Collecting interaction traces in distributed semantic wikis

Anh-Hoang Le, Marie Lefevre, Amélie Cordier, Hala Skaf-Molli

Collecting Interaction Traces in Distributed Semantic Wikis

Anh-Hoang Le, Marie Lefevre, and Amélie Cordier
University of Lyon, CNRS
LIRIS, UMR5205, University Lyon 1, France
firstname.lastname@liris.cnrs.fr

Hala Skaf-Molli
LINA– Nantes University, France
hala.skaf@univ-nantes.fr

ABSTRACT

In the Kolflow project, our general objective is to develop an assistance engine suitable for distributed applications. In order to provide contextualized and relevant assistance, we feed the assistance engine with interaction traces. Interaction traces record events occurring while users are interacting with applications. These traces become containers of valuable knowledge to providing assistance. Collecting interaction traces is a challenging issue that has been thoroughly studied in the context of local applications. In contrast, few approaches focus on collecting interaction traces in distributed applications. Yet, when applications are distributed, collecting interaction traces is even more challenging because new difficulties arise, such as data synchronization and multi-synchronous collaboration. In this paper, we propose a model and a tool for collecting traces in a distributed environment. The originality of the model is that it is tailored to fit distributed applications. We implemented the model in Collectra, a tool to collect interaction traces in distributed web applications. Collectra collects interaction traces and stores them in a dedicated trace-base management system. We report on the experiments we have conducted in order to evaluate performances of Collectra (both response time and memory space). Results of the experiments show that Collectra performs well and that it can be used to support the assistance tasks carried out by the assistance engine.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—Distributed applications; H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces—*Collaborative computing, theory and models*

General Terms

Design, Theory

Keywords

Distributed semantic wikis, interaction traces, model of trace, trace collection process, trace-based reasoning, user assistance.

1. INTRODUCTION

The success of Wikipedia is just one of many examples of the fact that collaborative editing of Web resources has become common practice. Wikis are one of the main tools enabling collaborative editing of Web resources. Semantic wikis [11, 4] introduce semantic annotations within wiki pages thus enabling knowledge construction through social interaction. Semantic wikis contribute to the development of the so-called "social semantic spaces", spaces enabling humans and machines to collaborate for producing knowledge usable by humans and by machines. Distributed semantic wikis [14, 18, 17] enable offline work and multi-synchronous editing [9]. They bring an additional dimension to collaboration tasks over the Web. Indeed, distributed semantic wikis draw inspiration from popular distributed version control systems (DVCS) [1] such as Git[1] and Mercurial.[2] Collaboration mechanisms in distributed semantic wikis are similar to mechanisms we can observe in DVCS though they involve more complex tasks such as knowledge management tasks.

The goal of the Kolflow project[3] is to design a social semantic space enabling humans and machines to collaborate as easily as possible in order to build knowledge understandable by humans and usable by reasoning engines. In this project, Distributed Semantic MediaWiki[4] (DSMW) is used to provide a community of user with a distributed social semantic space enabling them to build a shared knowledge base. This environment is designed to provide users with numerous features similar to those we can find in DVCS: local copies, main repository, push and pull operations between users, conflict management tools, versioning, etc.

The Kolflow environment is complex and has numerous features. It is quite difficult for users to master all the features at once. Therefore, the environment includes an assistance engine responsible for providing a contextual help to users (automating complex actions, creating communication channels in the network of users, help during conflict resolution, etc.).

[1] `http://git-scm.com`
[2] `http://mercurial.selenic.com`
[3] `http://kolflow.univ-nantes.fr`
[4] `http://www.mediawiki.org/wiki/Extension:DSMW`

This assistance engine is based on the paradigm of Trace Based Reasoning [7]. The main input of this assistance engine are interaction traces. Interaction traces are records of the events occurring when a user interacts with a system. These traces enable the assistance engine to gather information about the context in which the user is performing a task, and therefore to provide him with contextualized help. Indeed, as shown in previous work, interaction traces are a rich source of knowledge for building personalized and contextualized assistance in distributed semantic wikis [5]. The challenge of efficiently collecting relevant interaction traces is therefore of major importance.

In this paper, we tackle the problem of collecting interaction traces in a distributed environment such as DSMW. We introduce a new approach to collecting interaction traces in distributed wikis and we propose an original trace model taking into account collective and collaborative dimensions of traces. We present Collectra, a tool we have developed to collect interaction traces according to this model. Through experiments, we show how the performances of Collectra comply with the requirements of the assistance engine developed in the Kolflow project. We also show that the designed model allows us to fully collect elements needed for building assistance. Though the scope of this paper is to present Collectra, more broadly, we are interested in studying how the use of traces supports a dynamic and collaborative process of knowledge construction. We discuss the applicability of our approach to other areas at the end of this paper. It must be noted that Collectra is available as an open source plugin for DSMW.[5]

The paper is organized as follows. Section 2 describes the context of the Kolflow project and the motivation for our approach. Section 3 introduces background knowledge about traces and reviews related work. Section 4 describes the Collectra approach and the distributed trace collection methodology. Section 5 presents the implementation of Collectra. Section 6 reports on the experimentation performed on Collectra. Experimentation show that Collectra is efficient enough to support the assistance process. Results and future work are discussed in section 7.

2. BACKGROUND AND MOTIVATIONS

In the Kolflow project, DSMW is used as a distributed tool for collaboratively editing knowledge bases. DSMW is a tool difficult to master. This is the reason why we have decided to equip the Kolflow environment with an assistance engine. To better understand the requirements of this assistance engine, in this section, we present situations where assistance may be required. We give examples of some difficult tasks users have to address. We pinpoint difficulties arising from these tasks and we show how an assistance engine could be helpful.

Semantic MediaWiki (SMW) is the most popular semantic wiki developed by the semantic web community. It allows mass collaboration for creating and maintaining ontologies. In SMW, users add semantic annotations to wiki page texts to represent relations and properties available on the page. Users may choose their own vocabulary to type links. For instance, a link between the wiki pages "France" and "Paris" may be annotated by a user as "capital".

Annotations express semantic relationships between wikis pages. Semantic annotations are usually written in a formal syntax so they are processed automatically by machines and they are exploited by semantic queries.

Although successful, semantic wikis still have limitations such as they do not support a multi-synchronous work mode [14, 18, 17]. Consequently, users cannot work in isolation or disconnected. Distributed semantic wiki [14, 18, 17] has been designed to overcome this limitation. DSMW [14] is a distributed extension of Semantic MediaWiki that supports multi-synchronous collaborations. Multi-synchronous systems are different from synchronous and asynchronous ones by managing multiple streams of activities instead of giving the illusion of one stream. In standard collaborative applications, when a modification is made by a user, the modification is immediately visible by others. However, in multi-synchronous systems, modifications made by users are not immediately visible by others. In that case, modifications become visible only when users validate them (*e.g.* commit changes).

DSMW allows users to build their own cooperation network. The construction of the collaborative community is declarative, in the sense that users declare explicitly with whom they would like to cooperate. Users may have a DSMW server installed locally if they like. They may create and edit their own semantic wiki pages as in a normal semantic wiki system. Later, they may decide to share these semantic wiki pages, and choose with whom to share.

The replication of data and the communication between servers is made through *channels* (feeds). The channel usage is restricted to few servers with simple security mechanisms that require no login or complex access control. The key point is that channels are read-only and can be hosted by users themselves. When a semantic wiki page is updated on a DSMW server, the server generates a corresponding operation.

According to DSMW replication model, an operation is processed in four steps: (1) It is executed immediately against the current page, (2) it is published on the corresponding *channels*, (3) it is pulled by the authorized servers, and (4) it is integrated to their local replica of the page. If needed, the integration process merges this modification with concurrent ones, either generated locally or received from remote servers.

In practical, to exchange modifications, DSMW users have to set up communication channels using **push-pull** operations. They start by creating **pushFeeds** to send modifications performed on a set of semantic wiki pages on one server and **pullFeeds** to fetch these modifications on a remote server. Once the communication channels are created, **Push** and **Pull** operations are available to actually push and pull each modification.

Figure 1 gives an example of a collaboration network built upon a network of DSMW. $User3$ is working on $site3$ and decides to share content with $User4$ who works on $site4$.

[5] `http://sourceforge.net/projects/silexcollectra/`

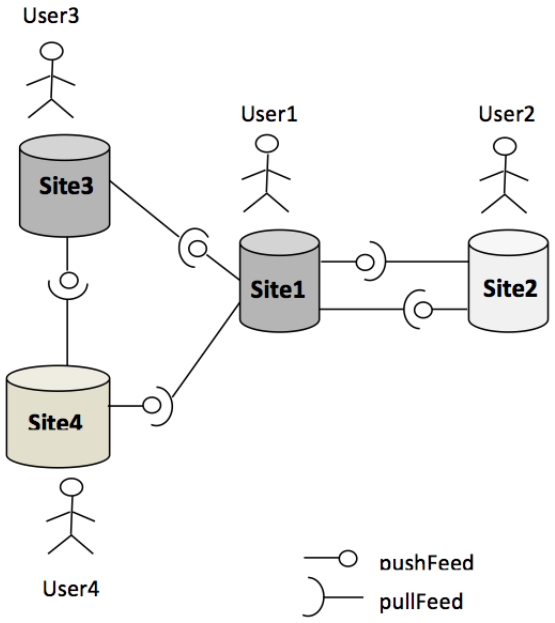


Figure 1: Collaboration network in DSMW.

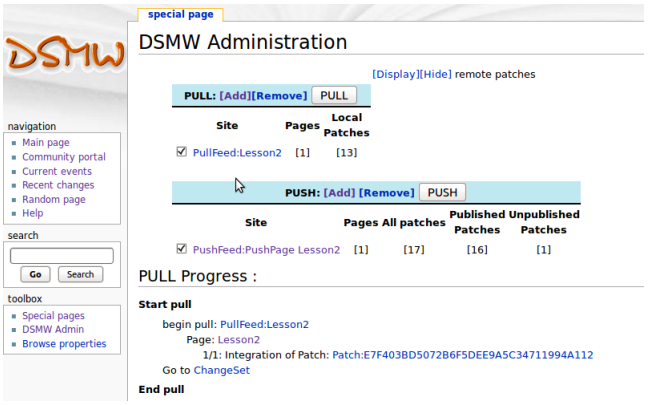


Figure 2: Screenshot of DSMW administration interface.

$User3$ creates a *pushFeed* for $User4$ and selects exactly the pages that must be shared (using a set of semantic queries). $User4$ creates a *pullFeed* corresponding to the *pushFeed* of $User_3$. Now, $User4$ can fetch modifications published by $User3$. In the same way, other users create their pushFeeds and pullFeeds to exchange modifications.

Even for experimented users, setting up communication channels and manipulating the environment can be tedious. The administration interface of DSMW is displayed on figure 2. This administration interface is quite difficult to use, especially for novice users. As we can see on that interface, administration tools for creating feeds are not user-friendly, and are difficult to understand for novice users. This is the first reason why we need to provide assistance to DSMW users. We want to help them using the tool. For example, we can automate complex actions, such as the creation of communication channels.

The second motivation to develop user assistance is to support collaborative activities in a community of users. Indeed, problems can arise very quickly in distributed collaborative activities. To illustrate this point, let us consider the following scenario. A community of cooks is using DSMW to share cooking knowledge. The DSMW consists of recipes, food ontologies and general cooking knowledge. A master version of DSMW, named DSMW-Cook, is available on the web. A community member, Bob, owns a DSMW instance on which he pulled all the resources available in DSMW-Cook. He has added his Melon Pie recipe. He has also enriched the ontology with new ingredients (melon, *etc.*). Alice also has an instance of DSMW based on DSMW-Cook. In her instance, Alice has added her own recipe of Melon Pie, and modified an existing recipe of Chocolate Pie. She also has modified the ontology, but, unfortunately, her modifications differ slightly from Bob's. Bob has put Melon in the category Tropical Fruit, while Alice has put them in the more general category Fruit. When Bob and Alice try to share the content of their wikis, conflicts arise (different recipes with identical names, different ontologies, *etc.*). Therefore, they have to face a new task: resolving conflicts. Conflict resolution is a complex task involving several sub-steps such as analysis of the provenance of the conflict, search for additional knowledge, negotiation with other users, *etc.* It is usually difficult for end-users to deal with these problems. An assistance system could help them negotiate the knowledge contained in their wikis and present them the steps to resolve conflicts, to show the change history of the resources involved in the conflict, *etc.*

The assistance engine developed in Kolflow makes use of interaction traces to provide contextualized assistance. Therefore we need to collect interaction traces. The collection process has to match some specific requirements to make sure that the assistance will be efficient. These requirements are as follows. Firstly, collected traces have to match a model depending on the task at hand. For the Kolflow project, this model is described hereafter. Secondly, the collection process has to be fast enough not to slow down users' activities. Thirdly, collected traces have to be stored using a decent memory space, thus ensuring that tracing all the users' activities will not overload servers. Lastly, the retrieval time of collected traces must be fast enough to ensure that assistance can be provided real-time. Evaluations presented at the end of this paper assess the efficiency of the collecting process implemented in Collectra with regard to these requirements.

3. RELATED WORK

In a previous work, we made a review of various assistance strategies and we demonstrated why trace-based assistance is relevant for the Kolflow project [5]. The goal of the work described in this paper is to collect traces of interaction in distributed environments in order to provide the assistance engine with traces. In this section, we introduce briefly the trace theory, in order to better explain requirements that the trace collection process must fullfil. Then, we review related work on trace collection. We compare these methods against the criteria that the collection process must satisfy in order to make the assistance possible.

3.1 Trace Theory

In this paragraph, we briefly recall the trace theory presented in [16]. We also provide some basic definitions about traces.

A trace is defined as a set of observed elements, called *obsels*. Obsels are timestamped within traces. A *trace model* defines the structure and the types of obsels that are contained in the trace, as well as the relationship between these obsels. A *modeled trace* (M-Trace) is a trace together with its trace model. A *Trace-Base Management System* (TBMS) is a knowledge based system enabling storage and management of traces. All the traces are stored in a Trace-Base Management System.

There are two types of M-Traces in a TBMS: primary trace and transformed trace. A primary trace is collected from external sources and stored as an M-Trace. M-Traces created after performing transformation operations on existing traces are called transformed traces. A TBMS has three main components:

- **Collecting system:** collects the observed data from different input sources (*i.e.* log files, video records, interface events, server messages, *etc.*) and stores them into traces as obsels.
- **Transformation system:** performs several transformations such as filtering, rewriting, merging obsels in one or more M-Traces. Transformations can be applied on any existing M-Trace, called source trace. The result of a transformation is a new trace, called transformed trace.
- **Querying system:** executes queries on traces. Queries enable us to compute various results from existing traces (number of obsels, frequency of an obsel type, frequency of a pattern, *etc.*). Queries also enable us to produce rich output resources such as documents, other traces, episodes (sub-parts of existing traces), *etc.*

Each resource in a TBMS is identified by a unique URI (Uniform Resource Identifier[6]). Through these URIs, users and other systems (*e.g.* trace-based application, collecting system) can manipulate and process TBMS resources.

A complete formalization and semantics of trace models, traces, queries, and transformations can be found in [16]. In the following, we focus on the collection process.

3.2 Tracking interaction on web applications: a review

In the context of the Kolflow project, we need to collect traces of user interactions in a distributed environment (namely, DSMW). As a consequence, we must not only collect traces of multiple users, but also traces of exchanges between servers. Based on the requirements of the assistance engine, we have defined a set of criteria that the collection process has to satisfy. These criteria are presented in columns on table 1.

[6] `http://en.wikipedia.org/wiki/Uniform_resource_identifier`

The collection module must collect all the users actions (key and mouse actions). It must also track requests on servers and enable sharing of elements between servers. The module also must enable collection of traces of multi-users. Lastly, we seek for a module providing formalized traces so that they can be easily reused in a pre-existing system. Thereafter, we study various approaches to collecting traces from user actions in web environments with regards to these criteria. We summarize our observations in table 1 and we provide a synthesis illustrating the main problems related to our environment.

The collection of interactions between users and a website can be performed manually by human observers, semi-automatically by external equipments (*i.e.* video and audio captures) or automatically through tracing system implemented within the application or the tool. We briefly review these different approaches and we give several examples of tracing systems.

Manual collection is performed by human observers who observe users live or from a distance with the help of information tools (*i.e.* text editor, text notes, etc.). The result of the collection process usually takes the form of a set of notes in natural language.

Collection with external equipments requires the use of cameras or audio recording devices. This approach produces audiovisual traces.

These two approaches give detailed information about end-users, but are not well formalized. Consequently, the exploitation of collected information often requires complex processing systems like human operations (*i.e.* text processing, image processing). Therefore, these two approaches are not suitable for building an online assistant based on interactive traces.

Tracing systems implemented within existing applications allow obtaining formalized traces. These traces are then processable by reasoning engines. In the context of websites, tracing systems can be located on client side (*i.e.* browser plugin) or scripts integrated in the traced application on server side. Tracing on client side produces very fine-grained observations but has important limitations (portability, speed, compatibility). Tracing on server side does not produce such detailed observations, but is usually more generic and simpler to implement. In the following, we give several examples of these automatic tracing systems.

Web browser: a Web browser is in itself a tracing system. Information about user activity can be recorded into the logs of Web browsers. This information generally focuses on the links that users have visited [13].

Web browser extension: may be implemented to obtain more detailed information than in the logs of Web browsers. These extensions are often known as *key-loggers*. They are capable of capturing key events and mouse events as well as retrieving information related to user actions. These extensions are used for several purposes such as facilitating the repetitive tasks (*i.e.* input assistance in search engine). However, these extensions have to be installed before being

Trace sources	K	M	Rq	Sh	Mu	F	Example
Manual collection	√	√	√	∅	∅	∅	Heraud et al., 2005 [10]
External equipment	√	√	√	∅	∅	∅	Avouris et al., 2005 [2]
Web browser	∅	∅	√	∅	∅	√	Oh et al., 2011 [13]
Web browser extension	√	√	√	∅	∅	√	Bell et al., 2012 [3]
Local Web proxy	∅	∅	√	∅	∅	√	Mathieu et al., 2010 [8]
Web server	∅	∅	√	∅	√	√	Schechter et al., 1998 [15]
Web server extension	√	√	√	∅	√	√	May et al., 2007 [12]

Table 1: Type of collection process and observable information. (K) Key action, (M) Mouse action, (Rq) Server request, (Sh) Sharing between servers, (Mu) Multi-users, (F) Formalized trace.

used. This may be considered as a disadvantage because the sensitive data can be observed and recorded.

Local Web proxy: web activities of users can be recorded through a specialized tracing system called *local Web proxy* [8], running on the local computer of the user. This HTTP logging mechanism allows observing all requests going out of the user's computer from local Web agents (not limited to a particular browser). This approach has benefits in monitoring and analysing various aspects of personal web activities. However, its traces are not suitable for use by an assistance system. Because, while technically on the client side, this method collects the same kind of information as server side techniques.

Web server: Like Web browsers, Web servers (*e.g.* Apache) also have tracing systems, which are capable of generating log files containing information about user actions. These log files follow predefined formats(*i.e.* Common Log Format[7]). They usually consist of the address of the client, date and time of request, executed request, status code returned to client, size of document returned to client. These traces can be used as inputs of assistants for navigation of Web [19]. However, like browser log files and local Web proxy, the Web server log files do not contain enough detailed information about user actions (*i.e.* actions performed on client side).

Web server extension: collecting traces on both client side and server side raises problems. However, these problems can be solved with the aid of server extensions. For example, [12] proposed an extension which allows tracing discussion forums. This extension consists of a collector in JavaScript on client side associated with a collector on server side. However, the proposed extension is limited to the observation of a single server and its users.

The analysis of table 1 shows that there are various types of tracing systems implemented on Web applications. These tracing systems allow obtaining traces which have various contents and representations. This analysis also shows that in order to have detailed information about actions of users on a website, we should develop extensions on server side, connected to tools on client side.

In the context of distributed servers (*i.e.* distributed semantic wikis), the collaboration between servers must be traced and shared. Existing tracing system seldom enable sharing of traces between tracing components. Moreover, they are not suited to multi-user traces collection. Thus, we propose a new tracing system which enables trace collection in a distributed Web environment. This model, presented in section 4.2.2, is based on a trace theory that we have introduced before.

[7] `http://en.wikipedia.org/wiki/Common_Log_Format`

4. AN APPROACH FOR COLLECTING DISTRIBUTED TRACES

In this section, we describe our approach for designing a plug-in for collecting interaction traces in DSMW and for storing collected traces in a Trace-Base Management System. The main characteristic of our approach is to implement a trace collecting module on each SMW server in order to observe and record a maximum of users' actions with satisfactory performances. A Trace-Base Management System (TBMS) is associated to each SMW. Traces are shared between the different servers through a specific sharing protocol. To implement this protocol, we had to enrich the pre-existing features of the TBMS (see section 3.1). Thus, in order to design our collection module, we had to work at two levels: implementation of an independent collection module and extension of the existing programming interface of the TBMS.

4.1 Architecture

Figure 3 illustrates our architecture for collecting and sharing traces in DSMW. In this architecture, each Semantic MediaWiki (SMW) server is used by a group of users and is connected with a TBMS which stores the collected traces and shares them with other servers. The collecting module acts as an interface between DSMW and the TBMS. The collecting process implemented in DSMW sends messages to this module. The module parses the messages in order to build obsels that are stored in a trace managed by the TBMS.

To get shared traces from other servers, we use an Inter-TBMS communication ("Sharing traces" on the figure) that can be established through a special method allowing the collection of traces from external sources. These external sources can be other TBMS. This special method is an extension of standard methods of the TBMS.

The collected traces stored in the TBMS can be reused by any trace-based application. In our project, we will use these traces as knowledge sources for our trace-based assistant for DSMW. With this architecture, we can collect both individual and collaborative traces. Collaborative traces are described in details in the next section.

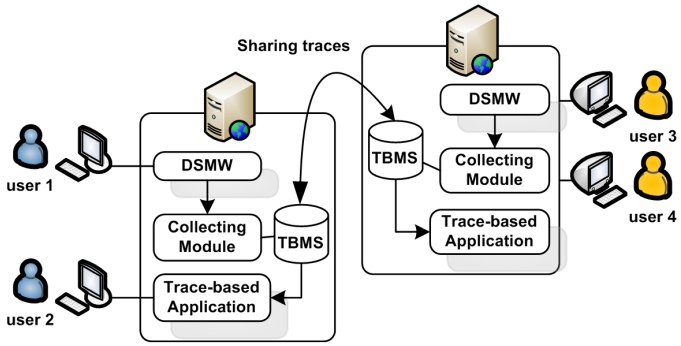


Figure 3: Collecting and sharing traces in DSMW.

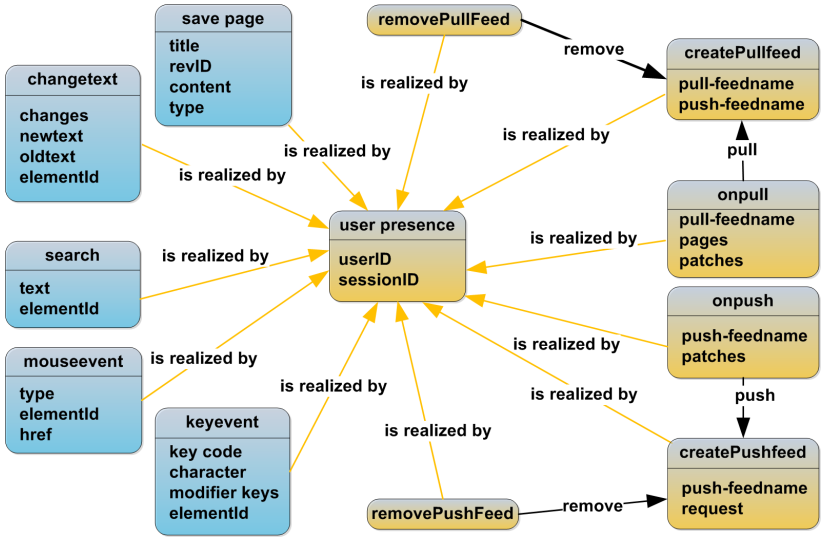


Figure 4: Trace model of primary traces in DSMW.

In Kolflow, we use kTBS[8] (Kernel for Trace-Based Systems) [6] to store traces. kTBS is a TBMS implementation in Python. It provides all the required features of a TBMS (primary trace, transformed trace, trace model, obsel types, attribute types and relation types of obsel types). Section 5 give details about the implementation of our collection plug-in and shows how we connect to the kTBS.

4.2 Model of Collaborative Trace

DSMW is a multi-synchronous collaborative editing system allowing a group of users to perform editing activities on semantic wiki pages and to synchronize their semantic wiki pages. Therefore, we are interested in collecting traces of collaborative activities. We call these traces *collaborative traces*.

In order to manipulate collaborative traces, we start by dividing user's activities in DSMW into two main activity types: *individual activity* and *collaborative activity*. An individual activity defines an activity that is performed by the user on his server and that does not require awareness of activities performed on other servers. A collaborative activity defines an activity performed by a user on the network and that requires awareness of other users and other servers.

This distinction allows us to define two types of traces: *individual trace* and *collaborative trace*. At the semantic level, an "individual trace" is a term used to indicate traces that reflect individual activities of a semantic wiki user. The term "collaborative trace" is used for traces reflecting collaborative activities. At the design level, an individual trace is identified by one user, and is called a *private trace*. A trace reflecting a part of the individual activity that is shared with others is called a *shared trace*. A fusion of shared traces is called a *group-shared trace*. The objective of this organization is to facilitate the process of building shared traces from individual traces. Thereafter, we describe the elements of our model of collaborative traces.

4.2.1 Obsel Classification

Considering the specific context of a collaborative work as it is in Kolflow, we propose a simple categorization of obsels. This categorization allows us to model users' activities in DSMW. In order to build this categorization, we have listed the most frequent actions in DSMW. At this stage, we have not focused on other interactions (such as log on/log off) but obsels corresponding to these actions can easily be built by combining basic obsels. A description of the collected obsels

[8] `http://liris.cnrs.fr/sbt-dev/ktbs/`

Type of Obsel	Description
KeyEvent	Key and character code that a user pressed
MouseEvent	Element on the interface that a user clicked (button, link, *etc.*)
ChangeText	Text and changes that a user made
Search	Search information collected when a user performed a search
SavePage	Inputs and revision numbers of the saved wiki page and its type of action ("new", "edit", "undo" or "rollback")
UserPresence	Root obsel created when the system creates a new session for a user. Contains information about the user and his session
CreatePushFeed	Name and URL of PushFeed and its semantic query designed when a user created a PushFeed
Push	Push operations including the published changes
RemovePushFeed	Removed PushFeed
CreatePullFeed	URLs of the PullFeed and the PushFeed associated with it
Pull	Pull operation including the downloaded changes and pages related to those changes
RemovePullFeed	Removed PullFeed

Table 2: Description of collected obsels.

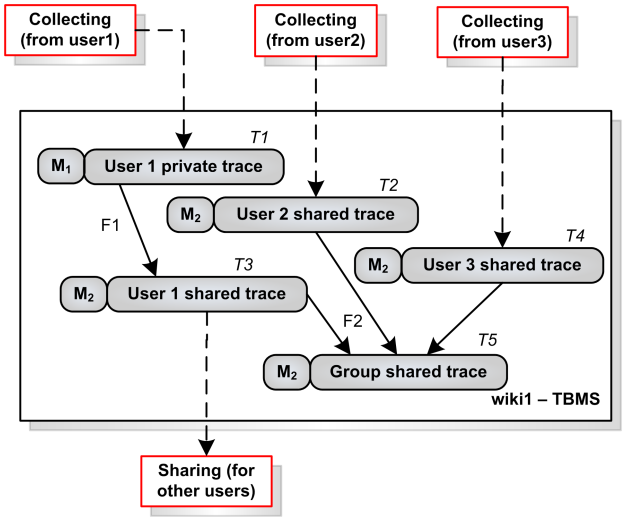


Figure 5: Transformation model for collaborative traces.

is given in table 2. Note that each obsel has a timestamp set by the collection module.

The categorization consists of three meaningful levels: low level, medium level and high level. At the low level, obsels reflect a single action (basic unit of interaction) performed by a user on the system. Usually, obsels at this level reflect events that occur on the user's interface, *i.e.* key events and mouse events. At the medium level, obsels characterize a set of single actions which are associated with a subtask. For example, a text modification ("ChangeText") can be made by mouse or/and keyboard and it is often a part of a search or a page editing. At the high level, the obsels characterize a set of subtasks which are necessary for the realization of a general task (*i.e.* search, edit, access a page).

These three levels constitute a hierarchical model of obsels (low: interaction, medium: subtask, high: task) in which any obsel can be represented by obsels in the lower levels ("a part of" semantic relation). Technically speaking, there could be many more levels, depending on the complexity of the obsels. In the context of Kolflow, these three levels appeared to be the most suitable to describe the objects we were interested in.

4.2.2 Trace Model

Figure 4 presents the organization of our trace model for modelling activities in DSMW (obsels and relationships between obsels). This trace model describes the structure and types of obsels which are contained in primary trace and which are collected directly from the target application. In this trace model, we use the obsels at the different levels in order to provide different inputs for the TBMS and to evaluate the suitability of the model. Results of the experiment (see section 6.2) show that using obsels at the intermediate level (change text) instead of obsels at the low level (key events) in some cases (page editing) can improve the performance of collecting trace while still ensuring action replay feature of obsels at high level (wrt. information loss).

There are two groups of obsels in the trace model: obsels only used for private traces (save page, change text, search, mouse event, key event) and obsels used for shared traces, called *shared obsels* (user presence, create PushFeed, create PullFeed, on push, on pull, remove PushFeed, remove PullFeed). Our trace model is *extensible*, meaning that it can be enriched to take into account unforeseen elements. For example, it can be extended for integrating new obsels for other entities (pages, patches, *etc.*) and other activities (create new account, view log history, *etc.*). Attribute types and relation types of obsels also can be modified or added, depending on the specific needs of trace-based applications. In the same way, it can be simplified in order to be better adapted to specific needs and to provide better performances.

4.2.3 Transformation Model

Transformations enable us to manipulate traces (filtering, merging, *etc.*). Here, we use transformations to build collaborative traces from individual traces. In the following, we distinguish three types of traces:

- Private trace: it is the trace of an individual user on a single wiki.
- Shared trace: for each private trace, a shared trace is built. Shared traces are shared with other wikis through the Inter-TBMS sharing protocol.
- Group shared trace: when several shared traces are merged in a single trace, this single trace is called a group shared trace. It reflects the collaborative activity of users in the group.

A first transformation is applied on a private trace, and the result of this transformation is a new trace that only contains shared obsels. A second transformation is applied on all shared traces, and the result of this transformation is a new trace which contains all shared obsels. Both transformations are performed automatically at run-time.

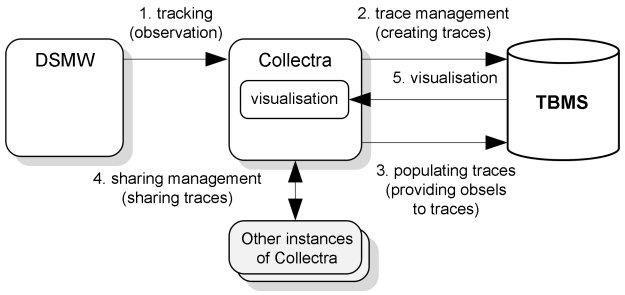


Figure 6: Features implemented in Collectra.

Figure 5 shows an example of our transformation model applied to a situation involving three users (user1, user2 and user3). User1 works on wiki1. User2 and user3 work on different servers, but share their traces with user1. The figure represents the TBMS of wiki1. We can see that the TBMS stores the private trace of user1 (T1) and two shared traces collected from user2 (T2) and user3 (T4). The model of the trace T1 is M1. It contains a description of the obsel types of the trace T1. It is the same for all the stored traces.

In order to create a shared trace for user1, we use a *filter* method F1. This method is used to find shared obsels in the trace T1 and to copy them into a new trace (trace T3) sequentially. The new trace links to the model M2 which contains only shared obsels. In order to create a group shared trace for the three users, we use a *fusion* method F2. This method is used to copy all obsels in the shared traces of three users into a new trace (trace T5) sequentially. The new trace also links to the model M2 because it only contains shared obsels.

5. COLLECTRA: A TRACE COLLECTION MODULE FOR DSMW

We implemented the architecture described in the previous section in a collection module named Collectra. It has a server-side part (as a DSMW plug-in) and a client side part (embedded in the pages by the plug-in). The role of Collectra is to collect elements to characterize user interactions with DSMW. Collected data (obsels) are stored into a specific TBMS (kTBS), which stores and transforms collected traces.

In addition, Collectra has another important features that enables the creation of traces in its kTBS and the update of shared traces from other servers in DSMW. Collectra also handles creation and management of new resources that are not handled by default in the TBMS. For example, when a new user is created, a new private trace must be created in the TBMS.

Figure 6 shows the main features of Collectra. These features are detailed bellow. First, users' actions on client side (browsers) and their consequences on server side are captured and collected by Collectra (1). The user's actions (*i.e.* key event, mouse event, change text) are recorded live on client side. These actions are then serialized in XML and sent to the server at regular time intervals by using AJAX[9] technique. Next, they are synchronized with elements collected on server side. For example, a mouse event "click on the search button" causes a search event on the server. In Collectra, the collected objects (obsels) are serialized in a format accepted by the TBMS. When new obsels are collected, Collectra retrieves the trace of the current user from TBMS. If no trace is found for the user, a new trace is created automatically for him (2). Then, the formatted obsels are added into their traces (3).

In addition, Collectra has features allowing it to synchronize with the other servers on the network. When traces are modified on the network, servers update their statuses to reflect these changes. Collectra synchronizes itself at regular time intervals in order to take into account these changes (4).

Lastly, we implemented a simple visualization tool to observe the effects of collaborative traces visualization on the awareness of DSMW users (5).

[9]`http://en.wikipedia.org/wiki/Ajax_(programming)`

6. EXPERIMENTS

To ensure that the collection module meets the requirements of the assistance engine, we conducted several experiments. These experiments focused on performances and stability of the collection module. The objectives were twofold. First, experiments demonstrate that it is possible to collect, in real-time, traces matching the model we have defined. Next, experiments show that it is possible to perform queries on collected traces with a good response time. These experiments and their evaluations are described in this section.

6.1 Experimental protocol

In order to establish a data set to evaluate our approach, we set up an experiment involving two teachers. Teachers were to collaborate through DSMW in order to create a web programming course. Different stages of collaboration were predefined. The experiment consists of two main parts.

Before the experiment, a web server was set up, and DSMW, Collectra and kTBS were installed on this server. The server ran on a personal computer (Processor: Intel Core 2 Duo CPU E6550 2.33GHz x 2, RAM: 3.8GB, OS: Ubuntu 11.10 64 bits). Three instances of wikis were installed on the server: wiki1 for user1, wiki2 for user2 and wiki3 to integrate the productions of the previous two wikis.

The first part of the experiment was designed to evaluate the trace collection process. This part consisted of two stages. During the first stage, user1 and user2 worked on their own wikis to create their own parts of course (*single-user mode*). This stage lasted 45 minutes and involved all types of operations (creation of pages, push, pull). During the second stage, both users published their content on wiki3. Both users pulled their wiki pages created from wiki1 and wiki2 (content of pages from wiki1 and wiki2 is different) and continued working simultaneously during 25 minutes (*two-users mode*). In particular, they had to collaborate in order to create the introduction page. During these 70 minutes, Collectra collected all interactions, defined in our traces model, between user1, user2 and server, on the three wikis.

The second part of the experiment was designed to evaluate the trace exploitation process. For that purpose, we

Obsel type	Number of obsels		
	Wiki1	Wiki2	Wiki3
Key event	799	927	680
Change text	409	518	404
Mouse event	49	98	130
Created page	4	3	10
Modified page	1	1	1
Search	7	3	1
Pushfeed	5	3	3
Pullfeed	3	2	8
Push	5	5	4
Pull	0	1	7
Others	1	1	1
Total	1283	1562	1249

Table 3: Number of obsels created on each wiki.

	Size (kB)	Number of attributes	Creation time(s)	Retrieving time(s)
Min	6.2	9	0.02741	0.00394
Max	6.7	17	0.20917	0.01586
Average	6.3	12.6	0.04100	0.00672

Table 4: Characteristics of obsel.

performed various transformations on collected traces (*i.e.* filter by date, by type, by property and fusion). Then, we displayed transformed traces on the visualization interface of Collectra. Transformations and visualization interfaces are integrated in each wiki in DSMW, through the Collectra plug-in. This enables us to evaluate response time from the server (computation of the result and transfer).

6.2 Results

With regard to collection of traces, Table 3 shows the number of obsels associated with each wiki. The 45 minutes of activity produce about 1200 obsels (approx. 3 seconds/obsel), among which a majority are obsels used for editing (KeyEvent and ChangeText, approx. 91%). The number of obsels is relatively low, but the main problem lies in the number of attributes obsels, as we discussed below.

Table 4 shows that the average number of attributes of obsels is about 12 and the average size of an obsel is about 6 KB (uncompressed). And after compression of trace files, the average size of an obsel is reduced to 2 KB. We analysed this problem and we found that the cost of building the structure of obsels (*i.e.* properties with the long name or the repetition of RDF tags) is bigger than the memory space used for actual information. The average creation time is approx. 0.04 s (capable of handling up to 25 obsels/s) and the average retrieving time is approx. 0.0067 s, which is sufficient for real-time retrieval of parts of traces.

On graphs 7(a) and 7(b), lines represent the time taken by the creation of each one of the three types of obsels with the highest occurrence: KeyEvent(K), ChangeText(C) and MouseEvent(M). Obsels are ordered by identifier. We observe that the time of creation of obsels depends on the number of attributes of that obsel (C=11, K=13 and M=17; the number of attributes is defined in the trace model). and the number of simultaneous users (the creation time in two-users mode is *slightly* higher than in single-user mode). In addition, graphs show that there is no strong correlation between time of creation and number of obsels in the trace. In other words, the creation time of obsels does not increase with the number of previously created obsels in the same trace.

In the second part of the experiment, we focused on the exploitation of traces. Retrieving a trace consists of two basic phases: (1) Search and compute obsels of the trace, and (2) Serialize the trace in a given format (*i.e.* RDF, JSON, *etc.*).

In the last column of table 4, we observe that the average retrieving time is approx. 0.0067 s. This retrieval time is satisfactory. To give an idea, it allows to fetch approximately 150 obsels per second.

Graph 8(a) shows the comparison of serialization times of traces in single-user mode and two-users mode. We can see that the serialization times of traces in single-user mode and two-users mode are similar. The serialization time (t) is proportional to the number of obsels (n) in trace: $t \approx n \times \alpha, \alpha = 0.0066$ s.

Graph 8(b) shows the comparison of computation time of transformed traces (fusion and filter) in single-user and two-users mode. The computation time for filtering traces is longer than the one for merging traces. Indeed, for filtering traces, searching successive obsels in source traces is time consuming, whereas there is no need to do it for fusion. The search and computing is incremental. This means that the search and computation time(t') only depends on the number of unprocessed obsels(n'): $t' \approx n' \times \beta, \beta = 0.026$ s.[10]

The previously processed (transformed) obsels are omitted in this phase thanks to an incremental counter which ensures that an obsel is processed only once. Thus, at run-time, the retrieving time almost reaches the serialization time because the number of unprocessed obsels is often small.

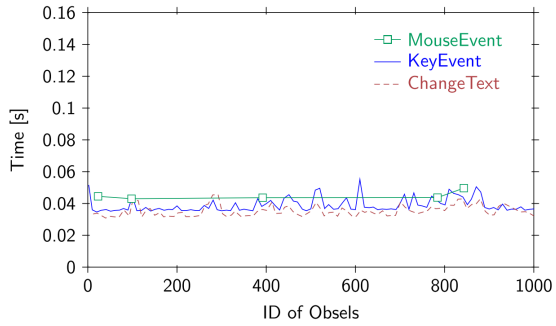
6.3 Discussion on the results

Thereafter, we analyse the results with regard to the four requirements described in section 2.

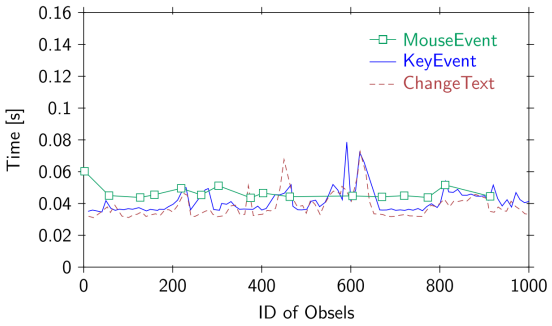
The first requirement is to collect traces matching a model depending on the task. In Collectra, we have defined a trace model suitable for the tasks conducted in Kolflow. This first requirement is then met.

The second requirement is about time performances. On graph 7(b), we observe a peak during the creation of the six hundredth obsel. This sharp increase in the time of creation is explained by simultaneous actions of two users in the same page of the wiki3. Indeed, wiki are not designed to support simultaneous edition of pages. When user1 and user2 edit a page at the same time, Collectra sends the same obsel twice to the kTBS. Therefore, creation time is necessarily higher.

[10] Alpha (α) is the linear regression coefficient of serialization time on the number of obsels in a trace. Beta (β) is the linear regression coefficient of search and computation time on the number of unprocessed obsels in source traces. These two values depend on the process capability of the server.

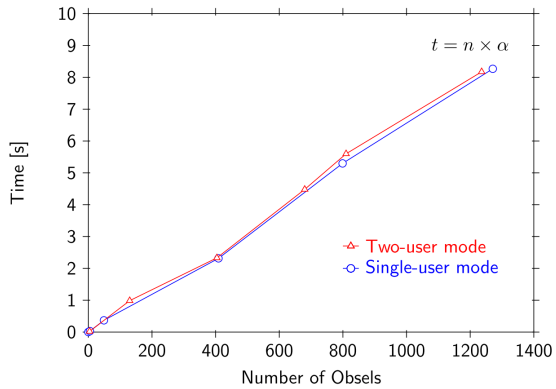


(a) Time to create obsels in single-user mode

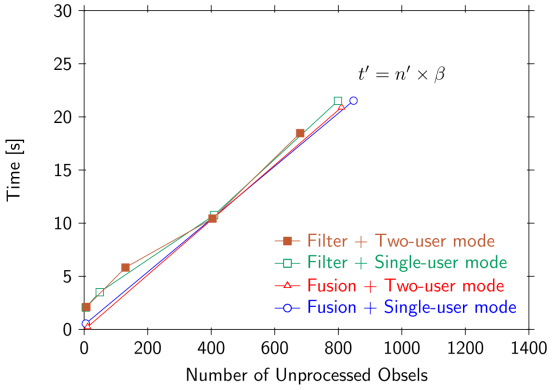


(b) Time to create obsels in two-users mode

Figure 7: Time to create obsels.



(a) Serialization times of traces in single-user mode and two-users mode.



(b) Computation times of transformed traces (fusion and filter) in single-user mode and two-users mode.

Figure 8: Experimental results.

Apart from this peak, we observe that the creation time of obsels remains stable over time and do not slow down users' activities.

The third requirement is about memory space used to store collected traces. Results presented in section 6.2 show that our approach works satisfactorily on a local scale. Furthermore, due to the distributed nature of the approach, response time is linear and therefore scales well. However, simple estimations show that memory space quickly becomes an issue. For example, collecting traces of a single user during 4500 hours produces 500GB of data. The same amount of data is collected when observing 2 users during 3000 hours. For 12 users working during 21 days, we also collect 500GB of data. Such performances are satisfactory in the context of the Kolflow project, but for a bigger application (such as Wikipedia), they could rapidly become an issue. These results also show that the required memory space depends on the number of attributes of each obsel. To obtain satisfactory performance on a large scale, it is therefore important to carefully choose the attributes to be stored in each obsel. In this project, we have decided to provide a complete model. However, if one is interested in improving performances, the best thing to do is to simplify the collect model in order to reduce the number of attributes per obsel.

The last requirement is to ensure that retrieval time of collected traces is fast enough to enable online assistance. Results on the second part of the experiment show that retrieval time is satisfactory. However, further experiments in real conditions with the assistance engine are required to better evaluate this point.

7. CONCLUSION

In this paper, we presented an architecture and a tool, Collectra, to collect interaction traces of multi-user in distributed environments. Collected interaction traces are intended to be used by the assistance engine developed in the Kolflow project. To guaranty the success of the assistance, we must ensure that both the collection process and the retrieval of collected traces are performed in satisfactory response times.

We conducted experiments to evaluate Collectra with respect to these issues. Experiments show that our architecture ensures satisfactory response times in real-world situations. However, memory space remains an issue. Memory usage is not a problem for small scale problems, but it can become an issue on the long run. This result raises new issues regarding memory management, storage methods and forgetting policy in trace-base management systems. We consider this aspect as a major research question for this

work.

Now that we can rely on efficient collection module, we can focus on the next step for providing assistance to Kolflow users. This step is to continue developing the assistance engine, and its interface. This interface will be built on top of requests that we already have designed and experimented. Based on user feedback, we will assess the quality and relevance of the different modes of assistance provided by the assistance engine.

More broadly, we want to study the relevance of our framework in other contexts than the one of the Kolflow project. At the moment, Collectra has been implemented as a plug-in for DSMW and has been experimented in this context. The trace model has been designed in order to collect interaction traces of DSMW users. In order to apply our model to other application domains, two important steps have to be accomplished. Firstly, a new trace model has to be defined, with respect to the chosen application. Secondly, slight modifications have to be made to the Collectra API in order to connect it to the observed application. In future work, we will experiment our approach with other distributed and collaborative activities, in order to identify strengths and weaknesses of our approach.

Acknowledgment

We are grateful to Pierre-Antoine Champin, Emmanuel Desmontils for their comments on this work. This work is supported by the French National Research agency (ANR) through the Kolflow project (code: ANR-10-CONTINT-025), part of the CONTINT research program. More information about Kolflow is available on the project website: `http://kolflow.univ-nantes.fr/`

8. REFERENCES

[1] L. Allen, G. Fernandez, K. Kane, D. Leblang, D. Minard, and J. Posner. ClearCase MultiSite: Supporting Geographically-Distributed Software Development. *Software Configuration Management: Scm-4 and Scm-5 Workshops: Selected Papers*, 1995.

[2] N. Avouris, V. Komis, G. Fiotakis, M. Margaritis, and E. Voyiatzaki. Logging of fingertip actions is not enough for analysis of learning activities. In *AIED 2005*, pages 1–8, 2005.

[3] O. Bell, M. Allman, and B. Kuperman. On browser-level event logging. Technical Report TR-12-001, International Computer Science Institute, Berkeley, CA, Jan. 2012.

[4] M. Buffa, F. L. Gandon, G. Ereteo, P. Sander, and C. Faron. Sweetwiki: A semantic wiki. *Journal of Web Semantics*, 6(1):84–97, 2008.

[5] P.-A. Champin, A. Cordier, E. Lavoué, M. Lefevre, and H. Skaf-Molli. User Assistance for Collaborative Knowledge Construction. In A. DL, editor, *WWW 2012 - SWCS'12 Workshop*, pages 1065–1073, Apr. 2012.

[6] P.-A. Champin, Y. Prié, O. Aubert, F. coise Conil, and D. Cram. kTBS: Kernel for Trace-Based Systems, 2011.

[7] A. Cordier, B. Mascret, and A. Mille. Extending Case-Based Reasoning with Traces. In *Grand Challenges for reasoning from experiences, Workshop at IJCAI'09*, July 2009.

[8] M. d'Aquin, S. Elahi, and E. Motta. Personal Monitoring of Web Information Exchange: Towards Web Lifelogging. 2010.

[9] P. Dourish. The parting of the ways: Divergence, data management and collaborative work. pages 215–230. Kluwer Academic Publishers, 1995.

[10] J.-M. Heraud, J.-C. Marty, L. France, and T. Carron. Helping the Interpretationof Web Logs: Application to Learning Scenario Improvement. In *AIED'05*, 2005.

[11] M. Krötzsch, D. Vrandecic, M. Völkel, H. Haller, and R. Studer. Semantic wikipedia. *Journal of Web Semantic*, 5(4):251–261, 2007.

[12] M. May, S. George, and P. Prévôt. Tracking, analyzing, and visualizing learners' activities on discussion forums. In *Proceedings of the sixth conference on IASTED International Conference Web-Based Education - Volume 2*, WBED'07, pages 649–656, Anaheim, CA, USA, 2007. ACTA Press.

[13] J. Oh, S. Lee, and S. Lee. Advanced evidence collection and analysis of web browser activity. *Digital Investigation*, 8(0):S62–S70, 2011.

[14] C. Rahhal, H. Skaf-Molli, P. Molli, and S. Weiss. Multi-synchronous Collaborative Semantic Wikis. In *10th International Conference on Web Information Systems Engineering - WISE '09*, volume 5802 of *LNCS*, pages 115–129. Springer, October 2009.

[15] S. Schechter, M. Krishnan, and M. D. Smith. Using path profiles to predict HTTP requests. In *Proceedings of the seventh international conference on World Wide Web 7*, WWW7, pages 457–467, Amsterdam, The Netherlands, The Netherlands, 1998. Elsevier Science Publishers.

[16] L. S. Settouti, Y. Prié, P.-A. Champin, J.-C. Marty, and A. Mille. A trace-based systems framework : Models, languages and semantics. *Framework*, 5205:40, 2009.

[17] H. Skaf-Molli, G. Canals, and P. Molli. DSMW: a distributed infrastructure for the cooperative edition of semantic wiki documents. In *Proceedings of the 10th ACM symposium on Document engineering*, DocEng'10, pages 185–186, New York, NY, USA, 2010.

[18] H. Skaf-Molli, C. Rahhal, and P. Molli. Peer-to-peer semantic wikis. In *20th International Conference on Database and Expert Systems Applications- DEXA 2009, Lecture Notes in Computer Science 5690, Springer*, volume 5690 of *Lecture Notes in Computer Science*, Linz, Austria, August 2009.

[19] Q. Yang, C. X. Ling, and J. Gao. Mining web logs for actionable knowledge. In *Intelligent Technologies for Information Analysis*, pages 169–192. Springer Heidelberg, 1998.