

WQA: an XSL Framework for Analyzing the Quality of Web Applications

PIERO FRATERNALI,
fraternal@elet.polimi.it
and
MARISTELLA MATERA,
maternal@elet.polimi.it
and
ANDREA MAURINO
maurino@elet.polimi.it

Dipartimento di Elettronica ed Informazione
Politecnico di Milano
Piazza Da Vinci 32 - 20133 – Italy

Nowadays Web applications are very complex and high sophisticated software products, where the application quality perceived by users is one of the central factors determining the success or failure of the application. Recently, design patterns have been proposed in the context of Web conceptual modeling methods, as a way to support Web designers and developers with proven successful solutions that can be reused in different contexts where the correspondent problems arise. The correct use of these methodological tools allows the designer to create complex and consistent applications. However, even when a gallery of design patterns is ready for use, the problem arises of verifying if during the application development such patterns are applied consistently. This paper therefore presents Web Quality Analyzer, an XSL-based framework, which is able to automatically analyze the XML specification of Web applications designed through WebML, with the aim of identifying the occurrence of some design patterns, and calculating metrics revealing if they are used consistently throughout the application. Metrics are defined by XSL files, which are applied over the XML specification of the application conceptual schema, so that to compute the metric. Concepts behind our approach will be exemplified by showing how the use of WQA during the development of a real Web application designed with WebML has allowed designers to improve the application quality.

1. INTRODUCTION

In the last years we have assisted to an impressive evolution of the Web, from an environment for publishing simple HTML-based multimedia documents, to a new paradigm for accessing large data sources trough complex applications. Also, current Web applications more and more address the b-to-c and b-to-b sectors, where competition among companies providing services through the Web is largely based on the acceptability of applications by users. Such a scenario raises the need for new methods enhancing both the quality of the development process and the usability of the final application.

Recently, design patterns have been proposed in the context of Web conceptual modeling methods [SCHWABE et al. 1997, BERNSTEIN 1998, NANARD et al. 1998, GARZOTTO et al. 1999] as a way to support Web designers and developers with proven successful solutions that can be reused in different contexts where the correspondent problems arise. On one side, reuse based on design patterns allows facing the complexity

of Web application development, improving the development process. On the other side, the adoption of successful design patterns can also increase the quality of final applications. First of all design patterns provide tested and successful solutions, and their use implies a major reliability of the final application. Additionally, if design patterns are consistently used whenever the problems they solve occur, it is easier for users to identify reliable expectations about the overall application structure and behavior. They are in fact able to apply past experiences to current explorations or, even better, to predict how unfamiliar sections of the application will be organized. A major consistency thus corresponds to a major usability of the final application.

This paper discusses on the experience conducted within the WebML method about the use of design patterns as conceptual tools for increasing the quality of Web applications. WebML (Web Modeling Language) is a conceptual model for data-intensive applications, which provides some high-level abstractions for modeling data-intensive Web applications [CERI et al. 2000]. The WebML language is also complemented by a CASE tool, which offers a visual environment supporting the designer in the specification of the application conceptual schemas, according to the WebML language. The tool then stores such visual specifications as XML documents, and automatically translates them into the application code.

Several design patterns have been identified and translated according to the WebML language [BRAMBILLA et al. 2000]; they suggest how WebML primitives can be composed for solving recurrent design problems. Even when a gallery of design patterns is ready for use, the problem however arises of verifying if during the application development such patterns are applied consistently. Patterns may in fact consist of a core specification, i.e., an invariant composition of WebML units that characterizes the pattern, and by some variant fragments, which extend the core specification with valid modalities by which the pattern composition can be started or terminated. According to our experience in designing application by adopting design patterns, it results that the consistency of the final application may depend not only on the adoption of design patterns for solving some recurrent problems, but also on the consistent use of the same pattern variants across the application.

This paper therefore presents *Web Quality Analyzer*, an XSL-based framework, which is able to automatically analyze the XML specification of Web applications designed through WebML, with the aim of identifying the occurrence of some design patterns, and calculating metrics revealing if they are used consistently throughout the application. Concepts behind our approach will be exemplified by showing how the use of WQA during the development of a real Web application designed with WebML has allowed designers to improve the application quality.

The paper is organized as follows: Section 2 depicts the rationale and background of our research, by briefly introducing the WebML method for the design of data-intensive Web applications. Section 3 discusses how the availability of design patterns and their use during application design can improve the application quality, and introduces the definition of our pattern-based metrics, which allow verifying if patterns have been applied consistently across the application. A metric example is also reported. Section 4 illustrates the software architecture of WQA, while Section 5 reports on some results obtained by applying our framework for the analysis of a real Web application developed through WebML. Finally, Section 6 draws the conclusions and delineates further exploitations of WQA and our future work.

2. OVERVIEW OF WEBML

WebML is a language for modeling data-intensive Web applications. It provides some abstractions that a CASE environment is able to map onto data sources and translate into concrete page templates [CERI et al. 2000]. The typical development process based on WebML consists of different phases, from Requirement Collections to Deployment and Evolution. However, in accordance with other approaches to Web modeling [SCHWABE and ROSSI 1995, GÓMEZ et al. 2000, BARESI et al. 2001, ATZENI et al. 2001], out of the entire process, the adoption of a modeling language primarily impacts on *Data Design*, aimed at organizing core information objects previously identified into a data conceptual schema, and *Hypertext Design*, which produces schemes expressing the composition of content and the invocation of operations within pages, as well as the definition of links between pages.

The WebML models for data and hypertext design, and the CASE tool that support the WebML-based development of Web applications are briefly described in the following sections. A broader description of the language and of the tool can be found in [CERI ET et al. 2000], and at <http://webml.org>.

2.1. WebML conceptual dimensions

According to WebML, Web applications have two main orthogonal conceptual dimensions.

The **Data Model** enables describing the schema of data resources according to the Entity-Relationship Model. WebML Data Model is a suitable adaptation of conceptual models for data design, as already in use in other disciplines, such as database design, software engineering, and knowledge representation. It is compatible with the Entity-Relationship data model, used in conceptual database design. The fundamental elements of structural models are *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 **Hypertext Model** enables describing how data resources are assembled into information units and pages, and how such units and pages are interconnected to constitute a hypertext. The WebML hypertext model includes a *composition model*, concerning the definition of pages and their internal organization in terms of elementary interconnected units, and the *navigation model*, describing links between pages and content units to be provided to facilitate information location and browsing. The Hypertext Model also offers a set of operations for specifying the invocation of external operations for managing and updating content. The following sections will details the units in the WebML hypertext model.

The elementary units provided by WebML for hypertext specification are detailed in the following sections.

2.1.1. Composition model

As described in Table , the *composition model* is based on a set of building blocks, called *content units*, i.e., elementary information elements that may appear in the hypertext. Units represent one or more instances of the entities of the structural schema, typically selected by means of queries over the entity attributes or over their relationships. In particular, they allow representing a set of attributes for an entity

instance (*data units*), all the instances for a given entity (*multidata units*), list of properties (also called *descriptive keys*) of a given set of entity instances (index units), and also scrolling the elements of a set one-by-one (*scroller units*).





Data Unit	Multi-data unit	Index unit	Scroller unit
			

Table I: The five content units in the WebML composition model.

2.1.2. Navigation model

The *navigation model* is based on the definition of links that connect units and pages, thus forming the hypertext. Links can connect units in a variety of legal configurations, yielding to composite navigation mechanisms. Links between units are called *contextual*, because they carry some information (called context) from the source unit to the destination unit; in contrast, links between pages are called *non contextual*.

Figure 1 depicts a simple hypertext, consisting of two pages, Page 1 and Page 2. Page 1 includes an Index Unit and a Data Unit, defined over the same Entity A. The former shows the list of all the instances for the Entity A; the latter shows details about one single instance. The contextual link between the index unit and the data unit transports the context about the item selected from the index. The data unit uses this context for displaying the instance details. A non-contextual link allows instead navigating from Page 1 to Page 2, where a Multidata Unit shows data about all the instances of Entity B, independently from the content displayed in Page 1.

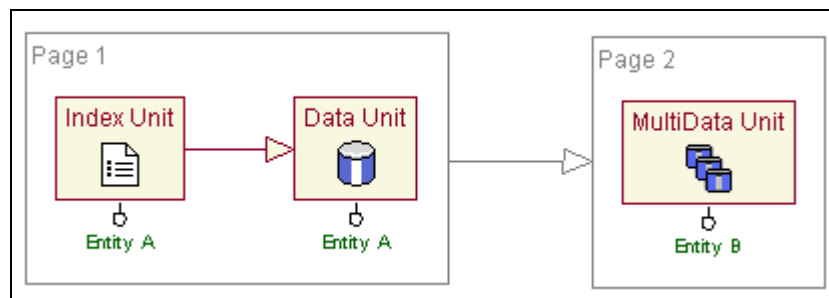


Figure 1: Example of contextual and non-contextual navigation.

2.1.3. Content management operations

In addition to the specification of read-only Web sites, where the user interaction is limited to information browsing, WebML also supports applications requiring write access to the information hosted in a site (e.g., the filling of a shopping trolley or the update of the users' personal information).

As reported in Table , WebML extends the *composition and navigation models* with additional primitives for collecting input values into fields (*data entry unit*), and for expressing built-in update operations, such as creating, deleting or modifying an instance of an entity (respectively represented through the *create*, *delete* and *modify* units), or adding or dropping a relationship between two instances (respectively represented through the *connect* and *disconnect* unit).

From the user point of view the execution of an operation is a side effect of navigating a contextual link; operations may have different incoming links, but only one is the activating-one.






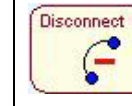
Data Entry unit	Create unit	Delete unit	Modify unit	Connect unit	Disconnect unit
					

Table II: The WebML operation units.

2.2. The WebML CASE tool

The WebML CASE tool (<http://www.webratio.com>) largely assists designers in the execution of data and hypertext design. First of all, it offers a visual environment for drawing the data and hypertext conceptual schemas. Such visual specifications are then stored as XML documents. On the basis of such documents, the WebML code generator produces a set of page templates and unit descriptors, which enable the execution of the application in the runtime layer. A page template is a template file (e.g., a JSP file), which expresses the content and mark-up of a page in the mark-up language of choice (e.g. in HTML, WML, etc.). A unit descriptor is an XML file, which expresses the dependencies of a WebML unit from the data layer (e.g., the name of the database and the code of the SQL query from which the population of an index must be computed).

Both the templates and the unit descriptors are produced by a set of translators coded in XSL and executed on the XML specification by a standard XSL processor. The WebML case tool provides also an interface to the Data Layer to assist the designer in mapping an abstract WebML structure schema to an existing data repository (e.g., a relational database).

3. PATTERN-BASED QUALITY METRICS

The concept of design pattern, as a cognitive tool to capture the human expertise and to make it reusable, was introduced by the architect C. Alexander, applied to architecture and urban planning: "... Each pattern describes a problem which occurs over and over again in our environment, and then describes the core solution to that problem, in such a way that you can use this solution a million of times over..." [ALEXANDER et al. 1977]. Patterns are now used in many other contexts. In software engineering, design patterns are a fashionable topic, and are increasingly used to describe design experience for object-oriented programming [GAMMA et al. 1995]. More recently, design patterns have been introduced in the Web modeling field for describing navigation and structural features [SCHWABE et al. 1997, BERNSTEIN 1998, NANARD et al. 1998, GARZOTTO et al. 1999].

In WebML, a gallery of design patterns records commonly applied and tested solutions that are general to any type of Web applications. Patterns have been discovered for data design, by identifying typical roles played by information objects within the data schema (i.e., core concepts, interconnection concepts, access facilitators), and typical data sub-schemas constructed around such roles (i.e., core sub-schema, interconnection sub-schema, access sub-schema) [CERI et al. 2002]. Patterns have also been defined for

hypertext design, by identifying unit compositions representing typical hypertext navigation chains [BRAMBILLA et al. 2000]. As a novelty with respect to other methods for conceptual design, WebML also introduces patterns for content management operations, i.e., unit compositions built around operations for creating new information objects, and modifying or deleting existing information objects.

Besides enhancing the quality of the development process, the use of design patterns also improves the quality of the final applications. The consistent use of patterns makes it easy for users to identify reliable expectations about the overall application structure, and the way to access and explore information and activate operations. In fact, consistent patterns allow users to apply past experiences with the application to future explorations or, even better, allow them to predict how an unfamiliar section of the application will be organized.

However, our experience in using design patterns also suggests that defining and exploiting design patterns during design is not enough for guaranteeing the quality of the final applications; rather, it is also necessary to verify if design patterns are applied consistently. In fact, a pattern typically consists of a *core specification*, representing the invariant WebML unit composition that characterizes the pattern, and a number of *pattern variants*, which extend the core specification with all the valid modalities in which the pattern can start (*starting variants*) or terminate (*termination variants*). Starting variants describe which units can be used for passing the context to the core pattern composition. Termination variants describe instead how the context generated by the core pattern composition is passed to successive compositions in the hypertext. In some cases, it may therefore happen that, even when the same pattern is adopted along the whole application, some problems may be caused by the use of different pattern variants that make the final applications inconsistent.

3.1. Definition of pattern-based metrics

In order to verify the consistent use of patterns, we have defined a set of metrics that allow quantifying the distribution of pattern variants throughout a WebML-based application. According to our definition, such metrics consist of three elements:

- **Pattern description**, representing all the valid compositions that implement the pattern in WebML, whose occurrence must be identified within the XML specification of the application conceptual schemas for metric computation. In particular, the pattern description illustrates the invariant core part of the pattern, together with some variants describing different ways of starting and terminating the pattern. A rationale, justifying the pattern use, and examples of use are also reported.
- The **calculus method**: a formula whose computation generates numerical values quantifying the consistency of the pattern usage throughout the whole application.
- The **scale of measurement**, which allows associating nominal values (e.g., Insufficient, Weak, Discrete, etc.) to ranges of the numerical values generated by the metric computation.

The following section reports an example of metric definition, by introducing the three elements above illustrated for one of the defined WebML metrics.

3.2. An example: the Modify-End-Point metric

The Modify-End-Point (MEP) metric is one of the metrics we have defined for verifying the consistent use of the WebML Modify pattern. The Modify pattern is centered upon the WebML Modify unit (see Table), which represents the invocation of a

content management operation for modifying an instance of an entity in the application information content. In particular, the metric concentrates on verifying the consistency of the Modify pattern with respect to its termination variants.

3.2.1. Pattern description

PATTERN TITLE: MODIFY PATTERN

RATIONALE

The modify pattern solves the problem of let the users modify one or more instances of an entity of the application content. It consists of inserting new attributes values and then invoking an operation for modifying the instance values in the underlying database.

In several Web applications users are required to modify instances of some entities in the application content. This is needed for example when users want to modify their personal data in Web applications where user registration is required, or the content of shopping trolleys (e.g., the quantity of ordered products) in e-commerce application. Content updating operations are also needed when Web site administrators want to update the application content through a content management front-end.

EXAMPLE OF APPLICATION

Figure 2 shows a page from the Amazon Web application (<http://www.amazon.com>), through which registered users can modify their personal data. This operation corresponds to modifying an instance of an entity in the application data schema, which serves the purpose of recording profiles of all registered users.

In order to modify their profile, users insert new values for some (or all) profile attributes, through the form reported in Figure 2. The “Submit changes” button then activates the updating operation.

The screenshot shows the Amazon.com website interface for account modification. At the top, there is the Amazon logo and navigation links: VIEW CART, WISH LIST, YOUR ACCOUNT, and HELP. Below this is a category menu with buttons for WELCOME, ANDREA'S STORE, BOOKS, ELECTRONICS, DVD, MUSIC, CARS, COMPUTERS, and SEE MORE STORES. The main heading is 'Your Account > Change Your Name, E-mail Address, or Password'. A sub-heading reads: 'If you wish to change the name, e-mail address, or password associated with your Amazon.com customer account, you may do so below. Be sure to click the Submit changes button when you are done.' The form contains the following fields: 'New name? Please enter it below:' with a text input field containing 'andrea'; 'New e-mail address? Please enter it below:' with a text input field containing 'maurino@elet.polimi.it'; 'Want to change your password? Please enter it twice below:' with two text input fields for 'New password:' and 'Re-enter password:'. A yellow 'Submit changes' button is located at the bottom of the form.

Figure 2: Page for personal data modification in the Amazon Web site (<http://www.amazon.com>).

CORE PATTERN SPECIFICATION

The modify pattern in WebML is built around the Modify operation, which allows updating the content of an entity instance with new data inserted through a data entry unit.

As reported in Figure 3, the core specification of the pattern consists of the composition of a data entry unit, providing a form for inputting new data for the instance attributes to be modified, with a Modify operation unit. The data entry unit may

eventually be pre-filled, with current values of the instance attributes. The contextual link defined between the entry unit and the operation transports all the new attributes values inserted by users, which must replace the values currently stored in the database. It also transports the identifier (OID) of the instance, which has been passed to the entry unit by some unit preceding it, as it will be better explained in the pattern variants section. If the operation fails, an error page is also displayed, showing a message that invites users to repeat the operation. A non-contextual link then allows users to go back to the initial page.

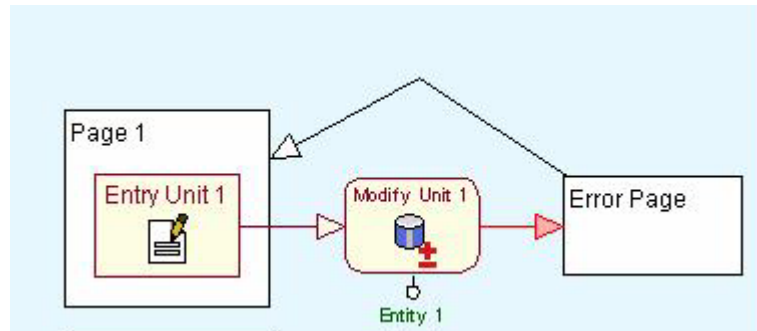


Figure 3: Core specification of the modify pattern.

PATTERN VARIANTS

Two sets of variants may be defined for the Modify pattern: the *starting* variants¹, referring to the modality in which the pattern begins (i.e., which unit passes to the entry unit the OID of the instance to be modified), and the *termination* variants, referring to pattern termination.

Three main compositions may be conceived as starting variants:

1. *Start-with-data-unit variant.* As shown in Figure 4, one possible pattern variant consists of a data unit preceding the data entry unit. The data unit, previously selected via any valid navigation chain, details the instance to be modified. A contextual link from the data unit to the entry unit then transports the instance OID, as well as parameters corresponding to the current values of the instance attributes to be eventually displayed in the entry unit, so that users can modify them. Several navigation chains are available for reaching the data unit displaying the entity instance, but this is outside the scope of this pattern.

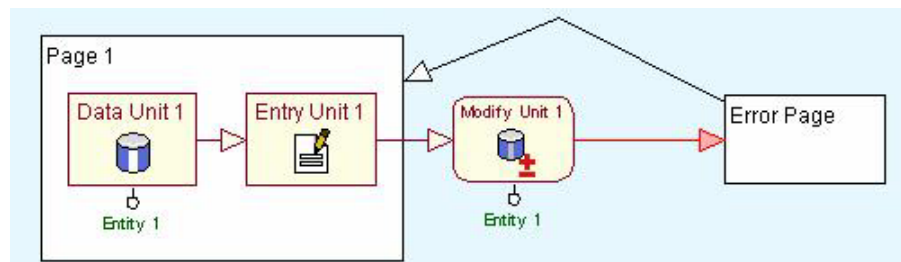


Figure 4: The start-with-data-unit variant of the modify pattern.

¹ Starting variants are here reported for sake of completeness; they are not addressed by the MEP metric. Another defined metric (Modify Start Point) serves the purpose of verifying the Modify pattern consistency with respect to its starting variants.

2. *Start-with-index-unit variant.* As depicted in Figure 5, a second variant of the Modify pattern consists of having an index unit preceding the entry unit. The index unit shows the list of all (or a set of) the instances of an entity. Each instance in the index is represented through some key attributes. The user thus selects the instance to be modified from the index unit, and the contextual link defined between the index units and the entry unit transports the OID Of the selected instance and the values of the attributes to be preloaded in the entry unit, so that users can modify them.

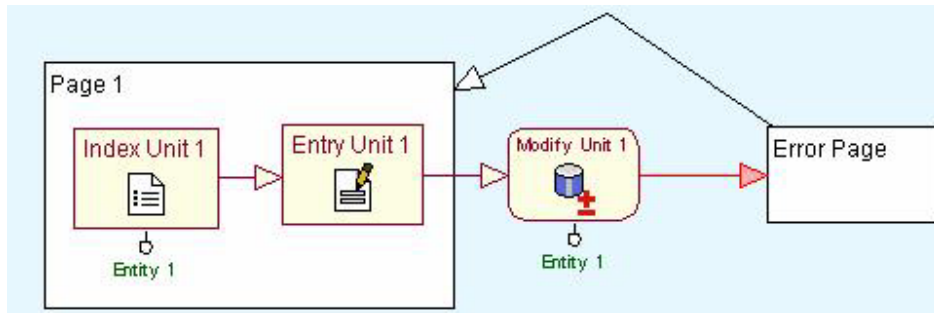


Figure 5: The start-with-index-unit variant of the modify pattern.

3. *Start-with-multidata-unit variant.* As shown in Figure 6, it is also possible to merge the two previous variants, by having a multidata unit preceding the entry unit. The multidata unit allows presenting multiple instances of an entity together, by repeating the presentation of several, identical data units.

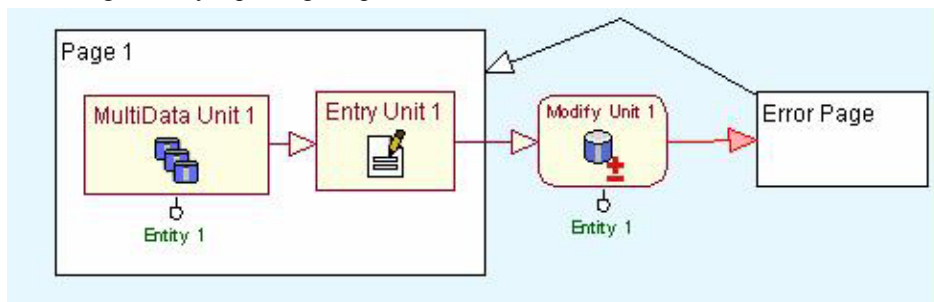


Figure 6: The start-with-multidata-unit variant for the modify pattern.

Termination variants include two meaningful variants:

1. *Same Page* termination variant. As shown in Figure 7, after the operation completion, users are lead back to the same page in which the operation has started. In this way they are allowed to further modify the same entity instance; also, if start variants 2 and 3 are adopted, users can choose to modify other entity instances, different form the one previously modified. It is worth noting that although Figure 7 depicts a non-contextual return to the initial page. Contextual return may also occur, so that the OID of the modified instance is passed to a unit within the page. This may be for example needed when the modified data must be shown in the page. The distinction between contextual and non-contextual return is not however relevant for the definition of our metric.

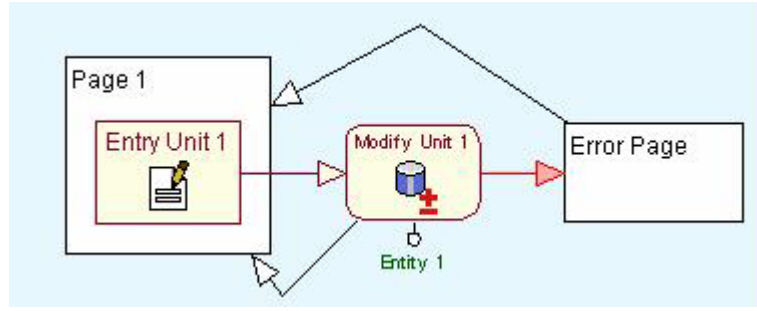


Figure 7: The Same Page termination variant for the Modify pattern.

2. *Different Page* termination variant. As shown in Figure 8, after the operation completion, users are led to a page different from the initial one. In this case, in order to further modifying the same entity instance or choosing to modify other instances, users must go back to the page where the operation starts. However, this last navigation step is not relevant for the pattern variant definition, because it is concerned with navigation chains external to the mere pattern specification. The observation about contextual and non-contextual return mentioned for the previous pattern also applies to this variant.

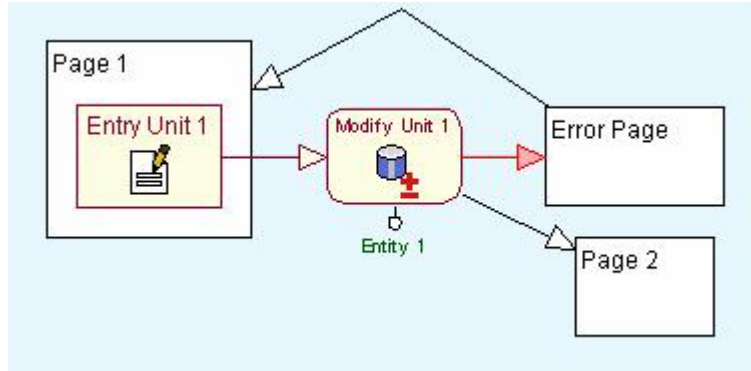


Figure 8: The Different Page termination variant for the Modify pattern.

3.2.2. Calculus method

The MEP calculus method defines a formula for computing the statistical variance of the occurrences of the two termination variants for the Modify pattern, normalized with respect to best-case variance. Therefore:

$$MEP = \frac{\sigma^2}{\sigma_{BC}^2} \quad (1).$$

σ^2 is the statistical variance of the termination variants occurrences, which is calculated through the following formula

$$\sigma^2 = \left(\frac{1}{N} \right) \sum_{i=0}^N \left(p_i - \frac{1}{N} \right)^2 \quad (2),$$

where N is the number of pattern variants addressed by the metric (in this case 2), while p_i is the percentage of occurrences for the *i*-th pattern variant.

σ_{BC}^2 is instead the best case variance, and is therefore calculated through the formula (2), assuming that only one variant has been coherently used throughout the application. It therefore follows that:

$$MEP = 4\sigma^2 \quad (3).$$

3.2.3. Measurement scale

The last step in the metrics definition is the creation of a measurement scale, which defines a mapping between the numerical results obtained through the calculus method with meaningful discrete values. According to the scale types defined in the measurement theory [ROBERTS 1979], the MEP metric adopts an ordinal nominal scale; each nominal value in the scale expresses a consistency level, corresponding to a range of numerical values of the metrics, as defined in the following table.

MEP range	Measurement scale value
0 <=MEP<0.2	Insufficient
0.2<=MEP<0.4	Weak
0.4<=MEP<0.6	Discrete
0.6<=MEP<0.8	Good
0.8<=MEP<= 1	Optimum

Table III: The measurement scale defined for the MEP metric.

4. WQA: A TOOL FOR ANALYZING WEB APPLICATION QUALITY

WQA is an XSL-based framework that can be applied over the XML specification of WebML projects for calculating pattern-based metrics, like the one defined in the previous section. The tool is based on the XSL coding of the three metric components reported in Section 0. In particular, as depicted in Figure 9, the WQA software architecture is based on the three repositories *Pattern Description*, *Calculus Method* and *Measurement Scale*, which contains the XSL/XSLT specification of the three components.

The XSL language [W3C-XSL 2001] allows writing pattern-matching rules that can be applied over an XML document for generating a new XML document. Each rule contains a matching part for selecting the target XML elements, and an action part to transform the matched elements. The XSL documents stored in the three WQA repositories serve therefore the purpose of extracting the instance of pattern variants from the XML specification of the WebML conceptual schema, thus generating a new XML document specifying all the retrieved occurrences, on which further transformation can be applied for calculating metrics according to its calculus method.

As reported in Figure 9, in order to calculate a metric, the *Pattern Extractor* module uses the pattern description for analyzing the XML specification of the WebML project and retrieving all the occurrences of the pattern variants the metric focuses on. An XML document is generated by this first analysis, which is stored into the *Pattern Occurrences* repository. The *Data Analyzer* module receives in input the pattern occurrences document, and calculates the metric by applying over this document the calculus method, stored as an XSL file in the *Calculus Methods* repository. The result of this analysis is a further XML document, which is stored in the *Aggregations* repository. The *Metric Elaboration* module then loads the XSL specification of the measurement scale from the *Measurement Scales* repository, and applies it to the aggregation file previously

generated. The result is a nominal value from the measurement scale, which is thus stored in the *Results* repository. The tool is then able to visualize the analysis results by using some graphic representations, as it will be shown in the following section.

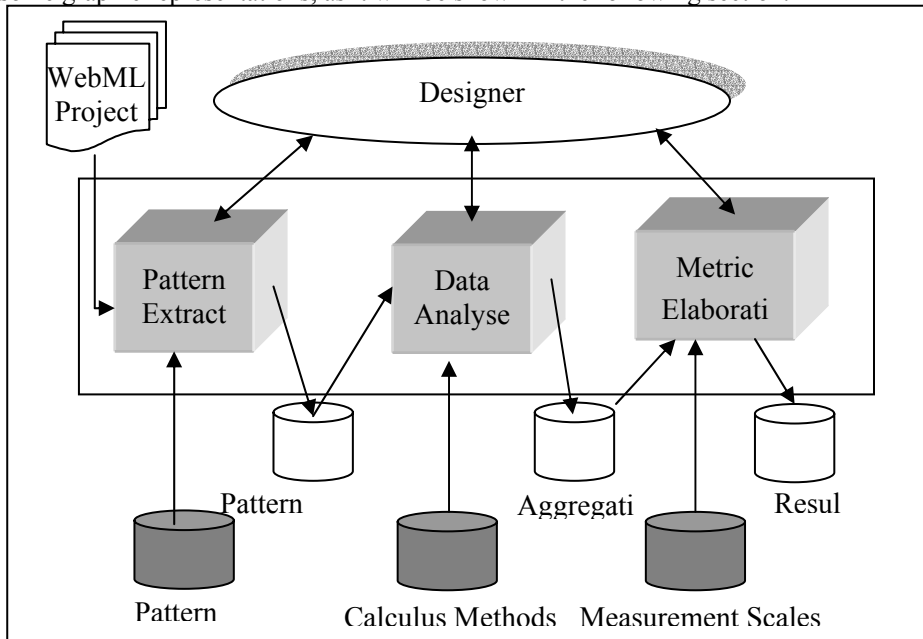


Figure 9: The WQA software architecture.

5. WQA AT WORK

We have used the WQA tool to support the development of the DEI application, the new Web application, currently still under development, of our department (Dipartimento di Elettronica ed Informazione) at Politecnico di Milano. The DEI application is aimed at presenting the main activities of the department and people working in it. Moreover, it also includes a content administration section, which provides Web administrators with an easy-to-use front end for creating or updating content to be published via the Web application. The development of the DEI application has been carried on by a company external to our department, which has however chosen to adopt the WebML language and method. We have therefore been able to apply WQA for analyzing this application.

WQA has been used within an iterative development process, in which different application versions are being released, tested and evaluated, and then improved with respect to the problems and bugs raised by evaluation. As better explained in the sequel, the analyses so far accomplished through WQA have helped designers discover and correct some problems, thus improving the application quality. WQA has been so far used for analyzing 14 successive versions of the DEI application by applying 11 different metrics to evaluate its internal consistency. However, for sake of brevity, in the following the analysis process will be exemplified by only discussing data obtained through the computation of the MEP metric (already presented in section 0).

5.1. Applying the MEP metric to the DEI application

Figure 10 plots the history of the MEP metric along the 14 releases of the DEI applications so far analyzed through the WQA tool.

The graphic in Figure 10 highlights four different phases (from A to D) in the application development. Phase A is characterized by a scarce care about design consistency. The initial high value of the metric for release 1 only depends on the limited number of modify operations featured at that time within the application. However, as soon as the number of modify operations has started growing in the following releases, the consistency value of the Modify pattern termination has decreased, until reaching its lowest value (0.04) in release 5. At this point, the evaluator using WQA raised the problem. Consequently, designer started modifying the application, by trying to use one only pattern variant (the Same Page termination variant) in almost every case. The release 6 already reflects this re-engineering, featuring an improvement of the metric value, from 0.04 to 0.54. The improvement can also be noted from the percentages of use of the two pattern variants in release 5 and 6 detailed in Table .

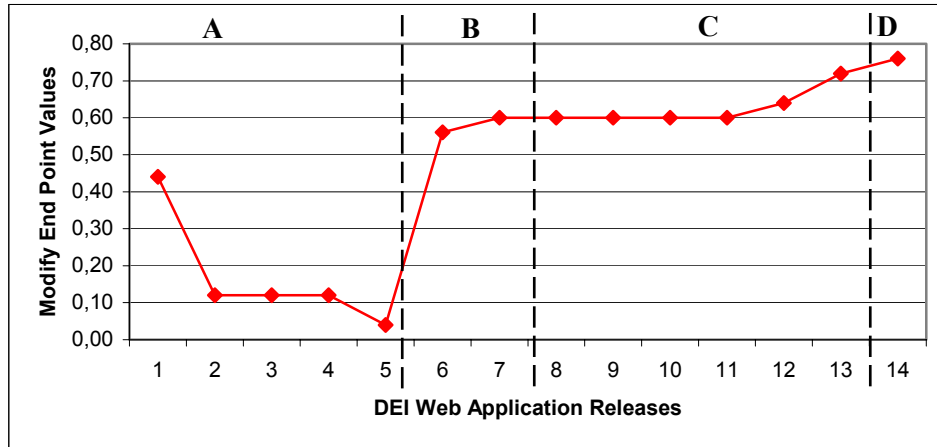


Figure 10: History of the MEP computation along the different DEI application releases.

	Different Page Termination	Same Page Termination
Release 5	42,86%	57,14%
Release 6	12,5%	87,5%

Table IV: The percentage of the occurrences of the two different Modify pattern variants within releases 5 and 6.

From release 6 a new phase starts, in which the MEP metric computation has produced a constant value - no modifications have been applied on the modify operation. In phase D, from release 12 to 14, we have instead assisted to the always-increasing effort of designers to improve consistency. The last computed value for the MEP metrics is 0.76 – which corresponds to a “good” level of consistency in the nominal measurement scale defined for the MEP metric.

Figure 11 reports the visualization generated by WQA for showing the results of the MEP computation for the last analyzed release of the DEI application. The metric value is reported in the bottom area of the widow. The graph in the central area represents the percentages of use of the two MEP termination variants. The left bar also gives indication about pages where the two pattern variants have been retrieved, so that the designer can easily locate the portion of the schema to be eventually modified. Through the graph of

Figure 11, it is therefore easy to identify that, among all the occurrences of the Modify pattern retrieved in the application, only in one case (within Page 52), the designer has used the Different Page pattern termination.

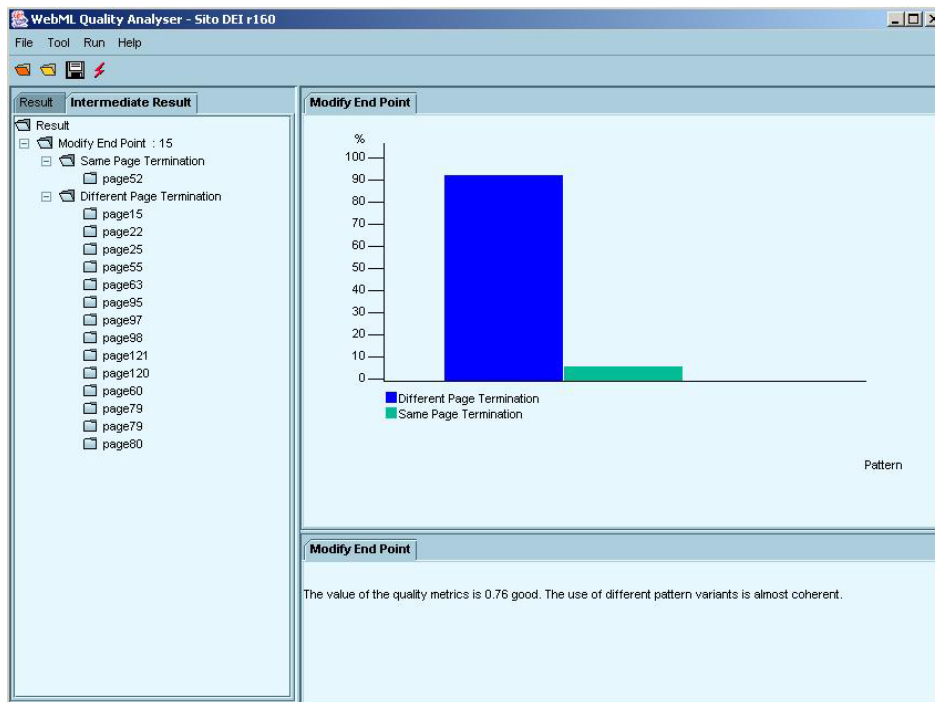


Figure 11: Number of occurrences for the two termination variants of the Modify pattern in the last analyzed release of the DEI Web application.

Figure 12 shows the page (titled *Project Editing*) where the problem has been identified. This page belongs to the application area through which Web administrators can modify the application content. It contains a modify pattern instance for updating data of a research project. A data unit (*Project details*) shows details of a specific project. This unit is connected to an entry unit (*Project Data Entry*) from which a link originates activating the modification operation (*Modify project*). The result of the operation is shown in a data unit (*Project*), contained in a different page. The same figure also shows that another occurrence of the *Modify* pattern, centered on the Modify research area operation and aimed at modifying data about research areas, ends in a different way with respect to the previous described operation.

It is worth noting that in some cases it could happen that inconsistencies are caused by conscious design choices, needed for responding to specific application constraints. However, in the situation above described, as also confirmed by the application designers, the retrieved inconsistency, as well as other inconsistencies identified for the previous application releases, were not explicit choices, but rather mistakes. This shows that WQA can improve the design process and the quality of the final application, by providing a “black box” analysis of the application schema, able to highlight potential design problems, with a low involvement of designers and evaluators.

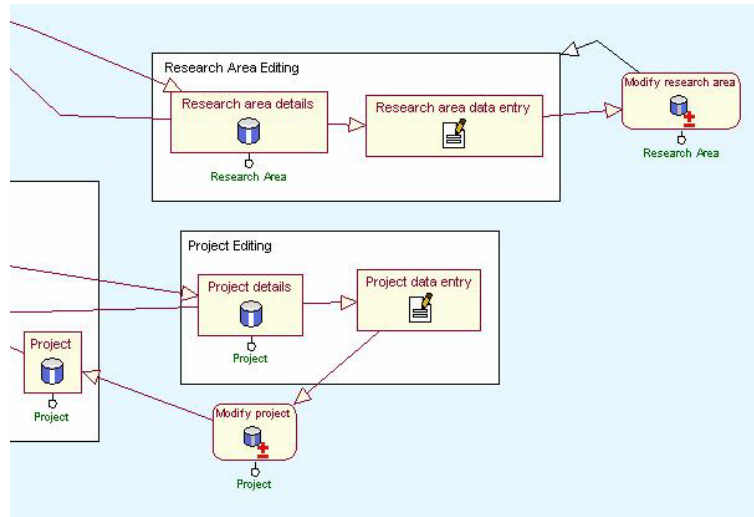


Figure 12: A fragment of the DEI WebML schema, highlighting the design inconsistency discovered for the Modify pattern.

6. CONCLUSIONS AND FUTURE WORKS

This paper has presented some preliminary results about the development and use of WQA, a tool that allows the design team to automatically monitor the internal consistency of WebML-based applications, by calculating metrics about the coherent use of design patterns throughout the application schema.

WQA is an open and flexible framework. In fact, the WebML metrics so far defined are not the only possible metrics that the WQA is able to compute. If needed, new metrics can be easily defined and automatically calculated through WQA - we are currently working on the identification of new metrics. Defining a new metric just requires specifying it through the three XSL files corresponding to the three metrics components, and then storing such files into the repositories on which WQA is based. Also, the WQA software architecture can be potentially adapted for analyzing the quality of applications designed through any other conceptual model; the only two mandatory requirements are that the application schemas be specified in XML, and the metrics must be represented in XSL.

The experience about the definition and computation of metrics based on design patterns has been conducted in a broader research context, which is aimed at defining efficient and effective methods for the development of data-intensive Web applications. In this context, several efforts are devoted to the definition of design patterns, which can improve the development process of Web applications. We are currently working on a software module, which provides a wizard enabling the automatic generation of WebML hypertexts via selection of design patterns from the gallery of WebML patterns. The wizard first allows designer to mark the information objects in the data schema, according to the role characterization reported in [CERI et al. 2002]. Then, it allows selecting the correspondent hypertext patterns, among the set of design patterns associated to the specific information object roles previously marked in the data schema.

We are also defining mining techniques, for supporting the discovery of new design patterns through the analysis of schemas of applications developed with WebML. So far, pattern definition has been performed “at hand” by expert designers. Based on rules of thumbs, they have tried to abstract and describe in a general format solutions that they

use for solving particular problems. Our idea is instead to extend the WQA tool with a mechanism for the automatic analysis of XML application schemas, aimed at capturing project data about the recurrent use of some design solutions. Such data can then feed a data warehouse, where sophisticated analysis can be conducted for discovering possible associations and sequentialities among unit compositions.

7. ACKNOWLEDGMENTS

We are very grateful to Marco Murro and Fabiana Donadini for their valuable help in implementing WQA. We are also grateful to the Wise-lab, and in particular to Simone Avogadro, for giving us the opportunity to apply WQA over the DEI application. We also thank also to Prof. Stefano Ceri for his encouragement to write this paper.

References

- ALEXANDER, C., ISHIKAWA, S., SILVERSTEIN, M., JACOBSON, M., FIKSDAHL-KING, I., AND ANGEL, S., *A Pattern Language*. Oxford University Press, New York, 1977.
- ATZENI, P., MECCA G., AND MERIALDO, P., Data-Intensive Web Sites: Design and Maintenance. *World Wide Web*, 4 (2001), pp. 21-47.
- BRAMBILLA, M., CERI, S., FRATERNALI, P., MATERA, M. The WebML Methodology. W3I3 Technical Report. Dipartimento di Elettronica e Informazione, Politecnico di Milano, 2000.
- BERNSTEIN, M. Patterns of Hypertext. Proc. of HT'98 (Pittsburgh, PA), ACM Press, June 1998.
- BARESI, L., GARZOTTO, F., PAOLINI, P. Extending UML for Modeling Web Applications. Proc. of the 34th Hawaii International Conference on System Sciences, 2001.
- CERI, S., FRATERNALI, P., BONGIO, A. Web Modeling Language (WebML): a modeling language for designing Web sites. Proc. of WWW9, and *Computer Networks*, 3 (1-6), 137-157 (2000).
- CERI, S., FRATERNALI, P., MATERA, M. Conceptual Modeling of data-intensive Web applications. *Internet Computing*. To appear.
- GÓMEZ, J., CACHERO, C., PASTOR, O. Extending a Conceptual Modeling Approach to Web Application Design. Proc. of the 12th International Conference CAISE 2000, LNCS 1789, Springer Verlag, 79-93, 2000.
- GAMMA, E., HELM, R., JOHNSON, R., VLISSSEDES, J. *Design Patterns - Elements of Reusable Object Oriented Software*, Addison Wesley, 1995.
- GARZOTTO, F., PAOLINI, P., BOLCHINI, D. VALENTI, S. Modeling-by-Patterns of Web Applications. Proc. of the ER'99 Workshop "World Wide Web and Conceptual Modeling", Paris, France, November 1999, Springer Verlag.
- NANARD, M., NANARD, J., KAHN, P. Pushing Reuse in Hypermedia Design: Golden Rules, Design Patterns and Constructive Templates. Proc. of ACM Hypertext '98 (Pittsburgh, PA), June 1998.
- ROBERTS, F.S. *Measurement Theory with Applications to Decision Making, Utility and the Social Sciences*. Addison Wesley, 1979
- SCHWABE, D., GARRIDO, A., ROSSI, G. Design Reuse in Hypermedia Application Development. Proc. of ACM Hypertext '97, Southampton, UK, April 1997.
- SCHWABE, D. AND ROSSI, G. The object-oriented hypermedia design model. *Communications of the ACM*, 38(8), 45-46, 1995.
- Extensible Style sheet Language – W3C Recommendation. October 2001 (<http://w3.org/TR/XSL/>).