

# Handbook of Research on Web Log Analysis

Bernard J. Jansen  
*Pennsylvania State University, USA*

Amanda Spink  
*Queensland University of Technology, Australia*

Isak Taksa  
*Baruch College, City University of New York, USA*

Information Science  
**REFERENCE**

**INFORMATION SCIENCE REFERENCE**

Hershey • New York

Director of Editorial Content: Kristin Klinger  
Director of Production: Jennifer Neidig  
Managing Editor: Jamie Snavelly  
Assistant Managing Editor: Carole Coulson  
Typesetter: Sean Woznicki  
Cover Design: Lisa Tosheff  
Printed at: Yurchak Printing Inc.

Published in the United States of America by  
Information Science Reference (an imprint of IGI Global)  
701 E. Chocolate Avenue, Suite 200  
Hershey PA 17033  
Tel: 717-533-8845  
Fax: 717-533-8661  
E-mail: [cust@igi-global.com](mailto:cust@igi-global.com)  
Web site: <http://www.igi-global.com>

and in the United Kingdom by  
Information Science Reference (an imprint of IGI Global)  
3 Henrietta Street  
Covent Garden  
London WC2E 8LU  
Tel: 44 20 7240 0856  
Fax: 44 20 7379 0609  
Web site: <http://www.eurospanbookstore.com>

Copyright © 2009 by IGI Global. All rights reserved. No part of this publication may be reproduced, stored or distributed in any form or by any means, electronic or mechanical, including photocopying, without written permission from the publisher.

Product or company names used in this set are for identification purposes only. Inclusion of the names of the products or companies does not indicate a claim of ownership by IGI Global of the trademark or registered trademark.

Library of Congress Cataloging-in-Publication Data

Handbook of web log analysis / Bernard J. Jansen, Amanda Spink and Isak Taksa, editors.

p. cm.

Includes bibliographical references and index.

Summary: "This book reflects on the multifaceted themes of Web use and presents various approaches to log analysis"--Provided by publisher.

ISBN 978-1-60566-974-8 (hardcover) -- ISBN 978-1-60566-975-5 (ebook)

1. World Wide Web--Handbooks, manuals, etc. 2. Web usage mining--Handbooks, manuals, etc. I. Jansen, Bernard J. II. Spink, Amanda. III. Taksai, Isak, 1948-  
TK5105.888.H3636 2008  
006.3'12--dc22

2008016296

British Cataloguing in Publication Data

A Cataloguing in Publication record for this book is available from the British Library.

All work contributed to this book set is original material. The views expressed in this book are those of the authors, but not necessarily of the publisher.

*If a library purchased a print copy of this publication, please go to <http://www.igi-global.com/agreement> for information on activating the library's complimentary electronic access to this publication.*

## Chapter XII

# An Integrated Approach to Interaction Design and Log Analysis

**Gheorghe Muresan**  
*Microsoft Corporation, USA*

### **ABSTRACT**

*In this chapter, we describe and discuss a methodological framework that integrates analysis of interaction logs with the conceptual design of the user interaction. It is based on (i) formalizing the functionality that is supported by an interactive system and the valid interactions that can take place; (ii) deriving schemas for capturing the interactions in activity logs; (iii) deriving log parsers that reveal the system states and the state transitions that took place during the interaction; and (iv) analyzing the user activities and the system's state transitions in order to describe the user interaction or to test some research hypotheses. This approach is particularly useful for studying user behavior when using highly interactive systems. We present the details of the methodology, and exemplify its use in a mediated retrieval experiment, in which the focus of the study is on studying the information-seeking process and on finding interaction patterns.*

### **LOGGING THE USER INTERACTION: AN INTRODUCTION**

A good understanding of people – what they are like, why they use a certain piece of software, and how they might interact with it – is essential for successful design of interactive systems, which help people achieve their goals. While each user

is unique, and may have a particular background, context, interest and motivation to use a system, it is necessary to learn what is generally true about the users of a system and what behavioral patterns are common. Specifically, the designer should learn (1) the users' goals in using a system; (2) the specific tasks undertaken in order to achieve some goals; (3) the language or terminology used

by users to describe what they are doing; and (4) the users' experience and skills at using a certain kind of system (Tidwell, 2006). Some common methods and techniques used before and during system design in order to understand the users' needs and to establish system requirements, as well as during the implementation and testing in order to evaluate the usability and effectiveness of a system, are direct observation, interviews, surveys, personas, focus groups.

While these methods are excellent tools for evaluating the quality of the interaction between human and system, the quality of the system in supporting the users to achieve their goals and the user satisfaction, they have a number of drawbacks. First, people are often incapable of accurately assessing their own behaviors, especially when removed from the context of their activities (Pinker, 1999) and therefore interviews and surveys may not provide true answers. Second, direct observation may be obtrusive – the users may be distracted, or they may not behave naturally. Third, they are expensive to run, and therefore provide information from a rather limited sample of users, so the results are often informative, but may lack statistical significance, may miss unusual cases, and may not capture behavioral patterns or trends.

Logging the user interaction with the system provides a complementary tool for analyzing the interaction and evaluating a system. It provides the means to acquiring large quantities of data about patterns of interface usage, speed of user performance, rate of errors, or frequency of requests for online assistance (Shneiderman & Plaisant, 2005). An important ethical issue, which indirectly affects user behavior and therefore the validity of the results, is whether users are told and know that their activity is logged. However, when logging is done in order to evaluate a system rather than user preferences or private activities, and when no personal information is captured, this problem is minimal.

An interesting set of constraints on what data can practically be logged, and on designing a logging system, is dictated by the software architecture of the system being investigated. The simpler situation is that of a standalone system, when the entire user activity runs on the same machine, and where all the data resides. In such situations, if the logging module is designed and built as part of the system, then all user actions, all user events and all data being generated or manipulated can potentially be logged. Logging the interaction with third-party software is more challenging: while operating system-level actions such as keystroke or mouse events, or opening/closing a file, or starting/stopping a certain application can be captured and logged, semantic events specific to a certain application are usually impossible to capture. For example, while it is possible to capture the text typed by a user, it is not easy or even possible to determine if the text was typed as a query for a search engine, or for filling in a form. This problem can be addressed by video-recording the interaction or by using screen-capturing software (e.g., Morae: <http://www.techsmith.com/morae.asp>; TaskTracer: <http://eecs.oregonstate.edu/TaskTracer>; uLog: <http://www.noldus.com>, so that the researchers can subsequently examine the interaction, interpret what is happening, insert annotations or mark significant events. While these tools can be helpful in analyzing the captured data, they rely on the manual-intellectual annotation done by the researcher, and are therefore very labor intensive and error-prone. Moreover, the format used for the logs is usually proprietary, which forces the researchers to buy proprietary analysis software that is not customizable. So, in order to fully benefit from the power of user activity logging, it is preferable that the designer of the logging module has access to the source code of the system being evaluated.

A more complex situation arises in the case of client-server architectures, common for using Web services. The client tier, usually a Graphi-

cal User Interface (GUI) application such as a Web browser or an email tool, runs on the user's machine and supports the interaction between the user and the system. Therefore, a logging module running on the client could capture all the user actions and system events (keystrokes, mouse moves, etc). This could even be synchronized with an eye tracking device to disambiguate some of the user's actions. On the other hand, the server tier runs on a server and provides services such as Web search or access to an email repository. Therefore, a logging module running on the server could capture such service requests, and possibly the results of these requests.

The data captured by server-side and client-side logging are complementary (with some overlap) and are typically used to answer different research questions. Moreover, data from the two types of logs is owned by different entities: server logs are owned by the operator of the server services, e.g. search engines, while the client logs may belong to the institution or research group that installed the client software and logging module on a number of workstations. Ideally, the two entities should collaborate and share data, so that answers to research questions can be corroborated. What is easier to corroborate is results obtained based on data from client-side logging, which support quantitative analysis of a user interface, with complementary results obtained from the qualitative methods and techniques discussed at the beginning of this section.

When talking about logging in a client-server architecture, one needs to clarify whether logging is done at the client side, or at the server side, or both. In the **Information Retrieval** (IR) context that interests us, search engine operators do server-side logging in order to capture, for example, trends in topical user interest or in the sophistication of the query formulation, e.g. the use of query Boolean operators. The users' selection of search results can also be used as feedback for adjusting the estimated quality of search results and thus the order of the search

hits, or the algorithm for generating Webpage summaries. While capturing a large amount of data about service requests coming from a high number of clients, server-side logging misses the details of the user-system interaction. On the other hand, a client-side logging module is able to capture the intricacies of the interaction, but only for the user running the user interface. Such data can be used for evaluating the usability and effectiveness of a user interface, typically with the purpose of improving it.

### **Log Analysis in IR and the Motivation for Our Work**

While much of the research work in Information Retrieval has focused on the systemic approach of developing and evaluating models and algorithms for identifying documents relevant to a well-defined information need, there is increasing consensus that such work should be placed in an Information Seeking framework in which a searcher's context, task, personal characteristics and preferences should be taken into account (Ingwersen & Jarvelin, 2005).

Since Robertson and Hancock-Beaulieu (1992) described the cognitive, relevance and interactive "revolutions" expected to take place in IR evaluation, the focus in interactive IR experimentation has shifted to exploring the dynamic information need that evolves during the search process, the situational context that influences the relevance judgments and the strategies and tactics adopted by information seekers in satisfying their information need. This paradigm shift to a cognitive approach to exploring search interactions and to studying Human Information Behavior has generated a large number of theories that attempt to model the search interaction and to predict the user's behavior in different contexts and at different stages of the interaction (Fisher, Erdelez, & McKechnie, 2005).

Of particular interest to this author are models of the search interaction process and empiri-

cal work to validate such models by observing consistent patterns of user behavior (Ellis, 1989; Kuhlthau, 1991; Belkin et al., 1995; Saracevic, 1996; Xie, 2000; Vakkari, 1999, 2001; Olah, 2005). The interest is not simply in validating theoretical models, but also in (1) developing methodologies to explore behavioral models; and (2) designing systems that implement appropriate interaction design patterns (Cooper, Reinmann, & Cronin, 2007), that better respond to user needs, that can adapt to support various search strategies, and that offer different functionality in different stages of the information seeking process.

Therefore, we are interested in methodologies for running interactive IR experiments, and especially in client-side logging of the interactions and analyzing the log data in such a way that the meaningful details of the interaction are captured and used for quantitative analysis. Let us clarify that we are not dismissing the techniques used for capturing qualitative data about the user interaction, such as direct observation and note-taking, questionnaires and interviews; such data is particularly useful for understanding the users' goals and motivation and for disambiguating user actions. However, we believe that the quantitative analysis of interaction logs is more suitable for observing patterns of behavior, for building a model of the interaction and possibly for predicting user behavior in certain contexts, or simply for testing the usability of a user interface. For example, we can capture the users' predilection for a certain kind of retrieval strategy (e.g. query-based searching vs. browsing), the users' use of advanced query operators or advanced terminology, or the common mistakes made by users, and correlate these with the users' search experience, familiarity with a domain, training, motivation, etc. in order to predict factors that could improve retrieval effectiveness and user satisfaction.

It is often recommended that the retrieval session be evaluated from multiple viewpoints, so that quantitative and qualitative measures are

corroborated, and so that objective measures of performance are compared to users' subjective perception of success and satisfaction (Belkin & Muresan, 2004; Sauro, 2004). However, there is no consensus on methodologies and measures for estimating retrieval effectiveness or success, especially for interactive retrieval on the Web. Therefore, there is no consensus on what data an interaction log should capture.

IR experiments are often run in order to answer some research hypotheses or questions, so capturing just the data predicted to answer these questions sounds reasonable. However, limiting the logging to such data may be too restrictive in the long run: new, more detailed questions may arise from the initial analysis, and richer data may be needed to answer them. On the other hand, one may be tempted to capture "all" that happens, in order to be able to conduct any post-hoc analysis. However, this approach may produce too much useless data and may be counter-productive. For example, if the state of the system is captured in tenth of a second increments, most of the data would probably be useless. On the one hand, capturing only changes in the system state, when they occur, would produce data that is relevant and easier to analyze. Also, capturing all mouse moves and clicks may be useless without context: while knowing that the user clicked on the "Search" button to submit a query is essential, knowing that the user clicked on the screen at position  $(x, y)$  is hard or impossible to interpret.

What we propose is that what should be logged is all semantic events and actions, i.e. events and actions that make sense and are interpretable for a certain system or user interface. For example, mouse moves or clicks are only semantic events if they represent interface actions such as button clicks, selection from a list or menu, copying or pasting, or scrolling of a list of search results. The essential question "*Which are the semantic events for a certain user interface?*" is addressed by our **integrated approach to interaction design and logging**. During the conceptual design

of the interaction and of the user interface, the design team builds the interaction model of the system, i.e. the functionality supported by the user interface and the valid sequences of actions and events that implement the model.

In particular, in the common case that the **Model View Controller (MVC)** design pattern (Gamma et al., 2005) is used for reifying the conceptual model of the system, the design of the controller drives the design of the logging module: the events that affect the model (which maintains the application data) and the views (for displaying data on the screen) are the semantic events that need to be captured in the logs. The consequence is a tight coupling between the design of the controller and that of the logger: all the events to which the controller reacts, and which affect the model or views, must be logged. Optionally, in order to increase the efficiency of the log analysis and to support testing of the log accuracy, intermediary data resulting from these effects can also be logged. Logging such data becomes necessary, rather than optional, when the data depends on the context and the time when the event occurs (e.g. the list of results returned by a Web search engine). The consequence is that the complete interaction flow and the changes of the system's state can be "re-played", analyzed and interpreted based on the log data.

## **A FRAMEWORK FOR MODELING THE INTERACTION AND THE LOGGING**

What we propose is a formal procedure that integrates the modeling of the interaction, the logging process and the log analysis, so that (i) a conceptual model of the interaction is developed to capture the functionality of the system, its states, the valid user actions in each state, and the possible flow of the interaction as the system is used; (ii) the user interface accurately implements the conceptual model of the interaction intended to be supported;

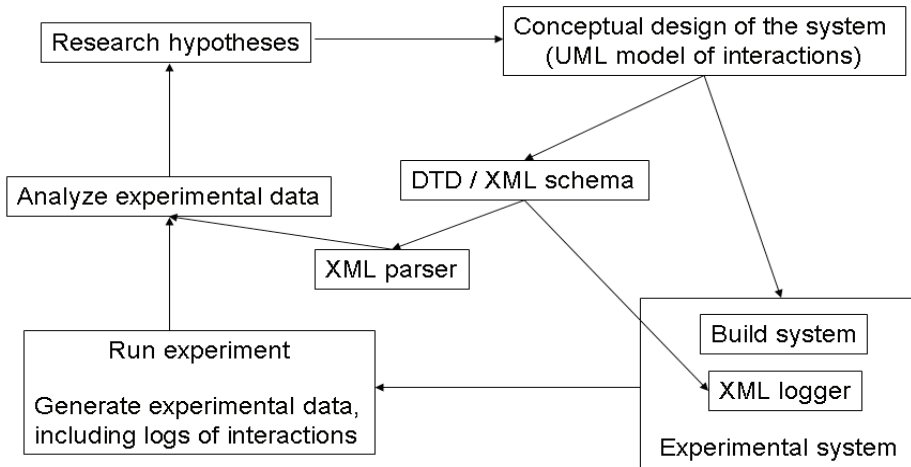
(iii) the valid, semantic events are captured in the logs, together with the state transitions, so that the sequence of state can be re-created when analyzing the logs (optionally, the states of the system can also be captured explicitly); and (iv) the logs can be analyzed in a systematic and at the same time flexible way. When applied to a particular kind of interaction (such as interactive information retrieval), the proposed procedure can be used to investigate user behavior or to test the usability of a user interface.

Naturally, the proposed approach is most suitable for standalone architectures, or for client logging in a client-server architecture, when the source code of the logger and of the actual application can be integrated easily, so that all the details of the human-computer interaction can be captured. In other configurations, a more restricted version of the approach could, in principle, be applied, based on the observable semantic events. For example, if user interaction with a third party system is studied (e.g. accessing a commercial search engine via a Web browser), then some effort is needed to recognize significant, semantic events and actions among the keyboard and mouse events that take place during the interaction.

Figure 1 captures the proposed experimental setting. What distinguishes this model from the typical experimental setting is the requirement for a conceptual model of the system and of the interaction, from which the design of the logger and of the log parser and analyzer are deterministically derived. It is common in experimental IR, especially for small teams and small budgets, to skip the formal modeling of the interaction, and to insert logging instructions in the application code in an ad-hoc, un-systematic fashion, rather than to formally design the logging module. Therefore, when analyzing the logs, it is difficult to relate the captured events to the states of the system or to the stages of the interaction.

While our approach means more work at the onset, and may seem un-necessary when the experimental schedule is tight, it pays off in the

Figure 1. Integrated approach to design, logging and analysis



long run. Moreover, the entire research team can participate in the conceptual design, with the advantages that some mistakes and omissions may be avoided, the team members have a better understanding of the underlying interaction model, and the work can be more easily shared. This contrasts with the common situation when the designated programmers build the system and other members of the research team do the log analysis, with insufficient collaboration.

In practice, our approach is based on statecharts (Harel, 1988) or, in the more modern Unified Modeling Language (UML)<sup>1</sup> terminology, on state diagrams. These are extensions of finite state machines (Wagner, 2006), in which the use of memory and conditional transitions make it practical to describe system behavior in reasonably compact diagrams. Such a model of a system describes: (i) a finite number of existence conditions, called **states**; (ii) the **events** accepted by the system in each state; (iii) the **transitions** from one state to another, triggered by an event; (iv) the **actions** associated with an event and/or state transition (Douglass, 1999; Fowler, 2004). Such diagrams have the advantage that they describe in detail the behavior of the system and, being relatively easy to learn and use, allow the

participation of the entire research team in developing the conceptual model of the IR system to be employed in an experiment. It also makes it easier for the designated programmers to implement and test the system, as the logic is captured in the model.

While UML is well suited to design the interaction supported by a user interface, XML is an excellent choice of format for logging user actions and state transitions. The Extensible Markup Language (XML)<sup>2</sup> is a World Wide Web Consortium (W3C)<sup>3</sup> standard for document markup that offers the possibility of cross-platform, long-term data storage and interchange. XML is more than a mark-up language: it is a meta-markup language, in the sense that it can define the tags and elements that are valid for a particular document or set of documents. For our purposes, it has the advantage that it is non-proprietary and can be examined with any text editor or open-source XML editor. Also, there are plenty of XML parsers available, written in various programming languages, so processing the logs and extracting relevant information is easy. Moreover, it allows a variety of access modes: (i) sequential access to each event in the log (via SAX<sup>4</sup>); (ii) random access to certain kinds of events, relevant for a

certain research hypothesis (via XPath)<sup>5</sup>; and (iii) complex visiting patterns (via DOM)<sup>6</sup>.

Closely related to XML are two other standards, Document Type Definitions (DTD) and the W3C XML Schema Language, which are used to describe the vocabulary and language of an XML document. A DTD or an XML Schema, (or simply “schema”, to refer to either) can be used by a human to understand or to impose the format of an XML document, or by a machine to validate the correctness of an XML document. Moreover, it can be used by an increasing number of tools (such as the open-source IDE NetBeans) to generate parsers for such XML documents.

While in principle both DTD and XML Schema can be used, there are some differences between the two. DTD’s are advantageous in that they are easier to write and interpret by a human, and since they have been around for longer, there are more tools to process them for XML validation and code generation (most commonly for Java or C++). The newer XML schemas allow more specificity in defining types of elements and attributes, but that comes at the cost of reduced readability and more human effort. It is envisaged that the two will co-exist in the future, and that a pragmatic choice can always be made according to the context as to which is more appropriate to use.

UML is ideally suited to support the design of systems, and XML for recording the activity logs. The problem is bridging the gap between the two. One approach fully supported by existing technology is to use the Java Architecture for Data Binding (JAXB)<sup>7</sup> specification to derive Java classes (or rather skeletons of Java classes, specifying name, attributes and method prototypes) from UML diagrams, and then XML DTDs or XML schemas from the Java classes. This approach has the advantage that the skeletons of the Java classes can be expanded with code either for implementing the user interface, or for processing the logs.

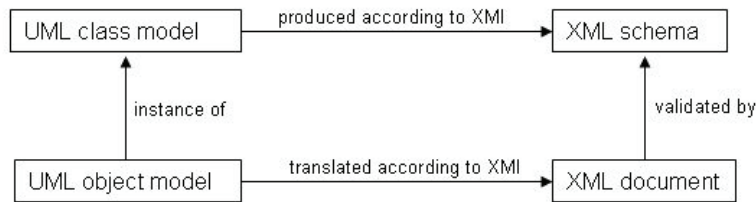
An alternative solution is to use the Object Management Group’s (OMG) XML Metadata

Interchange (XMI)<sup>8</sup> specification. Initially created as an open source specification that allowed modeling tools from different vendors (such as Rational Rose, TogetherJ) to export/import design models, XMI has grown to wider applicability by supporting the production of XML vocabularies and languages that enable the integration of many e-business applications (Carlson, 2001, 2006). XMI specifies a set of mapping rules between UML and XML in terms of elements, attributes and relationships. It must be noted that mapping UML to XMI is not an exact science, and different levels of strictness can be applied, and tradeoffs between a number of mapping decisions can be specified. For example, attributes specified in a UML class diagram can be converted to either XML elements or XML attributes. Carlson (2001) discusses at length such tradeoffs, as well as the use of XPath, XPointer<sup>9</sup> and XLink<sup>10</sup> in implementing more complex relationships from UML diagrams, such as inheritance, association or composition.

Figure 2 captures this approach. UML class diagrams provide the blueprints for UML object diagrams, and XML schemas provide the template for XML documents. XMI specifies the translation of UML class models into XML schemas and of UML object models into XML documents. The obvious and direct application of this approach to logging the interaction appears to be the following: (i) derive UML class diagrams from state diagrams (this is trivial, as the states at different levels of granularity correspond to classes); (ii) use XMI to derive XML schemas from the UML class diagrams; and (iii) capture in XML logs the successive states of the user interfaces, after each event or user action. The approach that we actually propose is a variation of this and will be described later in this section, after we discuss various design decisions.

Finally, in order to avoid the learning curve imposed by the JAXB or XMI automatic alternatives, a “manual-intellectual” approach is feasible for relatively small projects. We followed such

Figure 2. Mapping UML models to XML schemas and documents



a procedure on the case study described in the next section, deriving the design of the logger and of the log parser from the state diagram of the interaction.

In summary, the expected gains of our vision are:

- Generating user interfaces that accurately implement a certain interaction model.
- Client-side logs that accurately capture user interactions, such as a search session.
- Support for building user models that capture usability problems as well as user preferences. This in turn can contribute to building better interfaces, and to building personalized systems that adapt to the user’s needs and preferences.

An apparent disadvantage of this approach is the limitation of what events are logged. One may argue that, once a first log analysis is conducted, the set of research hypotheses/questions may be extended, so data initially viewed as irrelevant may become important. First of all, let us clarify that it is not the research hypotheses that determine if an event is semantic or not, but the interaction model: all the events to which the interaction is designed to respond are logged, whether they are considered relevant to the research questions or not. Secondly, the designers of the experiment have the option of logging additional, non-semantic events for the sake of completeness, and such data can prove useful: e.g. the amount of mouse moving may indicate the frustration of the user;

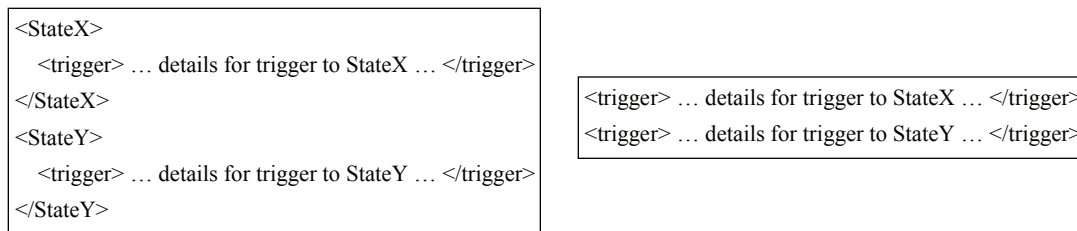
the number of invalid actions attempted by the user may reveal problems with the usability of the user interface, etc. It is up to the designers to reach a balance between logging “everything” and potentially wasting time and resources, and logging only events and actions that have an effect.

### Explicit vs. Implicit Logging of States

An essential design decision is whether the logs should capture the states explicitly, or whether logging just the events or actions that trigger state transitions is sufficient, or perhaps even preferred. Figure 3 depicts the conceptual difference between the two cases; of course, the details about a certain trigger will be described in proper XML.

At first sight, explicitly logging the system states appears natural, so that someone examining the logs can clearly see what happened while the system was in a certain state, and when a state transition occurred. However, logs are usually so large and contain so many details, that the researcher is unlikely to gain much knowledge from examining them visually. Rather, the logs should be processed automatically and the information pertinent to a certain research question should be summarized, and possibly visualized, so that it can be interpreted by the researcher. Therefore, *explicitly* capturing the states in the logs is not necessary, as long as they can be re-created at analysis time, based on the events and actions captured in the logs, and on the model captured by the state diagrams. As a result, capturing just

Figure 3. Explicit vs. Implicit capturing of states in interaction logs



the triggers to states may be sufficient, as long as the state diagrams capture the determinism of the transitions.

One can argue that the data captured in the logs, such as the buttons clicked, the text typed or the menu items selected by the user, are all attributes of user events rather than attributes of the states. Therefore, logging the events, with their attributes, makes it possible to log all the data relevant to the interaction. Let us now consider some more complex situations and design decisions.

In the case of hierarchic states, if we explicitly log states, then a further decision is needed, as depicted in Figure 4. If StateX1 and StateX2 are substates of StateX, then a decision is needed as to whether to explicitly capture the state hierarchy; in practice, one needs to decide whether to log all the levels of the state hierarchy, or only the leaf nodes. For a visual inspection, the explicit choice appears better: the log makes it obvious that, when in StateX1, the system is also in StateX. Again, for the automatic processing of the logs, that is not an advantage; on the contrary, a more complex DTD, and therefore parser, is a disadvantage. Note that, if only the triggers are logged, then the parsing of the log is even simpler, and the knowledge about the state hierarchy is only relevant in the data analysis stage.

Another special situation is the transition to the same state; for example, while the user types

the words of a query, the system stays in the same state until the query is submitted.

Figure 5 describes this situation. If the system stays in the same state, it does not make sense to capture multiple instances of the same StateX in the logs; the states can be “collapsed”. The problem that appears is that, in this case, a state will appear to have multiple triggers, which makes the DTD more complicated. Again, logging just the triggers removes this problem.

One more situation that we are considering is that of complex systems with orthogonal states, e.g. the state diagram captures, in parallel “swim-lanes”, the actions of the user scrolling a document, and the actions of a graphical module rendering a visual display of the search results. The problem is that state transitions in different swim-lanes are independent, so a situation like that depicted in Figure 6 can occur (where StateX and StateY are in one swim lane, and StateA and StateB are in another). It is apparent that the resulting log is not well-formed XML, so parsing it is not possible with regular XML parsers. On the other hand, if only the triggers are captured, this problem is removed.

Overall, there seems to be overwhelming evidence in support of logging just the events and actions that trigger state transitions, rather than explicitly capturing the system states in the logs, and to re-create the states when the logs are parsed and analyzed.

Figure 4. Explicit vs. Implicit capturing of state hierarchy

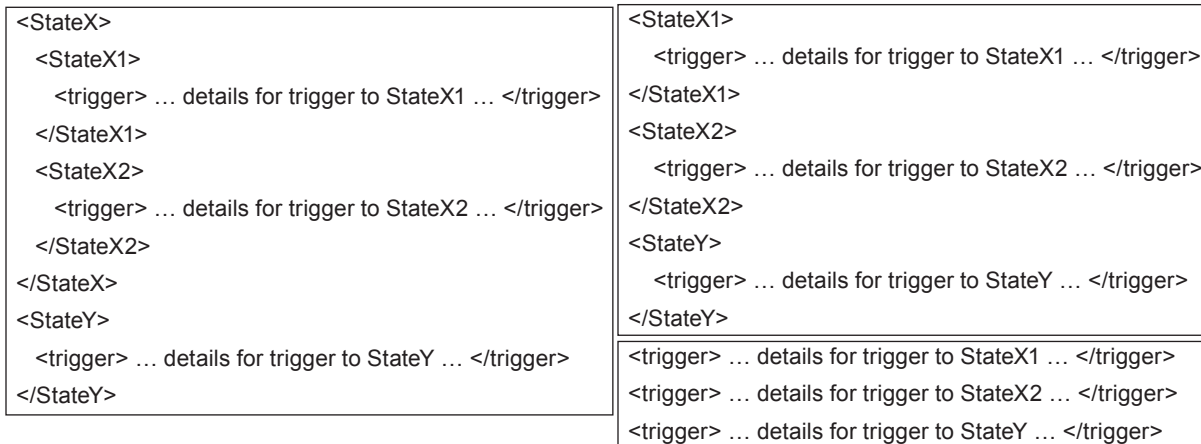
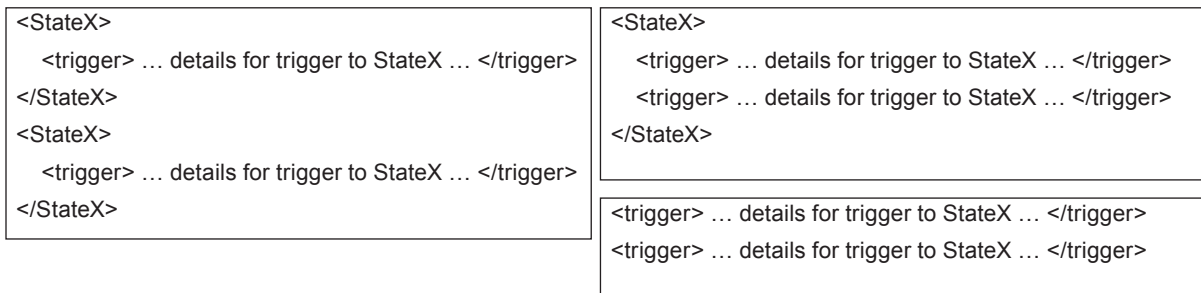


Figure 5. Collapsing identical states



## Design Patterns for System Design and Log Analysis

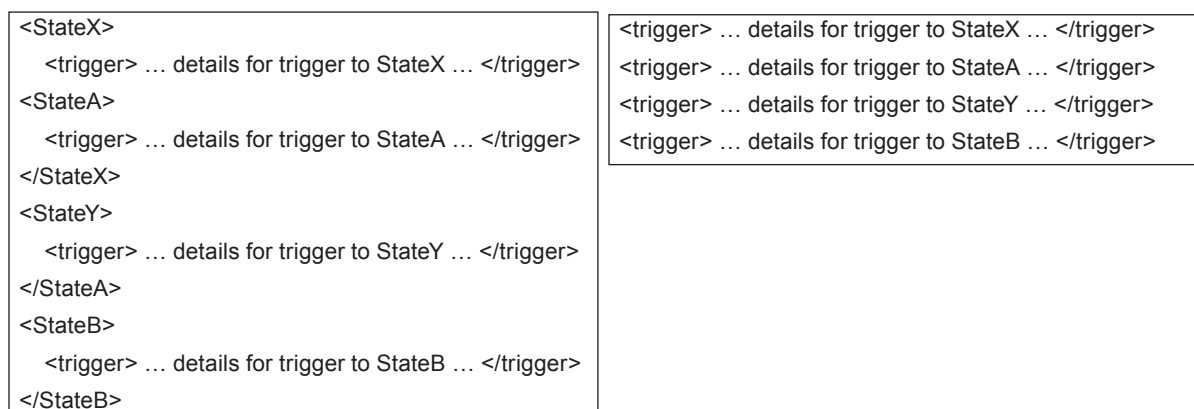
The **State** design pattern (Gamma et al., 1995) is a natural choice for a system whose behavior depends on its state, and may change its behavior during execution, based on a state change. It localizes state-specific behavior in different classes, one for each state, avoiding the need for complex *if* or *switch* statements in the code implementing behavior. If the statechart model of the system is available, coding it is relatively simple, as the states, the events and the state transitions are already identified.

One essential decision is whether to use just one set of classes, corresponding to the states of the system, both for implementing the functionality of the system and for analyzing the logs, or to

use two sets of classes with the same names, in different packages, one for the system functionality and one for log analysis. Using a unique set of classes can have the advantage that some of the analysis, and the computation of summaries describing the interaction can be done during the interaction, rather than as a separate, offline procedure. However, we prefer the advantage of simplicity and clarity offered by two sets of classes with distinct purposes.

Another essential decision is how the state objects are created and stored when analyzing the logs. One solution is to apply the **Singleton** design pattern (Gamma et al., 1995), so that a unique (singleton) object is created for each state. This is typically the preferred solution when an application has a small number of states and a large number of state transitions: state objects

Figure 6. Capturing transitions between orthogonal states



can be reused rather than new objects created, which makes the application more efficient. Also, a state object can accumulate information over multiple occurrences of the same conceptual state. While in most situations using the singletons is the better solution, for our specific application that solution is not appropriate, due to the level of detail that we want to capture. For example, for an IR application, the researchers may want to analyze not only how many queries were edited and submitted overall, but also how much time was spent formulating each of them, if words were typed or pasted into the query box, the number of corrections that were made on the query, etc. For capturing specific information for each instance of a state, the better solution is to create a new state object every time a state transition occurs.

Finally, it is common for XML parsers generated automatically based on DTD (such as the one produced by NetBeans<sup>11</sup>) to implement the **Visitor** software design pattern. This allows flexibility in specifying which elements of the log tree should be visited and in what order, in order to collect, process and summarize information.

## The Procedure

The previous sub-sections have covered the vision of our approach, as well as a discussion of

alternatives, with a number of preferences stated. In this sub-section we revisit the conceptual model of our approach, shown in Figure 1, and comment on the implementation of the specific steps.

*Building the conceptual model* of the interaction is the crucial step of this approach, as everything else depends upon it. The statechart captures the state, the events and the state transitions. Note that the states and the events allow the specification of attributes (e.g. a QuerySubmission event, for example, could specify the text of the query, the targeted search engine, the number of desired hits, etc). While the diagram may become overcrowded if too many details are displayed explicitly, these attributes need to be specified in order to support the next steps.

The list of possible events, together with their attributes, are extracted from the state diagram and used for two purposes: (i) for *specifying a logging module*, which has a function associated to each event so that, when one of these functions is called, it logs the appropriate event and its details; (ii) for *specifying the DTD or XML schema* of the interaction. These two sub-steps should be done in sync, as the DTD specifies the format of the log files written by the logging module. They can be performed either manually, for small systems, or automatically, based on XMI or JAXB technology. An *XML parser specific for the modeled*

interaction, and therefore for the log file, can be derived immediately from the DTD model; in fact, there are a number of open-source tools that perform the code generation automatically (such as NetBeans).

The names of the states, extracted from the state diagram, constitute the names of the classes for *building the log analysis module*. It uses the log parser to identify events and to derive state transitions, and it creates instances of the subsequent states, virtually re-creating the interaction. These state objects, which contain useful data read from logs as attributes of events (and possibly of states) can be stored in a list (or another data structure) which can be subsequently filtered according to the research questions investigated, and the information stored by them can be summarized and analyzed.

## CASE STUDY: MEDIATED INFORMATION RETRIEVAL

In order to help the reader more easily understand the proposed methodology, we are going to describe its application on our **MIR** (Mediated Information Retrieval) project. The focus of this chapter is the experimental methodology that we designed and employed, rather than the actual

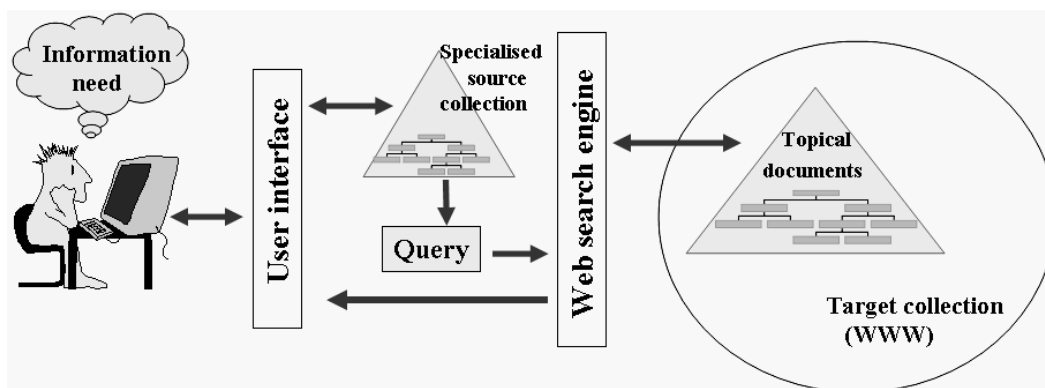
research questions and the experimental results of that project. Therefore, the description of the project will be limited to the minimum necessary. A more complete description of the project and an analysis of the results appear elsewhere (Lee, 2006).

## The Mediated Retrieval Model

We proposed the concept of **mediated information retrieval** (or access) in previous work (Muresan & Harper, 2001, 2004; Muresan, 2002) as a way to address the problem of exploratory searches, when the searcher may be unfamiliar with a problem domain, uncertain of what information may be useful for solving a particular task, or unsure as to what query terms would be helpful in retrieving relevant information. The idea is to emulate the function of the librarian or intermediary searcher, who interacts with the information seeker, elicits more information and helps the searcher refine, clarify and formulate her information need.

Our reification of the mediation interaction model is based on so-called source collections, specialized collections of abstracts or documents that cover the searcher's problem domain. These collections, which emulate the librarian's knowledge of a certain domain, are either manually

Figure 7. The interaction model in mediated information retrieval



structured (based on some ontology that describes that domain) or are automatically clustered in order to reveal the concepts and structure of the domain, in order to inform and educate the searcher.

The interaction model is captured in Figure 7. In the first stage the searcher interacts with the *source collection* so that (i) she becomes more familiar with the terminology, concepts and structure of the problem domain, and better able to convey her information need; and (ii) the system monitors the user's interaction and her selection of documents, and learns the type of documents that she is interested in. Following the mediation stage, the search target moves to the Web or any other *target collection* where the user hopes to find new information to satisfy her need and complete some task. At this point the system is able to support the searcher by suggesting query terms; also, the user is expected to be more familiar with the problem domain, and able to formulate better queries than before the mediation.

### **The MIR Project**

In previous work we demonstrated the *potential* effectiveness of mediation through pilot studies and user simulations. In the MIR project, we run formal user studies to verify if mediation can indeed improve retrieval effectiveness. Moreover, we are interested in observing patterns of interaction, which could help us design better interfaces.

In the first stage of the project, which we have completed, the human searcher did not get any support from the system in formulating their queries to be submitted to the Web search engine. The mediation consisted in the user exploring the source collection in order to better understand the topic investigated and to enrich her vocabulary. In a future stage of our investigation, the system will suggest a "mediated query" and the searcher will be able to edit it before submitting it to the search engine.

From among the candidate source collections that we were able to obtain, we selected the New Jersey Environmental Digital Library (NJEDL) collection because: (i) with approximately 1,300 documents, it is relatively small so, once clustered, it can be searched and browsed relatively easily in a reasonable amount of time; (ii) it provides a good coverage of environmental issues; (iii) we were able to generate a number of training and test topics for the experiment. A good test topic is one for which there are relevant documents in the target collection (the Web), but finding them requires good queries.

Our experimental design was inspired by work in Interactive TREC<sup>12</sup> (Dumais & Belkin, 2005). We compared a baseline system, with no mediation, against the experimental system, based on mediation. Each of the 16 subjects was randomly assigned a condition that specified the systems to be used and the topics to be investigated, two with the first system and another two with the second system. The systems and the queries were rotated in a latin square design, in order to avoid any order effect. Figures 8 and 9 depict the user interfaces for the baseline and experimental systems.

An effort was made to make the systems identical, with the exception of the mediation functionality, so that any differences in results can be attributed to mediation. Each interface has a *Task control* panel where the task is displayed, and where the subject can formulate their information needs and submit them as queries. Search results from the target collection are shown in the "WEB" tab of the *Search results* panel. When a document is selected, it is displayed in the Web browser. The subject can use the right mouse button to save a document from the hit list; the document snippet will be shown in the *Saved documents* panel. When a document is saved, the searcher is asked to specify the aspects of the topic that the document deals with. Retrieval effectiveness is measured both by recall (the number of relevant Web documents saved by the searcher, relative to the total number of relevant documents known

Figure 8. The baseline MIR interface (no mediation)

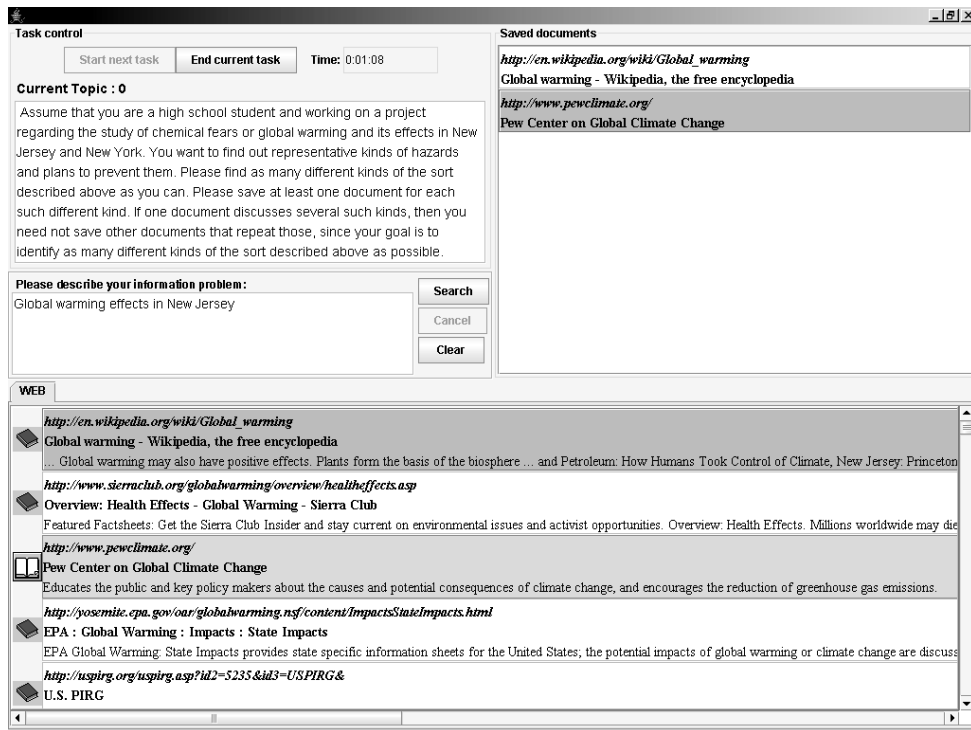
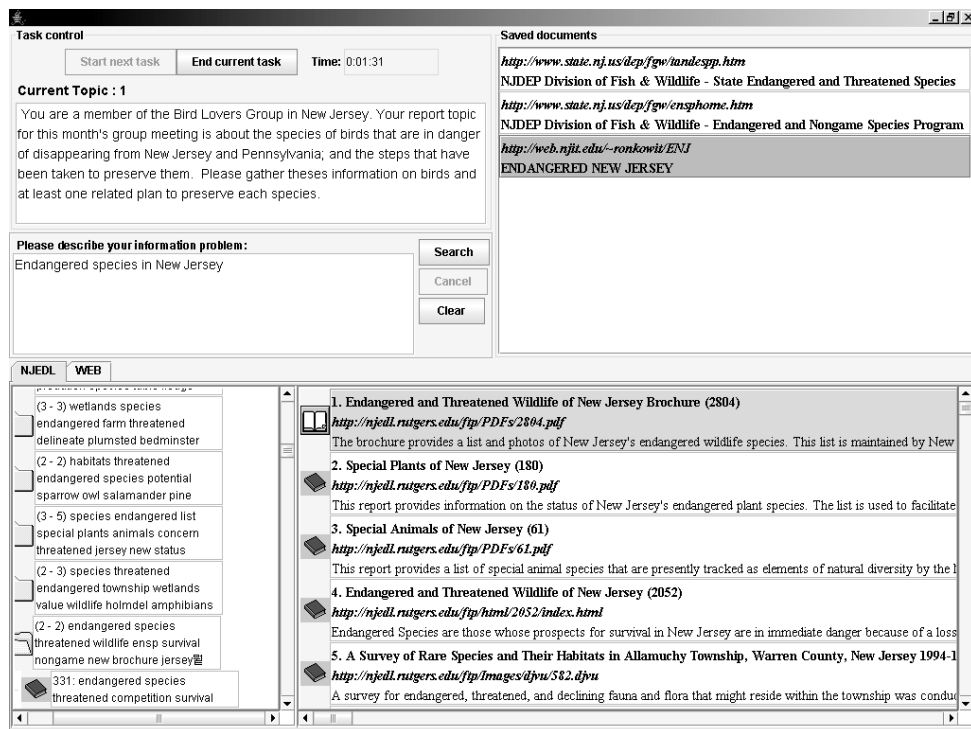


Figure 9. The experimental system (with mediation)



by the researchers to be relevant) and aspectual recall (the number of distinct topical aspects identified by the searcher, relative to the total number of aspects found by the researchers). In order to identify relevant documents and aspects, we employed a pooling procedure similar to what has become a standard procedure in such IR experiments (Voorhees & Harman, 2005): we judged the relevance of the documents saved by all the subjects, and of the candidate documents identified by ourselves when exploring candidate test topics.

The experimental system has an additional tab, “NJEDL”, which supports the exploration of the full source collection. The source collection is clustered, and the subjects can use a combination of searching and browsing for its exploration. On the one hand, searching can provide starting points for browsing: when a document snippet in the result list is selected, not only is the full document shown in the Web browser, but the cluster hierarchy is expanded and scrolled automatically, so that the user can investigate the neighborhood of the selected document. On the other hand, browsing the clusters and documents of the source collection is expected to reveal serendipitous relevant information and to suggest new query terms.

At the beginning of the experiment subjects are given a tutorial, and the experimental system is demonstrated to subjects through the prescribed mediation interaction: after seeing the current topic, the searcher explores the source collection available in the NJEDL tab, in order to understand the topic and its context better, and to grasp its terminology. Then, the interaction moves to the WWW tab, where a query can be submitted to the Web search engine, like in the baseline system. In the experiment the user is not forced to adopt this interaction model: if the topic is familiar and formulating a good query is perceived as easy, she may choose to go straight to the WWW tab and search the Web. However, the source collection is always available, and the searcher can always

explore it; this may happen if the Web search is perceived as unsuccessful, and more ideas for query terms are sought.

While the focus here is on methodology rather than on the experimental results, let us briefly describe some types of research questions and hypotheses investigated in the MIR project, with the purpose of highlighting the type of data needed to be captured in the logs and analyzed:

- **RH:** “During the mediation stage (exploration of the source collection) users are able to find documents relevant to their problem.”

In order to answer this research hypothesis, the logs need to capture the identifiers of the documents opened by the user, so that their relevance can be judged by the researchers. Additionally, capturing time-stamps in the logs allows the investigation of supplementary research questions: “Do users spend more time reading relevant documents than non-relevant documents?” Moreover, as the browsing of the hierarchically clustered source collection is captured, we can look at common behavior (e.g. depth-first vs. breadth-first exploration) and can compare searching with browsing in terms of efficiency (e.g. effort measured as amount of time spent, number of documents opened, etc.) and effectiveness (successful navigation to relevant documents).

- **RH:** “The mediation stage helps the user formulate better queries and thus achieve better retrieval effectiveness.”

While the research design is responsible for separating searchers that use mediation from searchers that do not use mediation, the logs have to capture the actual queries submitted to the search engine, the hits returned, the snippets clicked by the searchers and the saved documents (the relevance of the saved documents is then evaluated by the researchers for a more complete evaluation of search effectiveness). This data

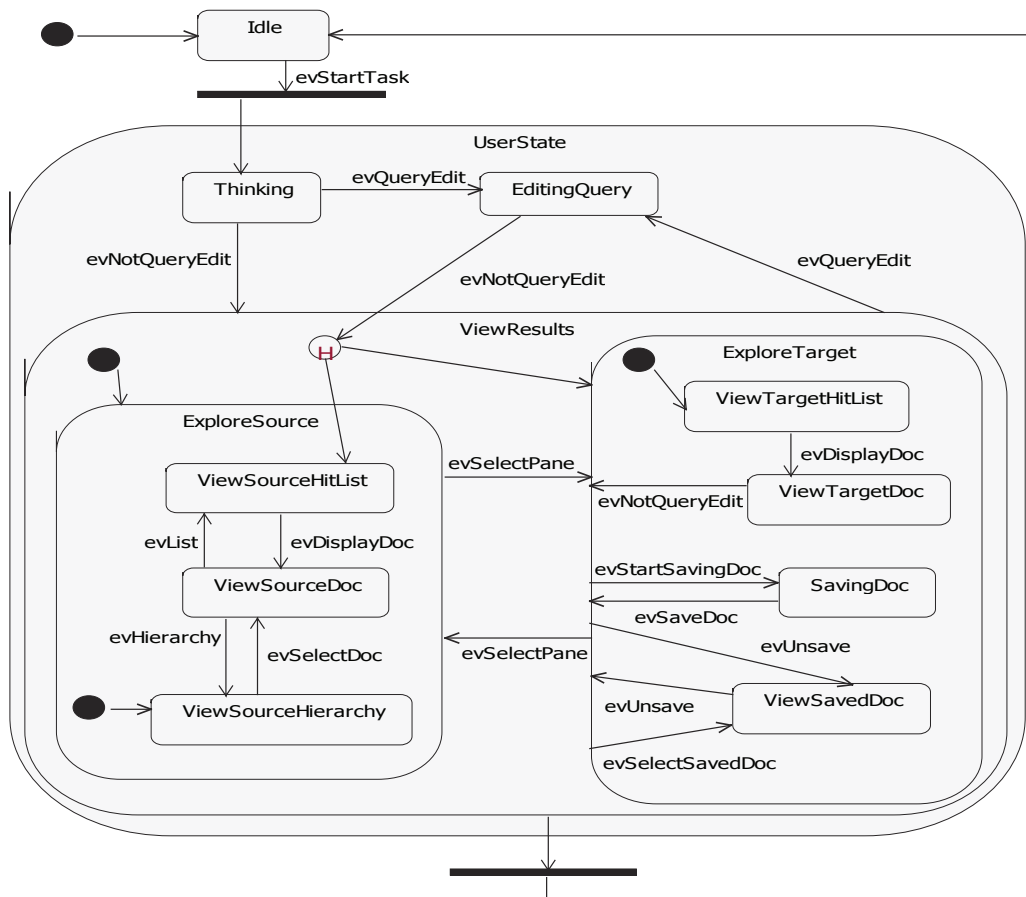
can support additional research investigation, for example looking for correlations between query length, more extended vocabulary, query clarity and search effectiveness. Moreover, such data can also be correlated with data from user questionnaires, for example to investigate the effect of search experience or familiarity with a certain topic on time to complete a search, on the number and quality of query submitted, and on task success.

It is apparent that, even if a research experiment is initiated with a small number of research hypotheses, the logging of all semantic events can support the exploration of many additional research hypotheses and questions.

### State-Based Design of Interaction and Logging in MIR

To exemplify our procedure on the case study, Figure 10 shows the state diagram that depicts the system states during the MIR interaction. We believe that such a diagram is fairly easy to understand or design even for a researcher not trained in software engineering. In the Idle state between search sessions, the user may perform related activities such as filling in questionnaires required by the experiment. When the session starts, triggered by an `evStartTask` event, the system displays the current search task and enters the Thinking state, in which the subject reads the task description and thinks of appropriate queries

Figure 10. State diagram for the MIR project



(or alternative actions) to be used. If the user starts typing a query (marked by an `evQueryEdit` event), there is a transition into the `EditingQuery` state. On the other hand, in the case of using the mediation system, the user has the choice of starting to browse the source collection first (marked by expanding the cluster hierarchy or selecting a cluster, i.e. an event different from query editing). While the user is editing the query (i.e. typing or using copy-and-paste), the system stays in the `EditingQuery` state. When the "Search" button is pressed, the history (H) pseudo-state will indicate which of the collections was being explored prior to editing the query; thus the query is submitted to the appropriate collection, the search results are displayed in the *Search results* panel of the appropriate tab, and the system enters the `ViewResult` state. This is a "superstate", which has a number of "substates": the `ExploreSource` state corresponds to the exploration of the source collection (NJEDL), while the `ExploreTarget` state corresponds to the exploration of the target collection (the Web). The searcher may choose between the two collections (and therefore between the two sub-states) by selecting one of the tabs, or the sub-state may be set automatically by the history mechanism.

Not depicted in this diagram are the orthogonal (or parallel) states, corresponding to different components of the system such as the *Task control* panel and the *Search results* panel. These states can also be modeled at different levels of granularity in order to support the design and implementation of the system. For example, the Query panel can be in a `Valid` state, when a query can be submitted, or an `Invalid` state, when there is no query, or a query has just been submitted and the search results are expected from the search engine. These system states, parallel to the user states (and hence the two synchronizations bars in the diagram), are essential in designing the functionality of the system. However, they are omitted here for space reasons.

A couple of clarifications are in order:

- Although think-aloud protocols can help, it is not possible to have a perfect image of the searcher's cognitive process. Therefore, what is represented in the diagrams is not the user's cognitive states, but system states. However, the user's actions and the sequence of system states do reflect the decisions taken by the user, and can therefore be used in modeling user behavior.
- The labels assigned to system states reflect the researchers' understanding of the interaction, and specify their understanding of what is going on. Similar to variable names in programming, these labels should convey the semantics of the interaction; however, a perfectly accurate depiction of the user's cognitive process is not necessary. In the example, the label "Thinking" was assigned to the state in which the searcher was instructed to read the assigned task and to think of a search query to submit. There is no guarantee that the user follows the instructions and is indeed thinking; conversely, it does not mean that this is the only state in which the user has to think. The label simply attempts to depict the researcher's best description of what is going on.

From the state diagram, we extract the names and attributes of the events, in order to specify the log format in a DTD and implement the functions of the logging module. Note that, especially for the manual version of the procedure, some adjustments of the names are acceptable, e.g. "evSelectPane" becomes "SelectPane" as XML element in the log file, and as class in the code.

Figure 11 presents a sample of the DTD that describes the MIR interaction, and Figure 12 depicts a sample extracted from a MIR log. It is apparent that the attributes of the events, such as the editing or submission of a query, are captured in the logs and can be used to address the

Figure 11. DTD sample for MIR Log

```

<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT log (record)*>
<!ELEMENT record (date, millis, message)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT millis (#PCDATA)>
<!ELEMENT message (StartSession|EndSession|
EditQuery|SubmitQuery|SearchResults|SelectPane|
DisplayDoc|StartSaveDoc|SaveDoc|UnsaveDoc|
TouchCluster|ShowMessage)>
<!ELEMENT StartSession EMPTY>
<!ATTLIST StartSession
  task CDATA #IMPLIED
  >
<!ELEMENT EndSession EMPTY>
<!ELEMENT EditQuery EMPTY>
<!ATTLIST EditQuery
  query CDATA #IMPLIED
  querySize CDATA #IMPLIED
  text CDATA #IMPLIED
  offset CDATA #IMPLIED
  count CDATA #IMPLIED
  action CDATA #IMPLIED
  >

```

research hypotheses. Moreover, based on the state diagram, the states can be re-created while the logs are parsed and the events interpreted. This supports research in analyzing state transitions and modeling user behavior.

Apart from being the source of the DTD / XML schema, the interaction state diagram also provides the state names and attributes that support the automatic or manual generation of the class skeletons (e.g. in Java) for the module running the application and for the log analyzer. The two sets of classes have the same names but are in different packages and have different purposes: (i) an application module, tightly coupled with the logging module which writes the XML

log files; and (ii) the log analysis module, tightly coupled with an XML parser that recognizes the log elements specified by the DTD file.

Note that the code generated is just a skeleton, and the research team needs to fill in the class methods with actual code that writes or reads data into or from a file. However, such code is trivial after the design of classes and methods has been generated. For writing, if Java is the implementation language, then the standard logging package<sup>13</sup> makes it extremely simple to output logs in XML: a Logger object uses XML by default to write logs into a file, adds a time-stamp automatically, and displays as content of a “message” element the text passed to it for logging (see Figure 12).

Even if not used directly in generating the XML schema of the interaction and subsequently the code for log recording and log parsing, the original state diagram describing the states of the system (Figure 10) can be used for automatically generating code for modeling state transitions and, for example, building a Markov model of user behavior (Jurafsky, 2000). Note that the classes depicted in Figure 13, corresponding to the states of the interaction, are actual classes (in an object-oriented programming language such as Java) of the log analyzer, and of the software for state modeling. State objects can capture events that took place for the duration of that state, and additional data structures can capture the sequence of states in chronological order.

## Discussion and Evaluation

The effectiveness of a methodological framework is best demonstrated by its flexibility as well as its ability to solve the problem it was designed for. In this section we highlight its power based on evidence from our experiments, as exemplified by the kind of data analysis and research hypotheses investigation that it supports. A more comprehensive analysis of the MIR logs and of the research hypotheses investigated is available elsewhere.<sup>14</sup>

Figure 12. Sample from a MIR Log

```

<?xml version="1.0" encoding="windows-1252" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
  <record>
    <date>2005-05-31T16:08:19</date>
    <millis>1117570099901</millis>
    <message><StartSession task='2' /></message>
  </record>
  <record>
    <date>2005-05-31T16:08:21</date>
    <millis>1117570101833</millis>
    <message><SelectPane title='NJEDL' /></message>
  </record>
  <record>
    <date>2005-05-31T16:08:30</date>
    <millis>1117570110185</millis>
    <message><TouchCluster op='select'><Cluster id='1413' level='2' parentId='1415' /></TouchCluster></message>
  </record>
  <record>
    <date>2005-05-31T16:08:31</date>
    <millis>1117570111026</millis>
    <message><TouchCluster op='collapse'><Cluster id='1413' level='2' parentId='1415' /></TouchCluster></message>
  </record>
  <record>
    <date>2005-05-31T16:08:42</date>
    <millis>1117570122001</millis>
    <message><EditQuery action='add' count='1' offset='0' text='n' querySize='1' query = 'n' /></message>
  </record>

```

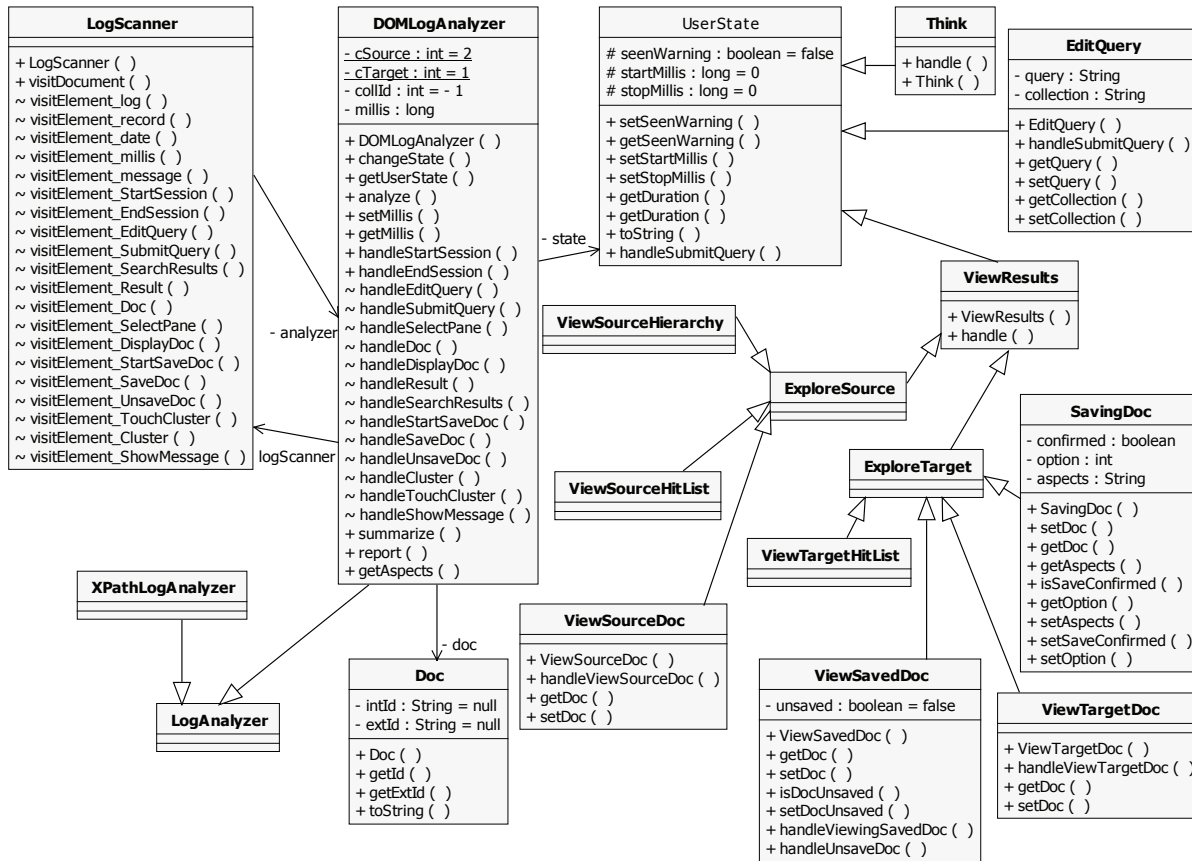
First of all, let us distinguish between two fundamentally different approaches to analyzing the logs. The “atemporal” approach can be applied when the interest is in processing information about a certain kind of event, with no regard to state transitions, or to the order of the states in the logs. Examples of such situations are: getting the list of all the documents viewed or saved by the user, getting the list of all queries submitted to the search engine, etc. In such situations, probably the most efficient solution is to implement an XPath-LogAnalyzer, which uses XPath to visit only the XML nodes in the log tree that are of interest (for example, the SaveDoc events can be visited by specifying “/log/record/message/SaveDoc” as the path to the nodes of interest).

If the time factor is essential in answering a certain research hypothesis or in getting a certain kind of information, then a DOMLogAnalyzer<sup>15</sup> can be employed instead, which will traverse and

process the nodes of the log tree (in XML format) in the desired (usually chronological) order. For more flexibility, the task of actually traversing the log tree can be delegated to a separate class (LogScanner in Figure 13), so that the function of traversing the log is decoupled from the function of taking action for each node. An even more flexible solution is to apply the **Strategy** design pattern (Gamma et al., 1995), by making LogScanner an abstract class and having different visiting strategies implemented by its concrete subclasses.

Let us now have a look at a sample of results obtained by applying this methodology in MIR. Box 1 shows a sample report obtained by listing the class names for each state object inferred from a log file, together with the duration of that state (in seconds). Subsequent processing could consist, for example, in building a transition matrix by compiling the states from all the log files in order to (i) observe patterns of behavior and be

Figure 13. State classes used in the MIR log analyzer



able to predict the next state at a given point; or to (ii) find the most common states and the most common transitions, and optimize the use of the interface for those situations; or to (iii) detect and correct usability problems (e.g. detect transitions that never happen, because some functions are not sufficiently visible in the user interface).

Note that the modeling and analysis of the state transitions can be done at various levels of granularity. For example, the sequence (EditQuery 9, ViewTargetHitList 15, ViewTargetDoc 78, SavingDoc 16, ViewTargetHitList 6, EditQuery 5) could be viewed as (EditQuery 9, ExploreTarget 115, EditQuery 5) if the details of exploring the target are considered irrelevant.

An essential piece of analysis for the MIR project regards the effectiveness of retrieval; we

are interested to see whether mediation improves effectiveness. The computation of recall and aspectual recall requires relevance judgments. Even without those, a simple extraction and comparison of data from the logs can give us an idea of how well our expectations were met. Note that in previous experiments, run as part of Interactive TREC, a high correlation was observed between recall and the raw number of documents saved by the subjects (Belkin et al, 2001). Moreover, in the current experiment, the subjects were asked to support their decision to save each document by stating the aspects addressed by the document; therefore, one can expect most saved documents to be relevant, and a higher than usual correlation between recall and number of documents saved. Obtaining from the logs the number of saved

Box 1.

Think	4
EditQuery	9
ViewTargetHitList	15
ViewTargetDoc	78
SavingDoc	16
ViewTargetHitList	6
ViewTargetDoc	31
ViewTargetDoc	9
ViewTargetDoc	35
SavingDoc	11
ViewTargetHitList	3
ViewTargetDoc	173
SavingDoc	16
ViewTargetHitList	14
EditQuery	7
ViewTargetHitList	4
ViewTargetDoc	17
ViewTargetDoc	59
ViewTargetDoc	51
ViewTargetDoc	39
EditQuery	13
ViewTargetHitList	25
ViewTargetDoc	38
SavingDoc	15
...	

documents and the number of queries submitted is trivial.

For the sake of exemplifying some of the statistical analysis supported by our approach, let us report that a set of ANOVA tests shows that most differences between the non-mediated and the mediated conditions are not statistically significant. Surprisingly, slightly more documents were saved on average in the non-mediated condition ( $m = 3.94$ ,  $sd = 1.76$ ) than in the mediated condition ( $m = 3.13$ ,  $sd = 1.62$ ) despite visibly more effort in the mediation condition. While spending roughly the same total amount of time in the overall search session ( $m = 1166.16$ ,  $sd = 185.98$  compared to  $m = 1190.91$ ,  $sd = 168.91$ )<sup>16</sup>, the mediation subjects submitted significantly more

queries ( $m = 8.69$ ,  $sd = 4.90$  compared to  $m = 5.69$ ,  $sd = 3.22$ ;  $F = 8.377$ ,  $p = 0.005$ ). In the mediation condition, subjects submitted an average of 2.22 queries to the source collection, and an average of 6.47 queries to the target collection.

Unfortunately, this is a bad result for the mediation hypothesis. Possible explanations are that (i) the subject could not find relevant documents in the source collection; or (ii) the subjects did not have time to read the identified source documents in order to improve their understanding of the topic or to enhance their terminological vocabulary in order to submit better queries. In order to answer these questions, our next steps are to examine the source documents viewed by the users (captured in the interaction logs) and to judge their relevance to the test topics. This will allow us to check if the statistical language models of the queries submitted following mediation show any significant difference. The power and flexibility of our methodology is obvious – the accurate logging of all semantic events, even those not related to the research hypotheses, affords the extension of the original hypotheses, and extra analysis not planned at the outset.

## CONTRIBUTIONS AND FUTURE WORK

The proposed methodology is a novel and significant contribution to experimental research in interactive systems, with applications in areas such as Human Computer Interaction or Information Seeking and Retrieval. It is particularly suitable for studying exploratory searching, where the research questions are usually related to understanding patterns of behavior in different stages of the interaction. This approach has been successfully applied in Interactive TREC work and in our Mediated Information Retrieval project.

One interesting issue to consider is the generality of our approach. What kind of systems can it be applied to? Is it not rather limiting to restrict

logging to semantic events? Is it possible to log everything that happens during the interaction? We will start addressing these issues by re-iterating the purpose of our work.

Our goal is to integrate the design of the user-system interaction (and implicitly of the user interface) with the design of the logger and of the log analyzer. This means the following:

- The user should be limited to performing actions judged by the system designer to be valid in a certain context; e.g. the user cannot submit an empty query, or save a document repeatedly, etc; this improves the usability of the system and, from a software perspective, it reduces the potential for bugs. It means that only valid actions need to be recorded in the logs. During testing, assertions in the log parsing software can help make sure that the XML documents perfectly match the interaction specification (the XML schema), and that all the recorded events and state transitions are valid.
- It is debatable whether user attempts to perform invalid actions (e.g. the attempt to re-submit a query while the search is active), or events ignored by the system (erratic moves of the mouse) should be logged. On the one hand, only lack of imagination as to what should be logged can limit the system designer, so the danger of recording too much irrelevant data is real (e.g. if a dedicated thread records the state of the system second by second). On the other hand, recording data that is judged irrelevant at the onset may be valuable if the relevance judgment is reconsidered, for example if new research hypotheses are proposed following the initial analysis of an experiment's logs. While recommending a balance between the extremes, we have addressed this issue by including a special action called *ShowMessage* (see the DTD in Figure 11), which records "other" events, i.e. events not included among the valid semantic events in the interaction design. In our own research experiment, we used this capability to record when the task panel's timer alerts the subject that just two minutes are left for completing the task; this is an event that does not affect the state of the system and can be ignored by the user. However, recording that event allows us to determine if the reminder affected the user's subsequent search behavior.
- On a related note, the designers need to decide the granularity of the events to be logged. For example, should the system log each keystroke used to edit a query, or just the final query? Our recommendation is to let the research hypotheses under investigation inform the decision. For example, we were interested in the effect of topic familiarity on the searcher's query formulation behavior (copying and pasting vs. typing, number of corrections made, etc), so we logged all keystrokes. On the other hand, we only logged the mouse events that had semantic interpretation (selection, cluster expansion, etc).
- Similarly, the system designer needs to decide whether orthogonal events (e.g. the search thread becoming active, or the Internet connection being lost, etc) are worth logging, at the expense of more design and implementation time. Our approach is applicable in two ways: (i) the state diagrams are built separately, and the logging is done in separate files; synchronization of logs, based on time-stamps may be required at analysis time; (ii) more complex state diagrams are used, with parallel swim-lanes, and all the events are logged into the same file; the disadvantage is the increased complexity of the software.

Our proposed approach is appropriate for client-side logging, especially when the research team design and implement both the user interface

software (which includes the conceptual interaction design) and the log analyzer. In this situation, the same class hierarchy, representing system states, can be used for implementing both the interaction with the user (keystrokes and mouse actions are interpreted in terms of semantic actions according to the state of the system) and the log analyzer (logged events are interpreted in order to re-create the system states). The proposed approach can be adapted in the following situations, with gradually increasing levels of difficulty:

- For adding logging and analysis functionality to existing code. The state diagram of the interaction needs to be reverse-engineered based on the code and on observing the functionality of the system. Although the benefits of an integrated design are lost, the logging of the events and analysis of the logs can work well.
- For analyzing existing logs produced by a different system. The success of our state-based approach depends on the quality of the user interface that generated the logs (whether or not it allows invalid events to take place and to be logged) and the amount of events logged (whether the sequence of events can unambiguously predict the sequence of system states).
- For server-side logging, our approach is only feasible if the logged information is sufficient to determine the client that generated each event, and if the states of the client can be predicted based on the logged events.

### **Related Work**

A clear distinction needs to be made between different stages of creating interactive systems when discussing and comparing approaches, methodologies or techniques, as these are different for (i) specifying the requirements of the system; (ii) designing the user interface; and (iii) designing and implementing the software.

The actual stage of designing the user interface (Tidwell, 2006), although essential for building usable and ultimately successful interfaces, is not one of the concerns of our work. We are interested in linking the system specification to the software design; therefore, we are only going to discuss work relevant to this activity.

Most often, the specification of an interactive system is in the designer's natural language, such as English, accompanied by a set of the sketches of the interface at different stages of the interaction. Unfortunately, natural-language specifications tend to be lengthy, vague and ambiguous, and therefore are often difficult to prove complete, consistent and correct (Shneiderman and Plaisant, 2004). Use cases use a graphical notation to describe user goals, but the emphasis is more on the user-system interaction than in the task itself (Sharp et al, 2007). Task analysis provides a more concise and systematic way to describe and analyze the underlying rationale and purpose of what people are doing: what they are trying to achieve, why they are trying to achieve it and how they are going about it. Task analysis produces models of the world and of the work or activities to be performed in it: it describes the entities in the world, at different levels of abstraction, and the relationship between them, either conceptual or communicative (Diaper & Stanton, 2004). Actually, "task analysis" is a rather generic term, an umbrella for a set of related methodologies such as Hierarchical Task Analysis (HTA), Goals, Operators, Methods and Selection rules (GOMS), Groupware Task Analysis (GTA), etc. Limbourg and Vanderdonckt (2004) provide a description of these, as well as a syntactic and semantic comparison.

The specifications above are generally at a high level of abstraction and task granularity. While useful in guiding the design of the system, they do not provide sufficient support for automatic processing in order to prove completeness or correctness of a system, or for code generation. A possible exception is Paterno's (2001, 2004) work

on graphical representation of task specification. He proposes the use of ConcurTaskTrees (CTT) and discusses a variety of ways to integrate task models which describe the activities that should be performed in order to reach users' goals; he employs UML diagrams, created for supporting object-oriented software design, but focused on the internal parts of the software system. Possible approaches discussed are: (i) to represent CCT models with standard UML notation, e.g. with class diagrams; (ii) to develop automatic converters between UML and task models; (iii) to extend UML by building a new type of diagram. Paterno favors the latter approach, proposing a notation for tasks similar to the existing UML activity diagrams, but that also captures hierarchic relationships between tasks. These could be used together with other UML diagrams such as use cases, which define pieces of coherent user behavior without revealing the details of the interactions with the system, and sequence diagrams, which reveal details of the interactions for a certain task or sub-task.

Paterno's work is related to ours in the sense that he also tries to bridge the gap between different levels of abstraction, moving from user tasks towards software implementation. Apart from the application of our methodology being rather different, a distinction is that we are looking at a more detailed level of the interaction, which connects keystrokes and mouse events to semantic actions, in the context of solving a certain task.

Shneiderman and Plaisant (2004) also discuss more specific and formal approaches such as grammars, transition diagrams or statecharts, which provide a more fine-grained view of the human-system interaction and provide support for automatic processing and a connection to software design. Winckler and Palangue (2003) propose a formal description technique based on statecharts, dedicated to modeling navigation in Web application. That work is indeed related to ours, but they focus and limit their attention to

modeling the interaction, with no interest to log it and further analyze it.

More closely related to our goal and approach is Trætteberg's (2003) work on DiaMODL, a dialog modeling hybrid language that combines a dataflow-oriented notation with statecharts that focus on behavior. This work is complementary to ours: rather than proposing a new notation or language, our intent is to use and integrate existing notations and languages in order to combine the advantages that they offer. In that direction, we were inspired by Carlson's (2001, 2006) work on linking UML and XML, which we already mentioned in the previous sub-section. However, his view is data-centric, with application in transferring data between applications, while we are mainly interested in modeling, representing, logging and analyzing user-system interactions. Similarly, Crawle and Hole (2003) propose an Interface Specification Meta-Language (ISML) which appears to be related but more generic than our Interaction Modeling Language, plus they also restrict their focus to modeling, rather than logging, the interaction.

In terms of logging and log analysis, our work also falls outside the mainstream research effort. Jansen's (2006) recent review of search log analysis research indicates that most work concentrates on the collection, preparation and analysis of logs, while we focus our attention on designing and/or generating log formats appropriate for certain interaction models, as well as matching parsers for validating the logs and for extracting relevant data from such logs in an efficient, effective and flexible manner.

Jansen concludes, based on an analysis of the literature, that transaction log analysis (TLA) refers, in general, to the use of data collected in transaction logs in order to investigate particular research questions concerning interactions among Web users, the Web search engine, or the Web content during searching episodes. Moreover, transaction logs are most often a server-side data collection method, capturing requests for services

from a large number of clients, but missing details of the user-interface interaction. In contrast, we are interested in capturing and analyzing the details of the interaction with client-side logging.

In order to address the drawbacks of server-side logging, a number of researchers have combined them with online questionnaires designed to clarify the users' motivations and intentions, and to disambiguate their behaviors (Hancock-Beaulieu, 2000). Others have used client-side logging. However, most of them did not attempt to capture semantic details of the interaction. For example, Choo, Detlor, & Turnbull (2000) used their WebTracker to log Web browser actions such as "Open URL or File," "Reload," "Back," "Forward," "Add to Bookmarks," "Go to Bookmark," "Print," and "Stop", while Jansen et al. (2006) used their Wrapper to capture operating system level events such as keystrokes, browser requests for URLs, and the start/end of desktop applications. Such tools do not capture the semantic interaction between user and system.

Efforts by Gonçalves et al. (2003) and Klas et al. (2006) toward standardization of log formats in certain types of applications, such as user interfaces for digital libraries, appear closest to ours (with the caveat that their publications focus on their research objectives, and not on the implementation details that could make a comparison possible). Moreover, we suggest that our approach of deriving logging formats from user interface design should help those efforts: the functionality provided by such user interfaces should be first standardized in UML format, and then standardization of the log formats can be achieved as an immediate consequence.

### **Future Research Directions**

One issue that we are currently investigating is an extension of this methodology to studying patterns of behavior by building Hidden Markov Models (HMM) based on the analysis of state transitions recorded in the logs (Jurafsky, 2000).

One decision in building such models regards the computation of the transition probabilities. The two potential approaches are based on: (i) macro statistics – the transitions are counted and the probabilities are computed for each individual user, then the probabilities are averaged over the users; and (ii) micro statistics – the transitions are counted and the probabilities computed over all the user logs. The former approach is expected to highlight the differences between individual subjects, and the latter to show common behavior. Both approaches should probably be used so that together they paint a better picture of what is happening. Moreover, where the difference between individual and common behavior is significant, correlations with individual factors (such as familiarity with the topic) should be sought.

Considering the hierarchical structure of states, it is obvious that another issue to consider is state granularity. Taking into account just the top levels may give too coarse a view of the interaction and may not provide sufficient details to answer research questions. On the other hand, the leaf states may provide too much detail and may hide patterns in higher levels. Moreover, due to the limited amount of data generated in a lab user experiment, some of the leaf states may appear infrequently, so drawing conclusions from such sparse data may be dangerous. It is probably better to repeat the analysis for different levels of granularity or to smooth detailed interaction models with models built for transitions between coarse granularity states.

Actually, the analysis described above may prove that, for complex interactions such as information seeking, pure Markov Models are inappropriate, and that more complex extensions should be considered. It may be the case that state transitions are not determined just by the current state and certain events, but also by some parameters of the state, such as the amount of time spent, or the number of documents examined.

A very different research direction is to investigate ways to automatically generate graphical

diagrams that show the frequency of each state transition and thus give a visual display of user behavior. So far we have extracted transition frequencies with the log analyzer, but have built such visualizations manually.

Finally, we intend to investigate a number of IR user interfaces and to compare their state diagrams, trying to identify common patterns. This would allow us to provide support, in the form of reusable toolkits of frameworks, for researchers designing and evaluating user interfaces for Information Retrieval.

## ACKNOWLEDGMENT

The methodology proposed and discussed here was successfully used in designing the user interfaces and analyzing the logs for the Mediated Information Retrieval (MIR) project, conducted in the School of Communication, Information and Library Studies, Rutgers University, the author's previous affiliation. This author thanks Hyuk-Jin Lee, who conducted the user experiment, Nicholas J. Belkin and Dan O'Connor, supervisors, and David J. Harper<sup>17</sup> from the School of Computing, The Robert Gordon University, UK, external advisor.

## REFERENCES

- Belkin, N. J., Cool, C., Kelly, D., Lin, S.-J., Park, S., Perez-Carballo, J., & Sikora, C. (2001). Iterative exploration, design and evaluation of support for query reformulation in interactive information retrieval, *Information Processing & Management*, 37(3): 403-434.
- Belkin, N.J., Cool, C., Stein, A., & Thiel, U. (1995). Cases, scripts, and information-seeking strategies: on the design of interactive information retrieval systems. *Expert Systems with Applications*, 9(3), 379-395.
- Belkin, N.J., & Muresan, G. (2004). Measuring Web search effectiveness: Rutgers at Interactive TREC. In *Measuring Web Search Effectiveness: The User Perspective*, workshop at WWW 2004, May 2004, New York.
- Carlson, D. (2001). *Modeling XML applications with UML: Practical e-Business applications*. Addison-Wesley.
- Carlson D. (2006). Semantic models for XML schema with UML tooling. *Proceedings of the 2<sup>nd</sup> International Workshop on Semantic Web Enabled Software Engineering (SWESE)*, Nov 2006, Athens, GA..
- Choo, C. W., Detlor, B., & Turnbull, D. (2000). Information seeking on the Web: An integrated model of browsing and searching, *First Monday*, 5(2).
- Cooper, A., Reinmann, R., & Cronin, D. (2007). *About Face 3: The essentials of interaction design*. Wiley.
- Crowle, S., & Hole, L. (2003). ISML: An interface specification meta-language, *10th International Workshop on Design, Specification and Verification of Interactive Systems*, Madeira.
- Diaper, D., & Stanton, N. (2004). *The handbook of task analysis of Human-Computer Interaction*. Lawrence Erlbaum Associates.
- Douglass, B. P. (1999). *Doing hard time: Developing real-time systems with UML, objects, frameworks, and patterns*. Addison-Wesley.
- Dumais, S. T., & Belkin, N. J. (2005). The TREC interactive tracks: Putting the user into search. In Voorhees, E. M., & Harman, D. K. (Eds.), *TREC – Experiment and evaluation in Information Retrieval*, MIT Press.
- Ellis, D. (1989). A behavioral approach to information retrieval system design. *The Journal of Documentation*, 45(3), 171-212.

- Fisher, K. E., Erdelez S., & McKechnie, L. (2005). *Theories of Information Behavior*. Information Today.
- Fowler, Martin (2004). *UML distilled: A brief guide to the standard object modeling language* (3rd ed.). Addison-Wesley/Pearson Education.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley.
- Gonçalves, M. A., Panchanathan, G., Ravindranathan, U., Krowne, A., Fox, E. A., Jagodzinski, F., & Cassel, L. (2003). The XML log standard for digital libraries: Analysis, evolution, and deployment. *The Third Joint Conference in Digital Libraries (JCDL)*, Houston, TX.
- Hancock-Beaulieu, M. (2000). Interaction in Information Searching and Retrieval, *Journal of Documentation*, 56, 431-439.
- Harel, D. (1988). On visual formalisms, *Communications of the ACM*, 31(5), 514-530.
- Horrocks, I. (1999). *Constructing the user interface with statecharts*. Addison-Wesley.
- Ingwersen, P., & Jarvelin, K. (2005). *The Turn – Integration of information seeking and retrieval in context*. Springer.
- Jansen, B. J. (2006). Search log analysis: What it is, what's been done, how to do it, *Library & Information Science Research*, 28, 407-432.
- Jansen, B. J., Ramadoss, R. Zhang, M., & Zang, N. (2006). Wrapper: An Application for Evaluating Exploratory Searching Outside of the Lab, *SIGIR 2006 Workshop on Evaluating Exploratory Search Systems*, Seattle, WA.
- Jurafsky, D., & Martin, J. H. (2000). *Speech and language processing*. Prentice-Hall.
- Klas, C.-P., Albrechtsen, H., Fuhr, N., Hansen, P., Kapidakis, S., Kovacs, L., et al. (2006). A logging scheme for comparative digital library evaluation, In *Proceedings of the 10th European conference on research and advanced technology for digital libraries (ECDL 2006)*, Alicante.
- Kuhlthau, C. (1991). Inside the search process: information seeking from the user's perspective. *Journal of the American Society for Information Science*, 42(5), 361-371.
- Lee, H.-J. (2006). Mediated Information Retrieval for the Web Environment, Ph.D. dissertation, School of Communication, Information and Library Studies, Rutgers University, New Brunswick, NJ, May 2006.
- Limbourg, Q., & Vanderdonckt, J. (2004). Comparing task models for user interface design. In Diaper, D., & Stanton, N. (Eds.) *The Handbook of Task Analysis of Human-Computer Interaction*. Lawrence Erlbaum.
- Muresan, G. (2002). Using document clustering and language modelling in mediated information retrieval, Ph.D. dissertation, School of Computing, The Robert Gordon University, Aberdeen, Scotland, January 2002.
- Muresan, G., & Harper, D. J. (2001). Document clustering and language models for system-mediated information access. In *Proceedings of the 5th European Conference on Digital Libraries (ECDL)*, Darmstadt.
- Muresan, G., & Harper, D. J. (2004). Topic modelling for mediated access to very large document collections, *Journal of the American Society for Information Science and Technology*, 55(10), 892-910.
- Olah, J. (2005). Shifts Between Search Stages During Task-Performance in Mediated Information-Seeking Interaction. In *Proceedings of the 68th Annual Meeting of the American Society for Information Science (ASIST)*, Charlotte, NC.
- Paterno, F. (2001). Towards a UML for Interactive systems. In *Proceedings of the 8th IFIP*

*International Conference on Engineering for Human-Computer Interaction*, Toronto.

Paterno, F. (2004). ConcurTaskTrees: an engineered notation for task models. In Diaper, D., & Stanton, N. (Eds.) *The Handbook of Task Analysis of Human-Computer Interaction*, Lawrence Erlbaum Associates.

Pinker, S. (1999). *How the mind works*. W.W. Norton & Co.

Robertson, S.E., & Hancock-Beaulieu, M. M. (1992). On the evaluation of IR systems. *Information Processing and Management*, 28(4), 457-466.

Saracevic, T. (1996). Interactive models in information retrieval (IR). A review and proposal. In *Proceedings of the 59th Annual Meeting of the American Society for Information Science (ASIST)*, 33, 3-9.

Sauro, J. (2004). Premium usability: Getting the discount without paying the price, *Interactions*, 11(4).

Sharp, H., Rogers, Y., & Preece, J. (2007). *Interaction design*. Wiley.

Shneiderman, B., & Plaisant, C. (2005). *Designing the User Interface*. Addison-Wesley / Pearson Education.

Tidwell, J. (2006). *Designing interfaces*. O'Reilly.

Trættemberg, H. (2003). Dialog modelling with interactors and UML Statecharts - A hybrid approach. In *10th International Workshop on Design, Specification and Verification of Interactive Systems*, Madeira.

Vakkari, P. (1999). Task complexity, problem structure and information actions, integrated studies on information seeking and retrieval. *Information Processing and Management*, 35, 819-837.

Vakkari, P. (2001). Changes in search tactics and relevance judgments when preparing a research proposal: a summary and generalization of a longitudinal study. *Journal of Documentation*, 57(1), 44-60.

Voorhees, E. M., & Harman (2005). *TREC – Experiment and Evaluation in Information Retrieval*. MIT Press.

Wagner, F. (2006). *Modeling software with finite state machines: A practical approach*. Auerbach Publications.

Winckler, M., & Palanque, P. (2003). StateWeb-Charts: a formal description technique dedicated to navigation modelling of Web applications. In *10th International Workshop on Design, Specification and Verification of Interactive Systems*, Madeira.

Xie, H. (2000). Shifts of interactive intentions and information-seeking strategies in interactive information retrieval. *Journal of the American Society for Information Science*, 51(9), 841-857.

## KEY TERMS

**Interaction Design:** Designing interactive systems that support certain functionality and a range a user behaviors.

**Interaction Schema/Model:** A formalized description of interaction rules and actions allowed in specific contexts.

**Log Analysis:** The analysis of user behavior based on the actions recorded during interaction.

**Logging Module/System:** Component of an interactive system that logs/records relevant interaction between the user and the system (events, user actions, system responses).

**Mediated Information Retrieval:** A model of IR interaction in which the systems supports the user's exploration of the information space and the formulation of queries.

**State Diagram (Statecharts):** Model of an interactive system that describes (i) a finite number of existence conditions, called states; (ii) the events accepted by the system in each state; (iii) the transitions from one state to another, triggered by an event; (iv) the actions associated with an event and/or state transition.

**User Behavior:** The set of actions taken by a user interacting with the system in order to reach a goal or complete a task.

## ENDNOTES

<sup>1</sup> <http://www.uml.org/>  
<sup>2</sup> <http://www.w3.org/XML/>  
<sup>3</sup> <http://www.w3.org>  
<sup>4</sup> <http://www.saxproject.org/>  
<sup>5</sup> <http://www.w3.org/TR/xpath>  
<sup>6</sup> <http://www.w3.org/DOM/>  
<sup>7</sup> <http://java.sun.com/webservices/jaxb/>  
<sup>8</sup> <http://www.omg.org/technology/documents/formal/xmi.htm>

<sup>9</sup> <http://www.w3.org/TR/WD-xptr>  
<sup>10</sup> <http://www.w3.org/TR/xlink/>  
<sup>11</sup> <http://www.netbeans.org/>  
<sup>12</sup> The Text Retrieval Conference (TREC) is an open evaluation event organized annually by the National Institute for Standards and Technology (NIST) which provides an evaluation framework for research groups worldwide. The experimental model and the evaluation metrics developed in TREC have been widely adopted as standards for IR evaluation. See <http://trec.nist.gov/>.  
<sup>13</sup> <http://java.sun.com/javase/6/docs/technotes/guides/logging/index.html>  
<sup>14</sup> Detailed results and conclusions from the MIR project are reported in Lee's Ph.D. dissertation (2006), supervised by this author.  
<sup>15</sup> Names such as XPathLogAnalyzer or DOMLogAnalyzer are by no means standard names. They were chosen in the MIR project to indicate that the scanning of the logs was based on XPath, respectively on traversing the DOM tree.  
<sup>16</sup> The subjects were told that they had 20 minutes (or 1200 seconds) for investigating each topic.  
<sup>17</sup> Currently affiliated with Google.