Journal of Universal Computer Science, vol. 8, no. 10 (2002), 913-923
submitted: 24/4/02, accepted: 15/10/02, appeared: 28/10/02 © *J.UCS*

The TrailTRECer Framework: Applying Open Hypermedia Concepts to Trails

Erich Gams
(Salzburg Research - SunTREC
Jakob Haringer Strasse 5/III
5020 Salzburg, Austria
egams@salzburgresearch.at)

Siegfried Reich
(Salzburg Research - SunTREC
Jakob Haringer Strasse 5/III
5020 Salzburg, Austria
sreich@salzburgresearch.at)

Abstract: Being lost in space and overloaded with information are two key problems users are confronted with, when searching for appropriate information. Trails built from information about the users' browsing paths and activities, are an established approach to assist users in navigating vast information spaces. However, existing trail-based systems are focusing on browsers only and therefore do not fully exploit the notion of trails. The *TrailTRECer* framework addresses these issues by being open to any application and any activity. The usability of the framework and the concept of user trails were tested by building a navigation support system with different trail-enabled clients.
Key Words: open hypermedia, trails, trail-based system, navigation support
Category: H.3.3, H.5.4

1 Introduction

Looking for specific information or finding colleagues working on similar topics, has been and still is an open issue. Growing numbers of documents on the World Wide Web makes it increasingly difficult to find appropriate information. The idea of using trails to assist users in navigation has been around since the early days of hypertext and several navigation support systems including recommender systems have been built (See [17, 10] for an overview). Most browsing advisor tools offer some kind of history mode of recently accessed web pages, however, they do not take into account the different types of activity or the types of documents [17, 18]. By activity we refer to the action performed on a document, rather than hypertext activity [16]. This type of activity is important for the documents' relevance and therefore has to be traced, too. Different types of activity also imply that trails can be generated by any application, i.e., mail clients, word processors, etc. not just Web browsers.

Based on these findings we introduce a data model and the architecture of *TrailTRECer*, an open framework that enables the development of trail-enabled applications, for assisting users in navigation or searching for appropriate information. Its name indicates that, similar to the way "Trackers" follow trails in nature, "TrailTRECers" make their way through digital information spaces. Furthermore, we will describe different attempts of third party tool integration into open hypermedia systems. Finally, we will demonstrate the applicability of our framework to arbitrary applications by integrating a client watching print jobs, a browser client and an office client, that are able of acquiring application specific activities and visualizing recommended documents.

2 A Data Model based on FOHM

Whereas most systems simply do some analysis on browsing behaviour, we introduce trails as objects, which can be edited, deleted, copied or exchanged (i.e. so called "first-class" objects). Trails can be defined as vectors of so called trail marks. A trail mark can be anything (e.g. document, website) that can be clearly identified by a URI. For the basic entities that a trailmark and trail are built of and we also call trail-data see [14].

Based on this core trail definition and the fact that any application can produce trails, we developed a trail data model and the architecture of the *TrailTRECer* framework. The basic requirements of trail-based systems were introduced in [6]. According to the OHSWG's (Open Hypermedia Working Group [15]) reference architecture, trail functionality could simply be implemented as another middleware service. To promote interoperability to e.g. other link servers (as suggested by the OHSWG), our framework integrates trails as another hypertext application domain such as the navigational, taxonomic or spatial domain. Mapping trails into an Open Hypertext Model, offers the possibility to interoperate with other link servers and thus different ways of navigation. In our approach, we followed the fundamental open hypermedia model (FOHM [11]), a common data model capable of representing structures and implementing operations from any of the three domains, and we have expressed trail data using that model.

The basic items of the data model are associations holding vectors of bindings, a relationship type, and a structural type. Following that definition, trails can be defined as associations. In order to allow time dependent ordering, an *ordered list* will be added to the structural types. Bindings hold data references and feature vectors together. Trail marks resemble bindings, i.e., they relate nodes (which are references to the actual documents) and trails together.

$$\begin{aligned}
\mathcal{A} &= \overrightarrow{\mathcal{B}} \times \mathcal{T} \times \mathcal{S} && \text{(Associations)}\\
\mathcal{S} &= \{\mathsf{heap}, \mathsf{list}, \mathsf{stack}, \mathsf{orderedlist}, \ldots\} && \text{(Structural Types)}\\
\mathcal{B} &= \mathcal{I}^r \times \overrightarrow{\mathcal{F}} && \text{(Bindings)}\\
\mathcal{N} &= \{\mathsf{direction}, \mathsf{shape}, \mathsf{activity}, \mathsf{ranking}, \mathsf{date}, \ldots\} && \text{(Features Spaces)}
\end{aligned}$$

The feature space vector enumerates the different properties that must be defined in a feature vector of each binding. With respect to retrieving relevant trail marks, we extended the feature space with the trail mark properties *activity*, *ranking* and *date*. Summarizing, FOHM could be adapted with some simple modifications to hold trail data. We argue that these modifications are useful for other domains as well. For example, in the navigational domain, a ranking feature could be used for prominently displaying those bindings with high ranking values.

3 Framework Architecture

By framework we stick to the definition of Bernstein [3], who argues that "a framework is a software environment that is designed to simplify application development and system management for a specialized application domain." The framework is a kind of reuseable design of a system that decomposed into a set of interacting components. Thus, the framework and its components are cooperating technologies [9]. An open trail-based framework has to provide a set of services for recording, storing, processing and navigating trails.

The distributed management of trail information and the dynamically interacting components (e.g. different trail processing components need to re-use results provided by other components) makes a trail-based system and framework well suited to be based on an multi-agent architecture [5].

SoFAR [12], the **So**uthampton **F**ramework for **A**gent **R**esearch, is a multi-agent framework in Java that addresses the problem domain of distributed information processing and acts as a basis framework *TrailTRECer*. Users can combine an arbitrary number of agents, solving different tasks by processing trail data (e.g. recommendation of related items [6]). Users have the possibility to compose their individual workspaces. In order to communicate with each other, the agents have to agree on a common ontology as the topology of the exchanged messages. By representing the trail extended FOHM in an ontology, we want the framework to be open to any agent, that supports hypermedia functionality.

In order to facilitate information sharing, SoFAR provides a registry to help agents advertise and find information. Every agent of the system has to subscribe to a registry agent to inform about the own functionality and to find certain capabilities supported by other agents. To communicate and exchange information, agents use ontologies to define the content of messages.

```
<term name="Trailmark" extends="Predicate">
    <field type="String" name="node"/>
    <field type="String" name="activity"/>
    <field type="String" name="date"/>
    <field type="String" name="duration"/>
    <field type="String" name="user"/>
    <field type="String" name="rating"/>
</term>

<vector name="Trail" type="Trailmark"/>

<vector name="Trailvector" type="Trail"/>

<vector name="Personvector" type="String"/>

<term name="Related" extends="Predicate">
    <field type="Trailmark" name="trailmark"/>
</term>

<term name="RelatedPersons" extends="Related">
    <field type="Personvector" name="personvector"/>
</term>

<term name="RelatedDocuments" extends="Related">
    <field type="Trail" name="trail"/>
</term>

<term name="RelatedTrails" extends="Related">
    <field type="Trailvector" name="trail"/>
</term>
```

The root of the SoFAR ontology hierachy is an abstract term and a predicate, a term that can be queried about. As our framework aims at building trail based recommender systems, the basic term *Related* of the trail ontology enables the exchange of related documents, trails or persons. Any agent, that wants to receive related documents, can query for that *Related* predicate.

Figure 1 shows the basic parts of the framework that can be adapted to the users' needs. All the agents are connected to the Registry Agent and they are able to communicate with each other via the registry.

The architecture of the framework contains two main platforms, on which the agents reside. All the tasks, such as collecting data, or visualization are distributed on different agents. Each user holds an individual user platform, that

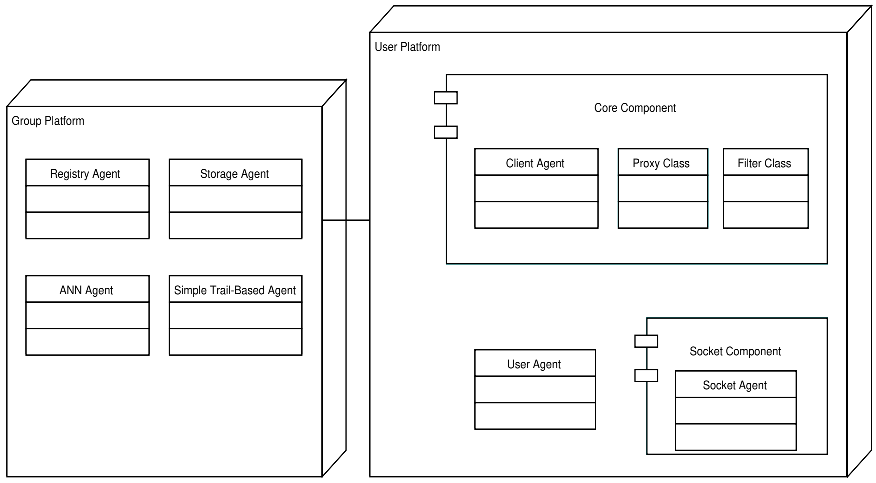

Figure 1: Framework architecture

contains the trail acquisition (Socket Agent, Client Agent) and trail visualization agents (User Agent). The group platform, holding the processing agents, is responsible for trail processing and storing. Our framework basically aims at supporting: trail visualization and trail acquiring.

4 Integration Scenarios

In this section we give examples of integrating different applications or functionalities into our framework to enable trail functionality. According to our trail data model [14] the document URL and the activity performed on the document are the two basic entities that we want to acquire. Our aims is to envisage how people handle documents, to later on use this acquired information for supporting their navigational and information seeking tasks. More precisely handling documents stands for how people, communicate and search for and leave information about documents. These activities correspond to functionalities provided by different groups of systems and applications. Basic activities (e.g. print) are enabled by functionalities provided by the operating system. Communication and exchange of information take place, when documents are sent over the Web (e.g. mail, newsgroups). Finally most of the documents are created or edited by some third party applications (e.g. Office). Visualization integration is useful in application providing an API for adding new functionality and providing control over presentation of trail data. The user can then make use of recommendations provided by the *TrailTRECer* system related to the document in work.

Efforts of integrating open hypermedia functionality into third-party applications are also relevant for our work (see [19, 4, 1] for remarkable examples). [19] promotes an architectural model which allows the characteristics of an application, relevant to its integration with an open hypermedia system, to be modeled. The initial state of an application's communication ability can be described as being either native, non-native or non-communicative. While an native application fully understands the communications protocol of an open hypermedia system, a non-communicative application does not have any API. A non-native application has an external application programming interface (API), differing from the open hypermedia system protocol. Considering the applications initial state application integration architectures fall into three categories:

1. The launch-only integration architecture only provides link traversals ending at the application. The possibility to mere start up an application is not relevant to a trail-based system.

2. A wrapper (or shim) is an integration containing a separate element, acting as an intermediary between the application and the hypermedia system. Translating unstructured information into trail data is main task in the trail-based framework.

3. Finally, the custom integration architecture is implemented by modifying the source code or writing code in the application's customization language.

Our approach also assumes that the *TrailTRECer* system and the trail-enabled process or application are independent processes and are run independently.

Our approach differs from the above models as we do not only need to integrate applications, but are mainly interested in different activities performed on documents. Based on these assumptions the framework offers three possibilities for a new applications to be integrated. In the following integration descriptions we also differ between application developer, who is the person adapting the framework for personal needs and the user, recording trails and customizing the recommendations.

4.1 Integration of Core Activities

Trail-related information is accessable via operations of the operation system. The Client Agent of our framework is able to retrieve trail data and forward it to the Storage Agent. The application developer is responsible for the retrieval of data of any application and convertion to trail data. A core template class (Filter Class) offered by the framework can be adapted to a users needs by implementing so called hook methods [13]. The hook method is an empty abstract class, providing an empty default implementation and parameterizing the template class. According to the integration architectures this type of integration is equivalent to a pure wrapper integration.

Example: Printed Documents One measure of importance of a document is, whether documents have been printed. Hence, we try to capture all documents printed. However, if you want an applications independent way of detecting whether a document is being printed, one possible way is pooling the print spooler. E.g. a C++ component enumerates all the printers (local and network) on the machine and dumps the information of any current job. The Client Agent of our framework instantiates a Java wrapper hosting this component. Whenever a print job is executed, trail data (the document name and the activity) is dumped and an event triggered.

4.2 Integration of Communication Activities

Many applications enable users to communicate with each other and thus exchange information over networks (e.g. attach a document to an email) The framework offers some functionality that considers capturing of trail-related information that is exchanged over a network. Like browsing advisors (e.g. [8], [7]) use proxies to retrieve and provide additional link information, we implemented a generic proxy (Proxy Class)for our framework being able to process every port by an own function and filter the trail relevant information. The application developer benefits from the fact that standardized protocols use well known port numbers. The developer has to maintain which port to watch and to implement the component responsible for filtering trail relevant data out of the protocol used.

Example: Mailed Documents A document mailed to a friend, may be an indication for importance of the document to the sender. Thus we record all documents mailed to someone. The User is not affected, but by just initially changing the proxy settings of the mail tool and can then continue using the favourite mail client. All mails are filtered and the name of the attached document is captured.

4.3 Integration of Third Party Applications

The third and last integration type is most independent from the TrailTRECer system. All the event functionality can be implemented in the third party application completely individual from the framework. Mostly applications offer some kind of API or internal customization language such as Visual Basic for (Windows) Applications. This integration is corresponding to the custom integration type introduced in [19]. Due to the large degree of manipulation control, it has the advantage of providing trail-data on a very fine grained level. These applications have to provide the ability to communicate either via a network socket connection or via a text stream to a captive process. The documents and activities that should be added to a current trail have to be converted to a fixed with the framework commited format. The trails have to be sent to the Storage Agent which holds a a server socket on the local machine. The framework runs a process retrieving the messages and processing them.

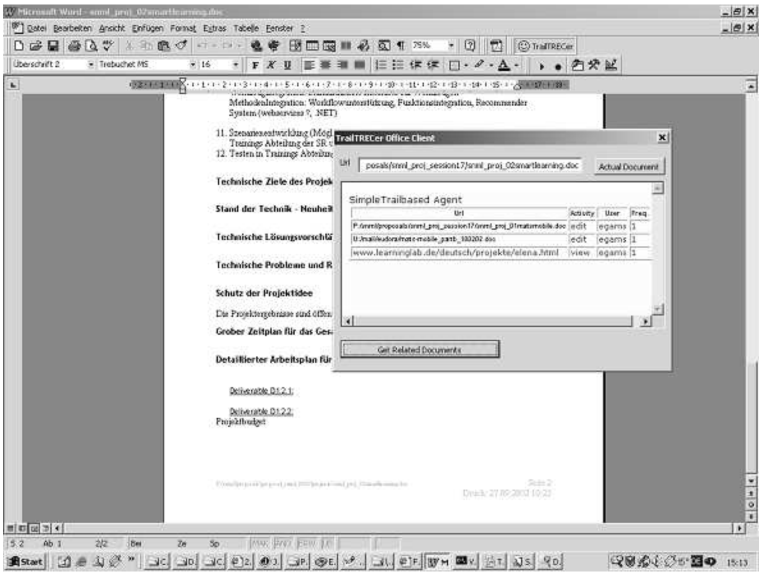

Figure 2: Office Client Screenshot

Example: Edited Documents For the integration of a third party integration we chose MS Office. MS Office allows the creation of custom user menus and the ability to add user defined functionality with Visual Basic for Applications (VBA). We decided to add a document to a trail, whenever it was changed (e.g. create a new document or making additions to an existing one). An internal event will be triggered and handled by our personally implemented macro functions. These functions communicate with the Socket Agent via a socket connection and send the required data in a fixed format. If recommendations to the current opened document are needed, the user can press the "Related Documents" button and the name and path of the actual active document is sent to the User Agent and processed by the individual agents. The result is returned and visualized with an integrated webbrowser control in the Winword dialog (Figure 2). To execute the macro it has to be imported in the currently used Winword template (Normal.dot).

4.4 Sample Navigation Assistance Scenario

In order to promote the feasability of trails for recommendation and to test the applicability of our framework, we integrated two trail recommending agents, one simply recommending the next neighbour node and the other one based on processing an artificial neuronal network, in the system. (For details on the implementation see [6, 2]). The recording of trails and the types of processing agents involved in the recommendation process can be controlled via a user interface component. The system searches for all agents available to deliver related documents and the user can decide which agents to use. The user can access the delivered recommendations and through a User Agent, connected to a sidebar in Netscape 6.2. and menu command in Winword. Users testing the system in first experiments gave positive feedback regarding the quality of the recommendation results, extracted from trail processing.

5 Experiences with the Integration

Summarizing, the effort of integration is highly depending on the type of data source filtered for trail data. Often simple core processes e.g polling the printer spooler, are not directly accessable via the Java API. Thus, implementing a component supporting low level operating system functionality can be very time consuming. For this integration of core activities the framework provides support through the Filter Class and a trail persistence mechanism, responsible for the storage of trails.

The most convenient way and less expensive way of integrating an application into our framework is to gather data that is communicated over a TCP/IP port. Supporting this case the framework provides additionally port scanning and filtering functionality. The application developer only needs to assign the port and the destination server in a simple text file, and to implement a function that structures the protocol and filters the relevant trail data. Knowledge only about the Filter Class and about the trail data model is required, when integrating core and communication activities.

The easiest and most independent way to integrate an application and is to use the Socket Component. Although no knowledge about the framework is needed, the effort to do this so-called custom integration can be high, depending on the degree of knowledge about the application customization language. However, if fine-grained activities (e.g. activities only associated with a particular application) and data are desired, custom integration provides the best results.

Trail visualization integration is mostly needed on application level and can easily be done if the applications supports HTTP access. The HTML data, sent back from the User Agent running a web server, can be visualized either via some

HTML viewer or can be interpreted and translated into an application specific format.

Finally we can say that all the integration examples need some kind of wrapper to convert the application data gathered into trail data according to the framework's data model.

6 Summary and Perspectives

In this paper we argued how open hypermedia concepts including the OHSWG's reference architecture, the FOHM data model and the openness with respect to applications, can be applied to trail-based systems. We presented the data model and the architecture of the *TrailTRECer* framework, an agent based platform for integrating trail processing components. The framework is adaptable for the integration of different processes and applications. We described three example integrations.

Future work will focus on capturing trails in an Intranet scenarios, e.g. shared project directories, and developing additional processing agents for more fine-grained recommendations. Further research will include analysing the user navigation and exploiting the results for trail similarity algorithms and topic extraction.

Acknowledgments

The TrailTRECer framework is part of the Trailist project which is supported by the Austrian Fonds zur Förderung der wissenschaftlichen Forschung (FWF) under grant No. P14006-INF.

References

1. Kenneth M. Anderson. Integrating open hypermedia systems with the world wide web. In *Proceedings of the '97 ACM Conference on Hypertext, April 6-11, 1997, Southampton, UK*, pages 157–166, April 1997.
2. Tobias Berka, Werner Behrendt, Erich Gams, and Siegfried Reich. Recommending internet-domains using trails and neuronal networks. *Int. Conf. on Adaptive Hypermedia and Adaptive Web Based Systems, Malaga*, pages 201–202, May 2002.
3. Philip A. Bernstein. Middleware: A model for distributed system services. *Communications of the ACM*, 39(2):86–98, February 1996.
4. Hugh C. Davis, Simon Knight, and Wendy Hall. Light hypermedia link services: A study of third party application integration. In *ECHT '94. Proceedings of the ACM European conference on Hypermedia technology, Sept. 18-23, 1994, Edinburgh, Scotland, UK*, pages 41–50, 1994.
5. David C. DeRoure, Wendy Hall, Siegfried Reich, Aggelos Pikrakis, Gary J. Hill, and Mark Stairmand. An open framework for collaborative distributed information management. *Seventh International World Wide Web Conference (WWW7), 14 - 18 April 1998, Brisbane, Australia*, 30:624–625, 1998. Published in Computer Networks and ISDN Systems.

6. Erich Gams, Tobias Berka, and Siegfried Reich. The TrailTRECer Framework: A platform for trail-enabled recommender applications. In *Conference on Database and Expert Systems DEXA '02, Aix-en-Provence, France (Berlin/Heidelberg/New York), Sept. 2002*, pages 638–647, September 2002.
7. Michel Jaczynski and Brigitte Trousse. Broadway: A case-based system for co-operative information browsing on the world-wide-web. In *Collaboration between Human and Artificial Societies*, pages 264–283, 1999.
8. Roger James, Alastair Hotchkiss, and Steve Loades. D 2.2 report on prototype test and evaluation programme. Technical Report UIC/MEMOIR/D2.2, ESPRIT Programme, MEMOIR Project (22153), February 1997.
9. Ralph E. Johnson. Frameworks = (components + patterns). *Communications of the ACM*, 40(10):39–42, October 1997.
10. Unmil P. Karadkar, Luis Francisco-Revilla, Richard Furuta, Haowei Hsieh, and Frank M. Shipman III. Evolution of the walden's paths authoring tools. In *Webnet 2000, San Antonio, TX*, pages 299–304, October 2000.
11. David E. Millard, Luc Moreau, Hugh C. Davis, and Siegfried Reich. FOHM: A fundamental open hypertext model for investigating interoperability between hypertext domains. In *Procs. of the '00 ACM Conference on Hypertext, 2000*, pages 93–102, 2000.
12. Luc Moreau, Nick Gibbins, David DeRoure, Samhaa El-Beltagy, Wendy Hall, Gareth Hughes, Dan Joyce, Sanghee Kim, Danius Michaelides, Dave Millard, Sigi Reich, Robert Tansley, and Mark Weal. SoFAR with DIM agents. An agent framework for distributed information management. In *Int. Conf. on The Practical Application of Intelligent Agents and Multi-Agents. PAAM 2000*, pages 369–388, 2000.
13. Wolfgang Pree. *Framework Development and Reuse Support.* Ed.: M. Burnett, A. Goldberg, T. Lewis, Prentice-Hall, 1994.
14. Siegfried Reich and Erich Gams. Trailist - focusing on document activity for assisting navigation. In *Procs. of the Twelfth ACM Conference of Hypertext and Hypermedia*, pages 29–30, 2001.
15. Siegfried Reich, Uffe K. Wiil, Peter J. Nürnberg, Hugh C. Davis, Kaj Grønbæk, Kenneth M. Anderson, David E. Millard, and Jörg M. Haake. Addressing interoperability in open hypermedia: The design of the open hypermedia protocol. *New Review of Hypermedia and Multimedia*, 5:207–248, 1999.
16. Jim Rosenberg. The structure of hypertext activity. In *Procs. of the '96 ACM Conference on Hypertext, 1996*, pages 22–29, 1996.
17. D. De Roure, W. Hall, S. Reich, G. Hill, A. Pikrakis, and M. Stairmand. Memoir - an open distributed framework for enhanced navigation of distributed information. *Information Processing and Management*, 37:53–74, 2001.
18. Alan Wexelblat and Pattie Maes. Footprints: History-rich tools for information foraging. In *Conf. on Human Factors in Computing Systems*, pages 270–277, 1999.
19. James E. Whitehead. An architectural model for application integration in open hypermedia environments. In *Proceedings of the '97 ACM Conference on Hypertext, April 6-11, 1997, Southampton, UK*, pages 1–12, 1997.