TStore: A Web-Based System for Managing, Transforming and Reusing Traces

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
[2] INSA-Lyon, LIRIS, UMR5205, F-69621, France
[3] Université Lyon 1, LIRIS, UMR5205, F-69622, France
[4] Department of Computer Science and Software Engineering,
Université Laval, Québec, Canada, G1K 7P4

{raafat.zarka, pierre-antoine.champin, amelie.cordier,
elod.egyed-zsigmond, alain.mille}@liris.cnrs.fr,
luc.lamontagne@ift.ulaval.ca

Abstract. This paper presents TStore, a Trace-Based Management System that handles the storage, transformation and reusing of Traces. Traces have been often stored without explicit structure. TStore is a web tool that allows anyone to store and reuse traces with their models from various applications. The transformation of Traces helps to move from a first simple interpretation (almost raw data coming from sensors) to the actionable knowledge level of abstraction. TStore provides predefined transformation functions as well as a customized transformation based on Finite State Transducers. Our experiments demonstrated the efficiency of TStore to handle the storage requests. They show that batch operations are more efficient than sequentially adding traces to the system.

Keywords: Trace-Based Management System, Finite State Transducer, Trace Transformation and Human Computer Interaction.

1 Introduction

Nowadays, many recent applications retain traces of their usage by collecting user information. These traces help to understand users' behaviors. Traces are usually stored in different ways and not formalized. We consider traces as objects satisfying a meta-model implying specific properties and associate methods (*M-Traces*) [1]. M-Traces are organized as sets of observed elements associated with their explicit models. In order to simplify the development of M-Traces Based Systems, it becomes important to develop a new kind of M-trace management system called Trace-Based Management System (TBMS).

In this paper, we describe our TBMS called TStore. It is a web tool that allows different agents (applications and users) to concurrently access, store and reuse M-Traces issuing 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. We also describe a customized approach for M-Trace transformation based on the Finite-State Transducer (FST) principle [2]. To reduce network traffic and avoid firewall blocking, TStore supports batch operations. Our experiments show that storing multiple observed elements as a chunk is better than storing them separate.

The remaining of the paper is organized as follows. Section 2 describes an illustrating scenario in a video recommendation application. Section 3 introduces the general concepts of trace-based systems. TStore structure and functionalities are presented in Section 4. In Section 5, the implementation details are discussed. We report our experiments and performance study in Section 6. We discuss the related work in Section 7, and conclude our work in Section 8.

2 Illustrating Scenario

Wanaclip[1] is a web application for generating video clips from different media: photos, videos, music and sounds. 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. Given the large amount of content available, the problem is to quickly find content that truly meets our needs.

In [3] we proposed an approach for contextual video recommendations based on a Trace-Based Reasoning approach. The collection process is the first step that builds the primary M-Traces by observing and storing the interactions between the user and the application. Collected M-Traces need to be stored and managed in a simple way to be retrieved and transformed efficiently. TStore is a TBMS designed for this scenario and other similar scenarios.

3 Background

An interaction trace is a rich record of the actions performed by a user on a system. In other words, a trace is a story of the user's actions, step by step. Here, we focus on a special kind of trace, called *modeled traces* (M-Traces). M-Traces differ from logs in the sense that they come with a model. As shown in Fig. 1, 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 among them [1].

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 *tracebase* is a collection of related M-Traces structured for a specific purpose.

[1] http://www.wanaclip.eu

Specific operations on M-Traces allow producing *transformed M-Traces* for several purposes (filtering, segmentation, reformulation, etc). Transformed M-Traces can be easier to reuse in a given context than the primary M-Trace. A TBMS guarantees the possibility, at any time, to navigate between transformed M-traces. Fig. 1 shows an M-Trace *T1* which is transformed into *T2* and *T3* (level 2). It also shows that *T2*, *T3* are transformed together 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* contains seven obsels (*o1*, *o2*... *o7*). Connected M-Traces represent a tracebase ([*M1*, *T1*], [*M2*, *T2*], [*M3*, *T3*], [*M4*, *T4*]).

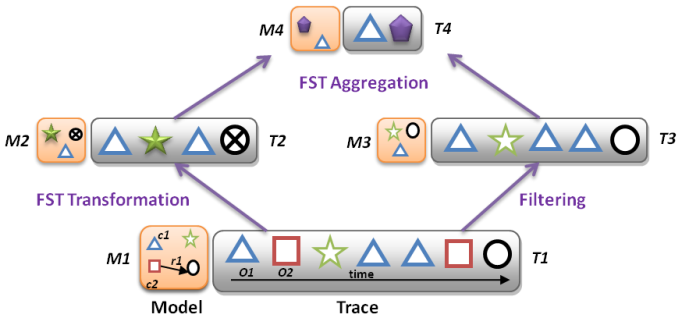

Fig. 1. An example of M-Traces and their transformations

4 TStore Architecture and Specifications

In this section, we describe the structure of TStore and its functionalities (see Fig. 2). TStore is a TBMS for storing M-Traces sent by application specific collecting modules. TStore contains four modules. 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 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.

4.1 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. 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. Conflicts cannot arise while creating models or storing M-Traces, since each user is responsible for his M-Traces.

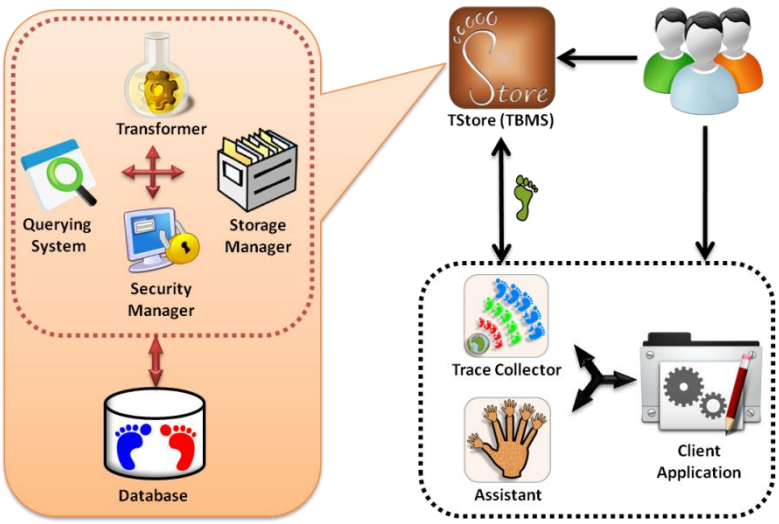

Fig. 2. TStore general structure

As TStore is a web tool, therefore, it is not convenient to make some services await the results of other services. That takes a lot of time for execution and messaging. Hence, TStore makes the storage flexible by scheduling a sequence of executions based on their dependencies. A service can be executed only after the completion of other services. For example, storing an obsel requires finishing the storage of all its hierarchical attributes and their values. TStore benefits from XML structure to store all the components. We will see an example of XML messages in Section 5.

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

4.2 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. TStore has two different approaches for M-Trace transformations: predefined functions and a customized approach based on the Finite-State Transducer (FST) principle. The Transformer 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 M-Trace containing only the input obsel types. Aggregation allows merging several M-Traces in one transformed M-Trace. Obsel types of an aggregated M-Trace contain a union of all obsel types of the original M-Traces. Unlike aggregation, segmentation

cuts M-Traces into smaller chunks to be more useful for understanding the behavior of client applications.

In these transformations, we notice that the transformed M-Trace preserves the same obsel types as the original M-Trace and it doesn't produce new obsel types. However, our FST transformation approach allows defining customized transformations using FST task signatures. The task signature concept has been introduced in [4] as a set of event declarations, entity declarations, relations, and temporal constraints. 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 [2]. TStore uses FST to produce new M-Traces. Such a transformation consists in replacing some obsels matching the FST with more abstract obsels. Currently in TStore, it is experts who define the structure of the new obsel in the same way they define a new obsel type. It is possible that two different transducers generate the same transformed M-Trace. However, FST should be deterministic to avoid ambiguity.

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,
- $F \subseteq Q$, the set of final states,
- Σ and Δ, finit sets corresponding respectively to the input and output alphabets of the transducer,
- δ, the state transition function which maps $Q \times \Sigma$ to Q,
- σ, 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 containing at most one element. Frequently, 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 [5].

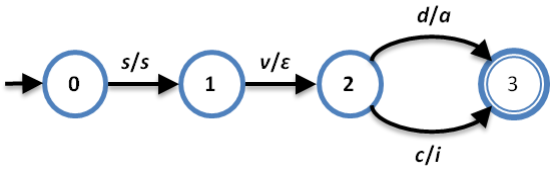

Fig. 3. Example of a FST transformation

By using FST we can apply a large variety of transformations. Fig. 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*), an obsel of type "accept" (*a*) is generated. Else, if the obsel type is "close" (*c*), 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.

4.3 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.

4.4 Security Manager

M-Traces can contain some sensitive data like passwords and credit cards numbers. Agents are responsible of what they send to TStore. 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 stores them. TStore uses a role-based access control (RBAC) approach [6]. The Security Manager allows creating roles for various task functions. 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 for his M-Traces and he can also specify their visibility to be public, private or custom. Private M-Traces can be used only for the analysis. Anyone can access and retrieve public M-Traces. While in custom M-Traces, only those with appropriate privileges and those who have the permission from the user himself can access them. It is possible to have privileges on any item such as an M-Trace, obsel and attribute type. For example, salary attribute can be hided from specific users.

5 Implementation

We have implemented TStore as a PHP web service system over a MySql RDBMS. TStore exchanges XML messages that have a predefined structure. For example, the following XML message can be sent to "storeObsel" service to add an obsel of type "SearchMedia" to the M-Trace (id=101) of the model (id=1). This obsel has 4 attributes: tags="Lyon", kind="right", duration="30" and rhythm ="0". If the obsel type and attribute types are not existed, it calls "storeObselType" and "storeAttributeType" services to create them.

TStore is independent of the client environment. Supporting client's APIs facilitate the generation the XML messages, but it is not mandatory. Thus, it is not important to develop APIs for client applications. M-Traces are indexed and partitioned in the database to enhance the retrieval performance. Obsels are sorted according to their traceID so that we can quickly retrieve all the obsels and their attributes of a specific M-Trace. M-Traces are also sorted and indexed according to their timestamps because most of search queries retrieve M-Traces for a specific time interval.

```
<obsel id="1" traceID="101" start="2012-05-21 17:31:26"
end="2012-05-21 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>
      <attribute id="4" name="rhythm" type="number">
        0
      </attribute>
    </attributes>
  </obselType>
</obsel>
```

6 Experiments

In order to test our environment, we have implemented a collection process in Wanaclip. 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. 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. To evaluate the storage performance a mixture of single obsel and multi obsel storage are used. After 1000 random tests to store a single obsel, we found 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 insert 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 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 (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 50.962 KB. Table 1 shows a comparison between single and multiple obsel storage. These values are calculated based on the average value of all the tests that we performed. Depending on these results, Wanaclip needs about 2.485 seconds to store 100 obsels as a chunk in TStore 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. Therefore, it shows that the multiple obsel mechanism reduces the require time for storage. This is due to the spent time for messaging, parsing command, inserting rows in the database. In multiple obsel, it reduces the number of exchanged messages and the required time for parsing the messages.

Table 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

As shown in Fig. 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. This incensement gradually decreases until very small changes in the end. 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.

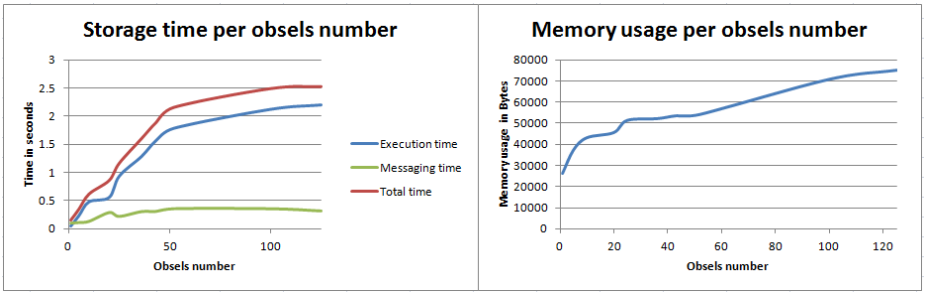

Fig. 4. Storage time and memory usage per obsels number

7 Related work

Many applications write log files to get some information about their usage. Log files typically consist of a long list of events in chronological order and they are usually 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 to indicate the status of variables, the results of decisions, warnings of potential problems and error messages. For example, WinSCP uses XML logging to find a list of files that were actually uploaded/downloaded and Record operations done during synchronization [7].

In the web, there are two tracing approaches. Tracing systems can be located on the client side (i.e. browser plug-in) or integrated in the traced applications on server side. In both cases, users' activities are usually recorded as web logs that contain mainly the visited links. CoScripter [8] is a Firefox plug-in created by IBM Research. 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 [9] and WebMacros [10] record web browser actions as a low-level internal representation, which is not editable by the user or displayed in the interface. The UserObservationHub [11] 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 [12].

A Kernel for Trace-Based Systems (kTBS) was the first TBMS developed in Python [13]. 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 sparql rule.

8 Conclusion and future work

In this paper we described TStore, a web Trace-Based Management System. We presented the modules of TStore that uses the notion of M-Traces. 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 trans-

formed M-Traces. The Security Manager ensures M-Trace protection and the distribution of roles and privileges. The major contribution is the customized transformation approach based on the Finite-State Transducer (FST).

Our experiments demonstrated the efficiency of TStore to handle the storage requests. The results showed that storing multiple obsels as a chunk is better than storing them separately. We discussed other related work like log files, web logs, DNA repositories and KTBS. 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 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.

9 References

1. Settouti, L.S.: M-Trace-Based Systems - Models and languages for exploiting interaction traces, http://liris.cnrs.fr/Documents/Liris-4984.pdf, (2011).
2. Roche, E., Schabes, Y.: Finite-State Language Processing. *MIT Press* (1997).
3. Zarka, R., Cordier, A., Egyed-Zsigmond, E., Mille, A.: Contextual trace-based video recommendations. *21st international conference companion on World Wide Web (WWW-XperienceWeb'12)*. pp. 751–754. , Lyon, France (2012).
4. Champin, P.-A., Prié, Y., Mille, A.: MUSETTE : a framework for Knowledge from Experience. *EGC'04, RNTI-E-2 (article court)*. pp. 129–134. Cepadues Edition (2004).
5. Mohri, M.: Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*. 23, 269–311 (1997).
6. Ferraiolo, D.F., Kuhn, D.R., Chandramouli, R.: Role-Based Access Control. *Artech House* (2003).
7. WinSCP, http://winscp.net/eng/docs/logging_xml, (2012).
8. Leshed, G., Haber, E.M., Matthews, T., Lau, T.: 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 (2008).
9. Anupam, V., Freire, J., Kumar, B., Lieuwen, D.: Automating Web navigation with the WebVCR. *Computer Networks*. 33, 503–517 (2000).
10. Safonov, A., Konstan, J.A., Carlis, J.V.: Beyond Hard-to-Reach Pages : Interactive , Parametric Web Macros. *Proc Human Factors and the Web*. 1–14 (2001).
11. Haas, J., Maus, H., Schwarz, S., Dengel, A.: ConTask - Using Context-sensitive Assistance to Improve Task-oriented Knowledge Work. *ICEIS (2)'10*. pp. 30–39 (2010).
12. Batley, J., Edwards, D.: Genome sequence data: management, storage, and visualization. *Biotechniques*. 46, 333–334, 336 (2009).
13. Champin, P.-A., Prie, Y., Aubert, O., Conil, F., Cram, D.: kTBS: Kernel for Trace-Based Systems, http://liris.cnrs.fr/publis/?id=5478, (2011).