An Integrated Approach to Interaction Modeling and Analysis for Exploratory Information Retrieval

Gheorghe Muresan
Rutgers University
School of Communication, Information and Library Science
4 Huntington St., New Brunswick, NJ 08901, USA
+1-732-932-7500-x8228
muresan@scils.rutgers.edu

ABSTRACT
In this paper, we describe a methodology that integrates the conceptual design of user interfaces with the analysis of interaction logs. It is based on formalizing, via UML state diagrams, the functionality that is supported by a system and the interactions that can take place, on deriving XML schemas for capturing the interactions in activity logs, and on deriving log parsers that reveal the system states and the state transitions that took place during the interaction. While this approach is rather general and can be applied in studying a variety of interactive systems, it was devised and subsequently applied in research work on exploratory information retrieval, where the focus is on studying the interaction and on finding interaction patterns.

Categories and Subject Descriptors
H.5.2 [Information Interfaces and Presentation]: User Interfaces – evaluation/methodology.

General Terms
Design, Experimentation, Human Factors, Theory.

Keywords
Interaction design, log analysis, UML, XML.

1. INTRODUCTION AND MOTIVATION
While much of the research work in Information Retrieval has focused on the systemic approach of developing end 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, and personal characteristics and preferences should be taken into account (Ingwersen and 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 there 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 et al, 2005).

Of particular interest to this author are models of the search interaction process and empirical 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). Of course, the interest is not simply in validating theoretical models, but also in designing systems that better respond to users’ needs, that adapt to support various search strategies, and that offer different functionality in different stages of the information seeking process.

We are interested in methodologies for running such experiments. Although no systematic study has investigated the methodological details for this kind of experiments, there is plenty of anecodal evidence to suggest that much of the investigation is manual: the researchers follow interaction transcripts or videos, and code significant actions that take place and shifts in interaction stages. This process is slow, expensive, and error prone. Logs of interactions are sometimes employed to address this issue. However, in our experience, there is usually little or no formal process in designing the logs, the logging process, and the log analysis, in order for the states of the system and the stages of the interaction to be captured. What we are proposing in this paper is a semiformal procedure that supports logging and log analysis, so that the stages of the interaction and the states of the system are captured accurately, and can be analyzed in a systematic and at the same time flexible way.

A second motivation for the proposed methodology comes from observations of interactive IR experiments where the systems had clear usability issues: “Save”, “Bookmark” or “View” buttons active when no documents were selected, or even before a search was conducted, “Search” button active when no query
was specified, “Back” button when no document was yet in the
history stack, etc. Such situations are common and not at all
surprising: these are experimental systems (as opposed to
operational systems), built for studying some aspects of
interaction, so little or no resources are available for high-
quality design and usability testing. Unfortunately, this can
potentially lead to compromised research results, as the usability
of the interface can potentially affect the searchers’ behavior.
Our proposed methodology, although imposing an initial design
overhead, promises to alleviate this situation and to support an
overall improvement in the quality of the experimental results.

2. SYSTEM STATE-BASED DESIGN OF INTERACTION AND LOGGING

2.1 General approach
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. Formal and
semiformal languages, usually used in fields such as
mathematics, physics or circuit design, have also proven their
value in modeling command language systems (Shneiderman
and Plaisant, 2004).

Our approach is based on statecharts (Harel, 1988) or, in the
more modern UML (Unified Modeling Language)\(^1\) speak, on
state diagrams. These are extensions of finite state diagrams\(^2,\)
in which the use of memory and of conditional transitions makes 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 and 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 not widely used in designing IR or other interactive
systems (according to our observations), the state diagrams are
certainly not new. What is novel is our proposed integration of
interface design with logging and log analysis and, at an
implementation level, between UML and the XML (eXtensible
Markup Language)\(^3\) family of languages. The general approach
is described here, with details discussed in the following
subsections.

From the state diagrams, an XML-based Interaction Modeling
Language (IML)\(^4\) can be derived, which will capture in a DTD
(Document Type Definition)\(^5\) or XML schema\(^6\) format the valid
states of the system, and the valid events and actions taking
place during the interaction. Subsequently, two software
modules can immediately be designed and implemented: (i) a
logger that captures each valid event and action that takes place,
and each state transition undergone by the system; (ii) a log
analyzer that uses an XML parser and identifies events, actions
and state transitions, and analyzes the data according to the
research hypotheses being investigated. Note that, apart from a
number of design decisions discussed below, these steps are
straightforward, once the state diagrams and the IML are agreed
upon. For example, if Java is the implementation language, then
the standard logging package\(^6\) makes it extremely simple to
output logs in XML. Also, open-source tools (such as
NetBeans\(^7\)) can automatically generate log parsers, given the
DTD or XML schema adopted for the logs.

![Figure 1. Integrated approach to design, logging and analysis](image)

State diagrams allow different levels of granularity via sub-
diagrams that optionally detail certain system states. It is up to
the research team to decide the level of granularity and the
precision for their model. On the one hand, a tight deadline may
force the design to include just the actions and transitions
relevant to the research questions investigated. On the other
hand, a more detailed design upfront can produce a much richer
log and support the exploration of un-anticipated research
hypotheses. For example, in a current mediated retrieval project,
we captured in the interaction model, and subsequently in the
logs, all the query edits performed by the searcher (including
backspace/delete corrections, copying and pasting, etc). While this
was not envisaged at the beginning of the project, we are now
able to investigate additional research questions, such as
whether the searcher’s familiarity with the search topic correlates
with the number of query corrections, with the query
length, or with the number of query terms typed in the query
box (as opposed to copied and pasted from the description of the
assigned search topic).

The following subsections discuss some of the decisions that
need to be taken when applying this methodology, and some
implementation details.

\(^1\) [http://www.uml.org/](http://www.uml.org/)
\(^3\) [http://www.w3.org/XML/](http://www.w3.org/XML/)
\(^5\) [http://www.w3.org/XML/Schema](http://www.w3.org/XML/Schema)
\(^6\) [http://java.sun.com/j2se/1.5.0/docs/guide/logging/index.html](http://java.sun.com/j2se/1.5.0/docs/guide/logging/index.html)
\(^7\) [http://www.netbeans.org/](http://www.netbeans.org/)
2.2 Design patterns in the log analyzer

Parsing XML has become routine due to the multitude of open-source parsers and parser generators available for a variety of programming languages. For extremely large logs, unlikely to fit in the computer memory for the analysis, a SAX (Simple API for XML)\(^8\) is needed. This type of parser identifies the beginning and end of various elements found in the log, and processes them based on the callback methods provided by the programmer/researcher. The more desirable approach, possible for logs of reasonable size, is to use a DOM (Document Object Model)\(^9\) parser, which builds a tree of the log, and allows the programmer to visit it in whatever order makes sense for a research hypotheses. For example, if the research hypothesis being investigated is related solely to the documents bookmarked by the searcher, it is possible and easy to visit just the nodes capturing document bookmarking.

It is common for XML parsers generated automatically based on DTD (such as the one produced by NetBeans) to implement the Visitor software design pattern, which 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. From our experience, we suggest combining that with the State design pattern, where different classes correspond to states in the state diagram. This allows the state objects to accumulate, summarize and report information in a simple and flexible fashion (Gamma et al, 1995).

For simple systems, the implementation of the State design pattern is straightforward. For complex states, class inheritance is used to implement subclasses, and composition is used for concurrent orthogonal states.

2.3 Explicit vs. implicit logging of states

At first sight, explicitly logging the system states is 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 and the information pertinent to a certain research question should be summarized, and possibly visualized, so that it can be interpreted by the researcher. Moreover, the log analyzer will be able to re-create the states based on the events and actions captured in the logs.

Some arguments in favor of not capturing the states explicitly, and in having the log analyzer infer them, are compelling. First, complex systems such as the user interface of a search engine are likely to have complex states, with nested sub-states, and often have concurrent orthogonal states. For example, if the user edits text in an “answer panel”, based on information collected via searching and browsing in a “search hits panel”, then the states of the two panels are components of the overall system state, and the state transitions in the two panels may happen independent of each other. Attempting to log the parallel states and the transitions is likely to produce nesting that cannot be captured in a well-formed XML document. Another advantage of capturing just events and actions in the log and re-creating the states via the log analyzer is that other interaction logs, obtained from previous experiments, or from experiments run by other researchers, can be analyzed based on the same approach., as long as these logs are converted from their native format into the XML format suggested by us.

2.4 Online vs. offline analysis

It is apparent that the State design pattern can be used both in designing the user interface, and in designing the log analyzer. A couple of related questions can be asked: (1.) is one set of classes sufficient, or should a set of classes be used in the user interface and a different one in the log analyzer ?; (2.) should the data be accumulated, summarized and analyzed online, while the experiment takes place, or should it be logged and analyzed at a later time ?

The second question is easier: we recommend logging all the events and actions, and doing the analysis offline. Here are some arguments: (i) occasionally, systems do crash during the experiment, in which case the information accumulated in the memory will be lost; (ii) for some of the statistical analysis, raw data rather than summaries or means are needed for between-subjects comparisons; (iii) previously not envisaged research questions may appear during the initial analysis, and these may be addressed if the entire raw data are available.

The answer to the first question depends on the complexity of the system, and on the designer’s preference. Our preference leans towards separating the software module for running the system from the software module for analyzing the data (in Java, these can be part of different packages) in the interest of increased cohesion and clarity.

3. CASE STUDY

We have started developing the proposed approach while running the Interactive TREC 2003 experiment\(^{10}\) and are continuing to refine it while applying it to a current project on Mediated Information Access. However, designing an IR system is a complex enterprise and the full state diagrams may be somewhat difficult to follow by an un-trained reader. Therefore, we are exemplifying here the first mock project on which we applied our methods: a JukeBox application adapted from sample code available with the Java SDK to demonstrate the use of audio and other media.

![Figure 2. JukeBox user interface](http://www.scils.rutgers.edu/~muresan/trec/inter2003.html)

---

8 http://www.saxproject.org/
9 http://www.w3.org/DOM/
Figure 3. JukeBox state diagram

```xml
<?xml version="1.0" encoding="UTF-8"?>
<elements log (record)>
  <elements record (date, millis, message)>
    <elements date (PCDATA)>
      <elements millis (PCDATA)>
        <elements message (StartSession, StopSession, Start, Pause, Forward, Backward, Stop, Resume)>
          <elements StartSession />
          <elements StopSession />
          <elements Start />
          <elements Pause />
          <elements Forward />
          <elements Backward />
          <elements Resume />
          <elements 1-welcome.wav />
          <elements 2-newest.wav />
          <elements space.wav />
        </elements>
      </elements>
    </elements>
  </elements>
</elements>
```

Figure 4. JukeBox interaction DTD

Figure 5. JukeBox log extract

Figure 6. JukeBox activity summary, extracted from log

Figure 7. JukeBox state transition diagram

Figure 8. Classes for implementing the JukeBox states

Figures 2-7 depict the user interface, the state diagram, the Data Document Definition specifying the possible actions, an extract
of the use log obtained following a user’s interaction with the system, and two versions of log analysis output: a summary of events and actions, in HTML format, and a diagram of states, in SVG (Scalable Vector Graphics)\[11\] format. Finally, Figure 8 shows the class diagrams corresponding to the states of the system. The attributes and methods of these classes are not specified and neither is the Client class, as two sets of such classes (with the same names) were used: one to run the application and the other to analyze the logs. The attributes and methods in the two sets correspond to different functionality, and are therefore different.

4. CONTRIBUTIONS AND FUTURE WORK

The proposed methodology is a novel and significant contribution to experimental 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 a mock project (the JukeBox) and on a real IR project (Interactive TREC 2003) and is being refined while being applied on a new project.

One issue that we are currently investigating is the automatic generation of the XML schema or DTD describing the interaction, based on the UML state diagram. In our projects we used a variety of modeling tools, each with its own file format, and we generated the XML schema manually (or rather intellectually). These days most modeling tools allow the export of the diagrams in XMI (XML Metadata Interchange)\[12\] format, and we are looking into converting state diagrams from XMI into XML schemas with no or minimal human effort.

We are also investigating 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 the diagrams 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.

5. REFERENCES


\[11\] http://www.w3.org/Graphics/SVG/

\[12\] http://en.wikipedia.org/wiki/XMI