Conceptual-Level Log Analysis for the Evaluation of Web Application Quality

Piero Fraternali, Maristella Matera, Andrea Maurino
Dipartimento di Elettronica e Informazione
Politecnico di Milano
P. zza Leonardo da Vinci, 32 – 20133 - Milano – Italy
{fraterna, matera, maurino}@elet.polimi.it

Abstract

So far, conceptual modeling of Web applications has been used primarily in the upper part of the life cycle, as a driver for system analysis. Little attention has been put on exploiting the conceptual models developed during analysis for application evaluation, maintenance and evolution. This paper illustrates an approach for integrating the use of conceptual models in the lower part of the application life cycle, by exploiting them in quality analysis and usage evaluation. A prototype tool for supporting the described evaluation activities is also presented.

1. Introduction

Current Web applications are very complex and high sophisticated software products, whose quality, as perceived by users, can heavily determine their success or failure. A number of methods have been proposed for evaluating their quality; some of them focus on analyzing the correctness and consistency of design specifications [1, 2, 3, 4]. Such techniques allow designers to improve the quality of the final product. However, they focus on a static description of the application, and do not take into account dynamic usage aspects, that can be only revealed by monitoring and analyzing the behavior of users interacting with the application. On the other hand, a menagerie of products exists that permit webmasters to analyze the Web server logs and extract information on application usage. However, these tools are unaware of the conceptual schema of the application (if it specification does exist) and thus understanding the relationships between the tool's output and the structure of the application is a non-trivial task, especially if the application is large and complex.

The main contribution of this paper is a framework that integrates the model-based design and development of Web applications with quality evaluation, based on the static (i.e., compile-time) analysis of conceptual schemas and on the dynamic (i.e., run-time) collection of usage data, which is automatically analyzed and contrasted with the conceptual schema of the application. Our approach is based on the Web Modeling Language (WebML) [5, 6] and its supporting CASE tool WebRatio, for the design and the development of data-intensive Web applications. However, the illustrated results are of general validity and apply to any application that has been designed using a model-driven approach, provided that the conceptual schema is available and the application runtime architecture permits the collection of customized log data.

In a previous paper [2], we have shown how, given the conceptual schema of a Web application, it is possible to automatically analyze it with respect to a quality model and identify at design time weakness of the deployed application.

In this paper we concentrate instead on the analysis of usage data, dynamically collected during the application's life. The main novelty with respect to previous works is the integration of common Web logs with additional log data, related to the content dynamically extracted from the application data source and the different hypertext structures (not only pages) the user interact with. These log data are represented in XML and analyzed by an XSL-enabled tool, fed with the XML representation of the conceptual schema of the application. The tool is able to provide useful feedbacks to the conceptual designer, represented using the conceptual schemas produced during model-driven analysis.

The paper is organized as follows: Section 2 describes the WebML development process, by shortly introducing the WebML model, some peculiarities of the proposed design activities, and the automatic generation of code by means of the WebRatio CASE tool. Section 3 introduces the WebML evaluation framework, by shortly describing the Design Schema Analyzer, the original core of the framework, and then introducing the Web Usage Analyzer, able to elaborate rich log data, for verifying the design soundness against user behavior. Section 4 shows the Web Usage Analyzer at work for the evaluation of a real Web application. Section 5 shortly illustrates the main features of the most popular log analyzers. Finally, Section 6 draws our conclusions.

2. Web development process

As reported in Figure 1, the model-driven approach to the development of Web applications consists of different phases. The application lifecycle undergoes several
cycles, where each cycle produces a prototype or a partial version of the application. At each iteration, the current version of the application is tested and evaluated, and then extended or modified to cope with the already collected requirements, as well as with new emerged requirements. After application deployment, usage data and user feedbacks become available, which may prompt for further revisions.

Out of the entire process illustrated in Figure 1, the “upper” phases of analysis and conceptual modeling are apparently those most influenced by the adoption of a conceptual model. However, we believe that also the other activities benefit from the existence of the application conceptual schema. For example, as far as maintenance and evolution are concerned, requests for changes can be analyzed and turned into changes at the conceptual level, either to the data model or to the hypertext model. Then, changes at the conceptual level are propagated to the implementation. This approach smoothly incorporates change management into the mainstream production lifecycle, and greatly reduces the risk of breaking the software engineering process due to the application of changes solely at the implementation level.

The framework proposed by this paper will show that the evaluation activity can take great advantage from a model-based approach. As recognized in literature, when a precise specification of the application structure is available, the existence of some structural problems becomes obvious [4, 7, 8, 9]. In line with this assumption, our evaluation technique aims at analyzing the quality of conceptual schemas, for identifying at design time incorrectness and inconsistencies that reduce the quality of the final application. Once the application is deployed, our evaluation framework also allows analyzing user behavior, through computation of Web usage logs, so as to verify if the solutions envisioned at design time meet the users’ requirements. As it will be illustrated in the paper, log analysis is based on the availability of the application conceptual schema. Metrics and statistics are calculated over a conceptual log, that is over log data enriched with structural elements, deriving both from an advanced logging mechanism embodied into the runtime of the application, and also from the available conceptual schemas.

2.1. WebML models

Although in principle any Web conceptual model could be used, we exploit WebML [5, 6], with which we are familiar, and its accompanying CASE tool WebRatio [10], which facilitates the implementation of conceptual-level logging. WebML is a visual language for specifying the content structure of a Web application and the organization and presentation of contents in one or more hypertexts.

The WebML Data Model adopts the Entity-Relationship primitives for representing the organization of the application data. Its fundamental elements are therefore entities, defined as containers of data elements, and relationships, defined as semantic connections between entities. Entities have named properties, called attributes, with an associated type. Entities can be organized in generalization hierarchies, and relationships can be restricted by means of cardinality constraints.

The WebML Hypertext Model allows describing how data, specified in the data model, are published through elementary units, called content units, whose composition makes up pages; it also specifies how content units and pages are interconnected by links to constitute site views, i.e., the front-end hypertexts.

The WebML Hypertext Model includes:

- The composition model, concerning the definition of pages and their internal organization in terms of elementary pieces of publishable content, called content units. Content units offer alternative ways of arranging content dynamically extracted from entities and relationships of the data schema.

- The navigation model, describing links between pages and content units, which supports information location and hypertext browsing.

- The operation model, consisting of a set of units for specifying operations for creating and updating content and interacting with external services.

Besides having a visual representation, WebML primitives are also provided with an XML-based textual representation, used to specify additional detailed properties, not conveniently expressible in the graphic notation. Web application specifications based on WebML can be therefore represented as visual diagrams, as well as XML documents.

![Diagram](image-url)
2.2. Systematic design through WebML sub-schemas

Data publishing and content management on the Web typically exhibit some regularity, which can be exploited in the design of Web applications [11]. The WebML design method therefore proposes a sequence of steps for assembling the data schema and the hypertext schema, which assume that Web applications can be abstracted as complex arrangements of elementary sub-schemas, i.e., pairs of structural diagrams (describing portion of the data schema), and hypertext diagrams (describing composition of pages and navigational patterns).

The essence of the proposed method is the classification of the role that information concepts play within the Web application:

- **Core concepts**: are the “central” objects, that is those concepts that form the main asset and express the mission of the Web application (e.g., the product sold in a B2C site, or the personal message published in a community center).
- **Access concepts**: support the location of core concepts.
- **Interconnection concepts**: interconnect core concepts, and are typically expressed by means of relationships, which the user can navigate to move the focus from one core concept to another related one.

Such a classification helps building the data schema, because designers can progressively review data requirements in search of core, access, and interconnection sub-schemas. It also helps the definition of the hypertext schema, which can be assembled using design patterns that represent “canonical” configurations of pages and units built on top of the previously identified core, access, and interconnection data sub-schemas.

Recognizing sub-schemas certainly helps designers organize their work in a systematic way and build new applications by reusing common design experiences [12], which normally results into more consistent schemas. The proposed classification of data and hypertext elements also improves the evaluation phase, because it provides a systematic way for characterizing the targets of evaluation, which are the core portions of the application schema, the access paths leading to them, and the interconnection paths for moving among them. These targets are the potential sources of problems to be investigated. Such a characterization also simplifies the reconstruction of navigation paths starting from Web usage data.

---

**Figure 2. XML representation of the WebML schema of a page.**

Figure 2 reports an XML specification of a hypertext page, named “WebML materials”, taken from the WebML-based hypertext schema of the webml.org application (see Section 4 for major details about this application).

The page includes two content units. A data unit (lines 4-15) publishes two attributes taken from a single instance of entity TextChunk, which represents the explanatory text inserted in the various pages. The instance to be published is retrieved in the database according to a selector condition (lines 8-14), specified over the attribute Title of entity TextChunk. Also, an index unit publishes a list of instances of entity MaterialType (lines 16-32), extracted from the database according to the condition expressed on attribute Title (lines 25-31). Moreover, as specified at lines 20-21, from the index unit a link originates, whose destination is a data unit (dau86) defined in a separate page of the hypertext. For further details on WebML, the reader is referred to [5, 6].
2.3. Implementation and deployment of WebML applications

The XML representation of WebML schemas enables the automatic code generation by means of CASE tools. In particular, WebML is supported by the WebRatio CASE tool [10], which translates the XML specifications into concrete implementations.

WebRatio offers a visual environment for drawing the data and hypertext conceptual schemas, and an interface to the data layer that assists designers to automatically mapping the conceptual-level entities, attributes and relationships to physical data structures in the data sources, where the actual data will be stored. The core of WebRatio is a code generator, based on XML and XSL technologies, which is able to generate automatically the application code to be deployed on J2EE or .NET platforms. More specifically, the code generator produces the queries for data extraction from the application data sources, the code for managing the application business logic, and the page templates for the automatic generation of the application front-end.

The generated applications run in a framework implemented on top of an application server. The runtime framework has a flexible, service-based architecture that allows the customization of components. In particular, the logging service can be extended with user-defined modules, so as to log the desired data.

3. The quality evaluation framework

The proposed quality evaluation framework consists of two phases, addressing two types of analysis.

Design Schema Analysis verifies the correctness and consistency of design specifications [1, 2], to enhance the quality of conceptual schemas by looking for design inconsistencies and irregularities in the application of design patterns. This phase focuses only on a static description of the application and does not take into account dynamic usage aspects.

Web Usage Analysis operates on log data dynamically collected at runtime and produces quality reports on content access and navigation sequences. This analysis exploits the so-called conceptual-level log files, defined as "enriched" Web logs that integrate the conventional HTTP data about page requests with information about the database objects used to populate pages and about the elementary page units and link paths accessed by the users.

Section 3.1 will shortly recall the main feature of the Design Schema Analyzer, already described in [2], whereas Section 3.2 will describe in more details the Web Usage Analyzer, which is the original contribution of this paper.

3.1. The Design Schema Analyzer

Design schema analysis is centered on the identification within the WebML conceptual schemas of configurations, which we call patterns, representing some potential sources of problems, with respect to some attributes of a quality model [1]. The patterns addressed by our analysis consist of compositions of hypertext elements (pages, units, operation, links) serving a typical application purpose. Examples could be the arrangement of pages, units, and links for supporting the navigation between two core objects, for accessing a core object via one or more access objects, or for creating a new object through an operation.

The generated applications run in a framework implemented on top of an application server. The runtime framework has a flexible, service-based architecture that allows the customization of components. In particular, the logging service can be extended with user-defined modules, so as to log the desired data.

The generated applications run in a framework implemented on top of an application server. The runtime framework has a flexible, service-based architecture that allows the customization of components. In particular, the logging service can be extended with user-defined modules, so as to log the desired data.

The generated applications run in a framework implemented on top of an application server. The runtime framework has a flexible, service-based architecture that allows the customization of components. In particular, the logging service can be extended with user-defined modules, so as to log the desired data.

The generated applications run in a framework implemented on top of an application server. The runtime framework has a flexible, service-based architecture that allows the customization of components. In particular, the logging service can be extended with user-defined modules, so as to log the desired data.

The generated applications run in a framework implemented on top of an application server. The runtime framework has a flexible, service-based architecture that allows the customization of components. In particular, the logging service can be extended with user-defined modules, so as to log the desired data.
Pattern description rules represent the XSL specification of schema portions to be retrieved and analyzed;

Quality criteria rules represent the XSL specification of the analysis procedures to be performed over patterns retrieved with the previous rules.

As shown in Figure 3, given the XML specification of WebML schemas, first a Pattern Extractor module applies the XSL rules stored in the Pattern Descriptions repository, to detect all the occurrences of the described patterns in the WebML specification. The identified patterns are returned in XML format, and stored in the Pattern Occurrences repository. The Pattern Analyser module uses them for verifying if the extracted elements satisfy the quality criteria coded as XSL rules in the Quality Criteria repository. The result of this analysis is a further XML document, stored in the Results repository, containing the aggregate values of measured attributes or a set of warnings, depending on the analysis method. Finally, a GUI module displays the analysis results, by showing the aggregate values in form of charts and warnings in form of textual sentences.

Figure 4 reports the result visualization for the metrics Modify End Point, calculated over the schema of the Web application of the Department of Electronics and Information (http://www.elet.polimi.it), a very large application developed with WebML.

The metrics identifies all the occurrences of the Modify pattern, used for specifying data updates, and verifies if such a pattern is applied coherently by comparing the identified configurations with respect to two possible variants. The graph shows that, among all the occurrences of the Modify pattern retrieved in the application, the designer normally uses the Different Page version (as shown by the leftmost bar), and only in the minority of cases the designer uses the Same Page version (as shown by the rightmost bar). This use of the pattern reveals a potential design inconsistency, which can be signaled to the designer for schema correction. For more details about the WebML Design Schema Analyser and its application to the analysis of complex applications, the reader is referred to [2].

3.2 The Web Usage Analyzer

Web usage analysis takes an orthogonal approach to quality evaluation. It starts from run-time collected log data about the usage of the application and helps the designer evaluate if the conceptual schema of the application adheres to the actual way in which users browse the hypertext. Any discrepancy between the design schema and the actual usage experienced at runtime is a potential candidate for correction.

The distinguishing feature of our Web usage analysis is the exploitation of the application conceptual schema, which takes place in two ways:

- The raw log data are semantically enriched with information extracted from the conceptual schema, to make them more readable and easily analyzable;
- The result of analyzing the navigation behavior of user is displayed directly on top of the graphical representation of the hypertext schema, which

![Figure 4. Metrics computation in the Design Schema Analyzer.](image)

![Figure 5. Input and output for the Web Usage Analyzer components.](image)
greatly facilitates the understanding of quality assessment results.

Web usage analysis is supported by a tool named Web Usage Analyzer, whose components, with their input and output, are depicted in Figure 5. The sequence of modules reflects the steps in which the analysis proceeds. The first module is the Log Synchronizer, which takes in input:

- The Application Server Log, which is a log in the ECLF (Extended Common Log Format) format [13], also including user session identifiers;
- The WebML Runtime Log, which stores events and data produced and consumed by the application runtime for serving page requests.

The module synchronizes these data and produces a unified XML representation of the available logs, which we call Synch Log.

![Figure 6. Extract from the Application Server Log.](image)

As a second step, the Log Conceptualizer extends the Synch Log and produces a rich representation of user accesses, in which log elements are related to the high-level elements specified in the conceptual schema. Its output is the Conceptual Log.

Finally, the Log Analyzer, implemented as XSL rule sets, elaborates the XML Conceptual Log to extract significant information, and computes quality metrics and statistics. The GUI module visually shows the results.

3.2.1. Log data sources. The primary source of Web usage data is the Application Server Log. It is organized in the ECLF (Extended Common Log Format) format [13], and is enriched with session IDs, for identifying page requests belonging to a same user session. This feature greatly simplifies the pre-processing necessary for reconstructing users’ sessions; it also allows distinguishing among requests coming from different user agents, even when proxies filter them or when users turn off cookie support on their browser.

Figure 6 shows an extract from the Application Server Log that represents the request of a page from the webml.org application (the same page specified in Figure 2). For privacy reasons, the IP address of the requesting host has been replaced with a sequence of Xs. The log includes the JSESSIONID field, whose value (acbTwnzgkSz6) identifies the user session.

The second source of data consists of the Runtime Log, which stores in XML format information about the processing of requested pages. Runtime log data include all the events generated by the application runtime when serving a requested page and populating its content units. Each event is delimited in the XML log file by the <event> tag. Since each page request is managed by a specific thread, the events generated for a single page request are characterized by the same thread number. An <event> tag denotes either the request of an entire page, or the computation of an individual unit. It may contain further sub-tags:

- The <message> tag, comprising the event parameters. In case of content units population, it also includes the list of OIDs of the objects extracted from the data source.
- The <NDC> tag, storing the identifier of the conceptual element (page or unit) to which the event refers.

```
1 <log4j:event timestamp="Sat, 29 March 2003 02:03:34 +0100"
   thread="tcpConnection-80-17">
2   <log4j:message>
3     Requested page service for id=page33
4   </log4j:message>
5  </log4j:event>
6  <log4j:event timestamp="Sat, 29 March 2003 02:03:34.962" thread="tcpConnection-80-17">
7     <log4j:message>
8       Requested page service for id=page33
9     </log4j:message>
10  </log4j:event>
11  <log4j:event timestamp="Sat, 29 March 2003 02:03:34.969" thread="tcpConnection-80-17">
12     <log4j:message>
13       Requested page service for id=page33
14     </log4j:message>
15  </log4j:event>
```

![Figure 7. Extract from the Runtime Log.](image)

Figure 7 shows an extract of the Runtime Log generated for the same page request illustrated in Figure 6. The recorded events are characterized by the same thread (tcpConnection-80-17). The first event (lines 1-7) denotes the page request. The other events (lines 8-14 and 15-21) denote the population of the units of the page. For example, the second event (line 8-14) refers to the population of a data unit (dau84). Its <message> tag (lines 10-12) includes the unitID (dau84), the IP address and the client SessionID, and a value (17) representing the OID of the single database instance extracted for populating the data unit.

The third event (lines 14-21) refers to the population of an index unit (inu9). Its <message> tag includes a list of values (line 18) representing the OIDs of the three database objects extracted for populating the index unit.

3.2.2. Log Synchronizer. This module synchronizes the Application Server Log and the Runtime Log, and generates a unified XML representation of all the
available log data. Not all the requests stored in the Application Server Log need to be considered for synchronization. Requests generated by software agents (spiders, Webcrawlers, search engines, etc.), or requests of image files included within pages can be filtered out. Also, errors generated by incorrect user requests or server exceptions, which are characterized by status code 400 or 500, can be excluded from synchronization, but they are however stored in different files used for calculating statistics about application faults. Each request in the Application Server Log is extended with the corresponding events and data coded in the Runtime Log, i.e., the identifiers of the units in the page, delimited by the `<Unit_Id>` tag, and the OIDs of the database objects extracted for populating such units, delimited by the `<Unit_Oid>` tag. The result is the Synch Log XML file. A fragment of such a file, deriving from the
3.2.3. Log Conceptualizer. The Log Conceptualizer is in charge of generating the final Conceptual Log, which merges the Synch Log and the XML-based conceptual schema of the application. The binding between log data and the elements of the conceptual schema is obtained by transforming the content of each `<Page>` and `<Unit_Id>` tag into a reference to an element of the conceptual schema. In this way, during the subsequent log analysis, relevant structural properties specified in the conceptual schema can be easily retrieved by following the references.

3.2.4. Log Analyzer. The Log Analyzer component elaborates the Conceptual Log, by retrieving relevant elements and computing metrics and statistics coded as XSL rules. As for the Design Schema Analyser, the conceptual log computation is based on two sets of rules, one dictating the elements to be retrieved, the other specifying computation procedures over them.

The present version of the Log Analyzer supports two kinds of assessments: access analysis and navigation analysis.

In general, access analysis aims at computing statistics on accesses, as commonly performed by log analyzers. However, our model-based approach, which separates the structure schema and the hypertext schema, supports two types of analysis:

- Data access analysis, which aims at computing access statistics related to database entities and their instances, purposely ignoring the hypertextual interface;
- Hypertext access analysis, which focuses on the usage of the hypertext elements (pages, areas, and site views).

Data access analysis can for example respond to such questions as "Which is the most/least accessed entity?" or "Which is the most/least accessed instance of Entity X?". Hypertext access analysis extends the statistics normally offered by state-of-the-practice log analysis tools. First, it enables the analysis of accesses to hypertext structures at different levels of granularity, covering not only page visits, but also accesses to site views, i.e., entire hypertexts defined as application front-ends, areas within site views, i.e., cohesive sets of pages publishing contents about some core objects of the application, and individual content units within pages. Second, thanks to the cross-links among the log entries, the elements of the data schema, and the elements of the hypertext schema provided in the Conceptual Log, Hypertext access analysis may take into account the actual data objects used to fill-in the requested pages. Therefore, it can respond to such questions as "Which is the most/least accessed page/area/site view displaying the content of Entity X?" or "Which is the most/least frequently used page for displaying a specific instance of Entity X?". These results greatly help designers evaluate the effectiveness of the hypertext in delivering the core...
content of the application to users and may quickly reveal design problems.

An important aspect of the Web Usage Analyzer is that the user can pose quality evaluation queries involving the accessed objects, without being aware of the physical structure of the application's URLs and, in particular, of the parameters that identify the actual objects used at runtime to fill-in the page. Objects identifiers are managed by the Runtime Log and abstracted as XML elements in the Conceptual Log.¹

Besides access analysis, also navigation analysis plays a central role in hypertext quality evaluation. While access analysis is relevant for understanding if published data are accessed by users, and for identifying possible elements that need more emphasis in the application interface, navigation analysis concentrates on verifying if the hypertext topology supports content accessibility. In particular, Navigation Analysis allows reconstructing navigation paths adopted by users for reaching the core application content, by taking into account the classification of data and hypertext sub-schemas explained in Section 2.2. The evaluator indicates which core units, publishing data about core entities, s/he is interested in. Once such units have been retrieved, access paths, defined over access entities, and interconnection paths, defined over interconnecting relationships, are reconstructed from the WebML links in the hypertext schema, and some statistics about their usage are calculated from the Conceptual Log. In this way, it is possible to highlight deviations between the reconstructed navigation paths mostly accessed by the users and the paths provided by the designer (for example, when users make frequent use of the browser's back button or neglect some navigation paths provided by the designer).

4. Case study

In order to prove the effectiveness of our framework, we have applied it to the webml.org application (http://webml.org), the official reference site about Web modeling and the WebML language. It publishes a rich set of resources, including excerpts of the WebML book [6], organized by chapters, downloadable materials, papers written by WebML researchers and by other research groups, tutorials and exercises useful for WebML teachers and students. The site has been designed with WebML and deployed with WebRatio on a J2EE platform. It consists of two site views:

- A content management site view, for gathering, storing and updating contents dynamically published in the public Web site. The access to such a site view is restricted to registered users only. This site view is organized along eight different areas, whose pages allow invoking content management operations, such as creating, modifying and deleting database objects.

- A public site view, accessible from any unregistered user interested in retrieving contents about Web Modeling, the WebML language, and its related activities. As shown in the navigation bar of the home page illustrated in Figure 10, the public site view consists of seven different areas (Overview, Book, Chapter, News, People, Resources, Research, and Industry).

The data schema of webml.org is centered on a few core entities (Book, Paper, Material, Exercise), which are interconnected among them and associated with a few additional access entities (e.g., MaterialType, PaperCategory, and so on) serving the purpose of categorizing the site's content.

The hypertext schema is quite elaborate and permits the users to reach the same piece of content (e.g., an exercise on hypertext modeling) in different ways. The application has been published online on 28/03/2003 and has an average of 60 unique visitors per day.

¹ In real Web sites, identifying the accessed objects may be difficult due to URL scrambling for security reasons, or due to the heavy use of POST-based HTTP requests, as common, for example, in the Microsoft .NET platform.
4.1. Access analysis

Figure 11 shows a data access analysis graph, generated by the Web Usage Analyzer from log data relative to a period of 15 days in March 2003. The graph shows that the most accessed entity is the one that represents the WebML book chapters, which gets 23% of the user's requests. As shown in Figure 12, by clicking on an entity, it is then possible to drill down to the distribution of accesses across the entity instances. In the specific case, Chapter TOC (Table Of Content) results the most accessed one.

Hypertext access analysis permits the further deepening of the evaluation. For example, Figure 13 ranks all the hypertext pages that have been used to access the Chapter “Table of Contents” sorted by the number of users' accesses.

The results of this analysis can suggest the introduction of navigation shortcuts for leading the users to the most accessed objects through few (hopefully one) navigation steps.

4.2. Navigation analysis

As an example of navigation analysis, let us consider the statistics Adopted Access Paths calculated over the core unit Paper, which publishes data about a WebML paper. The portion of schema of the webml.org application shown in Figure 14 represents the access sub-schema defined over the core entity Paper.

Two different access paths have been defined at design-time for accessing the core data unit Paper positioned in page WebMLPaper:

- A path defined over the access entity Category, which comprises the index unit named Categories (in page WebML papers), presenting the list of the available categories, the data unit Category (in page Papers Category), representing data about one selected category, and the index unit Papers, which finally presents the list of papers in the selected category.
- The hierarchical index unit Published Papers (placed in page WebML Papers), which publishes the list of papers hierarchically organized with respect to paper categories, and allows users to select a paper and jump directly to the Paper data unit, bypassing page Papers Category.
Another example of navigation analysis focuses on the interconnection paths that allow the user to navigate from one core entity to another one, following the semantic associations specified in the conceptual data schema. Figure 16 shows an interconnection schema between the two entities Person and Paper, which represents the association between WebML researchers and their published papers. This interconnection is rendered through two index units: one for showing the list of papers published by a person; the other for showing the list of authors of a given paper. The selection of one item in the index allows navigating to the related concept.

The result of this analysis suggests that, in order to reduce the use of the back button and the number of required navigation steps, the index of papers be replicated into the WebML Paper page.

For example, from page WebML Person, opened over a selected person, users chose one related paper, thus reaching the Paper page, where the Authors index unit shows all the authors of the chosen paper. However, in 90% of the cases, users select from the Authors index unit the same author they were previously accessing, using the index as a sort of shortcut to go back to the WebML Person page without losing the navigation context. It is worth noting that the identification of such a situation is possible thanks to the availability of log data about contents populating pages.

5. Related work

During last years, several methods and tools for the analysis of Web logs have been proposed, with the two emerging goals of calculating statistics about site activities and mining data about user profiles to support personalization.
The majority of the public and shareware tools (see for example [14, 15, 16]) are traffic analysers. As also described in [17], their functionality is limited to producing:

- Reports about site traffic, such as total number of visits, average number of hits, average view time, etc.
- Diagnostic statistics, such as server errors and page not found;
- Referrer statistics, such as search engines accessing the application;
- User statistics, such as top geographical region;
- Client statistics, such as user's Web browser and operating systems.

Only few of them (for example [15]) also track user sessions and present specific statistics about individual users’ accesses.

Our Web Usage Analyzer allows calculating traffic statistics as those previously listed. Also, thanks to the availability of session IDs within the Application Server Log, it is able to reconstruct user sessions, without needing any extra-effort for pre-processing computation, and being able to distinguish requests from different users, even when they are filtered by proxies, or when users disable cookies.

As already highlighted in Sections 3 and 4, our Web Usage Analyzer is also able to calculate advanced access statistics, related to data base entities and instances, and also to hypertext elements of any granularity. While in our approach such kinds of analysis do not require any additional effort, they are not trivial for several log analyzers. In many cases (for example for [14, 15, 16]), they are not supported at all. When provided (see for example [18]), they often require the inclusion of some scripts in the page code, able to generate ad hoc log data.

Several data mining projects have demonstrated the usefulness of a representation of the structure and content organization of a Web application [19, 20]. As reported in [21], “the description of the application structure is considered a critical input to the pre-processing algorithms that can be used as filter before and after pattern discovery algorithms, and can provide information about expected user behaviours”. However, Web usage mining approaches often require additional, sometimes complex, computations for reconstructing the application schema [21].

The Conceptual Log, which characterizes our approach, can also be easily tailored on specific mining algorithms used in Web Usage Mining, with the great advantage of eliminating the typical Web Usage Mining preprocessing phase completely. In fact, as illustrated in [22], according to our approach: i) Data cleaning is mainly encapsulated by the Log Synchronization module; ii) the identification of user sessions is done by the WebML runtime; iii) the post-mining retrieval of content and structure information is unnecessary since this information is available from the WebML conceptual schema, and is also integrated into the final Conceptual Log.

6. Conclusions

The ever-increasing spread of the Web asks for new methods for improving the quality of Web applications. Most current Web applications are centered on large sets of data and require very complex hypertext structures that very often are difficult to understand by users and do not meet their requirements. Quality enforcement has been so far pursued mostly by addressing the application analysis and design with the help of structured development methods, possibly based on conceptual models of Web applications. However, although conceptual modelling does improve the final quality of the application by fostering regularity and the definition and reuse of effective design patterns, a gap exists between the model-driven analysis and design phases and the application maintenance and evolution phases, where most of the quality evaluation activities and of the corrective actions are performed, thanks to the implicit or explicit feedback of "real" users. In particular, the conceptual schemas developed in the "upper" part of the application life cycle are not used in the post-delivery phases.

This paper has proposed an approach to quality evaluation based on the integration of the design-time conceptual schemas of the application and the usage data collected at runtime. The central contribution is the Web Usage Analyser, a software module that collects application server and runtime log data and merges them with the conceptual schema of the application, using XML as a common format for representing both data sources. The resulting conceptual log is then fed to an XSL-based processor, which extracts useful reports about the data access, hypertext access, and navigation paths occurring at run time. These reports quickly highlight weak design decisions and can be used to put in action the necessary corrections.

The original features of the proposed technique and tool can be summarized as follows:

- Web usage data are expressed in the same vocabulary as the conceptual models (e.g., they refer to entities, relationships, content units, pages, areas, and so on). This eliminates the impendence mismatch between log data and design documents, which occurs when conventional, low-level log analysis tools are used, and even permits to display Web usage statistics directly on the conceptual design diagrams.
Thanks to the ubiquitous use of XML and XSL, the quality evaluation framework is very flexible and extensible: new quality evaluation queries and measures can be easily specified by means of XSL rules and added to the rule repositories. In this way, each design team can define its own quality criteria, and code their measures within WQA by simply writing XSL code, an extensively used W3C standard.

The proposed framework is easily implementable and adapts well to any Web conceptual model. The pre-requisites for its implementation is the capability of logging a few data on the objects and object types involved in the computation of page contents. The required data have minimal size and their calculation does not impact sensibly the overall runtime performance.

Our future work will concentrate on applying the proposed evaluation framework to larger Web applications, so as to further validate our method and tools, and on the incremental enrichment of the quality metrics and statistics based on Web usage data, for the assessment of the quality criteria and attributes already identified in [1].

We are also working on the definition of a user interface for allowing designer to define new quality evaluation queries and reports, without the need of manual XSL programming.

Finally, we are investigating the application of data mining techniques for the extraction of interesting information from our rich conceptual log [22].

7. References