TStore: A Trace-Base Management System
Using Finite-state Transducer Approach for Trace Transformation

Raafat Zarka[1,2], Pierre-Antoine Champin[1,3], Amélie Cordier[1,3], Elöd Egyed-Zsigmond[1,2], Luc Lamontagne[4] and Alain Mille[1,3]
[1]*Université de Lyon, CNRS, Lyon, France*
[2]*INSA-Lyon, LIRIS, UMR5205, F-69621, Villeurbanne Cedex, France*
[3]*Université Lyon 1, LIRIS, UMR5205, F-69622, Villeurbanne Cedex, France*
[4]*Department of Computer Science and Software Engineering, Université Laval, Québec, G1K 7P4, Canada,*

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

Abstract: This paper presents TStore, a Trace-Base Management System (TBMS) handling storage, transformation and exploitation of traces collected by external applications. As the experiments reported in this paper demonstrate, TStore brings a solution to performances and storage issues usually encountered by TBMS. To exploit traces, transformations are used. TStore provides predefined transformation functions as well as a customized transformation based on Finite State Transducers (FST), which are also presented in this paper.

1 INTRODUCTION

Nowadays, many applications collect traces of their users' interactions. These traces can be used in many ways: analysis of users' behaviours, visualization of interactions, debugging, mining, etc. Storing these traces, and developing efficient mechanisms to exploit them, represents a considerable challenge. In this paper, we report on TStore, a Trace-Base Management System (Settouti, 2011) that addresses this challenge.

TStore was developed to solve trace management issues we had while working on an application called Wanaclip (www.wanaclip.eu). Wanaclip is a web application for generating video clips from different media: photos, videos, music and sounds. Users enter keywords, the system searches video sequences annotated with these keywords and lets the users drag them into a timeline in order to compose a video clip. Given the large amount of content available, the problem is to quickly find content that truly meets users' needs. Therefore, our goal was to develop a recommendation mechanism for this application. We decided to use a trace-based approach to provide contextual video recommendations. For that, we needed a TBMS. However, existing TBMS presented performance 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. After identifying the main problems, we designed TStore.

In this paper, first, we present the architecture of TStore. TStore is implemented as a web tool allowing agents (software systems or human users) to concurrently access, store and reuse traces issued from various applications. It is composed of four modules: Storage Manager, Querying System, Transformer and Security Manager. TStore manages M-Traces in a relational database and benefits from its storage and querying facilities. Experimentations reported at the end of this paper demonstrate TStore efficiency.

The second contribution described in this paper is the use of Finite-State Transducer (FST) principle (Roche and Schabes, 1997) to implement specific trace transformations. Trace transformation is a process allowing the transformation of existing traces in order to better exploit them. Transformations are manifold (filtering, segmentation, reformulation, etc.). Here, we show how we apply FST transformations to the recommendation problems we have in Wanaclip.

Zarka R., Champin P., Cordier A., Egyed-Zsigmond E., Lamontagne L. and Mille A..
TStore: A Trace-Base Management System - Using Finite-state Transducer Approach for Trace Transformation.
DOI: 10.5220/0004317301170122
In *Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2013), pages 117-122
ISBN: 978-989-8565-42-6

The remaining of the paper is organized as follows. Section0presents background knowledge on trace-based systems. TStore structure and functionalities are presented in Section 0. We report the implementation, experiments and performance study in Section0. We discuss the related work in Section0, and conclude our work in Section0.

2 BACKGROUND

An interaction trace is a rich record of the actions performed by a user on a system. Therefore, traces enable to capture users' experiences. Here, we focus on specific traces, called *modeled traces* (M-Traces). M-Traces differ from logs in the sense that they come with a model. An M-Trace includes both the sequence of temporally situated observed elements (*obsels*) which is the instantiated trace and the model of this trace which gives the semantics of obsels and the relations between them (Settouti, 2011).

The obsels are generated from the observation of the interaction between the user and the system. Each obsel has, at least, a type and two timestamps (begin and end). 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. A *trace base* is a collection of M-Traces.

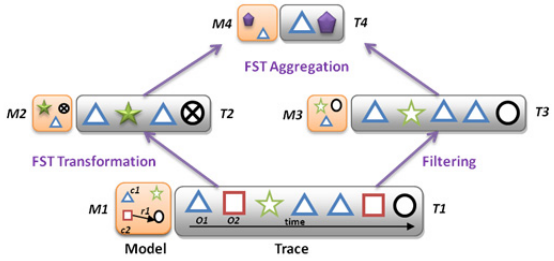

Figure 1: An example of M-Trace transformations.

A Trace-Base Management System (TBMS) is a tool for managing trace bases. A TBMS guarantees the possibility, at any time, to navigate between transformed M-traces. Figure 1 shows an M-Trace *T1* that is transformed into *T2* and *T3* (level 2). It also shows that *T2*, *T3* are transformed (merged) into *T4* (level 3). Each trace model contains different types of obsels that may have relations between them. For example, in the M-Trace *T1*, *M1* is its trace model that contains four obsel types (*c1*, *c2*, *c3*, *c4*) and one relation type *r1*. *T1* consists of seven obsels (*o1*, *o2*... *o7*).

3 TSTORE ARCHITECTURE AND SPECIFICATIONS

In this section, we describe the structure of TStore and its functionalities (see Figure 2). TStore is a TBMS for storing M-Traces collected by the Trace Collector from the client application (Wanaclip). In additions, TStore answers the queries of the assistant that observes the way the user interacts with the system, to help him doing his task effectively. TStore relies on a Database Management System (DBMS) and therefore benefits from performance and storage facilities. TStore contains four modules. The **Storage Manager** receives messages from external clients and stores them in the database in the form of M-Traces. The **Querying System** retrieves M-Traces from the database to answer queries of agents. The **Transformer** contains different functions to perform operations on M-Traces to produce transformed M-Traces. Finally, the **Security Manager** ensures M-Traces protection and the distribution of roles and privileges.

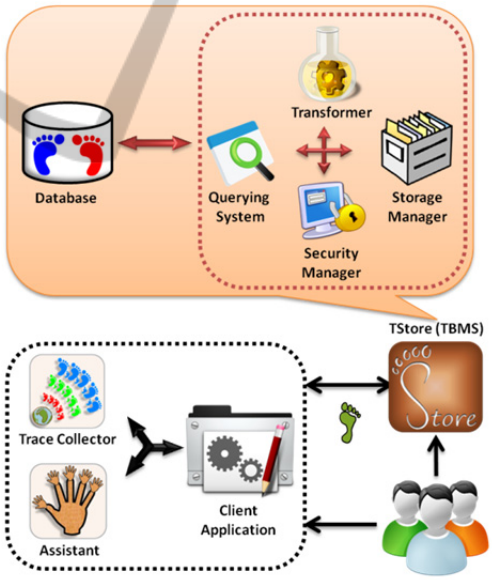

Figure 2: TStore general structure.

3.1 Storage Management

The Storage Manager is a module responsible for the communication between the M-Traces stored in the database and the client application connected to TStore. It contains services for creating models, storing M-Traces, obsels and their attribute values. External trace collectors send messages containing the collected M-Traces to the Storage Manager for

storing them in the database. The Storage Manager allows concurrent access, so multiple users can interact with the system and store or retrieve their M-Traces simultaneously.

Some objects cannot be created until the end of creation of all objects related to them. TStore schedule 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. TStore benefits from XML structure to do that.

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 it allows M-Trace collectors to store their M-Traces without defining all the details of their models, like in web logs.

3.2 M-Trace Transformation

The transformation of M-Traces helps to move from a first simple interpretation (almost raw data coming from sensors) to the actionable knowledge level of abstraction. The Transformer has predefined functions for frequently used transformations such as filtering, aggregation and segmentation. In these transformations, the transformed M-Trace preserves the same obsel types as the original M-Trace and it doesn't produce new ones. However, simple rules are not suitable for generating outputs. Therefore, we need a mechanism able to recognize and generate transformed M-Traces.

We propose to use a Finite-State Transducer (FST) transformation approach that allows defining customized transformations 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. As shown in Definition 1, transducers are automata that have transitions labeled with two symbols. One of the symbols represents input, the other is 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 1. A deterministic finite state transducer (DFST) is described as a 7-tuple $(Q, i, F, \Sigma, \Delta, \delta, \sigma)$ where:

- Q is the set of states,
- $i \in Q$ is the initial state,
- $F \subseteq Q$ is the set of final states,
- Σ and Δ are finite sets corresponding respectively to the input and output alphabets of the transducer,
- δ is the state transition function which maps $Q \times \Sigma$ to Q,
- σ is the output emission function which maps $Q \times \Sigma$ 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 $Q \times \Sigma \rightarrow Q \times \Delta$, mapping a pair of a state and an input symbol onto a pair of a state and an output symbol (Mohri, 1997).

In TStore, FST transformation consists in replacing some obsels matching the FST with more abstract obsels. Currently, it is experts that define the structure of the new obsel type. It is possible that two different transducers generate the same transformed M-Trace. However, FST should be deterministic to avoid ambiguity. By using FST we can apply a large variety of transformations. To create new FST transformation, the 7-tuple $(Q, i, F, \Sigma, \Delta, \delta, \sigma)$ described in Definition 1 should be defined. TStore creates new obsel types for all the output symbols of the transducer. These obsel types are contained in the transformed M-Trace model. Each transformation is represented by one transducer. TStore is able to make the output symbols of a transducer inputs for another transducer which represents the hierarchical M-Trace model.

When a new M-Trace comes, the relevant transducer is called depending on the predefined trace model. The transducer reads the current 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 TStore with the output symbols that represent the transformed M-Trace to store it.

Figure 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. We define a transducer as:
$T = (\{0,1,2,3\},0,\{3\},\{c,d,s,v\},\{a,i,s\},$
$\{(0,s,s,1),(1,v,\varepsilon,2),(2,d,a,3),(2,c,i,3)\})$

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*) so an obsel of type "accept" (*a*) is generated; (*d/a)* means replace (*d)* by (*a*). Else, if the obsel type is "close" (*c*) so 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.

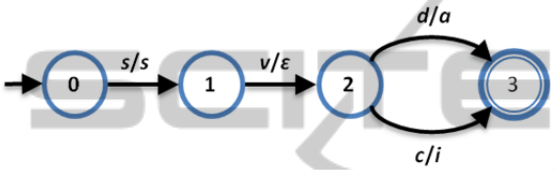

Figure 3: Example of a FST transformation.

3.3 M-Trace Querying

M-Traces are a large source of information. So, we need such a system that enables the extraction of episodes and patterns from M-Traces. TStore allow M-Traces to be retrieved at all levels and to navigate between transformed M-Traces. The Querying System contains some predefined methods allow M-Trace retrieval using different criteria. It can retrieve M-Traces for a specific user, in a specific period, contains a set of obsel types, etc. It provides statistics about M-Traces, obsels, users like their frequencies, relations, reusing, etc. Querying is a matter for our future work. We aim to define a querying language that is able to represent all the types of agents' queries.

3.4 Security Management

As 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 TStore implements a security policy. However, TStore ensures that only those with sufficient privileges can access the stored M-Traces. It protects the M-Traces by securing the underlying DBMS that holds that M-Traces. We use a role-based access control (RBAC) approach (Ferraiolo et al., 2003). 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 those role assignments acquire the privileges to perform particular functionalities like creating models, adding user, deleting M-Trace, etc. Each user is responsible of his M-Traces and he can also specify their visibility to be public, private or custom. Private M-Traces can be used only for statistics. Anyone can access and retrieve public M-Traces. It is possible to have privileges on any item such as M-Trace, obsel and attribute type. For example, video subtitle attributes can be hided from specific users.

4 EXPERIMENTATION

We have implemented TStore as a PHP web system over a MySql DBMS. TStore exchanges XML messages that have a predefined structure. One of the most important advantages of TStore is that it is independent of the client environment. Supporting client's APIs facilitate the generation the XML messages, but it is not mandatory.

In order to test our environment, we have implemented a collection process in Wanaclip. In this process, the M-Trace handler captures users' events, models them as obsels and stores them in TStore. We conducted tests to determine how TStore performs in terms of responsiveness, memory usage and stability under a workload. Both of TStore 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.

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, we 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 TStore spends to store an obsel in the database after receiving the message from the M-Trace collector. The messaging time is the time of exchanging a message between TStore and the M-Trace collector (sending + receiving). The average memory usage by to store a single obsel is 26.325 KB.

To examine the storage of multiple obsels, we tried 1000 random tests. At each test, we store a random number of obsels at once as a chunk. As a result, it takes on average 1.368 seconds (1.118 execution time + 0.250 messaging time) to store a

chunk. The average memory usage needed to store a chunk is 50.962 KB. Thus, we see that the multiple obsel mechanism reduces the require time for storage. This is due to the spent time for messaging, parsing command and inserting rows in the database.

As shown in Figure 4, the execution time and the memory usage for multiple obsels storage have logarithmic growth. The higher the number of obsels stored, it increases the memory usage and the required execution time. However, messaging time has very little 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 number of attributes need more time to be stored.

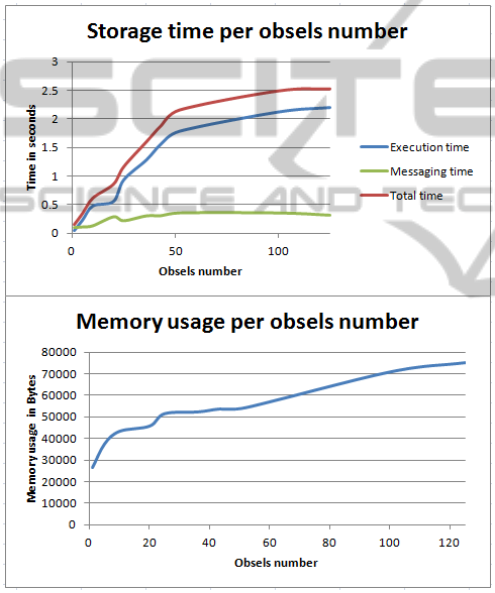

Figure 4: Storage time and memory usage per obsels number

5 RELATED WORK

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 chronological order and they are plain text files. Most often, one line of text corresponds to one log 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 well and truly wrong. For example, WinSCP uses XML logging to find a list of files that were actually uploaded/downloaded and Record operations done during synchronization ("WinSCP," 2012).

In the web, 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 a mixed approach. In all cases, users' activities are usually recorded as web logs that contain mainly the visited links. CoScripter (Leshed et al., 2008) is a Firefox plug-in created by an IBM Research group. 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 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).

The Kernel for Trace-Based Systems (kTBS) was the first TBMS developed in Python (Champin et al., 2011). Both kTBS and TStore implement the M-Trace concept. kTBS uses RDF files to store M-Traces, while TStore manages M-Traces in a database and benefits from its functionalities. In TStore, it is not important to develop APIs for client application while in kTBS it is important. kTBS currently support APIs for Java, PHP and Flex. kTBS stores obsels separately while TStore handles several obsels together to reduce network traffic. kTBS supports different message formats like, JSON, XML, and Turtle. Currently TStore only supports XML but we hope to support other formats. kTBS and TStore has predefined transformations, in addition, TStore supports FST transformations. kTBS transformations are filtering, fusion and SPARQLrule.

6 CONCLUSIONS AND FUTURE WORK

In this paper we described TStore, a web tool serving as a Trace-Base Management Systems. The originality of TStore resides on its performance and in the transformation facilities on M-Traces that it offers. We presented the four modules of TStore. The Storage Manager receives messages containing M-Traces from the clients and stores them in the database. The Querying System retrieves M-Traces from the database to answer queries of client applications. The Transformer contains different functions to produce transformed M-Traces. The Security Manager ensures M-Trace protection and the distribution of roles and privileges. The major contribution described in this paper is the customized transformation approach based on the Finite-State Transducer (FST) principle for M-Trace transformations. We proposed to use FST as a way to represent signatures of users' behaviours.

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. The implementation of TStore 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 develop our FST Transformation approach and try to define them automatically. A user interface is one of the important things to be provided since it allows users and admin to browse and manage M-Traces according to their privileges. Currently TSore only supports XML messages so we want to add new formats. Lastly, we want to develop a visualization module that helps to view and analyze M-Traces.

REFERENCES

Anupam, V., Freire, J., Kumar, B., & Lieuwen, D. (2000). Automating Web navigation with the WebVCR. *Computer Networks*, *33*(1-6), 503–517. doi:10.1016/S1389-1286(00)00073-6

Batley, J., & Edwards, D. (2009). Genome sequence data: management, storage, and visualization. *Biotechniques*, *46*(5), 333–334, 336. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/19480628

Berstel, J., & Reutenauer, C. (1988). *Rational Series and Their Languages*. *Springer-Verlag: Berlin-New York*. Springer-Verlag: Berlin-New York. Retrieved from http://www.springer.com/computer/theoretical+computer+science/book/978-3-642-73237-9

Champin, P.-A., Prie, Y., Aubert, O., Conil, F., & Cram, D. (2011). kTBS: Kernel for Trace-Based Systems. Retrieved from http://liris.cnrs.fr/publis/?id=5478

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

Ferraiolo, D. F., Kuhn, D. R., & Chandramouli, R. (2003). *Role-Based Access Control*. *Components* (Vol. 2002, p. 338). Artech House. doi:10.1016/S1361-3723(02)01211-3

Haas, J., Maus, H., Schwarz, S., & Dengel, A. (2010). ConTask - Using Context-sensitive Assistance to Improve Task-oriented Knowledge Work. *ICEIS (2)'10* (pp. 30–39).

Kuich, W., & Salomaa, A. (1986). *Semirings, automata, languages* (Vol. 5, p. v+374). Berlin: Springer-Verlag.

Leshed, G., Haber, E. M., Matthews, T., & 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*, 1719–1728. Retrieved from http://portal.acm.org/citation.cfm?id=1357054.1357323

Mohri, M. (1997). Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, *23*(2), 269–311. Retrieved from http://www.aclweb.org/anthology/J97-2003.pdf

Mohri, M. (2004). Weighted Finite-State Transducer Algorithms. An Overview. (C. Martín-Vide, V. Mitrana, & G. Paun, Eds.)*Studies In Fuzziness And Soft Computing*, *148*, 1–13. Retrieved from http://bi.snu.ac.kr/SEMINAR/ISMB2005/tutorials/AM5-handout.pdf

Roche, E., & Schabes, Y. (1997). *Finite-State Language Processing*. (E. Roche & Y. Schabes, Eds.)*Language* (Vol. 75, p. 850). MIT Press. doi:10.2307/417760

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

Settouti, L. S. (2011). *M-Trace-Based Systems - Models and languages for exploiting interaction traces*. University Lyon1. Retrieved from http://liris.cnrs.fr/Documents/Liris-4984.pdf

WinSCP. (2012). Retrieved from http://winscp.net/eng/docs/logging_xml