function and the emission function are combined into a single function, which may also be called δ, in $\mathcal{Q} \times \Sigma \longrightarrow \mathcal{Q} \times \Delta$ mapping a pair of a state and one input symbol onto a pair of a state and an output symbol [see Mohri 1997].

In $\mathcal{T}$Store, FST transformations consist in replacing some obsels matching the FST with more abstract obsels. Currently, it is experts who define the structure of the new obsel-type. It is possible that two different transducers generate the same transformed $\mathcal{M}$-Trace. However, FST should be deterministic to avoid ambiguity. By using FST, a large variety of transformations can be applied. To create new FST transformation, the 7-tuple $(\mathcal{Q}, i, \mathcal{F}, \Sigma, \Delta, \delta, \sigma)$ described in Definition 5 should be defined. $\mathcal{T}$Store creates new obsel-types for all the output symbols of the transducer. These obsel-types are contained in the transformed $\mathcal{M}$-Trace model. Each transformation is represented by one transducer. The transformed $\mathcal{M}$-Traces generated by a FST can be reused by another FST for a new transformation.

When a new $\mathcal{M}$-Trace comes, the relevant transducer is called depending on the predefined trace model. The transducer reads the current $\mathcal{M}$-Trace from the first obsel to the last one. At each obsel, the transducer writes an output symbol or skips to the next obsel if the output symbol is the empty character (ε). At the end, the transducers provide $\mathcal{T}$Store with the output symbols that represent the transformed $\mathcal{M}$-Trace to store it.

Example 6. Figure 3.3 shows an example in Wanaclip. A user can search for videos then view them. If he likes the video, he adds it to the selection, otherwise he closes it. This task reflects his satisfaction by accepting or refusing the video depending on the actions. It represents a task that starts by an obsel of type "search" (s) and follows by an obsel of type "view" (v). If the next obsel-type is "add" (d), then an obsel of type "accept" (a) is generated;

(d/a) means replace (d) by (a). Else, if the obsel-type is "close" (c), then an obsel of type "ignore" (i) is generated. It transforms $(s\ v\ d)$ to $(s\ a)$, and $(s\ v\ c)$ to $(s\ i)$, where, $state0$ is the start state, $state3$ is the accept state and (ε) is the empty character. ◊

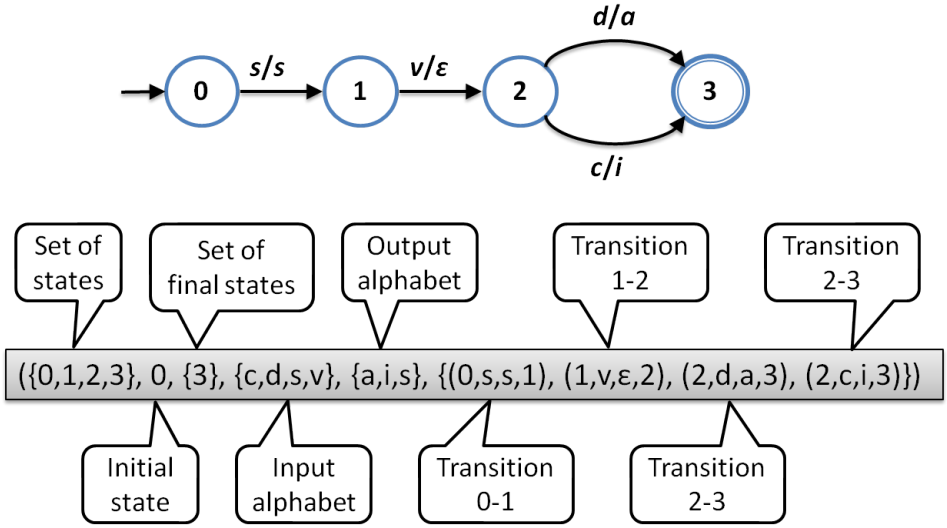

Figure 3.3: Example of a FST transformation.

3.3.3 Security Management

As $\mathcal{M}$-Traces are records of users' interactions on applications, they can carry personal and sensitive data (such as passwords and credit cards numbers). This is why $\mathcal{T}$Store implements a security policy. The needs for this is to ensure the ability to assign fine grained access rights to different users on different $\mathcal{M}$-Traces. To meet this need, we use a well-known policy called RBAC RBAC [see Ferraiolo et al. 2003]. RBAC is a role-based access control approach. The properties of RBAC fully answer our needs and thus RBAC assures the security requirements. Agents are responsible of what they send to $\mathcal{T}$Store. However, $\mathcal{T}$Store ensures that only those with sufficient privileges can access the stored $\mathcal{M}$-Traces. It protects the $\mathcal{M}$-Traces by securing the underlying DBMS that holds that $\mathcal{M}$-Traces. The Security Manager allows creating roles and associating them with tasks. The privileges to perform certain operations are assigned to specific roles. Users are assigned particular

roles, and through these role assignments acquire the privileges to perform particular functionalities like creating models, adding user, deleting $\mathcal{M}$-Trace, *etc.* Each user is responsible of his $\mathcal{M}$-Traces and he can also specify their visibility to be public, private or custom. Private $\mathcal{M}$-Traces can be used only for statistics by the system. Anyone can access and retrieve public $\mathcal{M}$-Traces. It is possible to have privileges on any item such as $\mathcal{M}$-Trace, obsel and attribute type. For example, video subtitle attributes can be invisible to specific users.

3.3.4 Querying system

$\mathcal{M}$-Traces are a large source of information. So, we need a system that enables the extraction of episodes and patterns from $\mathcal{M}$-Traces. $\mathcal{T}$Store allows $\mathcal{M}$-Traces to be retrieved at all levels and to navigate between transformed $\mathcal{M}$-Traces. The Querying System contains some predefined methods to allow $\mathcal{M}$-Trace retrieval using different criteria. It can retrieve $\mathcal{M}$-Traces for a specific user, matching a specific period of time, containing a set of obsel-types, *etc.* It provides statistics about $\mathcal{M}$-Traces, obsels and users, *e.g.* their frequencies, relations, reusing, *etc.* We aim to define a querying language that is able to represent all the types of agents' queries.

3.3.5 Similarity Measure

This module provides comparison functions to compare between $\mathcal{M}$-Traces. It is used during the process of retrieving similar $\mathcal{M}$-Traces in the querying system. A detailed description of the similarity measure mechanism is presented in Chapter 5.

3.3.6 Visualization System

The Visualization System allows to visualize $\mathcal{M}$-Traces and navigate between the different transformations. It enables to extract any trace to get its metadata, obsels and model. This visualization interface for Wanaclip is presented in Chapter 6.

3.4 Implementation

$\mathcal{T}$Store is implemented as a PHP web tool over a MySql DBMS. The Entity-Relationship Diagram (ERD) of $\mathcal{T}$Store is shown in Figure 3.4. It consists of a trace part and a security part. A client sends a request to a PHP script on the server where $\mathcal{T}$Store is loaded. $\mathcal{T}$Store parses the request, loads the requested service, calls it, and returns the answer accordingly. One of the most important advantages of $\mathcal{T}$Store is that it is independent of the client environment. $\mathcal{T}$Store exchanges XML messages that have a predefined structure.

Example 7. The following XML message can be sent to "storeObsel" service to add an obsel of type "SearchMedia" to the $\mathcal{M}$-Trace ($id = 27$) of the model ($id = 1$). This obsel has 3 attributes ($tags =$ "*Lyon*", $kind =$ "*right*" and $duration =$ "30"). If the obsel-type and attribute types are not defined, it automatically calls "storeObselType" and "storeAttributeType" services to create them.

```
<?xml version="1.0" encoding="UTF-8"?>
<obsel id="1" traceID="27" start="2012-09-21 17:31:26"
 end="2012-09-21 17:31:26">
  <obselType name="SearchMedia" modelID="1"
    <attributes>
      <attribute id="1" name="tags" type="string">
        lyon
      </attribute>
      <attribute id="2" name="kind" type="string">
        right
      </attribute>
      <attribute id="3" name="duration" type="number">
        30
      </attribute>
    </attributes>
  </obselType>
</obsel>
```

◊

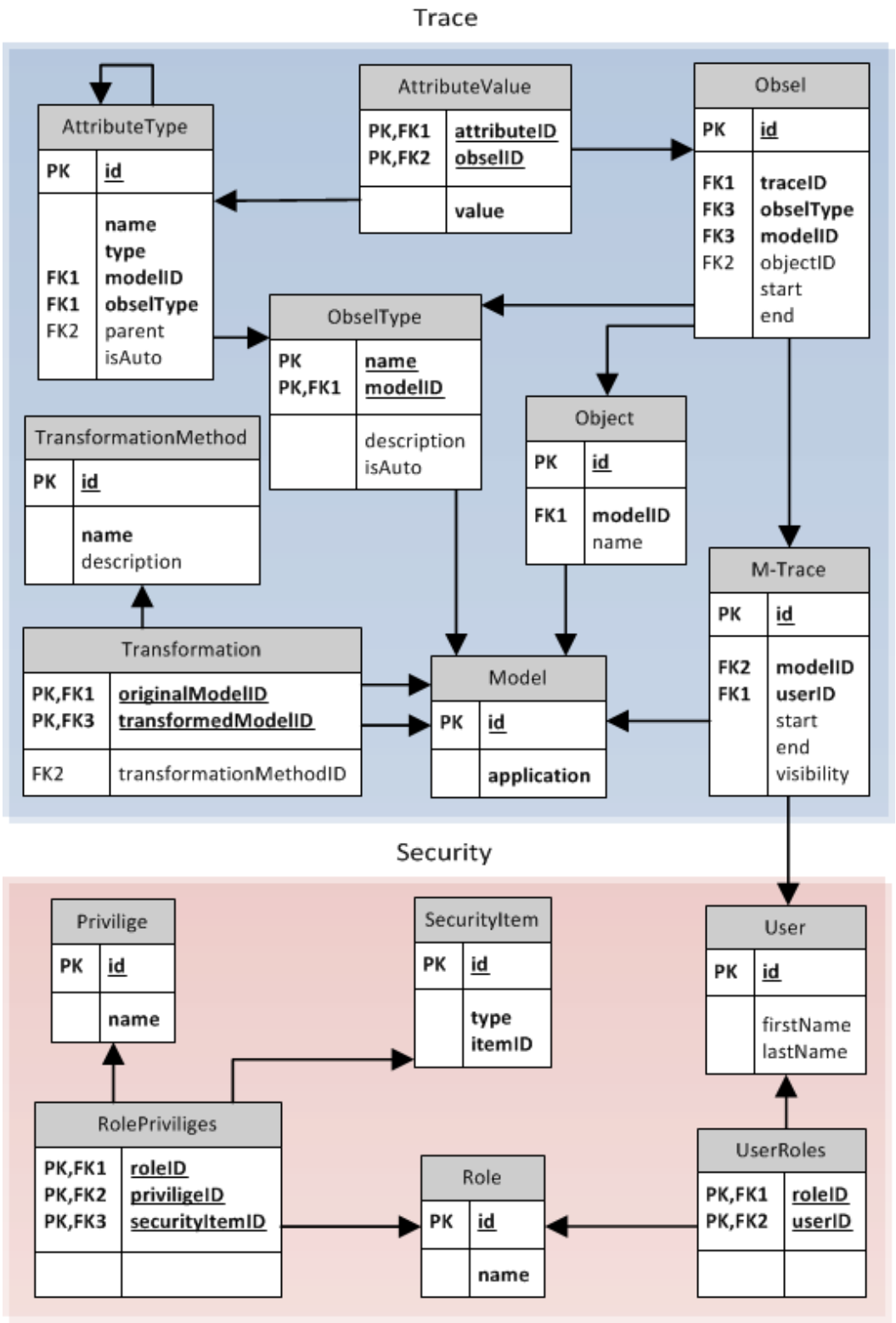

Figure 3.4: $\mathcal{M}$-Trace Entity-Relationship Diagram.

$\mathcal{M}$-Traces are indexed and partitioned to enhance the retrieval performance. Obsels are sorted according to their $traceID$, so that we can retrieve as fast as possible all the obsels and their attributes of a specific $\mathcal{M}$-Trace. $\mathcal{M}$-Traces are also sorted and indexed according to their time-stamps because most of search queries retrieve $\mathcal{M}$-Traces for a specific time interval.

3.5 Test and Performances

In order to test our environment, a collection process has been implemented in Wanaclip. In this process, the $\mathcal{M}$-Trace handler captures users' events, create the corresponding obsels and stores them in $\mathcal{T}$Store. Some tests have been conducted to determine how $\mathcal{T}$Store performs in terms of responsiveness, memory usage and stability under workload. Both $\mathcal{T}$Store and Wanaclip are running on the same computer. These performance measurements were performed using a laptop with an AMD Phenom II N930 Quad-Core 2.00 GHz processor, 4.00 GB RAM and a Windows7 32bit operating system. Evaluations of other services in $\mathcal{T}$Store such as the similarity measures and the querying services are presented in Chapters 5 and 6

Wanaclip collects about 50 different types of obsels. Each obsel-type contains a different number of attributes from 1 to 30. After 1000 random tests to store a single obsel, It is measured that the average storage time for an obsel is 0.148 seconds (0.054 execution time + 0.094 messaging time). The execution time is the time $\mathcal{T}$Store spends to store an obsel in the database after receiving the message from the $\mathcal{M}$-Trace collector. The messaging time is the time of exchanging a message between $\mathcal{T}$Store and the $\mathcal{M}$-Trace collector (sending + receiving). The average memory usage to store a single obsel is 26.32 KB.

To examine the storage of multiple obsels, we tried 1000 random tests. At each test, a random number of obsels are stored at once as a chunk (1 to 150 obsels in a chunk). As a result, it takes on average 1.368 seconds (1.118 execution time + 0.250 messaging time) to store a chunk. A chunk contains 41.7 obsels on average and each obsel contains 9.6 attributes on average. The average memory usage needed to store a chunk is 50.96 KB.

Table 3.1 shows a comparison between single and multiple obsel storage.

Table 3.1: A comparison table between single and multiple obsel storage.

Factor/Storage type	Single obsel	Multiple obsels
Messaging time	0.094 S	0.250 S
Execution time per obsel	0.054 S	0.043 S
Execution time per attribute	0.006 S	0.005 S
Storage time per obsel	0.148 S	0.048 S
Storage time per attribute	0.015 S	0.005 S
Memory usage per obsel	26.325 KB	1.243 KB
Memory usage per attribute	2.742 KB	0.129 KB

These values are calculated based on the average value of all the tests that are performed. Depending on these results, Wanaclip needs about 2.485 seconds to store 100 obsels as a chunk in $\mathcal{T}$Store and it uses 70.792 KB of memory. But, it needs about 14.8 seconds to store 100 obsels separate and 26.325 KB of memory for each process. Thus, it is seen that the multiple obsel mechanism reduces the time required for storage. This is due to the spent time for messaging, parsing command and inserting rows in the database. In multiple obsels, it reduces the number of exchanged messages and the required time for parsing the messages.

As shown in Figure 3.5, the execution time and the memory usage for multiple obsels storage have logarithmic growth. The higher the number of obsels stored, the higher the memory usage and the required execution time. However, messaging time hardly changes, thus it can be considered as a fixed value. The most important factor is the number of attributes in the obsels and their hierarchy. The obsels that have more attributes need more time to be stored. As an example, 5 minutes of activity on Wanaclip generates an average of 25 obsels. $\mathcal{T}$Store takes less than one second to store all of them. So, it is seen that this is a transparent process for the user and there is no risk to lose information.

3.6 Discussion and Related Works

Many applications write log files to get some information about the use of the application. Log files typically consist of a long list of events in temporal order in plain text files. Most often, one line of text corresponds to one log

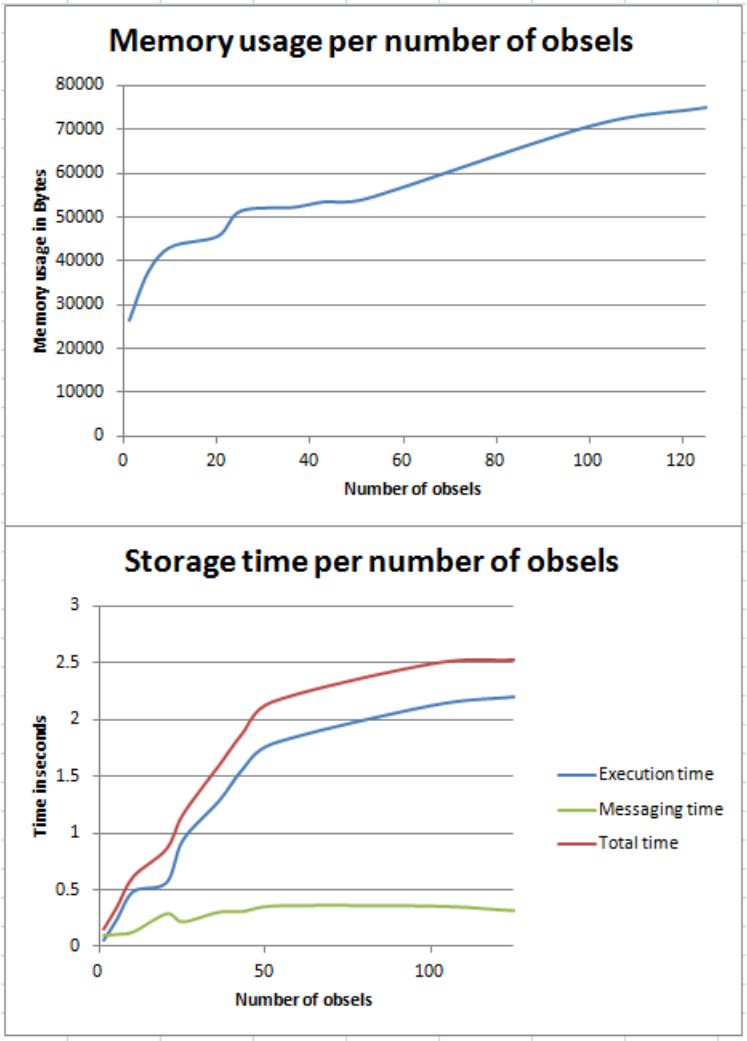

Figure 3.5: Storage time and memory usage per number of obsels.

entry. If an entry contains several fields, they are separated by a delimiter, *e.g.* a semicolon. The problem is that the delimiter may be part of the log information. Many programming environments and networking tools use XML log files for indicating the status of variables, the results of decisions, warnings of potential problems and error messages when things go wrong. For example, WinSCP [WinSCP 2012] uses XML logging to find a list of files that were actually uploaded/downloaded and Record operations done during synchronization.

In web applications, there are three tracing approaches. Tracing systems can be located on client side (*i.e,* browser plug-in), or integrated in the traced

applications on server side, or combined with approaches. In all cases, users' activities are usually recorded as web logs that contain mainly the visited links. Currently, [Ginon et al. 2013a] are working on developing a general trace collector. They want it to be able to collect $\mathcal{M}$-Traces from any web applications without modifying them. CoScripter is a Firefox plug-in created by an IBM Research group [Leshed et al. 2008]. It records user actions and saves them in semi-natural language scripts. The recorded scripts 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.

The UserObservationHub [Haas et al. 2010] is a small desktop service (daemon) that catches several registered user observation notifications and passes them on to interested listeners. Most of these tracing systems are mainly for collecting users' interactions but not for managing and reusing them. In bioInformatics, the International Nucleotide Sequence Databases provide the principle repositories for DNA sequence data. In addition to hosting the text sequence data, they host basic annotation and, in many cases, the raw data from which the text sequences were derived [Batley and Edwards 2009].

Recently, new TBMS systems implement the $\mathcal{M}$-Trace concept. Table 3.2 shows a comparison between the TBMS systems: Abstract [Mathern 2012], kTBS [Champin et al. 2013] and $\mathcal{T}$Store. The Kernel for Trace-Based Systems (kTBS) was the first TBMS developed in Python by [Champin et al. 2013]. Both kTBS and $\mathcal{T}$Store implement the $\mathcal{M}$-Trace concept. kTBS uses RDF files to store $\mathcal{M}$-Traces, while $\mathcal{T}$Store manages $\mathcal{M}$-Traces in a database and benefits from its functionalities. kTBS does not have predefined similarity measures to compare between traces. However, to do that, we need to retrieve all the traces matching a specific period of time and compare them outside of kTBS. Client applications use HTTP requests to send their $\mathcal{M}$-Traces in $\mathcal{T}$Store. kTBS currently support APIs for Java, PHP and Flex. $\mathcal{T}$Store handles several obsels together to reduce network traffic. kTBS supports different message formats like, JSON, XML, and Turtle. Currently $\mathcal{T}$Store only supports XML but further development is planned to support other formats. kTBS and $\mathcal{T}$Store have predefined transformations, in addition, $\mathcal{T}$Store sup-

ports FST transformations. kTBS transformations are filtering, fusion and SPARQL rule. Abstract was not initially designed as a TBMS, but as a full application. Therefore, no external applications are supposed to access Abstract database. This is the reason why there are some N/A (not applicable) in Table 3.2.

Table 3.2: Comparison table of existing Trace Base Management Systems.

Service/System	**kTBS**	**Abstract**	**$\mathcal{T}$Store**
Storage	RDF Files	XML/RDF files	XML/Database (multiple obsels)
Messaging	XML, JSON and Turtle	N/A	XML
Protocol C/S	APIs for Java, PHP, Flex	N/A	HTTP Requests
Transformation	SPARQL Rules	SPARQL Rules	Finite-State Transducers
Querying	Predefined functions	No	Predefined functions
Security	No	No	RBAC Approach
Similarity	No	No	Yes
Visualization	Yes	Interactive graphical visualizations	Yes

The implementation of $\mathcal{T}$Store is still in progress and a lot of services should be added. Future work will involve developing a querying language that allows answering different users' requests. We need to define FST Transformations automatically. A user interface is one of the important things to be provided since it allows users and administrators to browse and manage $\mathcal{M}$-Traces according to their privileges. Currently $\mathcal{T}$Store only supports XML messages. Work is in progress to add new formats.

3.7 Conclusion

In this chapter, we described $\mathcal{T}$Store, a web tool serving as a Trace Base Management System. The originality of $\mathcal{T}$Store resides on its performance and in the transformation facilities on $\mathcal{M}$-Traces that it offers. We presented the six modules of $\mathcal{T}$Store. The Storage Manager receives messages containing $\mathcal{M}$-Traces from the clients and stores them in the database. The Querying System retrieves $\mathcal{M}$-Traces from the database to answer queries of client applications.

The Transformer Module contains different functions to produce transformed $\mathcal{M}$-Traces. The Visualization System allows viewing $\mathcal{M}$-Traces and navigating between the transformed $\mathcal{M}$-Traces. The Similarity Measure module that can compare episodes of $\mathcal{M}$-Traces. The Security Manager ensures $\mathcal{M}$-Trace protection and the distribution of roles and privileges. The major contribution described in this chapter is the customized transformation approach based on the Finite-State Transducer (FST) principle for $\mathcal{M}$-Trace transformations. We proposed to use FST as a way to represent signatures of users' behaviors.

Our experiments showed that storing multiple obsels as a chunk is better than storing them separately. It shows also that the execution time and the memory usage for obsels storage have logarithmic growth. Next chapter will focus on our first application (SAP BusinessObjects Explorer) where we provide assistance in form of trace replay. It shows how to use traces to enable replay of user's interactions and how to manage impact propagation of changes during the replay process.

Chapter 4

$\mathcal{M}$-Trace Replay

Contents

Abstract To help end-users master complex applications, it is often efficient to enable them to "replay" what they have done before. In some situations, it is even more useful to enable them to modify some parameters of the actions they are replaying, so that they can see the consequences of the modification. Unfortunately, it is not always possible to replay the series of subsequent actions after a modification of a prerequisite. Hence, the replay process has to deal with change impact propagation. In this chapter, we describe our models to enable replay of user's interactions and to manage impact propagation of

changes during the replay process using impact rules to perform the adaptation. 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 BusinessObjects Explorer, an application assisting business users in managing large amounts of information. Our tool helps users to better understand the application.

Keywords: Impact Propagation, Macro Recording, Bookmarks, Trace-Based Reasoning, Replay Traces, User Assistance, Modeled Trace, Human Computer Interaction.

Related publications: [Zarka et al. 2011] and [Zarka et al. 2010]

Acknowledgments We thank Francoise Corvaisier, member of SAP-BO enterprise for her support, thoughts and for giving us the opportunity to do this work with SAP-BO.

4.1 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 complex, and difficult to master. While using these tools, users are often lost, usually because of the lack of 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 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*. The same assistance is provided to all users. Which is not always well suited to particular needs of specific users.

To overcome this issue, as presented in the previous chapters, we propose to use Trace-Based Reasoning in order to provide a contextualized help to end users. A trace can be seen as a story of the user's actions, one by one. Hence, traces enable us to capture users' experiences. More precisely, we use $\mathcal{M}$-Traces. Working with $\mathcal{M}$-Traces raises numerous research issues. How to collect, represent, store, and visualize $\mathcal{M}$-Traces? How to take into account privacy issues for $\mathcal{M}$-Traces? In the previous chapter, we have presented $\mathcal{T}$Store as a framework that provides some answers to these questions. However, in this chapter, we focus on a specific research question: how to replay an $\mathcal{M}$-Trace in a pre-existing system and which issues are raised by the replay when the initial situation has been modified? To better understand this problem, consider the following example:

Example 8. A user makes a sequence of interactions 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, add a background. If 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: "are the remaining actions still possible?", because for example, if the background picture is not available any more, then the replay will fail. ◊

The issue we address in this chapter is then: how to enable $\mathcal{M}$-Trace replay while monitoring the impact of a modification in the initial $\mathcal{M}$-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 an $\mathcal{M}$-Trace (*i.e,* with no modification) from any point in the $\mathcal{M}$-Trace. Then, we have defined a model to manage impact propagation after a modification of the $\mathcal{M}$-Trace. We make the hypothesis that the interactions change only the possibility of interacting with the system. These interactions do not change the state of the permanent objects of the application. The $\mathcal{M}$-Trace replay mechanism has been implemented in the SAP BusinessObjects Explorer application[1]; a web application enabling users to load, explore, visualize and export large quantities of data. SAP-BO[2] needed a tool to help their users better understand the application, so, we designed this solution for them. We have instrumented the initial application in order to collect interaction traces and we have developed a graphical interface that displays the $\mathcal{M}$-Traces according to an *ad-hoc* knowledge representation. Using this interface, any user can replay his recorded $\mathcal{M}$-Traces. The application is operational and a demo video is available[3]. However, we admit that this work has broad prospects of development to be completed. Many ideas in this work still need to be developed, tested and validated. We could not continue working on this project because of financial issues. This work is focused for the development of $\mathcal{T}$Store and the recommendation system of Wanaclip.

This chapter is organized as follows. In section 4.2, we survey related work. Then, in section 4.3, we show how we use $\mathcal{M}$-Traces in order to enable replay of user's interactions. In section 4.4, we discuss the consequences of a change during the replay, and we propose a rule-based impact propagation model. Section 4.5 gives implementation details. Open issues and discussion of our proposal are made in section 4.6. The chapter ends with a conclusion and a description of future research issues.

[1]SAP BusinessObjects Explorer: http://help.sap.com/boexpl

[2]SAP: http://www.sap.com

[3]A demo video of trace replay in SAP BusinessObjects Explorer: https://liris.cnrs.fr/~rzarka/ReplayTraceDemo/

4.2 Related Work

There are different methods to Replay past actions and go back to a particular state of the application. The most common methods are: macro recorders, bookmarks, undo commands, built-in functions, *etc.* Macro recorders are systems that record and play back mouse events and keystrokes. In most of existing macro recorders, 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 commands that can be replayed later. Recorded scripts are stored automatically on a wiki. CoScripter is a Firefox plug-in created by an IBM Research group [Leshed et al. 2008]. It allows users to record and share interactions with websites. It records user actions and saves them in semi-natural language scripts. The recorded scripts 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 and 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. Smart Bookmarks are an extended kind of internet bookmark. They directly give access to functions of web sites, as opposed to filling web forms at the respective web site for accessing these functions. While Smart Bookmarks allow users to save or share actions from ongoing browsing sessions, ActionShot [Li et al. 2010] enables users to share actions they have performed using a visual interface for browsing their entire history. ActionShot is built on top of CoScripter. 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 limitation because it cannot work with other browsers. Besides, they cannot work with dynamic pages (*e.g.* Ajax or Flash based).

In Smart Bookmarks, users can modify parameter 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. Briand et al. [2003] propose a UML model-based approach to impact analysis that can be applied before any implementation of changes in the UML model. This allows an early decision-making 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. Maule [2010] presents a novel method 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 a 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].

Some applications allow users to replay their actions like Photoshop [see Harrington 2009], by using undo or go back 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 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 client-server applications. More precisely, pages that use POST method to send web forms from client to server cannot undo the operation, thus cannot be replayed. Therefore, we aim to replay the whole navigation process including pages that use POST methods. In addition, we do not want to ask the user to start or stop recording his actions. However, we agree with [Dyke 2009] that not all traces can be replayed. In his thesis, [Dyke 2009] defines the concept of a *replayable*. A replayable is a generic artefact which can be replayed, synchronized with other artefacts, visualized,

transformed and enriched. According to him, $\mathcal{M}$-Trace which adds semantics and determines the well-formedness of queries and transformations. He chose to present a model which does not present a great degree of formalism because $\mathcal{M}$-Traces did not allow him to define generic transformations. However, we developed the notion of Finite-State Transducers in the past chapter to answer this problem.

$\mathcal{M}$-Trace replay is a step towards the goal of finding different personalized tasks to make applications more "task-aware", which is tackled. For example, TaskTracer [Dietterich 2012] collects user's interactions to organize the user's information naturally according to tasks. They use machine learning techniques to learn and adapt solutions according to user specificities. CaBMA (for: Case-Based Project Management Assistant) [Xu and Muñoz Avila 2007] is another work providing assistance with project planning tasks. The system uses Case-Based Reasoning by adding a knowledge layer on top of the traditional project management software, going beyond the editing and bookkeeping capabilities that this software is traditionally limited to. CaBMA automatically captures cases from previous project plans, and reuses them for planning.

Table 4.1: Comparison table of related work to $\mathcal{M}$-Trace replay.

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	Screen-shots, Firefox extension	Retroactive	Yes, without impact propagation	Classify buttons for side-effecting
ActionShot	Graphical text, Firefox extension	Retroactive	Yes, without impact propagation	No
Photoshop	Actions list	Macro, undo command	Yes	Yes
Power Designer	No	Undo command	No	Impact rules
$\mathcal{M}$-Trace Replay (Our approach)	$\mathcal{M}$-Trace with text explanations	Proactive	Yes	Impact rules and adapted values

Table 4.1 shows a comparison between all these systems. Representation column shows how each system represents and visualizes past actions. Both replay columns describe the ability of the systems to support the replay with or without changes of values. Adaptation column show the adaptation methods used by each system to allow replaying with changes.

4.3 Simple $\mathcal{M}$-Trace Replay (go back to a previous state)

To enable users going back to a previous state, we propose to implement an "$\mathcal{M}$-Trace replay mechanism". This mechanism enables users to replay their actions until they reach the expected state of the application. In order to

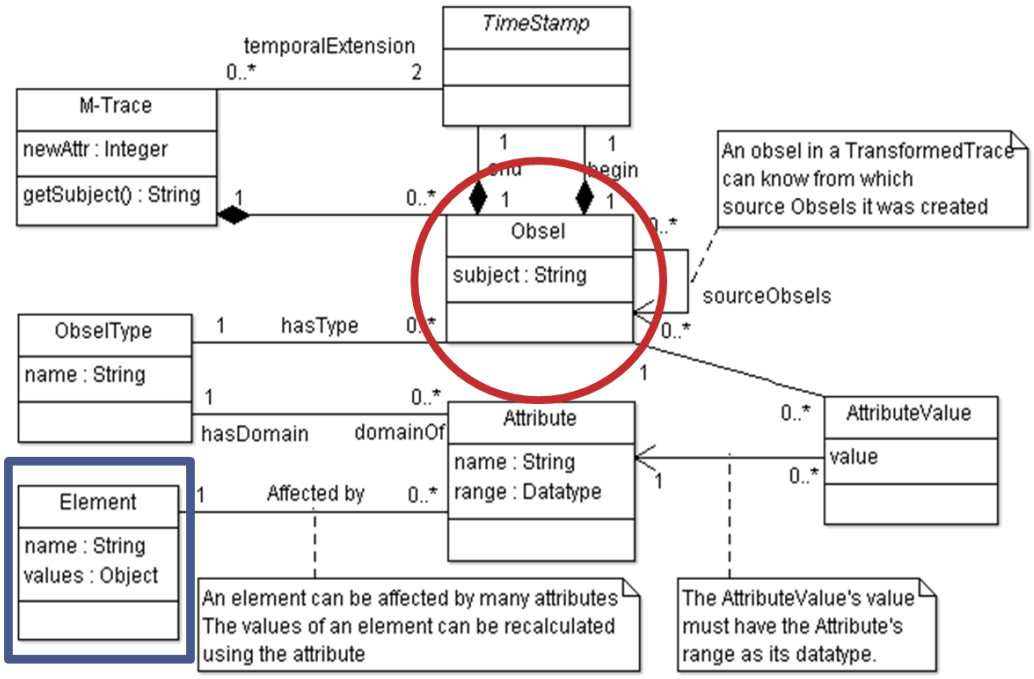

Figure 4.1: Modified $\mathcal{M}$-Trace model to support $\mathcal{M}$-Trace replay.

implement this mechanism, we propose to extend the $\mathcal{M}$-Trace meta-model to fit the needs of replaying traces. This extended meta-model has a new entity called "**Element**" (framed on the bottom left corner of the Figure 4.1). Element entity allows to keep all the necessary information about the different

objects of the application. Therefore, we can store their changing values over time after each new action.

In the diagram shown in Figure 4.1, we see that each user's session is represented by an **$\mathcal{M}$-Trace**. An $\mathcal{M}$-Trace consists of a set of observed elements (**obsels**). Each obsel has a type and two **time-stamps** representing its beginning and ending instants. Each **obsel-type** has a domain of **attributes** and indicates the values of its attributes within the range of the attribute-type. An obsel can affect many elements at the same time. For example, pressing "delete all" button can erase the values of many elements together. By using the obsel attribute-values, we can compute 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 an element. Respectively, we can get all the elements that can be affected by an obsel. When capturing the $\mathcal{M}$-Traces, it is not needed to store the values of elements at each time, but rather only the attributes and values of each obsel.

Example 9. In SAP BusinessObjects Explorer, 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. The complete information can be found in the "selected chart" element. ◊

4.3.1 Restarting $\mathcal{M}$-Trace process

Our solution to go back to a previous state of the system is to replay $\mathcal{M}$-Traces from a starting point (session start) and not by undoing last ones. Undoing last changes requires an execution for each undo command. However, replaying the whole process from a starting point can be done with one replay action. 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 interactions that happened from the beginning of the session until the selected obsel; let it be the ***triggered obsel*** (the obsel where we want the system to play back to).

Example 10. Figure 4.2 shows a simple $\mathcal{M}$-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 $\mathcal{M}$-Traces to restart the system to its

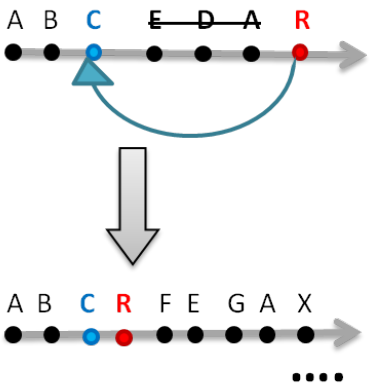

Figure 4.2: Simple $\mathcal{M}$-Trace Replay.

state when clicking on C. It is seen 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 $\mathcal{M}$-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. ◊

Algorithm 1 Simple $\mathcal{M}$-Trace Replay Algorithm

Input: $\mathcal{M}$-Trace: the current trace, $TriggerredObsel$: the obsel where we want the system to play back to.

Output: $Elements$: the new element values are updated making the system going back to a past state

1: $ReplayedTrace \leftarrow getSubTrace(0, getPosition(TriggeredObsel))$
2: $Optimize(ReplayedTrace)$
3: $Elements \leftarrow getDefaultValues()$
4: **for** $pos = 0 \rightarrow getObselCount(ReplayedTrace) - 1$ **do**
5: $\quad Obsel \leftarrow ReplayedTrace[pos]$
6: $\quad Attributes \leftarrow getAttributes(Obsel.Type)$
7: $\quad$ **for each** $attribute \in Attributes$ **do**
8: $\quad\quad value \leftarrow getAttributeValue(Obsel, attribute)$
9: $\quad\quad elem \leftarrow getAffectedElement(attribute)$
10: $\quad\quad Elements[elem] \leftarrow GenerateElementValue(value)$
11: $\quad$ **end for**
12: **end for**
13: **return** $Elements$

As shown in Simple $\mathcal{M}$-Trace Replay Algorithm, the algorithm gets $\mathcal{M}$-Trace and the triggered obsel as input and goes back to a previous state. Firstly, it gets the subset of the $\mathcal{M}$-Trace that should be replayed starting from the first obsel to the triggered one by a chronological order. Then, this $\mathcal{M}$-Trace will be optimized by using the optimization algorithm to filter extra obsels. Each element gets its default values and then a loop runs over 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.

4.3.2 Optimized $\mathcal{M}$-Trace replay process

As not all the obsels play a role for changing the state of the system, 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. Reset values can be considered as new starting points in the session, thus avoid manipulating the whole process. An optimized chronological list of obsels can be obtained from the beginning of the session to the triggered obsel; this list will be used to generate the values for each element. Optimization algorithm uses a Finite-State Transducer to apply a filtering transformation on $\mathcal{M}$-Traces. It produces a transformed $\mathcal{M}$-Trace by filtering all unnecessary obsels that induce loops in the $\mathcal{M}$-Trace.

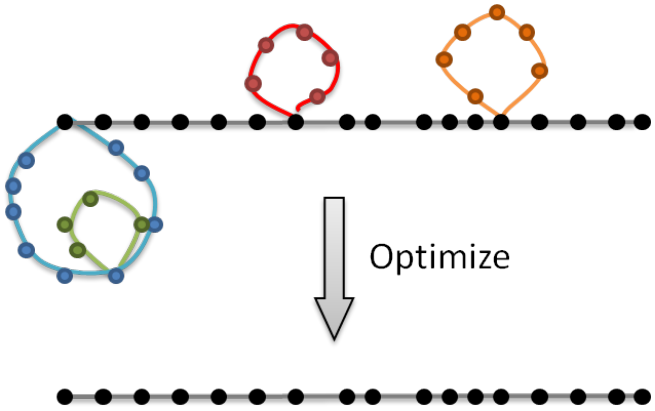

Figure 4.3: $\mathcal{M}$-Trace Replay Optimization.

Example 11. In the simple replay obsel, the subTrace from replay obsel to triggered obsel should be filtered. The same thing is also done for a reset obsel which means filtering all the obsels from the beginning to the reset obsel. Thus, it is considered that there is a list of unnecessary loop obsels in the $\mathcal{M}$-Trace that should be filtered as shown in Figure 4.3. ◊

4.4 Replay $\mathcal{M}$-Traces With Impact Propagation

This section describes how we can replay $\mathcal{M}$-Traces after modifying an obsel by handling the consequences of changes on elements. This is illustrated by the following example:

Example 12. As shown in Figure 4.4, R is a replay obsel that triggers a replay of the $\mathcal{M}$-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 find the adapted values and replay $\mathcal{M}$-Traces with these new values (if possible). After that the user can continue to use the system. ◊

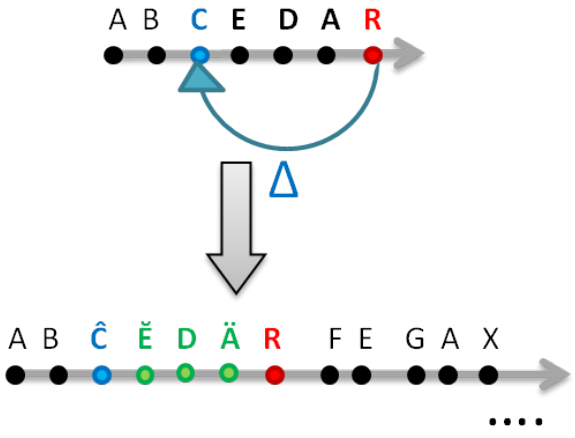

Figure 4.4: Replay $\mathcal{M}$-Traces With Change.

Many questions are raised. For example, how to identify the elements affected by a change? How to be proactive and specify the appropriate new values, without asking the user to enter the new values? How to replay the next interactions 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. However, the problem of the impact propagation of changes in the transformation graph of $\mathcal{M}$-Traces is raised. We think that modifications on the Finite-State Transducers can answer this problem. We could not continue working on this issue because this project is finished.

4.4.1 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 (***modified element***) and the condition on its values that specifies the dependence with a destination element (***affected 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.

For each application, system experts define impact rules for the dependencies between the elements. Impact rules determine the consequences of modifying an existing obsel. Usually, system experts define impact rules before deploying it. However, they can add new rules anytime during its usage. Each rule impacts some 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 $\mathcal{M}$-Traces.

Example 13. In SAP BusinessObjects Explorer, 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 an $\mathcal{M}$-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 as shown in Table 4.2: ◊

Table 4.2: Impact rule example.

$E_S = SelectedMeasures$	$E_D = Chart$
$C_S = (Count() > 1)$	$C_D = (type = \text{“}Pie\text{”})$
$Æ = (Type = \text{“}VerticalBar\text{”})$	

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 time-stamps, thus causing a change in the order between obsels, updating a value for an attribute of an obsel, or even filtering an obsel. By using impact rules, we can determine the consequences of a change and the adapted values. If it is hard to find an adapted value for an element or if we cannot find 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.

4.4.2 Retrieving adapted value from past $\mathcal{M}$-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 $\mathcal{M}$-Traces. For this purpose, we propose to use a retrieval algorithm similar to the algorithm presented in [Zarka et al. 2010]. In the original algorithm, obsel values are not taken in consideration when retrieving episodes similar to the current one, because we just wanted to know the next recommended interactions. An episode is a sequence of interactions that allow solving a problem. 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 interactions.

The algorithm starts by selecting a subset of the $\mathcal{M}$-Trace from its beginning to the modified triggered obsel (an episode). This episode represents the part that should be replayed without any modification. All the obsels after the triggered obsel are also in the replay process but their values may be changed during the replay process. The algorithm retrieves all the past similar episodes to the current one. To do that, it is needed to compare the current episode with past ones using similarity measures (see next chapter). These similarity includes values comparison.

For each similar episode, the algorithm computes 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, the algorithm takes only the one that occurs most often and it considers it as the adapted one. If the algorithm fails in retrieving any episode, it keeps this element as an invalid element until another obsel modifies its value later. Otherwise, the replay process will fail and the system will ask the user to choose the appropriate value manually.

4.5 Implementation

In the previous sections, we have described the models that we have defined to support replay of user interactions by exploiting $\mathcal{M}$-Traces. In this section, we show how we have implemented our $\mathcal{M}$-Trace replay model into the SAP BusinessObjects Explorer application.

4.5.1 $\mathcal{M}$-Trace collecting and visualization

Firstly, we have modified SAP BusinessObjects Explorer for being able to collect obsels. SAP BusinessObjects 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[4]; each user has a web application where he can do his exploration. Traces are collected on the client side. Figure 4.5 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

[4]Flex: http://www.adobe.com/products/flex.html

of the user is collected as an obsel presented in a XML format specifying the obsel-type, time-stamps, and the values of this obsel. We see that the interface of SAP BusinessObjects 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.*

Example 14. The "measures" block contains many types of obsels like "select measure", "add calculation", "edit calculation", *etc.* 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 time-stamp. ◊

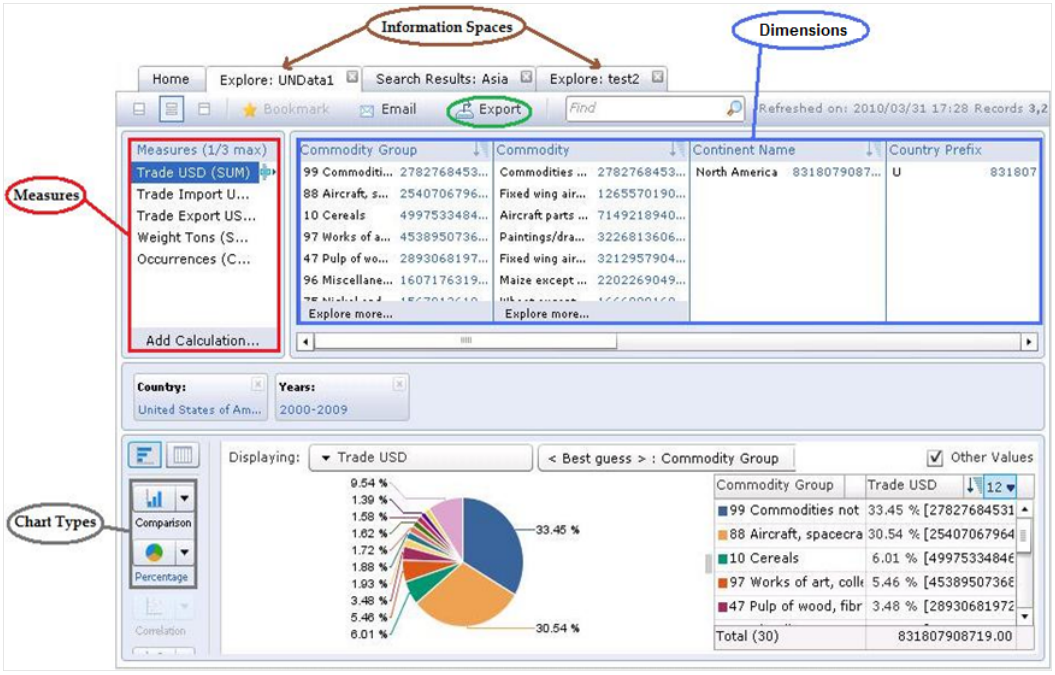

Figure 4.5: SAP BusinessObjects Explorer user interface.

Each session is presented as an $\mathcal{M}$-Trace structured in XML and has a unique session ID, contains the ID of the user who owns this session, and the temporal list of obsels that happened in this session. When a user log himself in SAP BusinessObjects 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 operation or activity. All the obsels of a session, whatever Information Spaces they belong to, are stored in the same file.

We have developed a new interface to visualize users' $\mathcal{M}$-Traces displaying a graphical representation of what they have done so far (see Figure 4.6). Each obsel is captured according to our model and is classified according to 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 values are visualized in the form of a tree of attributes and their values.

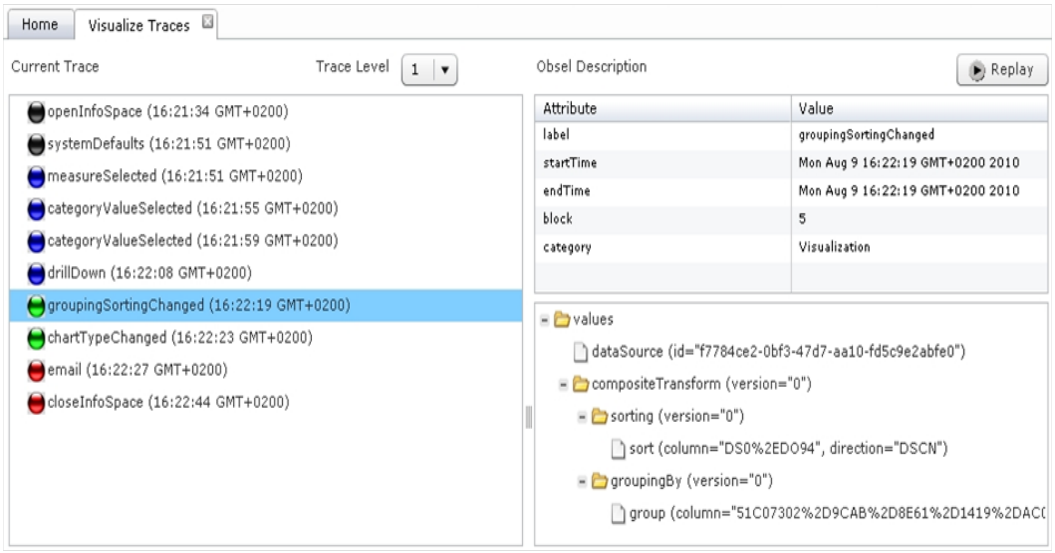

Figure 4.6: SAP BusinessObjects Explorer $\mathcal{M}$-Trace visualization interface.

4.5.2 $\mathcal{M}$-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 Figure 4.6). The system will automatically replay $\mathcal{M}$-Traces to go back to this state. A new obsel of type "Replay" will be added to the obsels list. 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

explained before, the optimization algorithm uses replay obsels to minimize the number of replayed obsels by filtering the obsels that are skipped in the replay action.

Each element has a different type and number of values. In order no to analyze each element in a different way, we need a general method. 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 a great deal of flexibility and control. To do so, the general type "Object" is used as a type of obsel attributes, which means that any obsel attribute can have any type for its values. We do introspection on the attribute in order to determine the content of it and then to manipulate it in a general way.

4.6 Discussion and Open Issues

We have implemented our replay method within the SAP BusinessObjects Explorer application. However, this method can be applied in any system where we can collect $\mathcal{M}$-Traces. To enable $\mathcal{M}$-Trace replay, the first step is to collect $\mathcal{M}$-Traces. We have experimented with our system by using many types of datasets and by considering all obsel-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. It takes only the time needed to exchange a message between a client and a server. A demo video is available online[5].

When implementing the system, many problems are raised. Protection of users from undesirable side-effects triggered by the replay *e.g.* modifying information without notifying the user, the robustness of the replay after doing some changes, *e.g.* replaying $\mathcal{M}$-Traces for already closed sessions, optimizing replay after modifying past obsels and rechecking impact rules after modifying elements values. In SAP BusinessObjects Explorer the same user can open

[5] A demo video of trace replay in SAP BusinessObjects Explorer: https://liris.cnrs.fr/~rzarka/ReplayTraceDemo/

many sessions at the same time. We had to deal with the problem of replaying the $\mathcal{M}$-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 reaching 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 solve loops problems and to know the dependencies between different obsels and elements. This is a future work.

When the $\mathcal{M}$-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. Our system continuously collects and records user's interactions which constitute a potential risk to privacy and security. This problem is shared by all the systems that record rich history traces (web browsers, recommendation systems, *etc.*). However we do notify our users that all their interactions are recorded. Side-effects are another issue have to be dealt with. Indeed, replaying an $\mathcal{M}$-Trace may have unexpected consequences and can damage the system or cause deletion of critical data. In our current implementation, we do not deal with this problem. However, we believe that it could be useful to define the "scope of the replay" *i.e,* the set of elements that we are allowed to modify when replaying past actions. We think that the proposed method described in [Hupp and 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 it easier the annotation of critical obsels. Last, we have to face robustness issues. Indeed, we have to make sure that the $\mathcal{M}$-Trace system is still usable after major changes either on data or on processes of the system. Robustness issues are handled by the Trace Base Management System ($\mathcal{T}$Store), responsible for $\mathcal{M}$-Traces management.

4.7 Conclusion

In this chapter we have described an approach using 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 (*e.g.* client-server applications). For this purpose, we use play-back of interaction traces. Replay 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 is conducted in collaboration with SAP-BO, and the application we used to implement our approach is SAP BusinessObjects 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 chapter is to show how we can replay 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, it is important to address the issues mentioned in the discussion concerning side-effects, robustness, and security. SAP BusinessObjects Explorer is our first test-bed application. However, our approach is generic enough to be applied to other application. Hence, another future work is to experiment with this approach in a different application domain and with a different category of end-user. Our goal is to study how interaction traces, and Trace-Based Reasoning can contribute to help user better understand and use applications. Next chapters will illustrate our contribution with the Wanaclip application. Chapter 5 reports on a similarity measure to compare episodes in modeled traces. In Chapter 6, these similarity measures will be used by a trace-based mechanism to provide contextual recommendations.

Chapter 5

Similarity Measures between Episodes of $\mathcal{M}$-Traces

Contents

Abstract This chapter reports on a similarity measure to compare episodes in modeled traces. A modeled trace is a structured record of observations captured from users' interactions with a computer system. An episode is a sub-part of the modeled trace, representing a particular task performed by

the user. Our method relies on the definition of a similarity measure for comparing elements of episodes, combined with the implementation of the Smith-Waterman algorithm for episode comparison. This algorithm is both accurate in terms of temporal sequencing and tolerant to noise generally found in the traces that we deal with. Our evaluations show that our approach offers quite satisfactory comparison quality and response time. We illustrate its efficiency in the context of an application for video sequence recommendation.

Keywords: Similarity Measures, Modeled Trace, Recommendations, Edit Distance, Human Computer Interaction.

Related publications: [Zarka et al. 2013b]

5.1 Introduction

Recently, there has been a growing interest in the analysis of user activity on the web. Indeed, from the observation of human activity, one can learn a lot about behaviors of users and use these findings for improving the quality of services. This chapter and the following chapter represent our second application domain. The aim is to provide assistance in form of contextual recommendations. We are working on the Wanaclip application. This application allows users to compose video clips from different audio-visual sources. We have built on Wanaclip a recommendation system that guides users in both the selection of videos, and the actions to perform in order to make a quality clip. The recommendation engine is fed by interaction traces left by previous users of the application.

To provide recommendations, we use a Trace-Based Reasoning mechanism. First, the system identifies the current situation of the user (the target problem in Case-Based Reasoning). Then, it searches for similar situations in memory (source cases). Once the source cases are retrieved, the system adapts them to fit the current situation, *e.g.* to make recommendations relevant and appropriate to the current context of the user. The whole process itself is traced. As a consequence, the experience base of the system is incrementally enriched as the system is used. We are aware that the size of the trace base will face scalability problems. To deal with them, forgetting methods are required but they are left for future work.

In the following, source cases are called **episodes**. An episode is a subsequence of **modeled traces** whose structure is often complex. A modeled trace is a structured record of **observed elements**, denoted **obsels**, captured from users' interactions with a computer system. Therefore, when computing similarity between episodes, we have to face new challenges that are not addressed by well-known CBR similarity measures. We have to take into account the fact that episodes are sequences of elements, but more importantly, we have to compare obsels which are complex objects.

In this chapter, we focus on the problem of assessing the similarity between two episodes identified in a modeled trace. For this, we defined a new similarity measure that is based on two main components: a similarity measure used to compare obsels having rich structures, and an algorithm that combines

obsel similarities to compare episodes. The algorithm we propose is an adaptation of the *Smith-Waterman algorithm* [Smith and Waterman 1981]. We implemented our proposal as a web service of $\mathcal{T}$Store, a Trace Base Management System that handles the storage, processing and exploitation of traces (see Chapter 3). The proposal is applied to Wanaclip in order to provide users with contextual recommendations. The recommendation mechanism itself is presented in the next chapter.

The rest of this chapter is organized as follows. Section 5.2 describes related work in the field of similarity measures for sequential data. Similarity Measures between obsels are presented in section 5.3. In section 5.4, we describe the similarity algorithm between the episodes of $\mathcal{M}$-Traces. We report our experiments and performance study in section 5.5, and conclude our work in section 5.6.

5.2 Similarity Measures for Sequential Data

There exist several approaches for comparing strings and defining similarity measures over sequential data. A detailed comparison between three major classes of such similarity measures (*i.e, edit distance*, *bag-of-word models* and *string kernels*) has been studied in [Rieck 2011]. In this section, we present some similarity measures methods and their usages in different domains.

5.2.1 Methods for Defining Similarity Measures

String distances of Hamming [Hamming 1950] and Levenshtein [Levenshtein 1966] are Edit Distances. They come from the domain of telecommunication for detection of erroneous data transmissions. These approaches enable the computation of the minimum edit distance between two strings, which is the minimum number of editing operations (insertion, deletion and substitution) needed to transform one sequence into another. Needleman-Wunsch [Needleman and Wunsch 1970] is a global alignment method which attempts to align every residue in every sequence. Smith-Waterman algorithm [Smith and Waterman 1981] is a local alignment method which is more useful for dissimilar sequences that are suspected to contain regions of similarity within their larger sequence context. This is the method we have chosen for extension in order

to address the similarity problem. More details about this method are given later in this chapter. The seminal concept of describing similarity in terms of edit operations has been extended in a variety of ways, as demonstrated by the Normalized Levenshtein Distance metric [Yujian and Bo 2007].

A different approach to sequence comparison is the vector space (or bag-of-words) model which originates from information retrieval and implements comparison of strings by embedding sequential data in a vector space [Salton et al. 1975]. This concept was extended to *n-grams* for approximate matching [Damashek 1995]. An n-gram is a contiguous sequence of n items from a given sequence of text. The vector space approach has been widely used for analysis of textual documents.

Kernel-based learning is a recent class of similarity measures derived from generative probability models. Various kernels have been developed for sequential data, starting from the original ideas of Watkins [1999] and extending to domain-specific variants, such as string kernels designed for natural language processing [Lodhi et al. 2002] and bio-informatics [Cuturi et al. 2006].

5.2.2 Applications of Similarity Measures to Sequential Data

Similarity measures are at the core of many research efforts in different disciplines. In bio-informatics, different algorithms were developed for sequence alignments to identify regions of similarity in the sequences of DNA, RNA or proteins. For example: Needleman-Wunsch [Needleman and Wunsch 1970] and Smith-Waterman [Smith and Waterman 1981] are edit distance algorithms, where FAST [Lipman and Pearson 1985] and BLAST [Altschul et al. 1990] are heuristic algorithms. The ease of mapping strings to vectors and subsequent application of vectorial similarity measures also influenced other fields of computer science like computer security [Rieck and Laskov 2007]. Measuring the similarity between documents and queries has been extensively studied in information retrieval [Metzler et al. 2007].

5.2.3 Similarity Measures in the Case-Based Reasoning Field

Similarity measures applied to complex sequences are also developed in the CBR field. The Episode-Based Reasoning framework provides mechanisms to represent, retrieve and learn temporal episodes [Sànchez-Marrè et al. 2005]. In the CR2N system [Adeyanju et al. 2009], the similarity assumption is used to determine reusable textual constructs. A confidence measure for a workflow adaptation based on introspection of the case base has been introduced in [Minor et al. 2012b]. The CeBeTA system [Valls and Ontañón 2012] combines a sentence similarity metric based on edit distance with a reuse approach based on text-transformation routines, in order to generate solutions for the text modification problem. CBR-WIMS framework [Kapetanakis and Petridis 2014, Kapetanakis et al. 2011] employs similarity measures to retrieve workflow execution cases similar to a given target case. [Mille et al. 1999] proposed a conceptual similarity measure for the episode elements. The objective of this similarity measure is to select the episode that is similar in terms of static context of supervision and has similar story of events. [Maarten Grachten 2004] developed a CBR system for expressive music processing that proposes a new melodic similarity measure. A proper case structure and a new distance measure have been proposed in [Montani and Leonardi 2012], that are exploited to retrieve traces similar to the current one. They use a graph edit distance definition by focusing on traces of executions. For them, it was guaranteed that the actions in the traces always matched reality. However, our approach is based on similarity measures between obsels by comparing their contents, *i.e,* their time-stamps, users, types, and values.

Most of these approaches enable comparison of homogeneous elements (such as characters, symbols or events). In this chapter, we focus on modeled traces, which are sequential records of complex elements. As the elements are complex, they cannot be directly compared. Therefore, we propose an approach inspired by the Smith-Waterman algorithm [Smith and Waterman 1981], that takes into account the richness of the compared elements. We chose this algorithm because, by combining it with our similarity measures between obsels, it has all the properties expected for the comparison of episodes, namely:

- Processing of sequential data,
- Tolerance to variations in representation,
- Noise resistance,
- High degree of customization,
- Satisfactory response time.

[Obweger et al. 2010] has developed a generic method for similarity searching in sequences of time-stamped complex events. This method is based on the assumption that similarity is computed through finding the deviations between the pattern sequence (input sequence) and candidate sequences (stored sequence). [Gundersen 2012] has also studied the problem of comparing the similarity between two sequences of complex events. They reduced the time complexity claimed by [Obweger et al. 2010] by transforming information about the sequence, such as the order of the elements, into a vector, and then comparing the vectors instead of the sequences. This approach compares all the events in the two sequences to find the global maximum similarity score, which is very similar to our approach where events corresponds to the obsels. This method is based on the assumption that events closer to a point of interest, like the current time, are more important than those further away. However, in our case, we are looking for local similar parts in the traces and not only the most recent obsels. Our approach takes into account the hypothesis that episodes may contain unnecessary obsels. These obsels are generated when unnecessary actions are performed (the user is lost, he is doing several things at the same time, *etc.*). These obsels are considered as "gaps", thus they should be ignored or "penalized" with a small factor.

5.3 Similarity Measures Between Obsels

In order to define a similarity between obsels, we need to define several local similarity measures for the obsel-types, users, attributes and time-stamps, which are the significant components of an obsel. These measures rely on the representation of $\mathcal{M}$-Traces described in Chapter 3.

Definition 6 (Obsel similarity measure)**.** *We define* $sim_{obs}(o_1, o_2)$ *as a similarity measure between the obsels:*

$o_1 = \{c_1, A_{o_1}, u_1, s_{t1}, e_{t1}\}$ *and* $o_2 = \{c_2, A_{o_2}, u_2, s_{t2}, e_{t2}\}$ *as:*

$$sim_{obs}(o_1, o_2) = \alpha \times sim_{obstype}(c_1, c_2) + \beta \times sim_{obsattr}(A_{o_1}, A_{o_2})$$

$$+\gamma \times sim_{obsuser}(u_1, u_2) + \delta \times sim_{obstime}(s_{t1}, e_{t1}, s_{t2}, e_{t2}) \quad (5.1)$$

where:

- $sim_{obstype}(c_1, c_2)$*: is the obsel-type similarity,*
- $sim_{obsattr}(A_{o_1}, A_{o_2})$*: is the obsel-attribute similarity,*
- $sim_{obsuser}(u_1, u_2)$*: is the obsel-user similarity,*
- $sim_{obstime}(s_{t1}, e_{t1}, s_{t2}, e_{t2})$*: is the obsel time-stamp similarity,*
- α*,* β*,* γ*,* δ*: are weights, with* $(\alpha + \beta + \gamma + \delta) = 1$ *to keep this measure normalized.*

The similarity measure between obsels $sim_{obs}(o_1, o_2)$ is a normalized value $\in [0, 1]$ since all its sub measures $(sim_{obstype}, sim_{obsattr}, sim_{obsuser}, sim_{obstime})$ produce normalized values and the sum of the weights $(\alpha + \beta + \gamma + \delta) = 1$. There are the application experts who define these values.

5.3.1 Obsel-Type Similarity $sim_{obstype}(c_1, c_2)$

Some obsel-types can be compared and some cannot. Thus, we propose to define a substitution matrix over obsel-types $S_{obstype}(|C| \times |C|)$ to express to which extent two types can be compared (not comparable, comparable, partially comparable). In bio-informatics and evolutionary biology, a substitution matrix describes the rate at which one character in a sequence changes to other character states over time. We consider that the substitution matrix has normalized values between 0 and 1. The simplest possible substitution matrix would be one in which each obsel-type in $(c \in C)$ is considered maximally similar to itself, but not similar to any other obsel-type. This matrix would look like:

$$S_{obstype} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

Definition 7 (Obsel-type similarity measure)**.** *For two obsel-types $c_1, c_2 \in C$, their similarity is defined as:*

$$sim_{obstype}(c_1, c_2) = S_{obstype}(c_1, c_2) \in [0, 1] \tag{5.2}$$

Where:

- *C is the set of all obsel-types*
- *S is a substitution matrix $|C| \times |C|$*
- *Both of the rows and columns of S are obsel-types. Each cell in this matrix has a normalized value representing the distance between a pair of obsel-types.*

The substitution matrix depends on the obsels defined on the application. By default, it is filled manually by an expert.

5.3.2 Attribute Similarity $sim_{obsattr}(A_{o_1}, A_{o_2})$

When two obsels are of the same type, it is easy to compare them. However, when two obsels are of different types, a more complex comparison method has to be defined.

Example 15. In Wanaclip, both obsel-types "playVideo" and "infoVideo" contain the meta-data of the video. To compare their attributes, we need to know the common attributes between these obsel-types. ◊

Definition 8 (Common attributes)**.** *We define the set of pairs of common type attributes belonging to the attribute sets A_{o_1} and A_{o_2} as:*

$$cA(A_{o_1}, A_{o_2}) = \{(ca_{1,i}, ca_{2,i})\}_{i=1,|cA(A_{o_1}, A_{o_2})|} \tag{5.3}$$

where o_1, o_2 are two obsels that can have different obsel-types. An obsel can not have two attributes of the same type unless they are not in the same hierarchy level.

Definition 9 (Attribute similarity measure)**.** *For two obsels $o_1 \in O$ and $o_2 \in O$, we define the similarity between the two sets of attributes of o_1 and o_2 ($sim_{obsattr}$) as:*

$$sim_{obsattr}(A_{o_1}, A_{o_2}) = \sum_{i=1}^{|cA(A_{o_1}, A_{o_2})|} w_{importance}(i) \times sim_{attr}(ca_{1,i}, ca_{2,i}) \quad (5.4)$$

where:

- *$sim_{attr}(ca_{1,i}, ca_{2,i}) \in [0, 1]$. For each attribute type, there has to be at least one similarity function provided to compare their values.*
- *$\sum_{i=1}^{|cA(A_{o_1}, A_{o_2})|} w_{importance}(i) = 1$. The different weights are defined separately and possibly modified by the user or the system. It ensures that the similarity measure between two sets of attributes is normalized.*

5.3.3 Obsel User Similarity $sim_{obsuser}(u_1, u_2)$

When comparing $\mathcal{M}$-Traces, we prefer to give more importance to the traces which belong to the same user or to users members of the same group. Social media give a lot of importance to this measure. Indeed users having similar profiles and interests are more likely to do similar activities, and therefore, to produce similar $\mathcal{M}$-Traces [Kietzmann et al. 2011].

Definition 10 (User similarity measure)**.** *The similarity measure $sim_{obsuser}$ between two users u_1, u_2 is defined as:*

$$sim_{obsuser}(u_1, u_2) = \begin{cases} 1 & u_1 = u_2 \\ \lambda \times sim_{profile}(u_1, u_2) + \mu \times sim_{groups}(u_1, u_2) & u_1 \neq u_2 \end{cases} \quad (5.5)$$

where:

- *$sim_{profile}(u_1, u_2)$ is the similarity between the profiles of the users according to their interests and activities. We will consider that this value is between 0 and 1.*

- *$sim_{groups}(u_1, u_2)$ is the Jaccard similarity coefficient [Lipkus 1999] between user groups. By considering G_1 as the groups that the user u_1 belongs to, G_2 for the user u_2, then we define the similarity measure between these groups as:*

$$sim_{groups}(u_1, u_2) = \frac{|G_1 \cap G_2|}{|G_1 \cup G_2|} \tag{5.6}$$

- *λ, μ: are weights, where $\lambda + \mu = 1$.*

5.3.4 Obsel Time-stamp Similarity $sim_{obstime}(o_1, o_2)$

According to the $\mathcal{M}$-Trace model, each obsel has two time-stamps (begin and end). At the moment, similarity is only defined with regard to the duration of the obsel. However, this measure could easily be extended (for example, by using Allen's interval relations [Allen 1983]).

Definition 11 (Time-stamp similarity measure)**.** *For two obsels o_1, o_2 having time-stamps (s_{t1}, e_{t1}) for o_1 and (s_{t2}, e_{t2}) for o_2, we define the obsel time-stamps similarity measure ($sim_{obstime}$) as:*

$$sim_{obstime}(o_1, o_2) = \begin{cases} 1 & |(e_{t1} - s_{t1}) - (e_{t2} - s_{t2})| = 0 \\ \frac{min(e_{t1}-s_{t1}, e_{t2}-s_{t2})}{max(e_{t1}-s_{t1}, e_{t2}-s_{t2})} & |(e_{t1} - s_{t1}) - (e_{t2} - s_{t2})| \neq 0 \end{cases} \tag{5.7}$$

where $(e_{t1} - s_{t1})$ is the duration of o_1. We calculate the ratio between the minimum and maximum duration of o_1, o_2 to get a normalized value. For example, if $t_1(2, 9), t_2(4, 14)$ then $sim_{obstime}(o_1, o_2) = 0.7$

5.4 Similarity Between Episodes

In the previous section, we defined a similarity measure for comparing two obsels. For that, we provided several similarity measures for obsel-types, users, time-stamps, and attributes. In this section, we present an approach for comparing episodes based on the minimum edit distance. This approach makes use of the similarity measure introduced earlier. This approach relies on a dynamic programming algorithm that solves the smaller problems optimally and

uses the sub-problem solutions to construct an optimal solution for the original problem. The Smith-Waterman algorithm [Smith and Waterman 1981] is used to compare the episodes. The algorithm is extended by introducing similarity measures between obsels.

5.4.1 The $\mathcal{M}$-Trace Smith-Waterman algorithm

The Smith-Waterman algorithm is a well-known algorithm for performing local sequence alignment. Instead of looking at each sequence in its entirety, it compares segments of all possible lengths. We adapted the algorithm using the similarity measure described in the previous section to make it more accurate in terms of temporal sequencing and tolerant to noise generally found in the traces that we deal with. For that, a substitution matrix is built using the similarity measures between obsels and a gap function to determine the reduced score according to the number of indels (insertions/deletions) between obsels. It represents the cost of replacing an obsel by another one. This method not only evaluates the similarity between two episodes but also the transformations needed to go from one episode to the other (alignment), which is particularly useful for producing the recommendations.

Definition 12 (Gap penalty function)**.** *Gap penalty function determines the reduced score according to the number of indels (insertions/deletions) in the sequence alignment. It determines the lost value when replacing an obsel by a gap. For the moment, for any obsel o, we consider that $gap(o) = -1$.*

Definition 13 (Substitution Matrix)**.** *For two episodes A, B, $\forall a \in A \cup \{`-'\}, b \in B \cup \{`-'\}$, we define the substitution matrix of obsels (S) as:*

$$S(a,b) = \begin{cases} gap(a) & b = `-' \\ gap(b) & a = `-' \\ (sim_{obs}(a,b) - 0.5) \times 2 & a \in A \wedge b \in B \end{cases} \tag{5.8}$$

where:

- *$gap(a) \in [-1, 0]$ is the gap penalty function for an obsel $a \in A$*
- *'$-$' is the gap-scoring scheme*

- $sim_{obs}(a, b) \in [0, 1]$ *is the obsel similarity measure between* a, b
- $(sim_{obs}(a, b) - 0.5) \times 2$ *means converting the values from* $[0, 1]$ *to* $[-1, +1]$

The $\mathcal{M}$-Trace Smith-Waterman algorithm 2 compares two episodes A, B. It computes the episode similarity measure $sim_{episode}(A, B)$ and the local alignments L_1, L_2 of A, B. The similarity-score matrix (H) between a suffix of $A[1 \cdots i]$ and a suffix of $B[1 \cdots j]$ with a size of $(|A| + 1 \times |B| + 1)$, is built during the algorithm. Firstly, the first row and the first column are initialized to zero. During the algorithm, a gap penalty is computed for each obsel in $(A \cup B)$. As shown in Figure 5.1, the algorithm iterates over each cell in H from the top-left cell to the bottom-right cell (see lines 4 to 8). At each time, it computes the similarity measure between the two obsels A_i, B_j according to Definition 6. The score of the current cell $H(i, j)$ is the highest score of three other scores in the similarity-score matrix (see line 7): the up-left neighbor cell $H(i - 1, j - 1)$ added to the similarity of obsels A_i, B_j, the left neighbor cell added to the gap penalty and the up neighbor cell added to the gap penalty. We keep pointers to the highest cells to keep directions of movement used to construct the matrix. This will be used in the trace-back step to obtain the best alignment. If we want the best local alignment, we

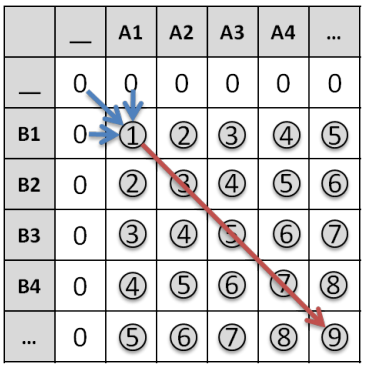

Figure 5.1: The iterations of the Smith-Waterman algorithm.

find $H_{opt} = max_{i,j} H(i, j)$ and we trace-back (follow the pointers). The cell of highest score can be anywhere in the array. Otherwise, if we want all local alignments, we find $H(i, j) > threshold$ for all i, j and we trace-back. To obtain the optimum local alignment, we start with the highest value in the matrix $H(i, j)$ (see line 12 of the $\mathcal{M}$-Trace Smith-Waterman algorithm 2).

Algorithm 2 The $\mathcal{M}$-Trace Smith-Waterman algorithm.

Input: A, B are episodes to compare

Output: The similarity measure $sim_{episode}(A, B)$ and the local alignments L_1, L_2

$H_{Score}(i, 0) \leftarrow 0 \quad \forall\ 0 \leq i \leq |A|$
$H_{Score}(0, j) \leftarrow 0 \quad \forall\ 0 \leq j \leq |B|$
▷ Initialization of the similarity-score matrix

$S(A_i, -) \leftarrow gap(A_i) \quad \forall\ 1 \leq i \leq |A|$
$S(-, B_j) \leftarrow gap(B_j) \quad \forall\ 1 \leq j \leq |B|$
▷ Compute gap penalties of obsels

for $i = 1 \rightarrow |A|$ **do**
  for $j = 1 \rightarrow |B|$ **do**
    $S(A_i, B_j) \leftarrow (sim_{obs}(A_i, B_j) - 0.5) \times 2$ ▷ Fill the substitution matrix
    $H(i, j) \leftarrow max$

Score	**Pointer**
0	None
$H(i-1, j-1) + S(A_i, B_j)$	Substitution
$H(i-1, j) + S(A_i, -)$	Deletion
$H(i, j-1) + S(-, B_j)$	Insertion

  end for
end for

$L_1, L_2 \leftarrow [\ ]$
$i, j = max(H_{i,j}(i, j))$
while $(i > 0) \wedge (j > 0) \wedge (H_{pointer}(i, j) \neq$ "$None$") **do** ▷ Trace-back
  if $(H_{pointer}(i, j) =$ "$Substitution$") **then**
    $push(L_1, A_i)$
    $push(L_2, B_j)$
    $i \leftarrow i - 1$
    $j \leftarrow j - 1$
  else if $(H_{pointer}(i, j) =$ "$Insertion$") **then**
    $push(L_1, -)$
    $push(L_2, B_j)$
    $j \leftarrow j - 1$
  else if $(H_{pointer}(i, j) =$ "$Deletion$") **then**
    $push(L_1, A_i)$
    $push(L_2, -)$
    $i \leftarrow i - 1$
  end if
end while

$sim_{episode}(A, B) \leftarrow max(H_{score}(i, j)$
return $L_1, L_2, sim_{episode}(A, B)$

Then, we go backwards to one of the positions $H(i-1,j)$, $H(i,j-1)$, and $H(i-1,j-1)$ depending on the direction of the movement used to construct the matrix (see lines 13 to 28). We repeat until we reach a matrix cell with zero value, or the cell $H(0,0)$. Once we have finished, we reconstruct the alignment as follows: starting with the last value, we reach $H(i,j)$ using the previously calculated path. A diagonal jump implies a replacement. A top-down jump implies a deletion. A left-right jump implies an insertion. The trace-back step stops when the score is 0 or when we reach the first row or column. The algorithm returns the maximum similarity score H_{opt} and the alignments L_1, L_2 found. Sometimes, we have different possible alignments especially when we have more than one maximum cell.

5.4.2 Illustrative Example

The following example comes from the web application Wanaclip. Let us consider that we have an episode $\mathcal{E}_1 = (S_8 P_1 A_1 A_2 X_1 A_3 G_7)$. It represents a user searching for videos about "Lyon" (S_8: S is the obsel-type, and 8 is the value). Then, he starts playing (P_1) and adding videos (A_1 and A_2) to his selection. After that, he adds the text X_1 to the first selected video and adds another video A_3. Finally, he generates the clip G_7. This episode is collected and stored in $\mathcal{T}$Store. Later, two users are using Wanaclip. The first one selects the tag "Lyon", plays and adds a video, then adds text to it. This episode is denoted $\mathcal{E}_2 = (T_8 P_1 A_1 X_1)$. The second user is searching for a different keyword "Paris". He plays and adds other videos. His episode is denoted $\mathcal{E}_3 = (S_9 P_3 A_3 A_5)$. Using the stored $\mathcal{M}$-Traces, an assistant recommends the next actions to do. For that, the assistant compares the current episodes with the stored ones and recommends the most similar ones.

Figure 5.2 shows how to compare the stored episode $\mathcal{E}_1$ with the current episodes $\mathcal{E}_2$ and $\mathcal{E}_3$ using Smith-Waterman algorithm without using the similarity measures between obsels (the default values: gap=-1, matching = 1, mismatch=-1). The results show that $\mathcal{E}_3$ is more similar to $\mathcal{E}_1$ than $\mathcal{E}_2$ ($sim_{episode}(\mathcal{E}_1, \mathcal{E}_2) = 2 < sim_{episode}(\mathcal{E}_1, \mathcal{E}_3) = 4$) (remember that the similarity measure between episodes is not normalized). However, in our context, $\mathcal{E}_2$ is more likely to be similar to $\mathcal{E}_1$ since both of them represent users trying to add similar videos to generate a clip about Lyon.

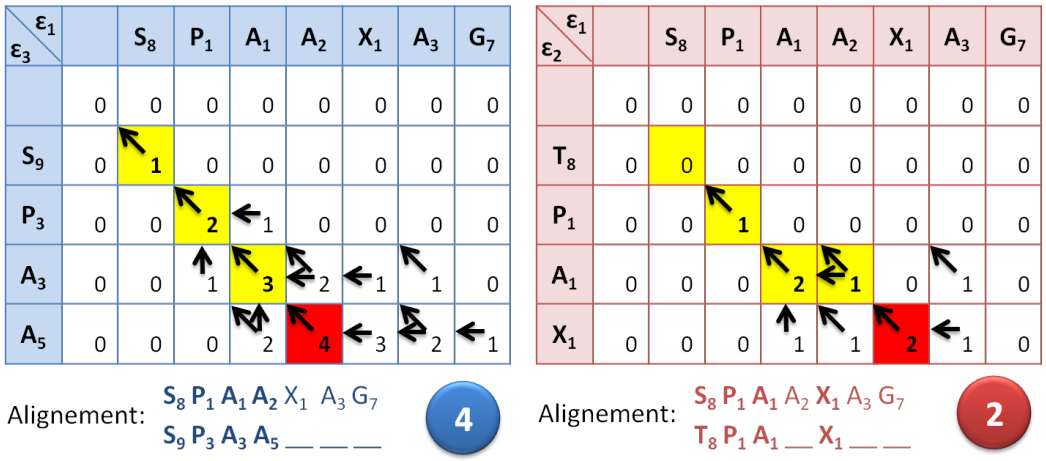

Figure 5.2: An example of the $\mathcal{M}$-Trace Smith-Waterman algorithm to compute $sim_{episode}(\mathcal{E}_1, \mathcal{E}_2)$, $sim_{episode}(\mathcal{E}_1, \mathcal{E}_3)$.

For that, we build a substitution matrix defined using the similarity measures between obsels, as the following: we ignore the similarity between users and time-stamps and we only give weights to obsel-types and attributes

$sim_{obs}(o_1, o_2) = 0.5 \times sim_{obstype}(c_1, c_2) + 0.5 \times sim_{obsattr}(A_1, A_2)$.

We also consider that the similarity between the obsel-types (search and selectTag) is $sim_{obstype}(S, T) = 0.5$. According to that, the substitution matrices for $\mathcal{E}_1, \mathcal{E}_2$ and $\mathcal{E}_1, \mathcal{E}_3$ are computed as shown in Figure 5.3. For example, to compute the substitution value for the obsels S_8, T_8 we compute the similarity measure between them $sim_{obs}(S_8, T_8) = 0.5 \times 0.5 + 0.5 \times 1 = 0.75$. Then we convert the value to be in the range $[-1, +1]$ as $S(S_8, T_8) = (sim_{obs}(S_8, T_8) - 0.5) \times 2 = 0.5$.

$\mathcal{E}_1$ / $\mathcal{E}_3$	—	S_8	P_1	A_1	A_2	X_1	A_3	G_7
—		-1	-1	-1	-1	-1	-1	-1
S_9	-1	0	-1	-1	-1	-1	-1	-1
P_3	-1	-1	0	-1	-1	-1	0	-1
A_3	-1	-1	-1	0	0	-1	1	-1
A_5	-1	-1	-1	0	0	-1	0	-1

$\mathcal{E}_1$ / $\mathcal{E}_2$	—	S_8	P_1	A_1	A_2	X_1	A_3	G_7
—		-1	-1	-1	-1	-1	-1	-1
T_8	-1	0.5	-1	-1	-1	-1	-1	-1
P_1	-1	-1	1	0	-1	0	-1	-1
A_1	-1	-1	0	1	0	0	0	-1
X_1	-1	-1	0	0	-1	1	-1	-1

Figure 5.3: The substitution matrices for $\mathcal{E}_1, \mathcal{E}_2$ and $\mathcal{E}_1, \mathcal{E}_3$.

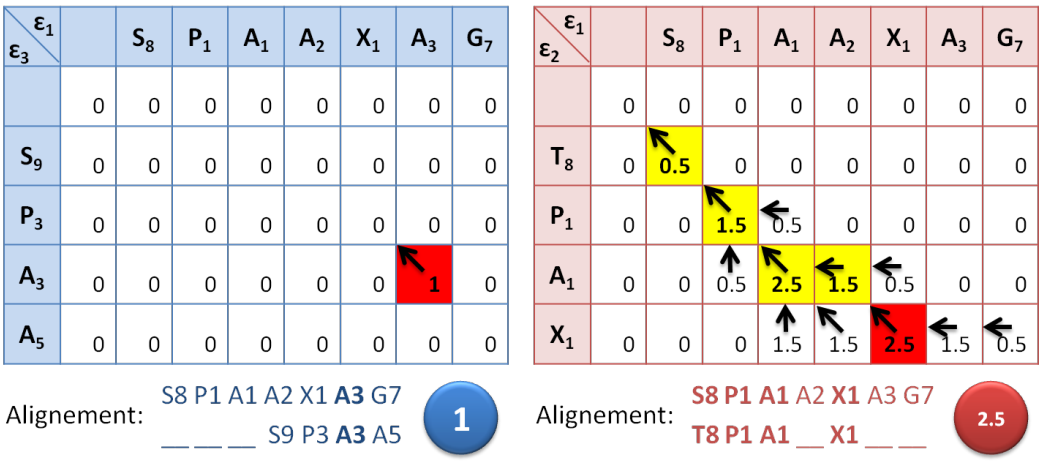

Figure 5.4: An example of the $\mathcal{M}$-Trace Smith-Waterman algorithm to compute $sim_{episode}(\mathcal{E}_1, \mathcal{E}_2)$, $sim_{episode}(\mathcal{E}_1, \mathcal{E}_3)$ after applying obsels similarity measures.

Figure 5.4 shows the $\mathcal{M}$-Trace Smith-Waterman algorithm after applying the substitution matrix based on the similarity measures between obsels. We see that $\mathcal{E}_2$ is more similar to $\mathcal{E}_1$ than $\mathcal{E}_3$ ($sim_{episode}(\mathcal{E}_1, \mathcal{E}_2) = 2.5 > sim_{episode}(\mathcal{E}_1, \mathcal{E}_3) = 1$) which is more reasonable and accurate than the past results. Using the alignments of the episodes, we can recommend the next actions. A detailed description of the recommendation mechanism is presented in the next chapter.

5.5 Implementation and Evaluation

We have implemented our local similarity measures and the $\mathcal{M}$-Trace Smith-Waterman algorithm as a PHP web service in the $\mathcal{T}$Store framework. Any client connected to $\mathcal{T}$Store can execute the service to compare obsels, episodes of $\mathcal{M}$-Traces or other types of sequences. These implementations allow us to customize the similarity measures by specifying all the weights used in the similarity measures between obsels defined in Definition 6. In addition, we can customize the $\mathcal{M}$-Trace Smith-Waterman algorithm by changing the substitution matrix. We also customize the gap penalty function and change the range of values of the similarity scores between obsels. We can extend it to be in $[-\infty, +\infty]$. For example, decreasing the score of similar obsels

will make the episodes more likely to be similar by giving more score to the matched obsels. The same thing applies for mismatching obsels. If we give a negative value for the substitution between two obsels, it means that a mismatch between them will decrease the probability of matching the episodes.

The $\mathcal{M}$-Trace Smith-Waterman algorithm is fairly demanding in time: to compare two episodes $\mathcal{E}_1, \mathcal{E}_2$, $O(|\mathcal{E}_1||\mathcal{E}_2|)$ time is required. In addition, improvements can easily be made. Indeed, through observation of the similarity matrix calculation process in Figure 5.1, we found that we can optimize the calculation time of the similarity scores. For each iteration, every element on an anti-diagonal line marked with the same number could be calculated simultaneously, taking into consideration the elements that could be calculated at the same time. For example, in the first cycle, only one element marked as (1) could be calculated. In the second cycle, two elements marked as (2) could be calculated. In the third cycle, three elements marked as (3) could be calculated, *etc.*

We have implemented a collection process in Wanaclip. The $\mathcal{M}$-Trace handler captures users' actions, and stores them as obsels in $\mathcal{T}$Store. In order to illustrate the efficiency of the proposed algorithm, we conducted run-time experiments on episodes regenerated randomly. We performed several experiments changing the weights used in the similarity measures between obsels and the similarity values in the substitution matrix. We compared the run-time of the $\mathcal{M}$-Trace Smith-Waterman algorithm over 100 episodes of different lengths from 2 to 20 obsels. We used about 50 obsel-types (of which 10 are frequent). Each obsel has on average 3 attributes. The procedure was repeated 10 times for various weights, and the run-time was averaged over all runs. As shown in Figure 5.5, the run-time of the algorithm shows a logarithmic growth. It is composed of the time required for the iterations of comparison and the time of the alignment part. We do not need to compute the alignments but it is important for the recommendations. In the chart on the right side, we see that about the half of the similarity scores of the comparison test we performed were between 0 and 5 points. Where 25% of the episodes have a similarity score of 0, thus they are eliminated before the retrieval process which facilitate the recommendations.

The most important things for the execution time are: the obsel-types, the

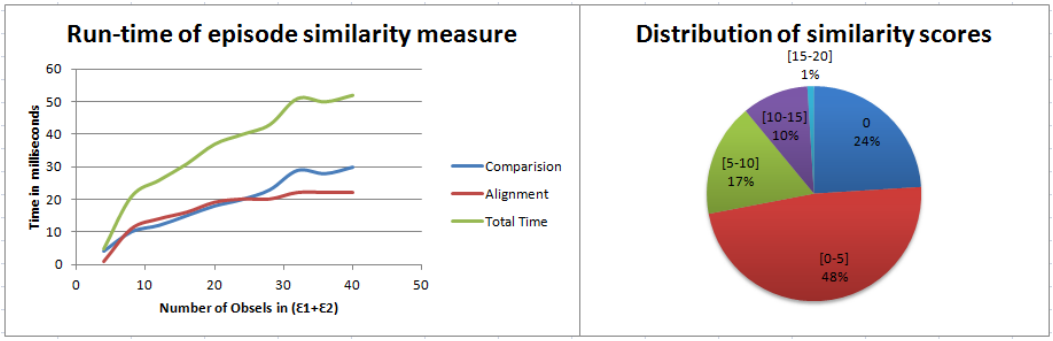

Figure 5.5: Evaluation of the $\mathcal{M}$-Trace Smith-Waterman algorithm.

length of the compared episodes (the number of obsels in each episode) and the number of common attributes between compared obsels. We have noticed also that the alignments of episodes are sometimes better when expanding the range of the substitution matrix.

5.6 Conclusion

In this chapter, we described a similarity measure for comparing episodes belonging to $\mathcal{M}$-Trace. Comparing such episodes is a difficult task, not only because they are sequences, but also because they are composed of obsels (observed elements) which are complex objects and therefore difficult to compare. Our method is based on two major components: a similarity measure for comparing obsels, and an adaptation of the Smith-Waterman algorithm, using the similarity measure described above, to determine the similarity between two episodes. This method not only evaluates the similarity between two episodes but also the transformations needed to go from one episode to the other (alignment), which is particularly useful given the recommendations.

Evaluations of the method showed that the algorithm is not time-consuming and that it has all the properties that we expected: ability to compare complex objects, calculate the similarity score, many customization options, *etc.* We implemented this measure in the platform $\mathcal{T}$Store (see Chapter 3). The experiments are carried out with the measure using the Wanaclip application. in order to provide Wanaclip users with contextualized recommendations bases on traces of previous users. More evaluations with real users are conducted in the next chapter while evaluating the recommendation approach.

A future work would be to work on the optimization of the $\mathcal{M}$-Trace Smith-Waterman algorithm and its application to different domains. In addition, using users' feedback for the computation of the similarity measures will enhance them. A long term goal is to develop sequence similarity measures that are faster than the current solution, and enhancing the accuracy of the similarity measurement. In the next chapter, we present a contextual recommendation mechanism that uses of the similarity measures described in this chapter. We perform a thorough evaluation of the quality of the recommendations in order to better tune the similarity measures.

Chapter 6

$\mathcal{M}$-Trace-Based Contextual Recommendations

Contents

Abstract Nowadays, many recent applications retain traces of their usage by collecting user information. These traces help to understand users' behaviors and their activities thus reflect the context. This chapter describes how interaction traces allow building contextual recommendations using a Trace-Based Reasoning approach. Trace-Based Reasoning is an artificial intelligence paradigm that draws its inspiration from Case-Based Reasoning. In TBR, modeled traces act as the main knowledge container. The application considered in this chapter ,Wanaclip, is an interactive online video clip composition application. We added a recommendation system that guides users in both video selection, and the actions to perform in order to make a nice clip. The recommendation engine is fed by interaction traces left by previous users of the application and stored in $\mathcal{T}$Store, a Trace Base Management System that handles the storage, processing and exploitation of traces. This approach uses similarity measures for finding and comparing episodes of traces. We validate our approach by an evaluation protocol to measure the performance, quality and users' satisfaction. For that, we propose a new metric called acceptance rate and we ask Wanaclip users to answer to a survey. Our evaluations show that this approach offers satisfactory recommendations and response time.

Keywords: Contextual Recommendation, Trace-Based Reasoning, Human Computer Interaction, User Assistance, Interaction Trace, Modeled Trace.

Related publications: [Zarka et al. 2012]

6.1 Introduction

The Web constitutes a worldwide interactive information system connecting more and more documents and people for a constantly growing number of human activities. Choosing the appropriate information to perform a given task is a complex process. Recommendation systems can save us time and efforts in this context. When performing an activity, the context is of major importance. However, most recommendation systems deal with applications having only two types of entities: users and items. Usually, these systems do not put the entities into a context when providing recommendations. For example, using the temporal context, a travel recommendation system should provide a vacation recommendation in the winter that can be very different from the one in the summer. Contextual recommendations are particularly important for domains like: e-commerce, e-learning, e-health, media applications, *etc.* In this chapter, we focus on recommendations in a semi automatic video clip systems.

We are contributing to a web application called Wanaclip[1]. The aim of this application is to generate a video clip by selecting and merging different media: photos, videos, music and sounds. This is similar to what MediaMixer[2] do. However, MediaMixer involves in the fragmentation of media sources, so that consumers can access and re-use only the parts they are interested in. Wanaclip allows users to import and annotate their own media in the media base. User content is added to the public content of the application. Users can organize media, adjust the duration, customize content by adding comments, choose their fonts and colors and adjust the display time. In Wanaclip, users enter keywords, the system searches video sequences (rushes) annotated with these keywords and lets the users drag them into a timeline in order to compose a video clip. Several search cycles can refer to the same result clip. Wanaclip offers publishing and sharing functionalities. It is plugged with social networks. In order to collect interaction traces, we have instrumented the Wanaclip application with a tracing and trace based recommendation functionality. In Figure 6.5), the users' trace is displayed on the top of the screen.

It is difficult to find the right video rush in the huge database. We show

[1]Wanaclip: http://www.wanaclip.eu
[2]MediaMixer: http://www.mediamixer.eu

how context-based recommendations can provide efficient help to users. For this, we rely on recorded interaction traces. We claim that interaction traces can be used to record user experiences over the web. Traces can then be reused by a reasoning process for different purposes (replay, mining, *etc.*). In order to provide relevant recommendations, we base ourselves on the notion of episode. An episode is a temporal pattern composed of an ordered set of events corresponding to a specific task. Given this definition, we use the defined similarity measures in the previous chapter to allow finding and comparing episodes. We added to Wanaclip a recommendation system that guides users in both the selection of videos, and the actions to perform in order to make a quality clip. The recommendation engine is fed by interaction traces left by previous users of the application and stored in $\mathcal{T}$Store (See Chapter 3). $\mathcal{T}$Store is a TBMS that handles the storage, processing and exploitation of traces. The whole system is called a Trace-Based Assistant for the Wanaclip application.

Trace-Based Reasoning draws its inspiration from Case-Based Reasoning and reuses past experiences to solve new problems. The main difference is that, in TBR, the main knowledge container is a set of traces [Cordier et al. 2013]. TBR is particularly suitable for dynamic Web applications. $\mathcal{M}$-Traces can be reused in different ways: task automation, replay, exploration, modification, user assistance, recommendation, stream mining, and visualization. In Chapter 4, we defined an approach to support replay of user's traces to return to a particular state of an application by exploiting traces. Here, we focus on recommendation mechanisms.

We have asked real users to test the Wanaclip application to evaluate the recommendation approach. The evaluation protocol is composed of three parts about performance, accuracy and user satisfaction. We define a specific accuracy metric called "acceptance rate" to measure the percentage of recommendations that are used by the users. In addition, to determine users' satisfaction, we have asked Wanaclip users to answer a survey. We analyze the results of this survey to confirm that the recommendations are contextual and useful.

The remaining of this chapter is organized as follows. Section 6.2 presents a state of the art about recommendations systems. In section 6.3, we present

our contextual recommendation mechanism followed by an example in section 6.4. The evaluation protocol and results are discussed in section 6.5. Finally, we conclude in section 6.6.

6.2 Recommendation Systems

Recommendation systems are a particular form of information filtering systems that provide assistance to users by helping them discover items they might not have found by themselves. They are software tools and techniques providing suggestions for items to be of use to a user [Burke 2007, Resnick and Varian 1997]. These systems exploit past behaviors and user similarities to generate a list of information items that is personally tailored to end-user's preferences. The suggestions relate to various decision-making processes, such as what items to buy, what music to listen to, what videos to watch, or what online news to read [Ricci et al. 2011]. The role of a recommendation system within an information system can be quite diverse. We are interested in its role for helping other users and sharing tasks and items with them. [Burke 2007] defines eleven popular tasks that a recommendation system can assist.

- *Find some good items:* Recommend to a user some items as a ranked list along with predictions of how much the user would like them (*e.g.* on a one to five-star scale).
- *Find all good items:* Recommend all the items that can satisfy some user needs.
- *Annotation in context:* Given an existing context, *e.g.* a list of items, emphasize some of them depending on the user's long-term preferences. For example, a TV recommendation system might annotate which TV shows are worth watching.
- *Recommend a sequence:* Recommend a sequence of items that is pleasing as a whole. *e.g.* recommending a TV series.
- *Recommend a bundle:* Suggest a group of items that fits well together. For example, recommending various attractions, destinations together as a package for a travel plan.

- *Just browsing:* The task of the recommender is to help the user to browse the items that are more likely to fall within the scope of the user's interests for that specific browsing session [Brusilovsky 1996].
- *Find credible recommender:* offering specific functions to let the users test the behavior of the recommender especially for users who do not trust recommendation systems.
- *Improve the profile:* Users need to provide information to the recommendation system about their interests. This helps to make the recommendations personalized.
- *Express self:* Some users like to contribute with their ratings and express their opinions and beliefs.
- *Help others:* Some users believe that the community benefits from their contribution, thus they rate the items.
- *Influence others:* There are users whose main goal is to explicitly influence other users into purchasing particular products. Recommendation systems have to avoid malicious users from promoting or penalizing items.

6.2.1 Approaches of Recommendation Systems

Recommendation systems have different approaches. Recommendation systems handbook [Ricci et al. 2011] classifies these systems into six broad categories.

6.2.1.1 Content-Based Systems

Content-based systems utilizes item characteristics and a profile of the users' interests [Pazzani and Billsus 2007]. The system learns to recommend items that are similar to the ones that the user liked in the past. The similarity of items is calculated based on the features associated with the compared items [Ricci et al. 2011]. For instance, if a Netflix user has watched many action movies, then the system can learn to recommend to this user other movies from this genre [Rajaraman Anand and Ullman. 2011]. This approach needs

little information but it is limited to the original item and does not take in account user's behavior.

6.2.1.2 Collaborative Filtering

Collaborative filtering builds models according to a user's past behavior as well as similar decisions made by other users [Su and Khoshgoftaar 2009]. The items recommended to a user are those preferred by similar users. This is why [Schafer et al. 2001] refers to collaborative filtering as "people-to-people correlation". According to [Ricci et al. 2011], Collaborative filtering is considered to be the most popular and widely implemented technique in recommendation systems. This approach requires a large amount of information on a user in order to make accurate recommendations. Facebook, MySpace, LinkedIn, and other social networks use collaborative filtering to recommend new friends, groups, and other social connections. When viewing a product on Amazon.com, the store will recommend additional items based on a matrix of what other shoppers bought along with the currently-selected item [Linden et al. 2001].

6.2.1.3 Demographic Recommendation Systems

It provides recommendations based on a demographic profile of the user. The assumption is that different recommendations should be generated for different demographic niches. For example, users are dispatched to particular Web sites based on their language or country. Or, suggestions may be customized according to the age of the user. There has been relatively little proper research into demographic recommendation systems [Mahmood and Ricci 2007].

6.2.1.4 Knowledge-Based Systems

A knowledge-based recommender suggests products based on inferences about user's needs and preferences. Notable knowledge-based recommendation systems are case-based [GÖKER et al. 2005] as presented later in this section. Constraint-based systems are another type of knowledge-based recommendation systems. Case-based recommenders determine recommendations on the basis of similarity metrics whereas constraint based recommenders predom-

inantly exploit predefined knowledge bases that contain explicit rules about how to relate customer requirements with item features.

6.2.1.5 Community-Based Systems

This type of systems recommends items based on the preferences of users friends. This technique follows the epigram "Tell me who your friends are, and I will tell you who you are.". The growing popularity of open social networks generates a rising interest in this systems or, as or as they usually referred to, social recommendation systems [Golbeck 2006].

6.2.1.6 Hybrid Recommendations

Recommendation approaches are often combined to create a hybrid recommendation system [Burke 2007]. A hybrid approach could be more effective in some cases. Netflix is a good example of hybrid systems. They make recommendations by comparing the watching and searching habits of similar users (*i.e,* collaborative filtering) as well as by offering movies that share characteristics with films that a user has rated highly (content-based filtering).

Video systems such as YouTube recommend videos that a user might like to watch based on user's previous ratings and watching habits. It is based on the behavior of other users and the characteristics of the video. [Davidson et al. 2010] want YouTube recommendations to be reasonably recent and fresh, as well as diverse and relevant to the user's recent actions. In addition, it is important for them that users understand why a video was recommended to them. For that, the set of recommended videos is generated by using a user's personal activity (watched, favorited, liked videos) as seeds and expanding the set of videos by traversing a co-visitation based graph of videos. The set of videos is then ranked using a variety of signals for relevance and diversity. However, [Baluja et al. 2008] present a recommendation system for YouTube videos based on random walks on the bipartite user-video graph. They evaluate their method on a three-month snapshot of live data from YouTube, and show that it outperforms using video co-views to recommend new videos.

6.2.2 Contextual Recommendations

[Lieberman and Selker 2000] define context as "everything that affects the computation except its explicit input and output". Most recommendation systems deal with applications having only two types of entities, users and items and do not put them into a context when providing recommendations. Context is providing information that can influence the perception of the usefulness of an item for a user. For this reason recommendation systems must take into account this information to deliver more useful (perceived) recommendations [Ricci 2010]. The fact that user's interaction with systems within a particular "context" and ratings for items within one context may be completely different from the rating for the item within another context [Anand and Mobasher 2007]. It is therefore not surprising that stories of inappropriate recommendations abound, for example:

- A student looking for scholarship offers but after finishing his studies he stills obtain recommendations about scholarships.
- A male customer buying a pregnancy book from Amazon.com as a present, persistently receiving recommendations on pregnancy related topics.
- To suggest a meaningful travel to a user, a recommendation system must know if the travel is scheduled in summer or winter, and if the user is traveling alone or with kids.
- Booking.com filters his clients reviews according to their situation (solo travelers, young couples, group of friends, families, *etc.*). If a recommendation service exists, it would take in consideration these contextual situations.
- YouTube video recommendations are mainly based on the videos that users have viewed but not on the order in which the user has actually watched these videos.

[Adomavicius and Tuzhilin 2008] introduced three algorithmic paradigms for incorporating contextual information into the recommendation process. In *reduction-based (pre-filtering) methods*, only the information that matches

the current usage context, *e.g.*, the ratings for items evaluated in the same context, are used to compute the recommendations. In *contextual post filtering*, the recommendation algorithm ignores the context information. The output of the algorithm is filtered/adjusted to include only the recommendations that are relevant in the target context. In the *contextual modeling*, the more sophisticated of the three approaches, context data is explicitly used in the prediction model.

The past decade has seen several approaches to integrating context into the recommendation process, each one focused on a single contextual factor, such as social networks [Said et al. 2010, Groh and Ehmig 2007, King et al. 2010], tags [Symeonidis et al. 2008], and reviews [Wijaya 2008]. None of these approaches have used the same context-aware algorithm, however, making it difficult to determine what the best solution is. [Anand and Mobasher 2007] presented a new approach to modeling users, based on research in psychology. It consists of short and long term memories (STM and LTM, respectively) that incorporates the notion of user context. STM are memory objects for current user interactions whith the system while LTM are for previous user interactions. They use a simple heuristic to segment user activities based on the context. Our approach reuses modeled traces that represent user's behavior to provide contextual recommendation of actions. It retrieves episodes that are semantically similar to the current sequence of interactions and adapts them to the current context.

6.2.3 Case-Based Recommendation Systems

Case-Based Reasoning has played a key role in the development of an important class of recommendation system known as content-based or case-based recommenders. In these systems, a similarity function estimates how much the user needs (problem description) match the recommendations (solutions of the problem). Here the similarity score can be directly interpreted as the utility of the recommendation for the user [GÖKER et al. 2005]. CBR research has shown how to guide the user by asking questions, giving advice, making further recommendations, processing user feedback of the recommended products, and providing explanations.

Explicit parallels between CBR and user-based collaborative recommenders

have been drawn. For example, a travel advisor system that enables the selection of travel locations, activities and attractions, and supports the bundling of a personalized travel plan [Ricci et al. 2002]. A new collaborative approach is introduced, Travel plans are stored in a memory of cases, which is exploited for ranking travel items extracted from catalogues. [Dong et al. 2013] described a novel approach to case-based product recommendation using experiential cases mined from user generated reviews. They use these cases as the basis for a form of recommendation that emphasizes similarity and sentiment. Group recommendations are well studied using CBR concepts. Group recommender systems suggest items taking into account the preferences and personalities of the members of a group [Jameson and Smyth 2007]. In the context of movie recommendation to groups of friends, [Quijano-Sánchez et al. 2012] consider a group recommender system that aggregates the results of running a single-person recommender system to predict movie ratings for each member of the group. They also offer a potential solution to the cold-start problem in group recommender systems. In addition, CBR has been also used to generate a sequence of songs customized for a community. [Baccigalupo and Plaza 2007] reuse the preferences of the audience to customize the selection for the group of listeners.

6.3 Trace-Based Recommendations

In this section, we describe our contextual recommendation mechanism. Figure 6.1 shows the general architecture of the Trace-Based Assistant (TBA). It is composed of four main parts. The *Client Application* (Wanaclip in our case) is the main application for which we want to provide the user with assistance. The *Trace Collector* observes the client application to collect the user' traces and to send them to the TBMS. $\mathcal{T}$Store is a TBMS allows concurrently accessing, store and reusing traces issued from various applications. It support functionalities like transformation, querying, and visualizing (See Chapterchp:tstore for a detailed description of $\mathcal{T}$Store). The *Assistant* observes the way the user interacts with the system and provide him with contextual recommendations based on recorded $\mathcal{M}$-Traces. The *Trace Adapter* is the connection layer between the Trace Base Management System and the client

application and its assistant. It is responsible of reusing and managing $\mathcal{M}$-Traces. We define a Trace-Based Reasoning (TBR) mechanism that applies a retrieval algorithm to get similar episodes and that reuses these episodes to provide recommendations.

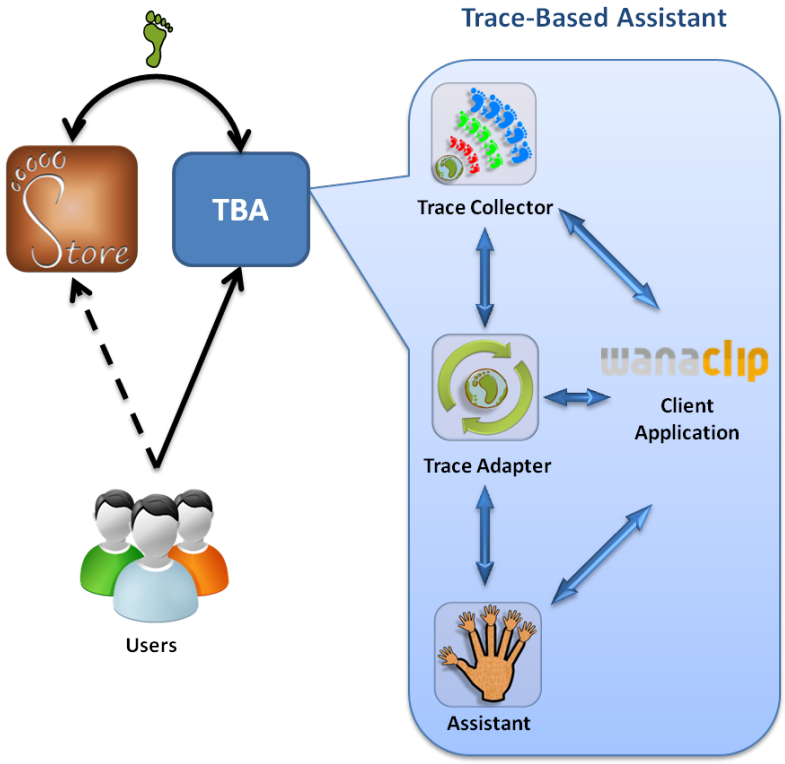

Figure 6.1: Trace-Based Assistant Architecture.

6.3.1 General Recommendation Mechanism

For Wanaclip, we want to develop a contextual recommendation engine that is able to retrieve similar tasks from $\mathcal{T}$Store. For this, we implemented in $\mathcal{T}$Store similarity measures for comparing episodes of $\mathcal{M}$-Traces (See Chapter 5). Using these similarity measures, we define a context-based recommendation mechanism of two main parts (see Figure 6.2), one for storing past experiences and the other for providing the recommendations.

The first part is about collecting and storing $\mathcal{M}$-Traces. $\mathcal{T}$Store applies a sequence of two different transformation types to the current $\mathcal{M}$-Trace (filtering then segmentation). These transformations produce a transformed $\mathcal{M}$-

Trace that represents temporal episodes, ordered sequences of performed actions. These episodes will be indexed and inserted to the episodes map in $\mathcal{T}$Store. This phase called the elaboration step. A temporal mining algorithm can be used to generate association rules to help obtaining better recommendations.

The second part provides contextual recommendations of actions related to the task. It also provides video recommendation that are related to the current selection. A Trace-Based Reasoning mechanism is applied to retrieve and adapt the similar episodes (see Figure 6.2). A retrieval algorithm starts at each new event leading to a change in a video selection. It searches all the similar episodes in the episodes map using episodes similarity measures. Then, a reuse step extracts the candidate videos and actions from the retrieved episodes. After ranking the results according to their similarity, they are adapted to the current context to be recommended to the user. According to the satisfaction of the user, feedback helps $\mathcal{T}$Store to enhance the results for a next usage.

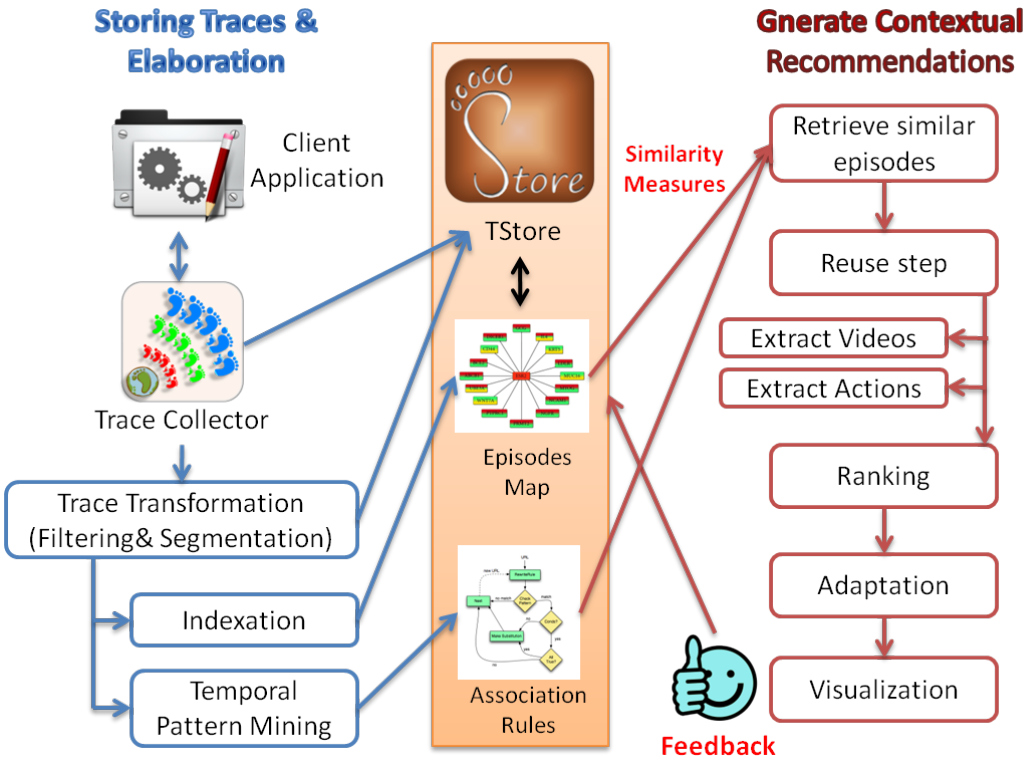

Figure 6.2: Trace-based contextual recommendations mechanism.

6.3.2 Transforming $\mathcal{M}$-Traces and Training

The primary trace issued from the collect process contains all the collected obsels (of all types). Depending on the task at hand, only a subset of these obsels is relevant. In addition, some obsels have to be combined during a transformation process to produce a transformed $\mathcal{M}$-Trace ready for its exploitation. Our idea is to use two different transformations to produce temporal patterns from $\mathcal{M}$-Traces.

As shown in Figure 6.3 , we apply two transformations: Filtering and Segmentation. First, we apply filtering transformation on the primary $\mathcal{M}$-Trace to get only the obsels that are important and can change the context. This filter is applied on specific obsel-types such as "Navigate Tab", "Mute", "StopVideo", *etc.* However, important obsel-types are kept in the transformed $\mathcal{M}$-Traces, *e.g.* "Add Media", "Add Subtitle", "RemoveVideo", *etc.* After that, a segmentation transformation is applied to the transformed $\mathcal{M}$-Trace produced from the filtering transformation. The segmentation splits the $\mathcal{M}$-Trace into several episodes where each episode corresponds to a specific task. In Wanaclip, segmentation is performed by defining specific obsels as terminations of tasks. For example, obsel-types such as "Reset", "ClearPlaylist" and "SaveProject" segment an $\mathcal{M}$-Trace containing them into different episodes. For a more complex tasks, Finite-State Transducer transformations (see Chapter 3) can to be used to define the segmentation process. The segmentation is important for comparing episodes of $\mathcal{M}$-Traces quickly. Comparing episodes of small sizes is much faster than comparing long episodes.

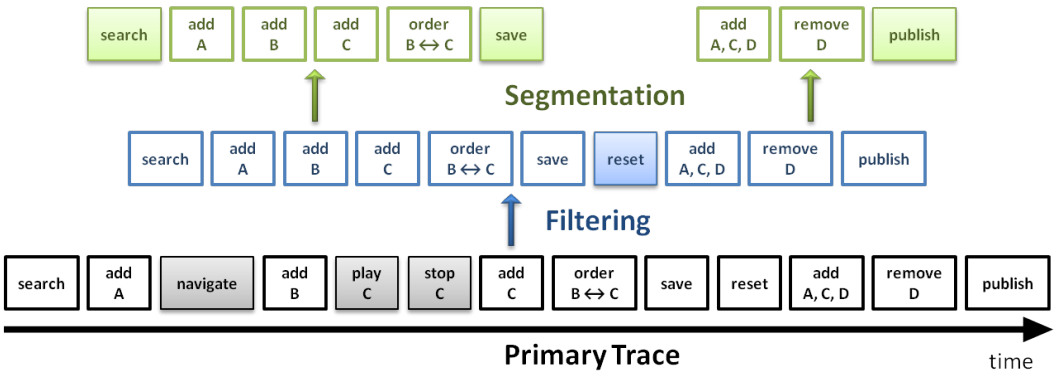

Figure 6.3: Trace Transformation Example.

We can use temporal pattern mining techniques to generate temporal as-

sociation rules between the episodes of $\mathcal{M}$-Traces. This step is optional as it helps to obtain better recommendations. The issue is to find all the episodes that have a user-specified minimum support. The problem of mining interesting sequential patterns is to find all episodes whose *support* or *confidence* are greater than a *threshold*. A *temporal association rule* is defined as a frequent pattern, if it satisfies user defined support and confidence with temporal constraints such as *window-size*, *min-gap* and *max-gap*.

There are different algorithms for sequential pattern mining such as GSP (Apriori-based), PSP (prefix tree), and SPAM [Mabroukeh and Ezeife 2010]. The choice of the algorithm and generating the temporal association rules is a matter of a future work. It is mentioned just to raise the attention to the possibility of adding this technology in order to obtain better recommendations. If this is applied, the generated rules can be generated and stored offline. They can be used later for the recommendations during the retrieval step. Each time a user changes the selection of videos, Relevant temporal association rules can be used to find related interactions or videos.

6.3.3 Generating Contextual Recommendations Using TBR Approach

TBR uses traces as a knowledge source. Reusing traces allows solving new problems by finding similar episodes of $\mathcal{M}$-Traces and adapting them to the current context. In our case, the TBR reuses the past $\mathcal{M}$-Traces in order to find episodes having similarity score bigger than a predefined similarity threshold. As shown in Figure, 6.4, the TBR process comprises three phases: elaboration, retrieve and reuse. A feedback at the end is used to enhance the results and learn the system.

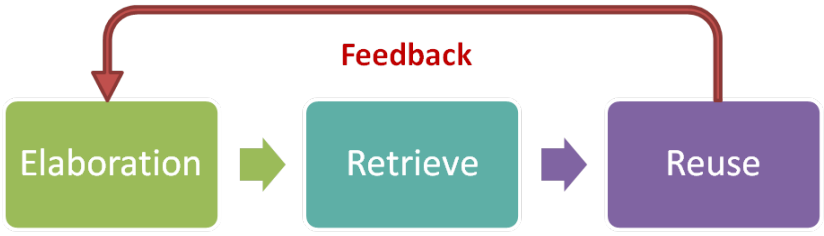

Figure 6.4: The Trace-Based Reasoning process

6.3.3.1 Elaboration step

The goal of the elaboration step is to identify an episode signature. An episode signature is a specification of the constraints that an episode must satisfy (pattern, duration, *etc.*) [Cordier et al. 2013]. Episode signatures are defined using trace transformations described previously. The used method is semi-automatic because it needs an expert in the application to define the task signatures. However, the system is able to enhance its results using users' feedback.

6.3.3.2 Retrieve step

This step consists in finding a set of episodes that can be used to solve the current problem. In this step, we define episode similarity measures for comparing episodes belonging to $\mathcal{M}$-Trace. Our method proposed in the past chapter is an edit distance method. It relies on the definition of a similarity measure for comparing obsels, combined with the implementation of the Smith-Waterman algorithm [Smith and Waterman 1981] for episode comparison. In order to define the similarity between the obsels, we defined several local similarity measures for the obsel-types, users, attributes and time-stamps, which are the significant components of an obsel. A substitution matrix is built using the similarity between obsels and a gap function to determine the reduced score according to the number of indels (insertion/deletion) between obsels. It represents the cost of replacing an obsel by another one. This method not only evaluates the similarity between two episodes but also the changes needed to go from one episode to the other (alignment), which is particularly useful for issuing the recommendations.

The retrieval algorithm is based on the similarity measure. Firstly it filters repeated episodes and uncompleted ones. For example, some sessions generates $\mathcal{M}$-Traces with only one action, these episodes are filtered. After that, for each episode, the retrieval algorithm computes the similarity measure between the current episode and the stored one. It returns all the candidate episodes for which similarity scores are bigger than an *episode similarity threshold*. Episode similarity threshold is defined by the application expert. It can be a static value or a function.

Example 16. An episode similarity threshold can be a function such

as (ln, log2, log10, sqrt, *etc.*) applied to the size of the current episode. *i.e,* if the current episode contains 9 obsels, then an episode similarity threshold can be defined using "sqrt" function on episode size ($\sqrt{9} = 3$). Therefore, only episodes that have a similarity score bigger than 3 are retrieved. ◊

An indexing method should be used to accelerate the retrieval step. To do that, we define a training mechanism that organizes the similar episodes like self-organizing map [Kohonen 1990] or Case Retrieval Nets [Chakraborti et al. 2006]. This issue will be treated later in a future work.

6.3.3.3 Reuse step

It uses the retrieved episodes to find the top N recommended videos and the next actions to perform. During this step, the user uses query operations on the retrieved traces to exploit their content. The obsels of the current episode helps to elaborate the target problem. This episode is used for the matching with stored episodes. In this step two different recommendations are provided: Video recommendations and action recommendations to be proposed to the user to perform them. For that, the system extracts from every retrieved episode (candidate episodes) a sub sequence of actions. This sub sequence is the part that follows the matched part between this episode and the current one (called *successors*), *i.e,* the obsels that follows the last obsel in the alignment of these episodes. Then, the system adapts the sequence of actions of the candidate episodes to the current context. This is performed by replacing some values by more adequate values. Finally, the system displays the adapted action in order to allow the user to execute them (if he wants to).

An adaptation step is important to assure that "impossible" or "already made" actions are not recommended to the user. For example, the action "publish clip" should not be recommended if the user has not added any videos to his timeline. Similarly, the action "Add video" should not be available if the video in question is already in the timeline. Some actions These actions are filtered. Some actions should not be taken into account at all, such as adding a private video that belongs to another user. They may also contain deleted values, *e.g.* adding a video that has been already deleted. In addition, the algorithm filters repeated actions or videos. For example, if a candidate

episode contains the same action repeated many times such as "Change subtitle color", it recommends only the last action. Otherwise, the system may add some actions to allow performing the recommended ones. For example, if the system recommends "change subtitle font", but there is not any subtitle added, then, the adaptation proposes to add a subtitle and selecting this font.

For video recommendations, the system applies transformation operations to extract the videos included in the retrieved episodes. As a result, the system obtains a sequence of videos and applies weight functions for ranking them. A weight function is composed of different measures applied to the videos in the retrieved episodes like: support, distance, common video annotations and the pre-computed episode similarity score. For example, a Video A can be similar to a video B, if they had similar annotations. Therefore, the distance between (ABC) and (XYZ) is not necessarily 0. Finally, the algorithm displays the top N recommended videos.

6.3.3.4 Feedback and Rating

The feedback mechanism enhances the accuracy of recommendations. During the time, accepted recommendations will be more likely to appear. The feedback helps to enhance the results for a next usage. For that, a new association rule is generated and the substitution matrix will be updated.

Each time a user click (accept) one of the suggested recommendations, an indication of the selected recommendation is stored automatically in $\mathcal{T}$Store. It is stored as an attribute in the collected obsel that is generated from the selected recommendation. For example, if a user selects a recommendation to add a subtitle, a new obsel is generate of type "Add Subtitle". This attribute contains its original attributes (such as the text, color, *etc.*) in addition to a new attribute called "recommendation". This attribute refers to the original obsel in the $\mathcal{M}$-Trace that this recommendation comes from. This also helps to keep a trace about recommendations and their origins (which $\mathcal{M}$-Traces contain the recommended obsels).

6.3.4 Visualization and Interactive Interface

We have instrumented the initial Wanaclip application shown in Chapter 1 in order to collect user's traces and display them on top of the screen (see

(1) in Figure 6.5). We have also developed a graphical interface that displays the recommendations and updates them automatically after a new action (see (2) in Figure 6.5). The recommendation interface inform users why they are receiving these recommendations.

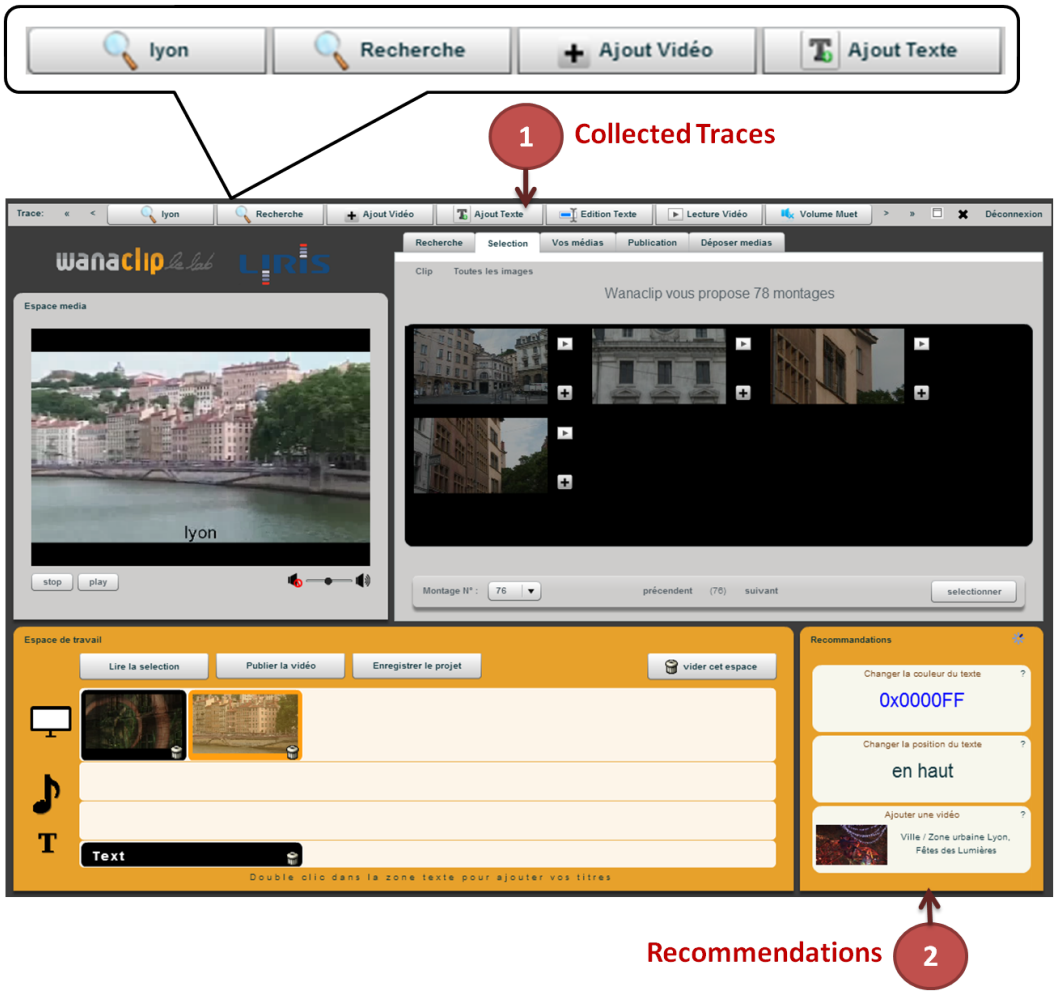

Figure 6.5: Wanaclip user interface with collected $\mathcal{M}$-Traces and recommendations.

The original interface of Wanaclip (see Figure 1.2) includes a player to view the videos, a timeline in which the videos, music and text that will form the video clip will be located and an area for the search of videos. We reduced the size of the timeline and created a zone of recommendations. This allows a simple visualization that avoids putting the recommendations into a place that could interfere with the normal use of the application.

Each time the user performs a new action, a trace handler generates a new obsel and stores it in $\mathcal{T}$Store. This obsel is displayed with other obsels on top of the application. Each action (obsel) is represented by a button with the action name (obsel-type) and an icon; Clicking on any of these actions displays a menu that provides more information on the action. The user can click on any action to see additional information about it. For example, when clicking on "addVideo", the user sees information about the added video, *e.g.* name, duration, keywords, *etc.* in addition to obsel id, obsel-type, and its time-stamps.

For certain new actions that changes the context, the recommendation engine asks $\mathcal{T}$Store to retrieve the appropriate recommendations. These recommendations are displayed on the bottom right of Wanaclip. Each visualized action is represented by a block with a descriptive name of the action (add video, add text, *etc.*) and potentially an image (for video) and/or a text describing in more detail. For example, for adding a video, the name of the video is displayed. For the subtitle actions, their names is shown as well as their values such as bold, italic, change the font, *etc.* When the user clicks on a recommendation, the associated action is executed. The user may also consult the source $\mathcal{M}$-Trace that contained this recommendation. This allows him to understand why this action has been recommended. In addition, if a user does not want to have his actions stored by the system, he can dis-activate the trace collector.

6.4 Recommendation Example

The following example comes from the web application Wanaclip and is related to the example shown in the past chapter. Let us consider that we have a stored episode $\mathcal{E}_1 = (S_8 P_1 A_1 A_2 X_1 A_3 G_7)$. It represents a user searching for videos about "Lyon" (S_8: S is the obsel-type, and 8 is the value). Then, he starts playing (P_1) and adding videos (A_1 and A_2) to his timeline. After that, he adds the text X_1 to the first selected video and adds another video A_3. Finally, he generates the video clip G_7. This episode is collected and stored in $\mathcal{T}$Store. Other episodes of $\mathcal{M}$-Traces for different users are also collected and stored in $\mathcal{T}$Store. Currently, a user is using Wanaclip. He selects the tag

"Lyon", plays and adds a video, then adds text to it. This episode is noted "$\mathcal{E}_2 = (T_8 P_1 A_1 X_1)$J.

Using the stored $\mathcal{M}$-Traces, an assistant recommends the next actions to do. For that, the assistant compares the current episode with the stored ones and recommends the most similar ones. Figure 6.6 shows how to compare the stored episode $\mathcal{E}_1$ with the current episode $\mathcal{E}_2$ using the Smith-Waterman algorithm after applying the substitution matrix based on the similarity measures between obsels. The substitution matrix is updated over the time using different criteria such as the temporal association rules between the obsels, video tags, user's profile, *etc.* The shown substitution matrix is just a sample one.

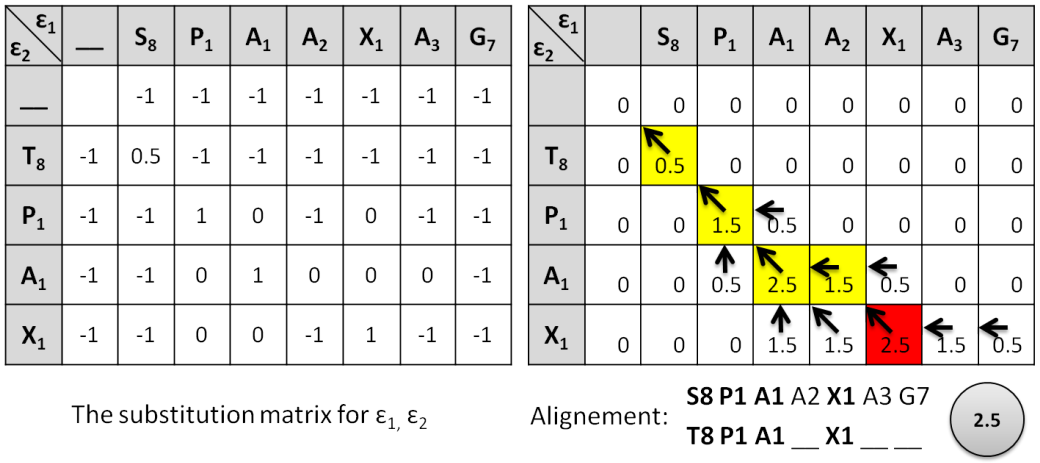

Figure 6.6: An example of the $\mathcal{M}$-Trace Smith-Waterman algorithm to compute the similarity between $\mathcal{E}_1, \mathcal{E}_2$.

Table 6.1 shows an example list of episodes and their similarity scores comparing to the current episode. They are computed, as shown in Figure 6.6, by considering an episode similarity threshold as the sqrt of the size of the current episode ($\sqrt{4} = 2$), the retrieve step returns 4 similar episodes from the episodes in this table.

In the reuse step, the system extracts the sequence of actions of the retrieved episodes in order to adapt them to the current context. We notice that the episode $(P_1 A_1 X_1)$ does not have any successor, so it is eliminated from the recommendations. After that, these actions are displayed as recommendations to the user ranked according to their similarity score. The user can execute

Table 6.1: A list of the retrieved episodes and their similarities comparing to the current episode $\mathcal{E}_2 = (T_8P_1A_1X_1)$.

Retrieved?	Episode	Successors	Similarity Score
√	$S_8P_1A_1A_2X_1A_3G_7$	A_3G_7	2.5
	$S_3P_2A_2T_2F_2C_2$	-	1
	$T_8P_3A_3X_3P_4A_4$	-	1
	$S_6P_2A_2P_7F_5$	-	0
√	$T_8P_2P_1A_1X_1A_3X_3A_4$	$A_3X_3A_4$	3
	$S_8P_1P_3P_4C_4$	-	1.5
√	$P_1A_1X_1$	-	3
√	$S_5P_1A_1A_2A_3X_3C_1$	$A_2A_3X_3C_1$	2

the recommended actions or replay the whole episode. He can also also see details about each action (its name, values and time-stamps). For video recommendations, firstly the system extracts the videos from the successors (we get 3 videos: 2, 3, 4). These videos are ranked according to the defined weight functions. If we considered the weight function is sum of the similarity scores of the episodes that contains the video, so the best video is $video_3$ (occurred 3 times $2.5+3+2 = 7.5$) then the $video_4$ (occurred 1 time = 3) and the last one is $video_2$ (occurred 1 time = 2). Finally, the recommendation engine displays the top videos to the user.

6.5 Evaluation

In this section, we perform a thorough evaluation of the quality of the recommendations, their performances and the users' satisfaction.

6.5.1 Evaluation Protocol

We conducted tests to determine how our recommendation mechanism performs in terms of accuracy, responsiveness and performance. We have asked real users to test the application. We have provided $\mathcal{T}$Store with functions that can calculate the execution time, memory usage and comparison time of storage, similarity measures and recommendations. We have also added a new table to $\mathcal{T}$Store database (called: executions) to store statistics about recommendations. Each row of this table represents a request of recommen-

dation. It contains fields about the used parameters of the algorithm, retrieval time, memory usage, size of the current episode, number of retrieved episodes, number of recommended interactions, *etc.* For each new action performed by the user, $\mathcal{T}$Store can automatically compute and store its storage time and memory usage. In addition, while generating recommendations, $\mathcal{T}$Store keeps all the important statistics of all recommendation requests. Our evaluation protocol is composed of three measurements:

1. **Performance:** This step is about analyzing the required time for computing similarity measures and generating recommendations. We also study the memory usage during the recommendation process. The goal is to prove that our recommendation mechanism provides recommendations within a reasonable time.

2. **Accuracy:** It is important to determine how close our recommendations are to the user's true expectations. For that, we define some metrics to measure how recommendations match user's needs and the percentage of responses of recommendations to the number of requests.

3. **User satisfaction:** It is based on a survey that we have asked Wanaclip users to answer, in order to analyze their satisfaction. This survey is composed of several questions that users answered after their usage of Wanaclip. We analyze the results to collect users' feedback to understand what are the weakness and issues that can be enhanced later.

6.5.2 Performance

Performance study is important to be sure that our recommendation approach has a good response time. The system should react quickly after each new action and provide the appropriate recommendations. Recommendations have to be shown to the user before performing the next step. This is important to help him decide which action to perform. In order to test the performance of our recommendation mechanism, we have asked real users to test Wanaclip. The total number of users who have tested Wanaclip is 128. They have generated 892 $\mathcal{M}$-Traces that are collected and stored in $\mathcal{T}$Store, with an average of about 7 $\mathcal{M}$-Trace (sessions) per user. These $\mathcal{M}$-Traces contain about 14742

obsels, an average of 16.5 obsels/$\mathcal{M}$-Trace, which means about 16.5 actions per session. In general, Wanaclip collects about 50 different types of obsels. The total number of attributes of all collected obsels is 107174 attributes. It means that each obsel contains on average 7 attributes.

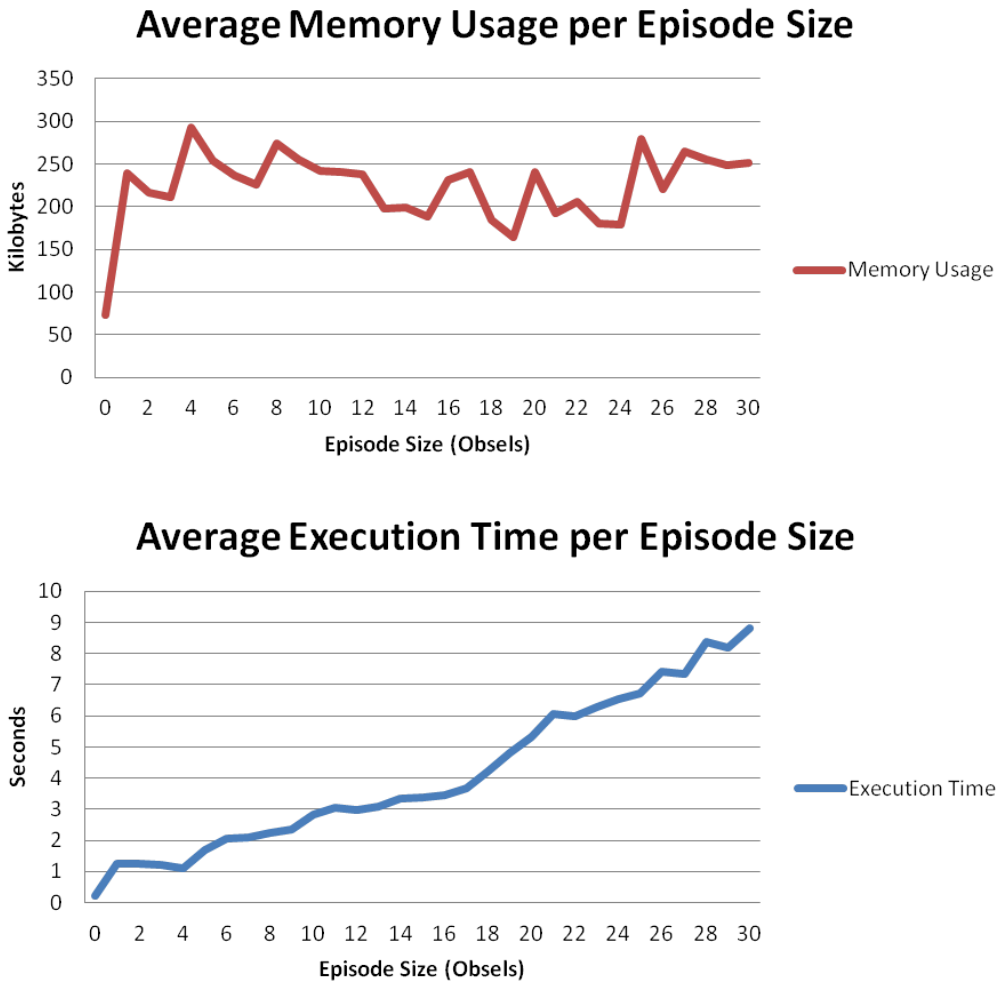

Figure 6.7: Average execution time and memory usage per episode size.

The recommendation process needs an average of 2.83 seconds to produce the results. The execution time is the required time to retrieve similar episodes and to extract and adapt the candidate obsels to be recommended to the users. This process uses an average of 223 Kilobytes from the memory. Figure 6.7 shows the average execution time of the recommendation process per episode size (number of obsels). It shows that the higher the size of the current

episode, the more required execution time increases. Therefore, it is important to segment the episodes. According to that, an episode should not have more than 20 obsels, otherwise, the recommendations will take a lot of time. This figure also shows that the memory usage is independent of the episode size.

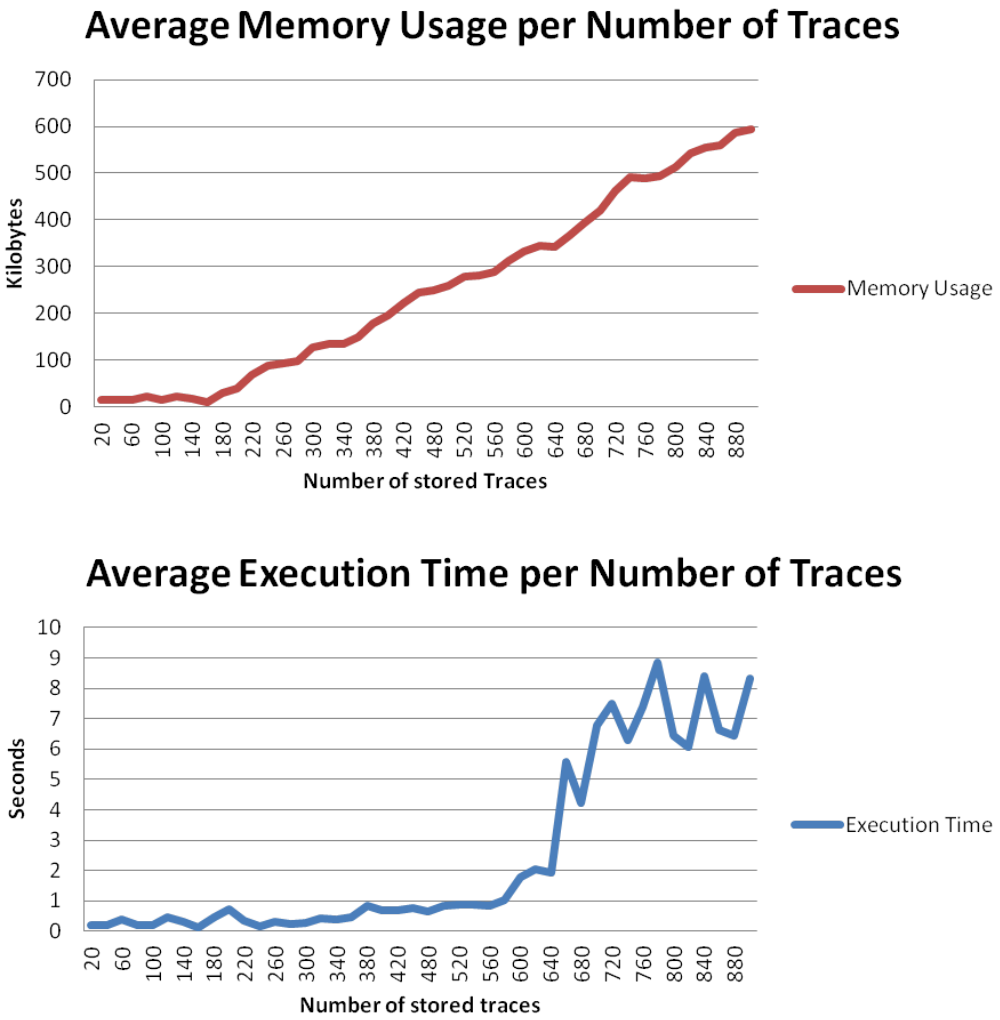

Figure 6.8: Average execution time and memory usage per number of stored $\mathcal{M}$-Traces.

Memory usage depends on the number of stored $\mathcal{M}$-Traces *i.e,* the number of episodes that are compared to the current episode. As shown in Figure 6.8, the more the number of stored $\mathcal{M}$-Traces in $\mathcal{T}$Store, the more memory usage and execution times are required. We notice that the execution time does not exceeded 2 seconds for the first 650 stored $\mathcal{M}$-Traces. After that it had a

high increase. We conclude that it is important to define indexing methods to decrease the number of compared episodes. The index categorizes the stored episodes to gather the similar ones together. This method helps to pre-filter a candidate set of episodes that are more likely to be similar. Indexation is an important matter for future work.

It is not enough to obtain the recommendations quickly, but it is also important to be accurate and satisfy the users. Next section defines some metrics to measure the quality and accuracy of our recommendations.

6.5.3 Accuracy and Acceptance Metrics

There are different measures focused on how close recommendation system predictions are to the user's true preferences. Whenever a recommendation system is evaluated, the metric used for the evaluation is built upon certain assumptions. Accuracy metrics measure of [Armstrong et al. 1995] are the most used and well-known metric in the field of Artificial Intelligence.

Definition 14 (Accuracy)**.** *According to [Hernández del Olmo and Gaudioso 2008], Accuracy metrics measure the quality of nearness to the truth or the true value achieved by a system. Accuracy can be formulated as in Equation 6.1 in general and as in Equation 6.2 for the recommendations.*

$$accuracy = \frac{number\ of\ good\ cases}{number\ of\ cases} \tag{6.1}$$

$$accuracy = \frac{number\ of\ successful\ recommendations}{number\ of\ recommendations} \tag{6.2}$$

Traditional calculations of accuracy require to know the real preferences of users. It is usually *estimated* for every particular user u and item i by means of the users' previous ratings. However, our trace-based recommendation approach provides recommendations that change over time after each new action. In other word, the context changes with each action. Our recommendations are temporally instantiated, thus we can not measure the accuracy. In addition, we do not recommend items, each recommendation is an obsel that has a type and attribute values, *i.e,* an action. Therefore, we cannot determine the number of items. Now, assuming that an "accepted recommendation" is equivalent to the recommendation that the user has selected from

the recommendation list, thus, it is *close* to the user's expectations. We use user feedback to define new metric (*acceptance rate*), similar to the accuracy metric with the temporal dimension.

Definition 15 (Acceptance Rate)**.** *We call an* ***Acceptance Rate*** $(\mathcal{A})$ *the metric that measures the number of recommendations that have been used (approved) by the users on the total number of recommendations overtime.*

There are specific obsels that trigger new recommendations. These obsels change the context, thus new recommendations need to be obtained. For example, the obsel-type "Change Tab" does not require computing new recommendations. Therefore, it is important to define a function that computes the total number of "*recommendation requests*" $(\mathfrak{Q})$.

Definition 16 (Recommendation Requests)**.** *Let us represent a request of recommendations as* $\mathfrak{q}(\tau, o)$ *for each obsel* (o) *in an* $\mathcal{M}$*-Trace* (τ)*.* $\mathfrak{q}(\tau, o)$ *offers only two values:* 0 *if the obsel* (o) *in the* $\mathcal{M}$*-Trace* (τ) *requires computing new recommendations and* 0 *otherwise. Using this function we can compute the total number of recommendation requests* $(\mathfrak{Q})$ *as the following:*

$$\mathfrak{Q} = \sum\nolimits_{\tau,o} \mathfrak{q}(\tau, o) \tag{6.3}$$

Each recommendation result shown to a user may contain a number of obsels (items) from 0 to N. In Wanaclip, $N = 3$ to fit the available space in the interface. To compute the acceptance rate we need to compute the number of recommendation requests which has at least one result shown to the user. We call it the "*recommendation responses*" $(\mathcal{P})$. We also need to compute the number of recommended obsels shown to the users.

Definition 17 (Recommendation Responses)**.** *Let us consider* $\mathfrak{r}(\tau, o)$ *the recommended obsels shown to a user after executing the obsel* (o) *in his* $\mathcal{M}$*-Trace* (τ)*. The number of answered recommendation requests (recommendation responses)* $(\mathcal{P})$ *is computed in Equation 6.4. For that, we use Dirac delta function* (δ) *[Dirac 1958] which converts any real number to zero, and zero in one. We need to inverse it, thus we use* $(1 - \delta(\mathfrak{r}(\tau, o))$ *to obtain 1 if there are responses and 0 if not. We can also compute the number of recommended*

obsels shown to the users ($\mathfrak{R}$) using Equation 6.5.

$$\mathcal{P} = \sum\nolimits_{\tau,o} \mathfrak{q}(\tau, o) * (1 - \delta(\mathfrak{r}(\tau, o))) \tag{6.4}$$

$$\mathfrak{R} = \sum\nolimits_{\tau,o} \mathfrak{q}(\tau, o) * |\mathfrak{r}(\tau, o)| \tag{6.5}$$

By using the previous equations (6.4) and (6.5), we can define the *acceptance rate* ($\mathcal{A}$) as in (6.6). In this equation, we consider that $\mathfrak{s}(\tau, o)$ is the selected (accepted) action (obsel) from a set of recommended actions shown to the user ($\mathfrak{s}(\tau, o) \subseteq \mathfrak{r}(\tau, o)$). If users can select more than one recommended obsel at a time, then the acceptance rate is the total number of accepted recommendation obsels $\sum_{\tau,o} \mathfrak{s}(\tau, o)$ on the total number of recommended obsels ($\mathfrak{R}$). Otherwise, it is the total number of accepted recommendation obsels $\sum_{\tau,o} \mathfrak{s}(\tau, o)$ on the number of recommendation responses ($\mathcal{P}$).

$$AcceptanceRate\ \mathcal{A} = \begin{cases} \dfrac{\sum_{\tau,o} \mathfrak{s}(\tau, o)}{\mathcal{P}} & (single\ selection) \\ \dfrac{\sum_{\tau,o} \mathfrak{s}(\tau, o)}{\mathfrak{R}} & (multi\ selection) \end{cases} \tag{6.6}$$

To compute the acceptance rate in Wanaclip, we have asked real users to test the application. From a total of recommendation requests of ($\mathfrak{Q} = 9108$ requests), the number of recommendation responses containing at least one recommended action is ($\mathcal{P} = 3122$ responses). This means that 29.1% of the requests has been responded. The total number of recommended obsels is ($\mathfrak{R} = 7616$ obsels). This means that an average of 2.43 recommended obsel have been shown to the user as a set of recommendations. In Wanaclip, each user can select only one recommendation at a time (single selection). The total number of accepted recommendation obsels is ($\sum_{\tau,o} \mathfrak{s}(\tau, o) = 825$ obsels). Therefore, the acceptance rate of Wanaclip users according to Equation 6.6 is ($\mathcal{A} = 825/3122 = 0.26$). We conclude that for every 4 interactions (obsels), 1 interaction has been selected from the recommendations. On average, Wanaclip users interact with the system 16.52 times each session. Thus, users use the recommendations about 4 times per session. However, it must be noted that the users knew that they were experimenting the recommendations of

Wanaclip. We have the problem of cold start in Wanaclip. Thus, we need a

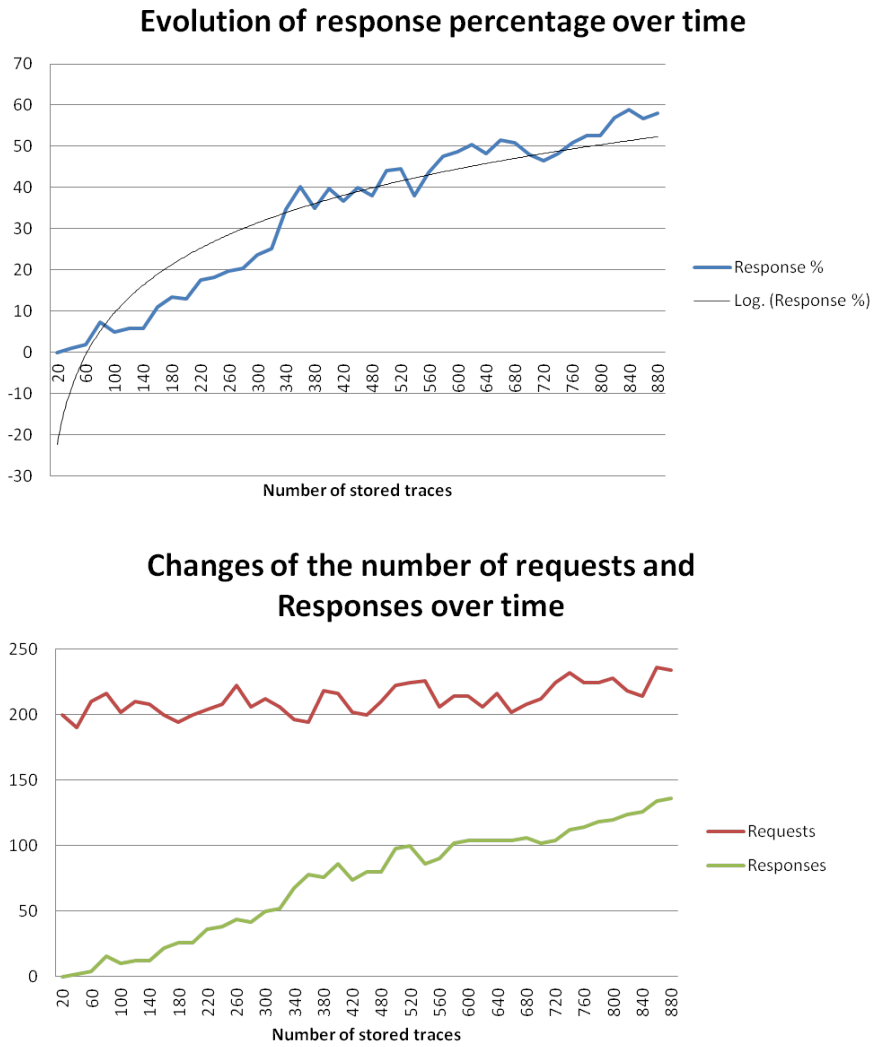

Figure 6.9: Changes of Responses/Requests of recommendation over time.

deep analyses of the results to understand the evolution of the recommendations. The more we have $\mathcal{M}$-Traces collected and stored, the quality and the number of recommendations is better. Figure 6.9 shows that the number of responses (recommendations) increases the more we have stored traces. However, the number of requests is changes within a reasonable range over time either by increasing or decreasing. Therefore, we see that the percentage of the number of responses on the number of requests have a great evolution. This is because the system has more chances to find similar traces when the number

of stored traces is bigger. But at the beginning the system was not able to find recommendations because of lack of stored traces. As a result we find that about 60% of the requests have been responded by at least one recommendation for about 900 stored $\mathcal{M}$-Traces. Otherwise, the size of the current episode (number of its obsels) affects the percentage of recommendation responses. As shown in Figure 6.10, an episode should have at least 2 obsels to be able to retrieve similar episodes, thus recommendations. However, if the episode is too long and contains more than 18 obsels, the response percentage drops down to 0. We also see that the maximum size of an episode generated by Wanaclip users is about 30 obsels. As a result we see that the recommendation algorithm has a better response when the episode size between 3 to 10 obsels.

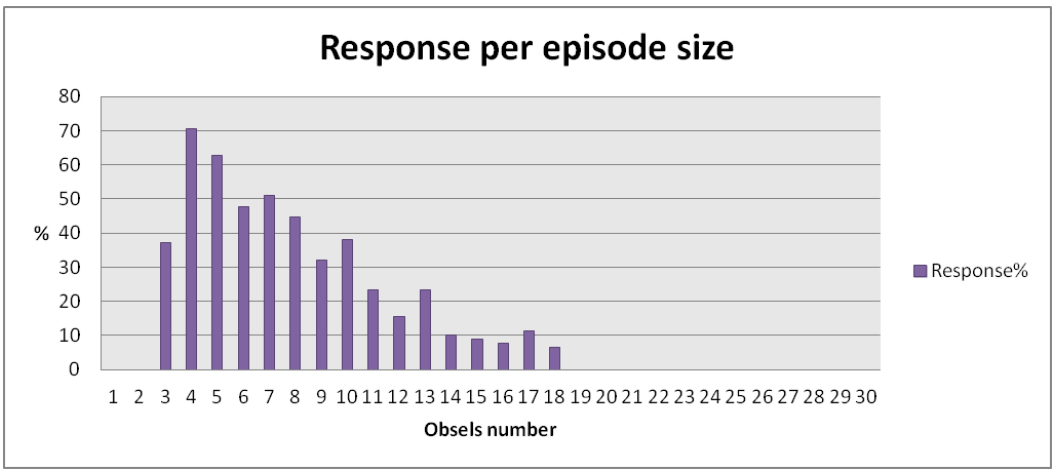

Figure 6.10: Response percentage of recommendation per episode size.

Acceptance rate gives an estimation on the useful recommendations that have been used by the users. However, it does not really reflect a precise user satisfaction, since it is not mandatory for users to click on the recommendations. Therefore, we need a method to determine the real satisfaction, for that we use a survey that users answered after their usage to Wanaclip.

6.5.4 Survey of Users' Satisfaction

In order to validate our research results and to measure the quality and relevance of the proposed recommendations, we have asked Wanaclip users to answer to a survey at the end of their use of Wanaclip. Only 67 participants

have responded to the survey from about 128 users who tested Wanaclip. Figure 6.11 shows a summary of the results of this survey. 72% of the participants have created at least 2 video clips using Wanaclip, 21% among them have created more than 5 video clips. Most of the participants (75%) prefer the current visualization of recommendations. However, 25% prefer to separate videos and actions recommendations. Only 10% find that neither action recommendations nor video recommendations are useful. In addition, more than half of the participants say that both recommendations (videos and actions) are useful for them, 7% think that neither of them is useful. We have also asked them if they accept to have their actions on Wanaclip tracked. A very low number of them refuse to be tracked (3%). However, 21% accept but only for their own use. 49% of participants accept for the recommendations, if the system explains its recommendations (as we do). Finally, 26% accept under opting out condition.

The most important thing is that 93% of the participants confirm that they feel that the recommendations are contextual. In order to measure user satisfaction, the participants are asked to answer this question explicitly. They have 5 options that have a weight from 0 to 4. Their answers are the following: 21% are very satisfied, 34% are satisfied, 31% are 50:50, 7% are dissatisfied and 2% are completely dissatisfied. Thus, user' satisfaction can be computed as the following:

$$Satisfaction = \frac{0 * 3\% \ + \ 1 * 10\% \ + \ 2 * 31\% \ + \ 3 * 34\% \ + \ 4 * 21\%}{4} = 64.5\% \tag{6.7}$$

6.6 Conclusion

In this work, we have described a TBR approach for contextual recommendation using $\mathcal{M}$-Traces. We have introduced a state of the art about recommendation systems. After that, we have presented the TBR process for recommendations. This process is composed of three steps: elaboration, retrieve and reuse. The elaboration step identifies task signatures using two transformations: filtering and segmentation. The retrieve step retrieves past episodes of $\mathcal{M}$-Traces stored in $\mathcal{T}$Store that are similar to the current episode.

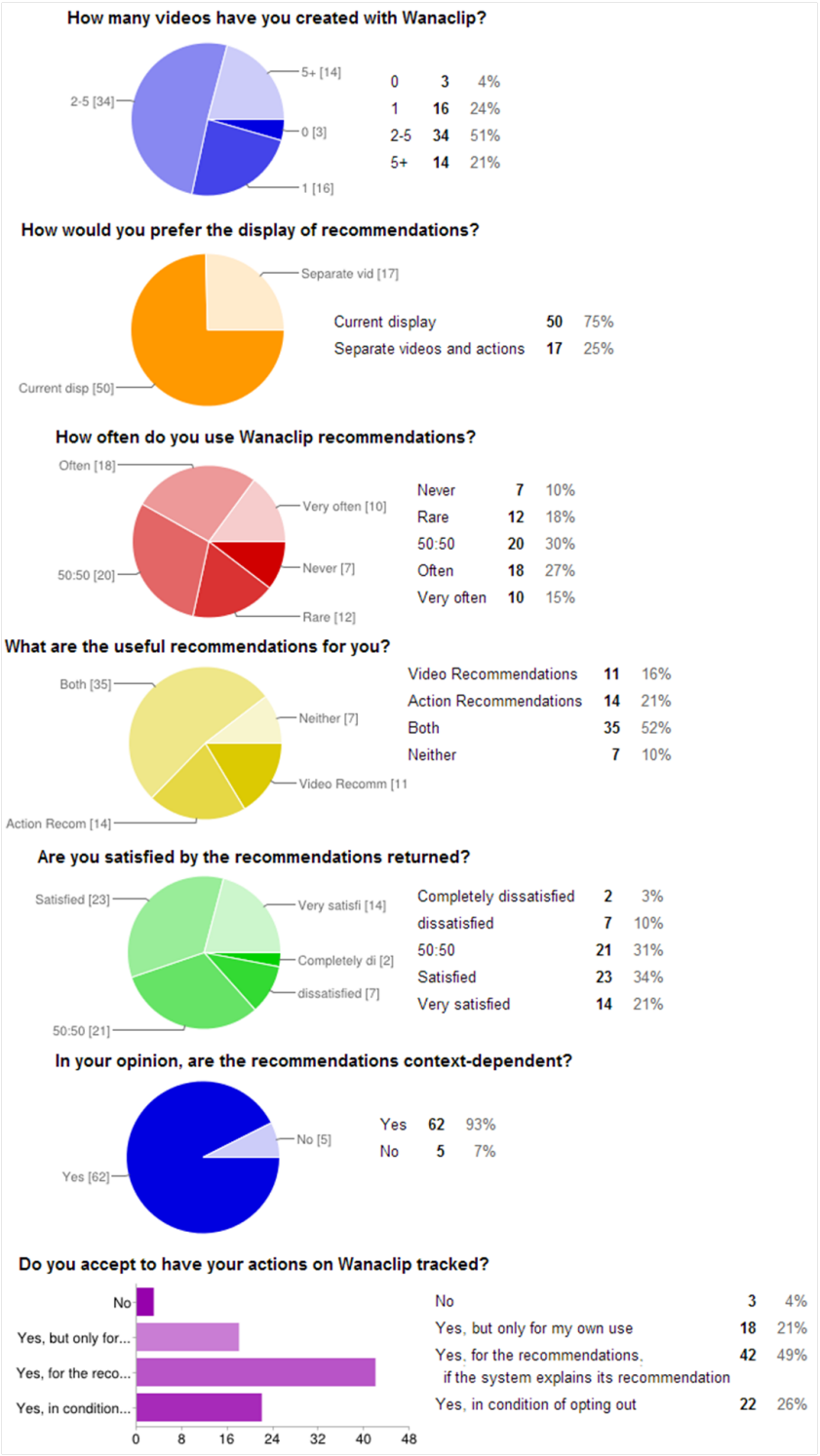

Figure 6.11: A summary of the results of Wanaclip survey.

This step uses similarity measures for comparing episodes belonging to $\mathcal{M}$-Traces. Finally, a reuse step ranks the retrieved episodes, adapts them to the current context and visualize them to the users. The recommendation approach has been applied to the Wanaclip application to guide users in both video selection, and the actions to perform in order to make quality clips.

To evaluate the recommendation approach, we have asked real users to test the Wanaclip application. The evaluation protocol is composed of three parts: performance, quality and users' satisfaction. Our evaluations show that the TBR approach offers quite satisfactory recommendations and response time. However, a number of important research questions remain unanswered Future work will involve developing an index of episodes of $\mathcal{M}$-Traces to accelerate the recommendations. We do not say that we provide the best recommendation mechanism, but we have succeeded in providing a new recommendation approach of actions by reusing interaction traces left by different users. To enhance the results, the TBR recommendation approach can be combined with other recommendation mechanisms.

Chapter 7

Conclusion and Future Work

Contents

We conclude the thesis by discussing the contributions with regard to the research questions stated in the introduction. The Trace-Based Assistant approach opens several possibilities for further research. Some possible future work is presented.

7.1 Synthesis

The main objective in this thesis is to model the creation of general assistance systems taking context into account. This issue raises four research questions that are addressed throughout this thesis. In order to answer these questions, we cover assistance systems and trace-based systems in our state of the art. We highlighted the importance of experience-based assistants and the need to provide contextual assistance. We proved that interaction traces can be used as a knowledge source in these systems. This is achieved using a Trace-Based Reasoning approach to provide contextual recommendations. We built, developed and validated our contributions working successively on two applications: SAP BusinessObjects Explorer and Wanaclip. Each research question lead to a specific contribution, for which we defined a particular method of validation.

The first research question addresses building a general framework of trace experience management for assistance. For that, a formalization of a generic trace meta-model called $\mathcal{M}$-Trace was defined. In this meta-model, each trace is described with a trace model, where a trace model represents the structure of the trace elements and their relations. We developed the $\mathcal{T}$Store web tool that implements $\mathcal{M}$-Traces. $\mathcal{T}$Store is a Trace Base Management System (TBMS) handling storage, transformation and exploitation of $\mathcal{M}$-Traces. We described the six modules of $\mathcal{T}$Store: Storage Manager, Querying System, Transformer, Visualization System, Similarity Measure and Security Manager. The originality of $\mathcal{T}$Store compared to other Trace Base Management Systems relies on the use of a Finite-State Transducer approach for $\mathcal{M}$-Trace transformations. It can also compare episodes of $\mathcal{M}$-Traces which is very important for reusing stored $\mathcal{M}$-Traces. In addition, $\mathcal{T}$Store uses a role-based access control (RBAC) approach to ensure the security of interaction traces. $\mathcal{T}$Store was used to store and manage $\mathcal{M}$-Traces collected from Wanaclip and it provides good performances. However, this is the first version of $\mathcal{T}$Store and it still

needs a lot of enhancements, optimization and additional services.

The second research question addressed providing a mechanism to replay experiences in a new context. For that, we described an approach using $\mathcal{M}$-Traces to allow users to return to a particular state of an application. For this purpose, we use playback of interaction traces. We also analyze the impact propagation of changes performed on past actions. Therefore, the concept of predefined impact rules was introduced. We built an algorithm that discovers adapted values of obsels affected by changes. We implemented the "$\mathcal{M}$-Trace replay" mechanism and demonstrated within the SAP BusinessObjects Explorer application.

The third research question is to define general similarity measures for comparing episodes of traces. This question was answered by defining a comparison method based on two major components: a similarity measure for comparing obsels, and an adaptation of the Smith-Waterman algorithm, using the similarity measure described above, to determine the similarity between two episodes. This method takes into account the richness of the compared elements. In addition, it takes into account the hypothesis that episodes may contain unnecessary obsels. These obsels are generated when unnecessary action are performed, thus they can be ignored or "penalized" with a small factor. Similarity measures are implemented as a component of $\mathcal{T}$Store. They can be called explicitly from any external application or implicitly in the retrieval process.

The last research question addresses providing recommendation of sequence of actions related to a task. We introduced a theoretical study comparing the different types of recommendation systems. The proposed recommendation mechanism is based on a Trace-Based Reasoning approach. For that, a retrieval step is needed to find episodes of traces that can be reused to provide contextual recommendations. This is done using similarity measures for comparing episodes belonging to $\mathcal{M}$-Traces. The recommendation approach has been applied to the Wanaclip application to guide users in both video selection, and the actions to perform in order to make quality clips. This approach can adapt the suggested actions to the current context. We developed an interactive visualization of the recommendations. It allows users to execute the suggested actions and videos by a one click and it obtains a

feedback of users' satisfaction.

To evaluate the recommendation approach and the similarity measures, we asked real users to test the Wanaclip application. The evaluation protocol is composed of three parts. The first part discusses analyzing the performance of the recommendation system in terms of execution time and memory usage. The evaluation results show that our approach offers quite satisfactory performances and response time. These results demonstrate the importance of segmenting episodes of $\mathcal{M}$-Traces to obtain good execution times. We have also measured the quality of recommendations and how close they are to the user's true expectations. For that, we defined a specific accuracy metric called "acceptance rate". It is based on the percentage of the number of accepted recommendations by the users on the total number of provided recommendations. Acceptance rate indicates that users of Wanaclip use the recommendations about 4 times per session. Finally, to determine users' satisfaction, we asked Wanaclip users to participate in a survey. Most of the participants confirmed that the recommendations are contextual. In addition, only 7% of the participants said that the recommendations are useless for them. We do not claim providing the best recommendation mechanism, but we succeeded in providing a new recommendation approach by reusing interaction traces left by different users. The results obtained show that about 65% of Wanaclip users are satisfied by the provided recommendations. To enhance the results, the recommendation approach can be combined with other recommendation mechanisms. Additional ideas for future work are presented in the next section.

7.2 Future Work

Designing a general architecture for a multifunctional assistant is an open question. The problem is to build an assistant that is able to react immediately and automatically to any application. Reusing $\mathcal{M}$-Traces can be useful to provide different forms of assistance. For that, a general trace collector must be defined to work without modifying the observed applications.

The implementation of $\mathcal{T}$Store is still in progress and many services can be added. Future work will involve developing a query language that allows

answering different users' requests. A user interface is one of the important things to be provided. It allows users and administrators to browse and manage $\mathcal{M}$-Traces according to their privileges. Currently $\mathcal{T}$Store only supports XML messages so we want to add new formats. Concerning the $\mathcal{M}$-Trace replay process, some changes in the graph of transformations of $\mathcal{M}$-Traces may be affected. We believe that Finite-State Transducer transformations can be utilized to solve this problem. In addition, Finite-State Transducer transformations should be defined automatically to alleviate the task of application experts.

A future work would be to use user feedback for the computation of the similarity measures. It is also important to develop other sequence similarity measures to enhance the accuracy of the similarity measurement. Optimizing the execution time for the similarity measures between episodes of $\mathcal{M}$-Traces is a critical issue. For a huge number of stored $\mathcal{M}$-Traces, our recommendation mechanism requires a lot of time for comparing between $\mathcal{M}$-Traces. Building an index between the episodes of $\mathcal{M}$-Traces can be a solution for optimizing the performance of the retrieval process. Another solution is to develop a pre-filtering mechanism that eliminate some episodes from being analyzed by the recommendation algorithm. Techniques such as Neural Networks, Self Organizing Maps and Hidden Markov Models, can be used.

Integrating our Trace-Based Reasoning recommendation approach with other recommendation approaches can help to obtain better results and thus to increase user satisfaction rate. In addition, social networks are great sources of knowledge. A revolutionary enhancement can be achieved by analyzing users' profiles and their activities. Information about "likes", "follows", interactions and relations between users can be combined with $\mathcal{M}$-Traces. Different Data Mining techniques can be applied to enhance the recommendations. This allows to pre-compute models that describe the correlations between episodes of $\mathcal{M}$-Traces. As a result, we can obtain recommendations that are more personalized and contextual.

Appendices

Appendix A

Obsel Types in Wanaclip

The trace model defined for the Wanaclip application contains about 54 obsel-types. These obsel-types are collected and stored in $\mathcal{T}$Store. Table A.1 shows a list of obsel-types and their attributes.

Obsel-Type	**Attributes**
Add Media	item(id, name, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, year, season, moment, exist, index, end, start, duration, volume)
Add Subtitle	item, italic, size, bold, position, duration, end, start, font, color, text
AddSelectedClips	item(id, name, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, year, season, moment, exist, index, end, start, duration, volume)
AddSelectedSequences	item(id, name, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, year, season, moment, exist, index, end, start, duration, volume)
AddUserVideo	id, title, link, image, tags, public, permanentLink, description, volume, end, start, duration, embedPlayer, sourceDrag, type, index
Advanced SearchMedia	kind, duration, rythme, tags
Bold Subtitle	item, item, italic, size, bold, position, duration, end, start, font, color, text
CancelUpload Media	filename, tags, description, title
ChangeProgress VideoPlayer	id, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, volume, end, start, duration, playheadTime, exist, moment, season, year
ChangeSlider VideoInOut	id, name, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, year, season, moment, exist, index, end, start, duration, volume
ClearPlaylist Selections	-
Close VideoPlayer	id, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, volume, end, start, duration, playheadTime, exist, moment, season, year

Confirm	id
Delete Subtitle	item, italic, size, bold, position, duration, end, start, font, color, text
Edit Subtitle	item, italic, size, bold, position, duration, end, start, font, color, text
Find UserMedia	text
InfoUserVideo	id, title, link, image, tags, public, permanentLink, description, volume, end, start, duration, embedPlayer, sourceDrag, type, index
Italic Subtitle	item, italic, size, bold, position, duration, end, start, font, color, text
Mute VideoPlayer	id, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, volume, end, start, duration, playheadTime, exist, moment, season, year
Navigate Selections	selection
Navigate Tab	tab
PauseVideo ClipPlayer	id, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, volume, end, start, duration, playheadTime, exist, moment, season, year
PauseVideo VideoInOut	id, name, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, year, season, moment, exist, index, end, start, duration, volume
PauseVideo VideoPlayer	id, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, volume, end, start, duration, playheadTime, exist, moment, season, year
PlayVideo ClipPlayer	id, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, volume, end, start, duration, playheadTime, exist, moment, season, year
PlayVideo Media	id, name, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, year, season, moment, exist, index, end, start, duration, volume
PlayVideo VideoInOut	id, name, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, year, season, moment, exist, index, end, start, duration, volume
PlayVideo VideoPlayer	id, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, volume, end, start, duration, playheadTime, exist, moment, season, year
PlayVideoList Selections	item(id, name, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, year, season, moment, exist, index, end, start, duration, volume)

Publish Clip	tags, image, username, description, mediatype, videoplaylist, textplaylist, audioplaylist1, audioplaylist2, title, debit, duration, filters, link
Random SearchMedia	kind, duration, rythme, tags
RemoveVideoItem Selections	item(id, name, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, year, season, moment, exist, index, end, start, duration, volume)
ResetFind UserMedia	text
SaveProject Selections	id
SearchProposition SearchMedia	kind, duration, rythme, tags
Select Subtitle	item, italic, size, bold, position, duration, end, start, font, color, text
Select UserMedia	id, title, link, image, tags, public, permanentLink, description, volume, end, start, duration, embedPlayer, sourceDrag, type, index
SelectTag SearchMedia	tag
SelectVideoItem Selections	item(id, name, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, year, season, moment, exist, index, end, start, duration, volume)
SimilarTags SearchMedia	kind, duration, rythme, tags
Sort Selections	item(id, name, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, year, season, moment, exist, index, end, start, duration, volume)
StopVideo ClipPlayer	id, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, volume, end, start, duration, playheadTime, exist, moment, season, year
StopVideo VideoInOut	id, name, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, year, season, moment, exist, index, end, start, duration, volume
StopVideo VideoPlayer	id, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, volume, end, start, duration, playheadTime, exist, moment, season, year
Subtitle Color	item, italic, size, bold, position, duration, end, start, font, color, text
Subtitle Duration	item, italic, size, bold, position, duration, end, start, font, color, text
Subtitle Font	item, italic, size, bold, position, duration, end, start, font, color, text

Subtitle Position	item, italic, size, bold, position, duration, end, start, font, color, text
Subtitle Size	item, italic, size, bold, position, duration, end, start, font, color, text
Toggle Trace	activated
Unmute VideoPlayer	id, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, volume, end, start, duration, playheadTime, exist, moment, season, year
Valid SearchMedia	kind, duration, rythme, tags
Volume ClipPlayer	id, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, volume, end, start, duration, playheadTime, exist, moment, season, year
Volume VideoPlayer	id, title, link, image, tags, sourceDrag, address(continent, country, region, town), themes, volume, end, start, duration, playheadTime, exist, moment, season, year

Table A.1: List of collected obsel-types from Wanaclip and their attributes.

Appendix B

Test Call and Queries for Evaluating Wanaclip

This appendix presents the test call e-mail that we have sent to the users of Wanaclip for evaluating the Wanaclip application. In addition, it shows also the "Executions Table" that is added to $\mathcal{T}$Store in order to collect statistics about recommendations. We also present some of queries that we used to analyze the results of the evaluation.

B.1 Test Call

We are researchers in computer science at LIRIS laboratory (UMR CNRS 5205). We conduct an evaluation of a recommendation service based on the reuse of user interactions. This study aims to validate our research results and to measure the quality and relevance of the proposed recommendations.

In collaboration with the company Webcastor, we are working on the Wanaclip application that allows users to compose video clips from different audio-visual sources. Our recommendation system has been integrated to Wanaclip to guide users in both the selection of videos, and the actions to perform in order to make quality clips.

This study takes places in two steps:

- **Usage of Wanaclip (between 5 and 10 minutes):** Run the application by clicking on the link `http:\www.wanaclip.eu`, you simply need to create an account, activate it using your email and log in. Unfortunately, only the French language is supported, however, it is still possible to run the application since it is not difficult to understand the services. Creating a video is easy: you start by searching with keywords and the system will display the related clips. After that you can select

the clips that interest you, add subtitles, upload your own media, *etc.* During your usage, recommendations will appear to guide you. Please, try to make different sessions (Refresh F5 or close/open the application page) to obtain better results.

- **Answer the survey (about 2 minutes):** at the end of your use of Wanaclip, it is important that you reply to all the questions. Your answers are anonymous and will only be for a statistical usage. To participate, please click on the following link: https://docs.google.com/forms/d/1MuE8lAk_d_2e5BG2HBJ78IMD3tol2Sf-27ZTj3MC_mA/viewform

We thank you in advance for your help. If you are interested, we will send you the results of the evaluation at the end of work.

Please spread this e-mail around you.

B.2 Queries for analyzig the results

We have added a new table to $\mathcal{T}$Store database (called: executions) to store statistics about recommendations. Each row of this table represents a request of recommendation. It contains fields about the used parameters of the algorithm, retrieval time, memory usage, size of the current episode, number of retrieved episodes, number of recommended interactions, *etc.* The creating command is:

```
CREATE TABLE Executions(
   id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
   time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   traceID INT NOT NULL REFERENCES Trace (id),
   executionTime DECIMAL(5,2),
   memoryUsage INT,
   nbObsels INT,
   nbTraces INT,
   nbSuccessors INT,
   threshold VARCHAR(10),
   alpha DECIMAL(5,2),
   beta DECIMAL(5,2),
   gamma DECIMAL(5,2),
   delta DECIMAL(5,2)
);
```

To get all the selected (accepted) obsels from a set of recommended obsels shown to the user $\sum_{\tau,o} \mathfrak{s}(\tau, o)$ and their sources, the following SQL query is used:

```
SELECT Obs.*, AttVal.value
FROM `attributetype` as AttType,
     `attributevalue` as AttVal,
     `obsel` as Obs
WHERE AttType.id = AttVal.attid
      AND Obs.id = AttVal.obselID
      AND AttType.name LIKE "recommendation";
```

To get the total number of the selected (accepted) obsels from a set of recommended obsels shown to the user $\sum_{\tau,o} \mathfrak{s}(\tau, o)$ and their sources, the following SQL query is used:

```
SELECT COUNT(*)
FROM `attributetype` as AttType,
     `attributevalue` as AttVal
WHERE AttType.id = AttVal.attid
      AND AttType.name LIKE "recommendation";
```

To get the number of recommendation requests ($\mathfrak{q}$), the following SQL query is used:

```
SELECT COUNT(*)
FROM `executions`;
```

To get the number of recommendation responses ($\mathcal{P}$), the following SQL query is used:

```
SELECT COUNT(*)
FROM `executions`
WHERE nbSuccessors > 0;
```

To get the total number of recommended obsels ($\mathfrak{R}$), the following SQL query is used:

```
SELECT SUM( nbRecObsel )
FROM `executions`;
```

To get all recommendation requests over time, the following SQL query is used:

```
SELECT (ROUND(traceID/10)) AS traceSlice, COUNT(*) AS request
FROM ‘executions‘
GROUP BY (ROUND(traceID/10));
```

To get all recommendation responses over time, the following SQL query is used:

```
SELECT (ROUND(traceID/10)) AS traceSlice, COUNT(*) AS response
FROM ‘executions‘
WHERE nbSuccessors > 0
GROUP BY (ROUND(traceID/10));
```

To get all recommendation requests per episode size (*i.e,* number of obsels), the following SQL query is used:

```
SELECT nbObsels, COUNT(*) AS request
FROM ‘executions‘
GROUP BY nbObsels;
```

References

Aamodt, A. and Plaza, E. (1994). Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *System*, 7(1):39–59. (Cited on pages xiii, 28, 30 and 31.)

Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., and Steggles, P. (1999). Towards a better understanding of context and context-awareness. pages 304–307. (Cited on page 21.)

Adeyanju, I., Wiratunga, N., Lothian, R., Sripada, S., and Lamontagne, L. (2009). Case Retrieval Reuse Net (CR2N): An Architecture for Reuse of Textual Solutions. In *ICCBR 2009*, volume 5650 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. (Cited on page 86.)

Adomavicius, G. and Tuzhilin, A. (2008). Context-aware recommender systems. *Proceedings of the 2008 ACM conference on Recommender systems RecSys 08*, 16(RecSys):335. (Cited on page 109.)

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843. (Cited on page 91.)

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410. (Cited on page 85.)

Anand, S. S. and Mobasher, B. (2007). From Web to Social Web: Discovering and Deploying User and Content Profiles. chapter Contextual Recommendation, pages 142–160. Springer-Verlag, Berlin, Heidelberg. (Cited on pages 5, 109 and 110.)

Anupam, V., Freire, J., Kumar, B., and Lieuwen, D. (2000). Automating Web navigation with the WebVCR. *Computer Networks*, 33(1-6):503–517. (Cited on pages 57 and 65.)

Armstrong, R., Freitag, D., Joachims, T., and Mitchell, T. (1995). Webwatcher: A learning apprentice for the world wide web. pages 6–12. AAAI Press. (Cited on page 126.)

Baccigalupo, C. and Plaza, E. (2007). A case-based song scheduler for group customised radio. *Lecture Notes in Computer Science*, 4626:433–448. (Cited on page 111.)

Baluja, S., Seth, R., Sivakumar, D., Jing, Y., Yagnik, J., Kumar, S., Ravichandran, D., and Aly, M. (2008). Video suggestion and discovery for youtube. In *Proceeding of the 17th international conference on World Wide Web - WWW'08*, page 895, New York, New York, USA. ACM Press. (Cited on page 108.)

Batley, J. and Edwards, D. (2009). Genome sequence data: management, storage, and visualization. *Biotechniques*, 46(5):333–334, 336. (Cited on page 57.)

Bergman, L., Castelli, V., Lau, T., and Oblinger, D. (2005). DocWizards. In *Proceedings of the 18th annual ACM symposium on User interface software and technology - UIST '05*, page 191, New York, New York, USA. ACM Press. (Cited on page 21.)

Bétrancourt, M., Guichon, N., and Prié, Y. (2011). Assessing the use of a Trace-Based Synchronous Tool for distant language tutoring. In *Computer Supported Collaborative Learning 2011*, pages 486–493. (Cited on page 35.)

Bigham, J. P., Lau, T., and Nichols, J. (2009). Trailblazer: enabling blind users to blaze trails through the web. In *Proceedings of the 14th international conference on Intelligent user interfaces*, IUI '09, pages 177–186, New York, NY, USA. ACM. (Cited on page 21.)

Briand, L. C., Labiche, Y., and O'Sullivan, L. (2003). Impact analysis and change management of UML models. In *International Conference on Software Maintenance 2003 ICSM 2003 Proceedings*, pages 256–265. IEEE Comput. Soc. (Cited on page 66.)

Brusilovsky, P. (1996). Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 6(2-3):87–129. (Cited on page 106.)

Burke, R. (2007). Hybrid Web Recommender Systems. *The Adaptive Web*, 4321(The adaptive web):377 – 408. (Cited on pages 5, 105 and 108.)

Capobianco, A. and Carbonell, N. (2001). Contextual online help: elicitation of human experts' strategies. *Proceedings of HCI*, 1:266—-270. (Cited on page 4.)

Capobianco, A. and Carbonell, N. (2007). Aides en ligne à l'utilisation de logiciels grand public : problèmes spécifiques de conception et solutions potentielles. *Intellectica*, 44:87–120. (Cited on page 22.)

Chakraborti, S., Lothian, R., Wiratunga, N., Orecchioni, A., and Watt, S. (2006). Fast Case Retrieval Nets for Textual Data. *Advances in Case-Based Reasoning*, pages 400–414. (Cited on page 117.)

Champin, P.-A., Cordier, A., Lavoué, E., Lefevre, M., and Mille, A. (2012a). Traces, Assistance and Communities, a review. Technical Report RR-LIRIS-2012-006, LIRIS UMR 5205 CNRS/INSA de Lyon/Université Claude Bernard Lyon 1/Université Lumiere Lyon 2/Ecole Centrale de Lyon. (Cited on page 35.)

Champin, P.-A., Cordier, A., Lavoué, E., Lefevre, M., and Skaf-Molli, H. (2012b). User assistance for collaborative knowledge construction. In *Proceedings of the 21st international conference companion on World Wide Web - WWW '12 Companion*, page 1065, New York, New York, USA. ACM Press. (Cited on pages 24 and 34.)

Champin, P.-A., Mille, A., and Prié, Y. (2013). Vers des traces numériques comme objets informatiques de premier niveau : une approche par les traces modélisées. *Intellectica*, (59). (Cited on pages 23, 25, 39 and 57.)

Champin, P.-A., Prié, Y., and Mille, A. (2004). MUSETTE: a framework for Knowledge from Experience. In *EGC'04, RNTI-E-2*, pages 129–134. Cepadues Edition. (Cited on pages 25, 45 and 48.)

Cordier, A. (2008). *Interactive and Opportunistic Knowledge Acquisition in Case-Based Reasoning*. Thèse de doctorat en informatique, Université Lyon 1. (Cited on pages xiii and 28.)

Cordier, A., Fuchs, B., and Mille, A. (2006). Engineering and Learning of Adaptation Knowledge in Case-Based Reasoning. In Staab, S. and Svatek,

V., editors, *15th International Conference on Knowledge Engineering and Knowledge Management EKAW'06*, LNAI, pages 303–317. Springer-Verlag. (Cited on pages xiii, 31 and 32.)

Cordier, A., Lefevre, M., Champin, P.-A., Georgeon, O., and Mille, A. (2013). Trace-Based Reasoning — Modeling interaction traces for reasoning on experiences. In *The 26th International FLAIRS Conference*. (Cited on pages 6, 31, 32, 34, 104 and 116.)

Cordier, A., Lefevre, M., Jean-Daubias, S., and Guin, N. (2010). Concevoir des assistants intelligents pour des applications fortement orientées connaissances : problématiques, enjeux et étude de cas. In Després, S., editor, *IC 2010 - 21emes Journées Francophones d'Ingénierie des Connaissances*, pages 119–130. Presses des Mines. (Cited on pages 5 and 23.)

Cordier, A., Mascret, B., and Mille, A. (2009). Extending Case-Based Reasoning with Traces. In *Grand Challenges for reasoning from experiences, Workshop at IJCAI'09*. (Cited on pages 31 and 34.)

Corvaisier, F., Mille, A., and Pinon, J.-M. (1997). Information retrieval on the World Wide Web using a decision making system. In *RIAO*, pages 284–296. (Cited on page 30.)

Cram, D., Fuchs, B., Prié, Y., and Mille, A. (2008). An approach to User-Centric Context-Aware Assistance based on Interaction Traces. In *MRC 2008, Modeling and Reasoning in Context*. (Cited on pages 22, 23 and 34.)

Cuturi, M., Vert, J.-P., Birkenes, O., and Matsui, T. (2006). A Kernel for Time Series Based on Global Alignments. *2007 IEEE International Conference on Acoustics Speech and Signal Processing ICASSP 07*, 2(i):II–413–II–416. (Cited on page 85.)

Damashek, M. (1995). Gauging Similarity with n-Grams: Language-Independent Categorization of Text. *Science*, 267(5199):843–848. (Cited on page 85.)

Davidson, J., Liebald, B., Liu, J., Nandy, P., Van Vleet, T., Gargi, U., Gupta, S., He, Y., Lambert, M., Livingston, B., and Sampath, D. (2010). The

YouTube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 293–296, New York, NY, USA. ACM. (Cited on page 108.)

Dietterich, T. G. (2012). TaskTracer project http://tasktracer.osuosl.org/. (Cited on page 67.)

Dirac, P. (1958). *Principles of quantum mechanics*. Oxford at the Clarendon Press. (Cited on page 127.)

Dong, R., McCarthy, K., O'Mahony, M., Schaal, M., and Smyth, B. (2012). Towards an intelligent reviewer's assistant. In *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces - IUI '12*, page 159, New York, New York, USA. ACM Press. (Cited on page 21.)

Dong, R., Schaal, M., O'Mahony, M., McCarthy, K., and Smyth, B. (2013). Opinionated Product Recommendation. In Delany, S. and Onta??n, S., editors, *Case-Based Reasoning Research and Development*, volume 7969 of *Lecture Notes in Computer Science*, pages 44–58. Springer Berlin Heidelberg. (Cited on page 111.)

Dufresne, A. and PromTep, S. (2006). ExploraGraph et la personnalisation des interactions pour l'apprentissage. In *Proceedings of the 18th international conference on Association Francophone d'Interaction HommeMachine IHM 06*, pages 153–157. ACM Press. (Cited on page 4.)

Dyke, G. (2009). Un modèle pour la gestion et la capitalisation d'analyses de traces d'activités en interaction collaborative. Technical report. (Cited on page 66.)

Egyed-Zsigmond, E. (2003). Gestion des connaissances dans une base de documents multimédias. *Thèse. Institut National des Sciences Appliquées de Lyon*. (Cited on page 18.)

Fayyad, U. M., Piatetsky-Shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery: an overview. In Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., editors, *Advances in Knowledge Discovery and Data Mining*, chapter 1, pages 1–34. American Association for Artificial Intelligence. (Cited on page 34.)

Ferraiolo, D. F., Kuhn, D. R., and Chandramouli, R. (2003). *Role-Based Access Control*, volume 2002 of *Computer Security Series*. Artech House. (Cited on pages 40 and 50.)

Gamewise (2012). Candy Crush Saga - Wiki Guide | Gamewise. (Cited on page 21.)

Gapenne, O. (2006). Relation d'aide et transformation cognitive. *Intellectica*, 44:7–16. (Cited on page 18.)

Georgeon, O. L., Mille, A., Bellet, T., Mathern, B., and Ritter, F. E. (2012). Supporting activity modelling from activity traces. *Expert Systems*, 29(3):261–275. (Cited on page 34.)

Ginon, B., Champin, P.-A., and Jean-Daubias, S. (2013a). Collecting fine-grained use traces in any application without modifying it. In *workshop EXPPORT from the conference ICCBR*. (Cited on page 57.)

Ginon, B., Jean-Daubias, S., and Champin, P.-A. (2013b). Adjonction de systèmes d'assistance personnalisée à des EIAH existants. (Cited on page 35.)

Ginon, B., Jean-Daubias, S., and Champin, P.-A. (2013c). Mise en place d'un système d'assistance personnalisée dans une application existante. In *Ingénierie des connaissances*. (Cited on page 34.)

Ginon, B., Jean-Daubias, S., and Champin, P.-A. (2013d). Une typologie de l'assistance aux utilisateurs : exemple d'application aux EIAH. Technical Report RR-LIRIS-2013-007, LIRIS UMR 5205 CNRS/INSA de Lyon/Université Claude Bernard Lyon 1/Universite Lumiere Lyon 2/Ecole Centrale de Lyon. (Cited on page 4.)

GÖKER, M. H., SMYTH, B., BRIDGE, D., and McGINTY, L. (2005). Case-based recommender systems. (Cited on pages 107 and 110.)

Golbeck, J. (2006). Generating Predictive Movie Recommendations from Trust in Social Networks. In *Proceedings of the 4th International Conference on Trust Management*, volume 3986, pages 93–104. (Cited on page 108.)

Greene, J. (2011). Google's Buzz kill completes shift to Google+ | Internet & Media - CNET News. (Cited on page 22.)

Groh, G. and Ehmig, C. (2007). Recommendations in taste related domains. In *Proceedings of the 2007 international ACM conference on Conference on supporting group work - GROUP '07*, page 127, New York, New York, USA. ACM Press. (Cited on page 110.)

Gundersen, O. E. (2012). Toward Measuring the Similarity of Complex Event Sequences in Real-Time. In *ICCBR*, pages 107–121. (Cited on pages 34 and 87.)

Haas, J., Maus, H., Schwarz, S., and Dengel, A. (2010). ConTask - Using Context-sensitive Assistance to Improve Task-oriented Knowledge Work. In *ICEIS (2)'10*, pages 30–39. (Cited on page 57.)

Hamming, R. W. (1950). Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160. (Cited on page 84.)

Harrington, R. (2009). *Understanding Adobe Photoshop CS4 The Essential Techniques for Imaging Professionals*. Peachpit Press. (Cited on page 66.)

Hernández del Olmo, F. and Gaudioso, E. (2008). Evaluation of recommender systems: A new approach. *Expert Systems with Applications*, 35(3):790–804. (Cited on page 126.)

Hupp, D. and 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*, pages 81–90. (Cited on pages 65 and 79.)

Jaczynski, M. and Trousse, B. (1999). Broadway: A case-based system for cooperative information browsing on the world-wide-web. pages 264–283. (Cited on page 29.)

Jameson, A. and Smyth, B. (2007). Recommendation to groups. pages 596–627. (Cited on page 111.)

Jr., W. S. and White, E. B. (1979). *The Elements of Style, Third Edition*. Macmillan. (Cited on page 14.)

Kang, H., Plaisant, C., and Shneiderman, B. (2003). New approaches to help users get started with visual interfaces: multi-layered interfaces and integrated initial guidance. pages 1–6. (Cited on page 21.)

Kapetanakis, S. and Petridis, M. (2014). Evaluating a Case-Based Reasoning Architecture for the Intelligent Monitoring of Business Workflows. In Montani, S. and Jain, L. C., editors, *Successful Case-based Reasoning Applications-2*, volume 494 of *Studies in Computational Intelligence*, pages 43–54. Springer Berlin Heidelberg, Berlin, Heidelberg. (Cited on page 86.)

Kapetanakis, S., Petridis, M., Ma, J., Knight, B., and Bacon, L. (2011). Enhancing similarity measures and context provision for the intelligent monitoring of business processes in CBR-WIMS. In *Process-oriented Case-Based Reasoning Workshop (PO-CBR)*. (Cited on page 86.)

Kelleher, C. and Pausch, R. (2005). Stencils-based tutorials. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '05*, page 541, New York, New York, USA. ACM Press. (Cited on page 21.)

Kietzmann, J. H., Hermkens, K., McCarthy, I. P., and Silvestre, B. S. (2011). Social media? get serious! understanding the functional building blocks of social media. *Business Horizons*, 54(3):241 – 251. (Cited on page 90.)

King, I., Lyu, M. R., and Ma, H. (2010). Introduction to social recommendation. *Proceedings of the 19th international conference on World wide web WWW 10*, page 1355. (Cited on page 110.)

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480. (Cited on page 117.)

Krzysztof Dembczynski, W. K. (2008). Effective Prediction of Web User Behaviour with User-Level Models. *Fundam. Inform.*, 89:189 – 206. (Cited on page 30.)

Lave, J. and Wenger, E. (1991). *Situated learning: Legitimate peripheral participation*, volume 95 of *Learning in doing*. Cambridge University Press. (Cited on page 4.)

Leake, D. B. (2010). Case-Based Reasoning Tomorrow: Provenance, the Web, and Cases in the Future of Intelligent Information Processing. In *Intelligent Information Processing*, page 1. (Cited on page 34.)

Leplat, J. (1998). À propos des procédures. Performances humaines et techniques. 94:6–15. (Cited on page 22.)

Leshed, G., Haber, E. M., Matthews, T., and Lau, T. (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*, pages 1719–1728. (Cited on pages 57 and 65.)

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710. (Cited on page 84.)

Li, I., Nichols, J., Lau, T., Drews, C., and Cypher, A. (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*, pages 723–732. ACM. (Cited on page 65.)

Lieberman, H. (1997). Autonomous interface agents. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '97*, pages 67–74, New York, New York, USA. ACM Press. (Cited on page 19.)

Lieberman, H. and Maulsby, D. (1996). Instructible agents: Software that just keeps getting better. *IBM Systems Journal*, 35(3.4):539–556. (Cited on page 19.)

Lieberman, H. and Selker, T. (2000). Out of context: Computer systems that adapt to, and learn from, context. (Cited on page 109.)

Linden, G. D., Jacobi, J. A., and Benson, E. A. (2001). Collaborative recommendations using item-to-item similarity mappings. (Cited on page 107.)

Lipkus, A. H. (1999). *A proof of the triangle inequality for the Tanimoto distance*, volume 26. (Cited on page 91.)

Lipman, D. J. and Pearson, W. R. (1985). Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441. (Cited on page 85.)

Little, G., Lau, T., Cypher, A., Lin, J., Haber, E., and Kandogan, E. (2007). Koala: capture, share, automate, personalize business processes on the web. (Cited on page 65.)

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C. (2002). Text Classification using String Kernels. *Journal of Machine Learning Research*, 2(3):419–444. (Cited on page 85.)

Lund, K. and Mille, A. (2009). *Analyse de traces et personnalisation des environnements informatiques pour l'apprentissage humain*, chapter Traces, traces d'interactions, traces d'apprentissages : définitions, modèles informatiques, structurations, traitements et usages , pages 21–66. (Cited on page 24.)

Maarten Grachten, J.-L. A. (2004). Melodic Similarity: Looking for a Good Abstraction Level. (Cited on page 86.)

Mabroukeh, N. R. and Ezeife, C. I. (2010). A taxonomy of sequential pattern mining algorithms. *ACM Computing Surveys*, 43(1):1–41. (Cited on page 115.)

Mahmood, T. and Ricci, F. (2007). Towards learning user-adaptive state models in a conversational recommender system. *Proceedings of the 15th Workshop on Adaptivity and User Modeling in Interactive Systems, ABIS*, 7:373–378. (Cited on page 107.)

Martín, F. J. and Plaza, E. (2004). Ceaseless Case-Based Reasoning. pages 287 – 301. (Cited on page 29.)

Mathern, B. (2012). *Découverte interactive de connaissances à partir de traces d'activité : Synthèse d'automates pour l'analyse et la modélisation de l'activité de conduite automobile*. Thèse de doctorat en informatique, Université Claude Bernard Lyon 1. (Cited on pages 24, 48 and 57.)

Mathern, B., Bellet, T., and Mille, A. (2010). An Iterative Approach to Develop a Cognitive Model of the Driver for Human Centred Design of ITS. In *European Conference on Human Centred Design for Intelligent Transport Systems*, Proceedings of European Conference on Human Centred Design

for Intelligent Transport Systems, pages 85–95. HUMANIST publications. (Cited on page 34.)

Mathern, B., Mille, A., Cordier, A., Cram, D., and Zarka, R. (2012). Towards a Knowledge-Intensive and Interactive Knowledge Discovery Cycle. In Luc Lamontagne, J. A. R.-G., editor, *20th ICCBR Workshop Proceedings*, pages 151–162. (Cited on page 16.)

Matthews, M., Pharr, W., Biswas, G., and Neelakandan, H. (2000). USCSH: An Active Intelligent Assistance System. *Artificial Intelligence Review*, 14(1-2):121–141. (Cited on page 4.)

Maule, A. (2010). *Impact analysis of database schema changes*. PhD thesis, ACM New York, NY, USA. (Cited on page 66.)

Metzler, D., Dumais, S., and Meek, C. (2007). Similarity Measures for Short Segments of Text. *Advances in Information Retrieval*, 4425:16–27. (Cited on page 85.)

Microsoft_Office (2010). Premiers pas avec Microsoft Office 2010. (Cited on page 4.)

Mille, A. (2006). Traces Based Reasoning (TBR) Definition, illustration and echoes with story telling. Technical Report RR-LIRIS-2006-002. (Cited on page 30.)

Mille, A., Caplat, M., and Philippon, M. (2006). Faciliter les activités des utilisateurs d'environnements informatiques : quoi, quand, comment ? *Intellectica*, 2(44):121–143. (Cited on page 4.)

Mille, A., Fuchs, B., and Chiron, B. (1999). Raisonnement fondé sur l'expérience : un nouveau paradigme en supervision industrielle. *Revue d'intelligence artificielle*, pages 97–128. (Cited on page 86.)

Minor, M., Bergmann, R., and Görg, S. (2012a). Case-based adaptation of workflows. *Information Systems*. (Cited on page 34.)

Minor, M., Islam, S., and Schumacher, P. (2012b). Confidence in Workflow Adaptation. In *ICCBR'12*. (Cited on page 86.)

Minsky, M. (1975). A Framework for Representing Knowledge. In Winston, P., editor, *The Psychology of Computer Vision*, pages 211–277. McGraw-Hill, New York. (Cited on page 2.)

Mohri, M. (1997). Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23(2):269–311. (Cited on page 49.)

Montani, S. and Leonardi, G. (2012). Retrieval and clustering for supporting business process adjustment and analysis. *Information Systems*. (Cited on page 86.)

Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453. (Cited on pages 84 and 85.)

Obweger, H., Suntinger, M., Schiefer, J., and Raidl, G. (2010). Similarity searching in sequences of complex events. In *2010 Fourth International Conference on Research Challenges in Information Science (RCIS)*, pages 631–640. IEEE. (Cited on page 87.)

Oxford Dictionary (2013). Trace: definition of trace in Oxford dictionary (British & World English). (Cited on page 23.)

Paquette, G., Pachet, F., and Giroux, S. (1994). Pitalk, un outil générique pour la construction de systèmes conseillers. *Sciences et techniques éducatives*, 1(3). (Cited on page 4.)

Pazzani, M. J. and Billsus, D. (2007). Content-Based Recommendation Systems. *The adaptive web*, 4321(2):325 – 341. (Cited on pages 4 and 106.)

Pesot, J., Antley, J., and Allsbrook, A. (2008). Dynamic user assistance in Eclipse-based applications. (Cited on page 17.)

Quijano-Sánchez, L., Bridge, D., Díaz-Agudo, B., and Recio-García, J. (2012). A Case-Based Solution to the Cold-Start Problem in Group Recommenders. In Agudo, B. and Watson, I., editors, *Case-Based Reasoning Research and Development*, volume 7466 of *Lecture Notes in Computer Science*, pages 342–356. Springer Berlin Heidelberg. (Cited on page 111.)

Rajaraman Anand and Ullman., J. D. (2011). *Mining of Massive Datasets - Recommendation Systems*. Cambridge University Press. (Cited on page 106.)

Rech, J., Ras, E., and Decker, B. (2007). Intelligent Assistance in German Software Development: A Survey. *IEEE Software*, 24(4):72–79. (Cited on pages xiii, 19 and 20.)

Resnick, P. and Varian, H. R. (1997). Recommender systems. *Communications of the ACM*, 40(3):56–58. (Cited on pages 5 and 105.)

Ricci, F. (2010). The context of a recommendation. In *Proceedings of the Workshop on Context-Aware Movie Recommendation - CAMRa '10*, page 1, New York, New York, USA. ACM Press. (Cited on pages 5 and 109.)

Ricci, F., Arslan, B., Mirzadeh, N., and Venturini, A. (2002). ITR: A Case-Based Travel Advisory System. pages 613–627. (Cited on page 111.)

Ricci, F., Rokach, L., and Shapira, B. (2011). Introduction to Recommender Systems Handbook. In Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B., editors, *Recommender Systems Handbook*, chapter 1, pages 1–35. Springer US. (Cited on pages 105, 106 and 107.)

Richard, B. and Tchounikine, P. (2004). Une approche centrée modèle pour la construction d'un système conseiller pour un site Web. In *Actes de IC 2004*, pages 151–162. (Cited on page 4.)

Rieck, K. (2011). Similarity measures for sequential data. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(4):296–304. (Cited on page 84.)

Rieck, K. and Laskov, P. (2007). Language Models for Detection of Unknown Attacks in Network Traffic. *Journal in Computer Virology*, 2(4):243–256. (Cited on page 85.)

Roche, E. and Schabes, Y. (1997). *Finite-State Language Processing*, volume 75. MIT Press. (Cited on pages 40 and 48.)

Safonov, A., Konstan, J. A., and Carlis, J. V. (2001). Beyond Hard-to-Reach Pages: Interactive , Parametric Web Macros. *Proc Human Factors and the Web*, pages 1–14. (Cited on pages 57 and 65.)

Said, A., Berkovsky, S., and De Luca, E. W. (2010). Putting Things in Context: Challenge on Context-Aware Movie Recommendation. In *Proceedings of the Workshop on ContextAware Movie Recommendation*, CAMRa '10, pages 2–6. ACM. (Cited on page 110.)

Salton, G., Wong, A., and Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620. (Cited on page 85.)

Sànchez-Marrè, M., Cortés, U., Martínez, M., Comas, J., and Rodríguez-Roda, I. (2005). An Approach for Temporal Case-Based Reasoning: Episode-Based Reasoning. In *ICCBR*, pages 465–476. (Cited on page 86.)

Schafer, J. B., Konstan, J. A., and Riedl, J. (2001). E-Commerce Recommendation Applications. *Data Mining and Knowledge Discovery*, 5(1-2):115–153. (Cited on page 107.)

Schank, R. C. (1983). *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, New York, NY, USA. (Cited on pages 2 and 28.)

Sehaba, K. (2012). Système d'aide adaptatif à base de traces. *Revue internationale des technologies en pédagogie universitaire*, 9(3):55–70. (Cited on page 35.)

Selker, T. (1994). COACH: a teaching agent that learns. *Communications of the ACM*, 37(7):92–99. (Cited on pages 18 and 19.)

Serres, A. (2002). Quelle(s) problematique(s) de la trace ? Technical Report 12, http://archivesic.ccsd.cnrs.fr/sic_00001397/fr/. (Cited on page 23.)

Settouti, L., Prie, Y., Marty, J.-C., and Mille, A. (2009). A trace-based system for technology-enhanced learning systems personalisation. In *Advanced Learning Technologies, 2009. ICALT 2009. Ninth IEEE International Conference on*, pages 93–97. (Cited on page 35.)

Settouti, L. S. (2011). *M-Trace-Based Systems - Models and languages for exploiting interaction traces*. PhD thesis, University Lyon1. (Cited on pages 25 and 41.)

Smith, T. F. and Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197. (Cited on pages 84, 85, 86, 92 and 116.)

Su, X. and Khoshgoftaar, T. M. (2009). A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence*, 2009(Section 3):1–20. (Cited on pages 4 and 107.)

Sybase (2010). Power Designer: Impact and Lineage Analysis. (Cited on page 66.)

Symeonidis, P., Nanopoulos, A., and Manolopoulos, Y. (2008). Tag recommendations based on tensor dimensionality reduction. *Proceedings of the 2008 ACM conference on Recommender systems RecSys 08*, page 43. (Cited on page 110.)

The Free Dictionary (2013). Trace - definition of trace by the Free Online Dictionary, Thesaurus and Encyclopedia. (Cited on page 23.)

Valls, J. and Ontañón, S. (2012). Natural Language Generation through Case-Based Text Modification. In *ICCBR'12*, volume 7466 of *Lecture Notes in Computer Science*. (Cited on page 86.)

Van Der Aalst, W. M. P. (2009). Process-Aware Information Systems: Lessons to be Learned from Process Mining. *Transactions on Petri Nets and Other Models of Concurrency II*, 5460:1–26. (Cited on page 34.)

Verillon, P. and Rabardel, P. (1995). Cognition and artifacts: A contribution to the study of though in relation to instrumented activity. *European Journal of Psychology of Education*, 10(1):77–101. (Cited on page 19.)

Wang, X., Zhang, D. Q., Gu, T., and Pung, H. (2004). Ontology based context modeling and reasoning using owl. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, pages 18–22. (Cited on page 22.)

Watkins, C. (1999). Dynamic Alignment Kernels. *Advances in Large Margin Classifiers*, 00(January):39–50. (Cited on page 85.)

Weber, B. G. and Ontañón, S. (2010). Using Automated Replay Annotation for Case-Based Planning in Games. In *ICCBR 2010 workshop on CBR for Computer Games*, pages 15–24. (Cited on page 34.)

Wijaya, D. T. (2008). A Random Walk on the Red Carpet: Rating Movies with User Reviews and PageRank. *Methods*, pages 951–959. (Cited on page 110.)

Wikipedia (2013). Trace - Wikipédia. (Cited on page 23.)

WinSCP (2012). XML Logging in WinSCP. (Cited on page 56.)

Xu, K. and Muñoz Avila, H. (2007). CaBMA: a case-based reasoning system for capturing, refining, and reusing project plans. *Knowledge and Information Systems*, 15(2):215–232. (Cited on page 67.)

Yeh, T., Chang, T.-H., Xie, B., Walsh, G., Watkins, I., Wongsuphasawat, K., Huang, M., Davis, L. S., and Bederson, B. B. (2011). Creating contextual help for GUIs using screenshots. In *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*, page 145, New York, New York, USA. ACM Press. (Cited on page 21.)

Yujian, L. and Bo, L. (2007). A normalized levenshtein distance metric. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1091–1095. (Cited on page 85.)

Zapata-Rivera, J.-D. and Greer, J. E. (2004). Interacting with Inspectable Bayesian Student Models. *International Journal of Artificial Intelligence in Education*, 14(2):127–163. (Cited on page 4.)

Zarka, R., Champin, P.-A., Cordier, A., Egyed-Zsigmond, E., Lamontagne, L., and Mille, A. (2013a). TStore: A Trace-Base Management System using Finite-State Transducer Approach for Trace Transformation. In *MODELSWARD 2013*. SciTePress. (Cited on page 38.)

Zarka, R., Cordier, A., Corvaisier, F., and Mille, A. (2010). Providing assistance by reusing episodes stored in traces: a case study with SAP Business Objects Explorer. In Le Ber, F. and Renaud, J., editors, *Case-Based Reasoning Workshop*, pages 91–103, Strasbourg, France. hal-00497210. (Cited on pages 16, 62 and 74.)

Zarka, R., Cordier, A., Egyed-Zsigmond, E., Lamontagne, L., and Mille, A. (2013b). Similarity Measures to Compare Episodes in Modeled Traces. In Springer, editor, *International Case-Based Reasoning Conference (ICCBR 2013)*. (Cited on page 82.)

Zarka, R., Cordier, A., Egyed-Zsigmond, E., and Mille, A. (2011). Rule-based impact propagation for trace replay. In Ram, A. and Wiratunga, N., editors, *19th International Conference on Case Based Reasoning (ICCBR-2011)*, pages 482–495. Springer. (Cited on page 62.)

Zarka, R., Cordier, A., Egyed-Zsigmond, E., and Mille, A. (2012). Contextual Trace-Based Video Recommendations. In *21st international conference companion on World Wide Web (WWW-XperienceWeb'12)*, WWW '12 Companion, pages 751–754. ACM. (Cited on page 102.)

Zarka, R. and Mille, A. (2010). Providing assistance by reusing episodes stored in traces: a case study with SAP Business Objects Explorer. Master's thesis, INSA de Lyon. (Cited on page 8.)

Zimmermann, A. (2003). Context-awareness in user modelling: requirements analysis for a case-based reasoning application. In *Proceedings of the 5th international conference on Case-based reasoning: Research and Development*, ICCBR'03, pages 718–732, Berlin, Heidelberg. Springer-Verlag. (Cited on page 28.)

List of Symbols

$S_{obstype}$	Obsel-type similarity measure	, page 88
s_t	Start time-stamp	, page 42
$sim_{episode}$	Episode similarity measure	, page 94
$sim_{obsattr}$	Attribute similarity measure	, page 90
$sim_{obstime}$	Time-stamp similarity measure	, page 91
$sim_{obsuser}$	User similarity measure	, page 90
sim_{obs}	Obsel similarity measure	, page 88
T	Temporal domain	, page 41
u	User	, page 42
v	Visibility of $\mathcal{M}$-Trace	, page 42
CBR	Case-Based Reasoning	, page 28
FST	Finite-State Transducer	, page 48
kTBS	kernel for Trace-Based Systems	, page 39
$\mathcal{M}$-Trace	Modeled Trace	, page 25
TBA	Trace-Based Assistant	, page 28
TBMS	Trace Base Management System	, page 25
TBR	Trace-Based Reasoning	, page 30
TBS	Trace-Based System	, page 25
ERD	Entity-Relationship Diagram	, page 51
obsel	OBServed ELement	, page 25
RBAC	Role-Based Access Control	, page 50

FOLIO ADMINISTRATIF

THESE SOUTENUE DEVANT L'INSTITUT NATIONAL DES SCIENCES APPLIQUEES DE LYON

NOM : ZARKA Prénom : Raafat **DATE de SOUTENANCE** : 04/12/2013

TITRE : Trace-Based Reasoning for User Assistance and Recommendations

NATURE : Doctorat **Numéro d'ordre** : ——

Ecole doctorale : InfoMaths **Spécialité** : Informatique

RESUME :
Dans le domaine des environnements numériques, un enjeu particulier consiste à construire des systèmes permettant aux utilisateurs de partager et de réutiliser leurs expériences. Cette thèse s'intéresse à la problématique générale des recommandations contextuelles pour des applications web dans un contexte particulier : tâches complexes, beaucoup de données, différents types d'utilisateurs (du débutant au professionnel), *etc.* Les recommandations sont une forme d'assistance à l'utilisateur. Plus précisément, nous cherchons à fournir une assistance à l'utilisateur en prenant en compte le contexte et la dynamique des tâches que l'utilisateur effectue. On cherche à fournir des recommandations dynamques qui sont enrichies au fur et à mesure des expériences. Pour fournir ces recommandations dynamiques, nous nous appuyons sur le Raisonnement à Partir de l'Expérience Tracée (RàPET). Le RàPET est un paradigme d'intelligence artificielle relativement récent, qui tire son inspiration du Raisonnement à Partir de Cas (RàPC). Dans le RàPET, les traces d'interaction constituent d'importants conteneurs de connaissances. Ces traces permettent de mieux comprendre le comportement des utilisateurs et leurs activités. Par conséquent, elles représentent également le contexte de l'activité. Les traces peuvent donc venir nourrir un assistant à partir d'expérience en lui fournissant des connaissances appropriées.

Dans cette thèse, nous présentons un état de l'art sur les systèmes d'assistances dynamiques et nous rappelons les concepts généraux des systèmes à base de traces. Afin de proposer une assistance à base d'expérience, nous avons effectué plusieurs contributions. Tout d'abord, nous avons proposé une formalisation des traces modélisées et des processus qui permettent de manipuler ces traces. Nous avons notamment défini une méthode pour établir des mesures de similarité afin de comparer des traces modélisées. Nous avons implémenté ces propositions dans un outil appelé $\mathcal{T}$Store. Cet outil permet le stockage, la transformation, la gestion et la réutilisation des traces modélisées. Ensuite, nous avons proposé un mécanisme de rejouage de traces pour permettre aux utilisateurs de revenir à un état précédent de l'application. Ce mécanisme gère les conséquences de la propagation des changements lors du processus de rejouage. Enfin, nous avons décrit une approche de recommandations à partir de traces. Le moteur de recommandations est alimenté par les traces d'interactions laissée par les précédents utilisateurs de l'application. Ces traces sont stockées dans un gestionnaire de traces tel que $\mathcal{T}$Store. Cette approche facilite le partage de connaissances entre des communautés d'utilisateurs, et s'appuie, entre autres choses, sur les mesures de similarité proposées plus haut.

Nous avons validé nos contributions théoriques à l'aide de deux applications web : SAP BusinessObjects Explorer pour l'analyse de données, et Wanaclip pour la génération semi-automatique de clips vidéos. Le mécanisme de rejouage de traces est démontré dans SAP BusinessObjects Explorer. Les recommandations à base de traces sont illustrées dans l'application Wanaclip. Elles guident l'utilisateur à la fois dans la sélection des vidéos, et dans les actions à effectuer pour réaliser des clips vidéo de bonne qualité. Dans la dernière partie du manuscrit, nous mesurons les performances de $\mathcal{T}$Store et la qualité des recommandations et des mesures de similarité qu'il implémente. Nous discutons aussi des résultats du sondage que nous avons appliqué aux utilisateurs de Wanaclip pour mesurer leur satisfaction. Nos évaluations montrent que notre approche offre des recommandations satisfaisantes et un bon temps de réponse.

MOTS-CLES : Assistance Interactive, Recommandations Contextuelles, Mesures de Similarité, Rejouage de Traces, Traces Modélisées, Système à Base de Traces, Raisonnement à Partir de Cas, Raisonnement à Partir de Trace, Interaction Homme-Machine

Laboratoire (s) de recherche : Laboratoire d'InfoRmatique en Image et Systèmes d'information (LIRIS)

Directeur de thèse : Alain Mille, Amélie Cordier et Elöd Egyed-Zsigmond

Présidente de jury : Sylvie Calabretto

Composition du jury :

Directeurs : Alain Mille, Amélie Cordier et Elöd Egyed-Zsigmond

Rapporteurs : Agnar Aamodt et Serge Garlatti

Examinateurs : Sylvie Calabretto, Sylvie Després et Miltos Petridis

Invité : Luc Lamontagne