DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN
UNIVERSIDAD POLITÉCNICA DE VALENCIA

P.O. Box: 22012 E-46071 Valencia (SPAIN)

Informe Técnico / Technical Report

Ref. No.:	DSIC-II/14/07	**Pages:**	126
Title:	A functional framework for analysis of undemandedness in websites		
Author(s):	M. Alpuente, D. Romero, and M. Zanella		
Date:	Julio 10, 2007		
Keywords:	web verification, maude, web service, undemanded, abduction		

V^o B^o
Leader of research Group Author(s)

UNIVERSIDAD POLITÉCNICA DE VALENCIA

DEPARTAMENTO DE SISTEMAS
INFORMÁTICOS Y COMPUTACIÓN

M. Alpuente[1], D. Romero[1], and M. Zanella[2]

A functional framework for analysis of undemandedness in websites

– Julio 2007 –

[1]Universidad Politécnica de Valencia
[2]Universitá di Bologna

Abstract

The explosive development of Internet and the Information Technologies, showed the problem related to the over-exposition information. We live in a world flooded by information, whose coherent management is a difficult task.

The automated verification and repairing of information is crucial. In particular the growing complexity of Web Sites makes it almost impossible to properly manage the information manually. It is a fact that is easier to find an inconsistent Web Site, than a Web Site properly maintained. The design, construction and maintenance of Web Sites are time-consuming tasks which should be managed with an engineering prospective.

This work aims to extend the Web Site verification and repair framework integrated by the system WebVerdi-M in order to detect "extra information" which shouldn't be contained in a Web Site when is not required by the Web Site's description.

The importance of publishing only information needed, is on one side to avoid to disturb the navigation experience of the user in order to not affect their decision to revisit the site, on the other to detect outdated information which derives from previous states of the system.

Moreover, our algorithm is endowed with the capability to learn patterns of undemandedness and recognize them in future uses. For that, a novel *difference* operator which identify in a web site all "extra information" is formalized.

Using the built-in facilities of the system WebVerdi-M, we are able to localize the errors and to identify the type of error generated. This is possible by exploiting the power of regular expression combined together with a rewriting-based technique which is called *partial rewriting* [11].

Finally, we provide a practical implementation which demonstrates that our methodology pay off in practice.

Sommario

Lo sviluppo esplosivo di Internet e delle Tecnologie dell'informazione, hanno evidenziato il problema della over-esposizione informativa. Oggi viviamo in un mondo inondato dalle informazioni, la cui gestione coerente un compito difficile.

La verifica e correzione automatica delle informazioni fondamentale. In particolare la crescente complessitdei siti web ne rende praticamente impossibile la gestione manuale. E' un fatto comune che pi facile trovare un sito web inconsistente, che uno consistente. Il design, la costruzione a la manutenzione di siti web sono compiti lunghi e complessi che dovrebbero essere gestiti ingegneristicamente.

Questa ricerca ha l'obiettivo di estendere il framework costituito dal sistema WebVerdi-M per la verifica e la riparazione di siti web, allo scopo di individuare "informazioni extra" che non dovrebbe essere contenuta in un sito, se non richiesta dalle specifiche del sito stesso.

L'importanza di pubblicare solo le informazioni necessarie determinata, da una parte dalla volontdi fornire la migliore esperienza di navigazione possibile all'utente in modo da aumentare la possibilitdi una sua ulteriore visita, dall'altra dalla allo scopo di individuare informazioni obsolete derivate da stati del sistema precedenti.

Inoltre il nostro algoritmo caratterizzato dalla capacitdi riconoscere pattern di ridondanza informativa e utilizzare questa conoscenza successivamente. A questo scopo un nuovo operatore *difference* che identifica tutte le "informazioni extra" in un sito web stato definito.

Usando le capacitdel sistema WebVerdi-M, quindi possibile la localizzazione degli errori e l'identificazione della tipologia degli stessi. Questo grazie alla potenza espressiva delle espressioni regolari combinata con una tecnica basata sulla riscrittura chiamata*riscrittura parziale* [11].

Infine presentata una implementazione del sistema che dimostra come la metodologia utilizzata ripaga in pratica.

Resumen

El explosivo desarrollo de Internet y la información relacionada con las tecnologías de la información han puesto de manifiesto los problemas de la sobrecarga de información: vivimos en un entorno saturado de información, en el cual la gestión de los datos se estconvirtiendo en una tarea pesada.

En este escenario, la verificación y corrección de la información asume un papel crucial, que no puede ser ignorado. En concreto, la creciente complejidad de los sitios Web ha convertido el problema de la verificación de datos en un verdadero reto. De hecho, es bastante m sencillo encontrar información inconsistente en la Web que un sitio debidamente mantenido en Internet. El dise, la construcción y el mantenimiento de sitios Web son fases que deberían ser tratadas desde una perspectiva ingenieril para poder proporcionar información consistente y fiable.

En este trabajo extendemos el framework de verificación y corrección integrado en el sistema WebVerdi-M para detectar "información extra", que no debería estar en un sitio web si no est requeridas en la propia especificación del sitio.

La importancia de publicar solo la información necesaria, por un lado evitamos molestar la experiencia de navegación del usuario para no afectar en su decisión de volver a visitar el sitio web, y por otro lado, detectar informacion obsoleta derivada de anteriores estados del sistema.

Adem, el algoritmo utilizado tiene la propiedad de aprender patrones de *undemandedness* para un uso posterior. Para esto utilizamos un novedoso operador *difference* que identifica en un sitio web todas la "informaciones extra".

Utilizando la característica del sistema WebVerdi-M, podemos localizar los errores y el tipo de estos. Esta localización es posible, utilizando la potencia expresiva de las expresiones regulares junto con una técnica basada sobre la rescritura que se llama *rescritura parcial* [11].

En fin, presentamos una implementación de nuestra metodología en la cual obtenemos muy buenos resultados.

Contents

1 Introduction **1**

1.1 Cooperation Project 2

1.2 Research Direction 2

1.3 Research Overview 3

2 Websites Verification and Repairing **5**

2.1 Verification and Repairing 6

2.2 Verification and Repairing of Websites 6

2.3 State of Art of Website Verification and Repairing 7

2.3.1 WebMaster: Knowledge-based Verification of Web-pages 8

2.3.2 Centaur and Typol for Semantic Verification of Web Sites Using Natural Semantics 8

2.3.3 xlinkit: A Consistency Checking and SmartLink Generation Service 10

2.3.4 Consistent Document Engineering Tool 10

2.3.5 STRUDEL Web-Site Management 11

2.3.6 Adding Semantix to XML 13

2.4 Comparison Between Systems 14

3 WebVerdi-M **17**

3.1 System Overview 18

3.2 Term Rewriting Systems 18

3.2.1 Description 18

3.2.2 Properties 19

3.3 Preliminaries 19

3.4 Rewriting-based Web Verification 20

3.4.1 Denotation of Web sites 21

3.4.2 Web specification language 22

3.4.3 Simulation and partial rewriting 24

3.4.4 Error diagnoses 25

3.5 Repairing a faulty Web site . . . 27
3.5.1 Fixing correctness errors . . . 29
3.5.2 Fixing completeness errors . . . 34
3.6 Accomplishments . . . 38

4 Undemanded Data Discovering 41
4.1 Undemanded Data Concept . . . 42
4.2 Genesis of the Idea . . . 44
4.3 Integration with WebVerdi-M . . . 45
4.3.1 Reverting Operator . . . 46
4.3.2 Reverting Operator Algorithm . . . 47
4.4 Testing Algorithm . . . 48
4.5 A Complete Example . . . 55

5 Implementation 65
5.1 Web Services . . . 66
5.1.1 Simple Object Access Protocol . . . 67
5.1.2 Java Implementation . . . 68
5.2 Software Structure . . . 68
5.2.1 WebVerdiService . . . 69
5.2.2 XML API . . . 70
5.2.3 Verdi-M . . . 70
5.2.4 WebVerdiClient . . . 72
5.2.5 DB . . . 73
5.3 Maude . . . 73
5.3.1 Functional Programming . . . 73
5.3.2 The Maude Language . . . 74
5.4 API . . . 79
5.4.1 Data Representation . . . 79
5.4.2 Methods . . . 81

6 Case Study 85
6.1 Scenario Description . . . 86
6.1.1 Web Site . . . 86

6.1.2 Preparation 86
6.2 Undemandedness Analysis Outcome 87

7 Conclusions 91
7.1 Achievements 92
7.2 Future Research Directions 93
7.2.1 Automatic HTML Translation to XML 93
7.2.2 Strategies to Optimize Completeness Repairing 93
7.2.3 Repairing Strategies Upon Node Labels 93
7.2.4 Extension of the Specification Language 94
7.2.5 Analysis of the Filtering Language for XML 94

A Example Data 95
A.1 Web Site XML 95

B Old API 101
B.1 Data Representation 101
B.1.1 Web Site 101
B.1.2 Rules 102
B.1.3 Errors 105
B.1.4 Actions 107
B.1.5 PairErrorAction 108
B.2 Methods 108
B.2.1 Methods Details 108

Bibliography 113

1 Introduction

This Chapter is dedicated to an introduction of the cooperation project – Section 1.1, of the topic chosen for it – Section 1.2 and of the research work accomplished – Section 1.3.

1.1 Cooperation Project

This works originates from the cooperation of the Dept. of Electronics, Computer Science and Systems (DEIS) - a research centre of the University of Bologna and the Department of Information Systems and Computation (DSIC) of the Technical University of Valencia (UPV).

The idea of cooperating and establishing a research relation between the group working on Logic Programming in Bologna and the Extensions of Logic Programming group of the Technical University of Valencia came up as the consequence of the shared interest of these research groups in the use of declarative programming technology to address relevant problems of the World Wide Web scenario. Whereas in Bologna logic programming is used to verify the correctness of Web Service interaction, in Valencia functional programming is used to verify the structure and content of Web Sites.

In our willing to find a common field of interest, analyzing the research interests of the two groups, we actually had some new ideas which were not exactly materialized on shared a research topic, but still take advantage of the synergy between the two approaches. The initial idea was to define a form of "abductional reasoning" in functional programming in order to be able to infer informations from Web Sites about the "extra" content which might appear in a Web Site. The idea of a form of "abductive reasoning" is almost unused in the context of functional programming and rewriting system.

1.2 Research Direction

The decision on working on website verification is to be found into the interest and experience of the Spanish research group in the area, which is justified by the increasingly high complexity of the task of building and maintaining complex Web Sites, as we illustrated in Chapter 2.

The analysis of "extra information" in WebSites smoothly integrates with the current research of the group [39], extending the WebVerdi-M framework for verification and repairing of WebSites. The system is able to identify erroneous code as well as suggesting optimal repairing action. Its architecture will be deeply described in Chapter 3.

The proposed methodology is able to identify a third type of information with respect to the "incorrect and incomplete" information identified by WebVerdi-M: the un-

demanded information w.r.t. a specification. This category include all the information not required by the users. The algorithm elaborated to deal with this type of error is able to: detect possible error of undemandedness, interact with the user to resolve them and to add new completeness rules to the original specification to avoid future occurrences of the same errors.

The theory has been concreted into the system prototype WebVerdi-M in the form of a Java web-server and a graphic client which interact with SOAP messages.

1.3 Research Overview

The choice of writing this publication in English is due to the nature of the project: as a cooperation between the Department of Electronics, Computer Science and Systems (DEIS) of the University of Bologna and the Department of Information Systems and Computation of the Technical University of Valencia (UPV), the best and most fair way to coordinate the effort was to write the manuscript in a third language, not to mention the relevance of English as knowledge sharing tool.

The structure is as follows. In this introductory chapter you can find a brief overview of the work written in English, Italian and Spanish respectively. Chapter 1 contains some introductory notes on the nature of the project. Chapter 2 analyzes the context in which this research stands. Chapter 3 presents the proposed technique. Chapter 4 exposes the theory of the project. Chapter 5 describes the implementation of the tool. Chapter 6 considers a case study in which the tool is applied. Chapter 7 outlines the accomplishments and possible future developments. Finally, the appendix includes some information which didn't deserve to be included in the main text, but which is still relevant.

2

Websites Verification and Repairing

The structure of this Chapter is as follows. Section 2.1 formalize the idea of verification and correction of a software system. Section 2.2 addresses its concretion in the Web Site realm. In Section 2.3 we recall the existing tools accordingly with current web technologies and detail their most relevant features in Section 2.4.

2.1 Verification and Repairing

In computer science, both in hardware and software systems, formal verification is the act of proving or disproving the correctness of a system with respect to a given formal specification or property, using formal methods borrowed from mathematics.

The idea is that software testing alone cannot prove that a system does not have a certain defect. Neither can it prove that it does have a certain property. Only the process of formal verification can prove that a system does have or doesn't have a certain property. It is impossible to prove or test that a system has "no defect" since it is impossible to formally specify what "no defect" means. All that can be done is to prove that a system does not have any of the defects that can be thought of, and has all of the properties that together make it functional and useful.

Formal verification can be used for example for systems such as cryptographic protocols, digital circuits or software systems. Software may be expressed as source code or information content, as a Web Site.

The verification of these systems is done by providing a formal proof on an abstract mathematical model of the system, the correspondence between the mathematical model and the nature of the system being otherwise known by construction. Examples of mathematical objects often used to model systems are: finite state machines, labelled transition systems, Petri nets, timed automata, hybrid automata, process algebra, formal semantics of programming languages such as operational semantics or denotational semantics.

2.2 Verification and Repairing of Websites

The increasing complexity of Web sites has turned their design, construction and maintenance into a challenging problem.

The World Wide Web has become a prime vehicle for disseminating information, as a result the number of large Web sites with complex structure that serve information derived from multiple data sources is increasing. Managing the content and the structure of such Web sites presents a data management problem: a site builder without any site creation tools must focus simultaneously on a pages content, its relationship to other pages, and its visual presentation.

Anybody who has used the World Wide Web has experienced the amounts of outdated,

missing and inconsistent information on many Web sites, even on those sites that are of crucial importance to individuals, companies or organisations. Websites are large, frequently updated and constructed by multiple authors. In order to achieve consistency, authors have to spend much time re-reading and revising their own and related documents. Worse, each document publication potentially invalidates consistency. Larger companies define guidelines and policies for writing; but still, a human reviewer is required to enforce them.

As a result, several important tasks, such as updating a site, restructuring a site, or enforcing integrity constraints on a sites structure, are tedious to perform.

Systematic, formal approaches can bring many benefits to Web site construction, giving support for automated Web site verification and repairing. The goal is to construct the tools that are necessary to produce and maintain complex and coherent Web sites. The main objective is to support the designers and the web masters in specifying and verifying semantically Web sites.

Moreover, these tools help to establishing a coherent Information Architecture of a Web site [3]. Information Architecture is the art and science of organizing information to help people effectively fulfill their information needs. A coherent Information Architecture results in a site that communicates a sense of order, unity and ease of use – qualities that Web visitors appreciate, especially if given the information they need.

2.3 State of Art of Website Verification and Repairing

The analysis of the techniques is based on a common framework in order to analyze them with a simple criteria. A general overview of the techniques is followed by a deeper analysis based on four points:

Technologies used: Technologies used to perform the verification;

Internals: How the tool works internally;

Repairing strategies: How detected errors are corrected, if so;

Use case: Main scenario.

The six tools which will be described are: WebMaster [48], Centaur and Typol for Semantic Verification [17], xlinkit [15], CDET [41], STRUDEL [20] and Semantix to XML [38].

2.3.1 WebMaster: Knowledge-based Verification of Web-pages

The aim of WebMaster is to support the maintenance of the contents of Web sites by verifying the contents of sites for outdated, missing and inconsistent information.

Technologies used: The semantic markup needed is domain specific XML defined by the user.

Internals: Using a knowledge-based approach to the verification of Web-page contents, the user exploits semantic markup in Web pages to formulate rules and constraints that must hold on the information in a site. An inference engine subsequently uses these rules to categorise Web-pages in an ontology of pages, while the constraints are used to define categories of pages which contain errors.

Repairing strategies: Repairing is not implemented.

Use case: WebMaster is intended to solve three types of problems that occur in Web sites: outdated information, missing information, and inconsistent (i.e. contradicting) information. A category of problems that is missing from this list concerns incorrect information.

2.3.2 Centaur and Typol for Semantic Verification of Web Sites Using Natural Semantics

These tools allow the specification of the semantics of Web Sites, in order to perform semantic verifications that will help both the conception and the maintenance of Web Sites.

Technologies used: The concept of Natural Semantics, traditionally used to specify the semantics of programming languages, is applied to Web sites specification and verification. This idea comes from the works done in semantics of programming languages drawing a parallel between the syntax of programming languages and the structure of Web sites or semi-structured documents. The advantage of using an

executable semantics for a specification is that tools as validators, compilers or debuggers can be produced. The main distinction between checking a program where all concepts (types, numbers, etc.) are very precisely defined in a mathematical way and checking a Web site where natural languages is manipulated and some representation of knowledge that cannot be manipulated by a theorem prover. The way Natural Semantics is conceived and implemented makes it possible to delegate some parts of the deductive process to external tools.

Internals: An XML validator checking that an XML page complies with a particular DTD is part of what can be generated with Centaur if the static semantics of XML is specified in Typol. In fact markup based languages are good to be specified semantically using Natural Semantics.

Repairing strategies: Repairing is not implemented;

Use case: Two kinds of Web site static specifications can be done:

Specifying a Thematic Directory : A thematic directory is a way of classifying documents found on the Web and presenting them to users in an organised manner. Topics are presented in a hierarchical manner as categories and sub categories. Most of the time, documents indexed in such a directory are selected and classified manually. Such directories must evolve frequently as new documents are very often available and because it is some time helpful to reorganise the hierarchy of categories. After each modification in such a directory one would like to check the integrity of it;

Specifying the Coherence of an Institutional Site : The hypothesises are the following: an authority is in charge of the general Web Site presentation that is the official view of the institution then some subparts of the site are edited by other groups of persons. This form of site is containing a declarative part and an other one than must follow these declarations even when the declarative part is updated.

2.3.3 xlinkit: A Consistency Checking and SmartLink Generation Service

xlinkit is a lightweight application service that provides rule-based link generation and checks the consistency of distributed Web content. xlinkit can be used as part of a consistency management scheme or in applications that require smart link generation, including portal construction and management of large document repositories.

Technologies used: It leverages standard Internet technologies, notably XML, XPath, and XLink.

Internals: xlinkit is given a set of distributed XML resources and a set of potentially distributed rules that relate the content of those resources. The rules express consistency constraints across the resource types. xlinkit returns a set of XLinks that supports navigation between elements of the XML resources.

Repairing strategies: The repair framework [35] which allow to complement the xlinkit tool have as main contribution the semantics that maps xlinkit's first order logic language to a catalogue of repairing actions that can be used to interactively correct rule violations. It does not predict whether a repair action can provoke new errors to appear. Also, it is not possible to detect whether two formulae expressing a requirement for the Web site are incompatible.

Use case: xlinkit is a highly generic technology. It can be applied wherever we want to establish links between Web resources, broadly construed, where those links reflect relationships between resource types. In particular, rather than directly authoring and maintaining links, xlinkit can provide semantically aware link generation. For example, information about important customers can be found in many places in sales files, service agreements, problem reports, logistics, and supply records. xlinkit can be used to build a Web-based customer relationship management system that allows us to navigate between all the pieces of information which reflect the interests of a single customer.

2.3.4 Consistent Document Engineering Tool

The CDET allows to approach flexible consistency management in heterogeneous repositories by explicit formal consistency rules. Formalization provides a common understand-

ing of consistency, which is vital for any collaborative document engineering process.

Technologies used: Explicit formal consistency rules for heterogeneous repositories that are managed by traditional Document Management Systems are used. Rules are formalized in a variant of first-order temporal logic. Functions and predicates, implemented in a full programming language, provide complex (even higher-order) functionality. A static type system supports rule formalization, where types also define (formal) document models. The concrete rule syntax is XML-based.

Internals: A rule designer defines domain specific rules in a full first-order temporal logic with linear time. In contrast to decidable subsets, full temporal first-order logic provides an expressive rule designer tool set and supports formalizing practically relevant consistency rules. Higher-order complexity is hidden from the rule designer; it is encapsulated in functions and predicates defined by the language designer. Automatically generated consistency reports precisely pinpoint inconsistencies within documents w.r.t. the rules defined.

Repairing strategies: Repair actions will be generated from enriched consistency reports. In [43, 42] an extension for the CDET is possible to remove inconsistencies from sets of interrelated documents, which first generates direct acyclic graphs representing the relations between documents and then repairs are directly derived from such direct acyclic graphs. In this case, temporal rules are supported and interference and compatibility of repairs are not completely neglected. This compatibility is expensive to check for temporal rules.

Use case: Cases in which consistency shall be improved in document engineering processes.

2.3.5 STRUDEL Web-Site Management

The Web Site construction and management problem is addressed from a database perspective. STRUDEL supports declarative specification of a Web sites content and structure and automatically generates a browsable Web site from a specification. STRUDELs key idea is separating the management of a Web sites data, the management of the sites structure, and the visual presentation of the sites pages.

Technologies used: STRUDEL is based on a semistructured data model of labeled, directed graphs. This model is introduced to manage semistructured data, which is characterized as having few type constraints, irregular structure, and rapidly evolving or missing schema. Semistructured data facilitates integration of data from multiple, non-traditional sources.

Internals: Using STRUDEL the site builder first creates an integrated view of the data that will be available at the site. The Web sites raw data resides either in external sources (e.g., databases, structured files) or in STRUDELs internal data repository. In STRUDELs mediator component, as in all of its other components, all external or internal data is modeled as a labeled directed graph, which is the model commonly used for semistructured data. A set of source-specific wrappers translates the external representation into the graph model. The integrated view of the data is called the data graph. Second, the site builder declaratively specifies the Web sites structure using a site-definition query in STRUQL, STRUDELs query language. The result of evaluating the site-definition query on the data graph is a site graph, which models both the sites content and structure. Third, the builder specifies the visual presentation of pages in STRUDELs HTML-template language. The HTML generator produces HTML text for every node in the site graph from a corresponding HTML template; the result is the browsable Web site.

Repairing strategies: Repairing is not implemented;

Use case: The following example shows how one authors homepage site is generated by STRUDEL. The main data source is the authors Bibtex bibliography file. The homepage site has four types of pages: the root page containing general information, a page containing all paper abstracts, and pages containing summaries of papers published in a particular year or category. The first two steps of the site-definition process are: creating the data graph from a Bibtex file and defining the site graph. The Bibtex wrapper converts Bibtex files into a Strudel data graph. It is important to note here that data in STRUDELs graph data model does not have a fixed schema, which facilitates integration of data from multiple sources. The homepages site graph can be defined as follows: firstly there are two objects called RootPage and AbstractsPage and a link between them; secondly there are two other

objects, AbstractPage(x) and PaperPresentation(x) for each member x of the Publications collection. These objects contain the publications information that will appear in different parts of the site. The general abstracts page is linked to the abstract page of each publication (Abstract-Page(x)). A page can be than created for each year associated with a publication; with a link clause each PaperPresentation object can be associated with its corresponding YearPage. Lastly, the root page is linked to each year page. A similar clause creates a page for each publication category and links category pages to PaperPresentation objects. The site-definition query only specifies what information will be available in the site and what relationships exist between pages in the site; it does not specify anything about the sites visual presentation. The result of applying a site-definition query to a data graph is another graph in Strudels data model; this permits to create site by composing multiple queries.

2.3.6 Adding Semantix to XML

Starting from the analogy between a document tagged by a mark up language (XML or SGML) and a source string generated by a BNF grammar, mark up specifications can be seen as meta grammars, so enriched with semantic attributes and functions. A way for defining the semantics of a document by a XML compliant specification an approach that is conceptually homogeneous and allows to exploit existing XML parser generators for analysing semantics and for generating attribute evaluators. The system architecture of a XML based syntax directed compiler compiler is analogous to a classical translator writing system.

Technologies used: The idea is to denote the semantic rules as an XML document which complies with an ad hoc DTD. The advantage of this solution is that the XML meta level does not have to be extended thus the semantic rules definition still complies with the XML standard.

Internals: The architecture is based on data flow diagrams in Figure 2.1.

Repairing strategies: Repairing is not implemented;

Use case: The applicative scenario is characterized by two main factors: XML will be mostly used as a document exchange format over Internet and XML can be used for

Figure 2.1: Data Flow Architecture of the Semantically Enriched XML Processing

any kind of information and document. The classical use is represented by textual documents, but XML can be successfully used for other issues such as defining the schema of an object oriented database and the objects stored in it.

2.4 Comparison Between Systems

To ease the comparison between the different system, they have been classified in three categories. Opposite categories have been identified to give a simple and precise view of the approaches used. In some cases, it might be a little forced to classify a system into a category, therefore in those circumstances is included in several (two) classes. Here comes the taxonomy:

Generation Oriented VS Repairing Oriented : While some tools focus on the initial generation of the Web Site, some other have the maintenance as primary objective. In the first class of systems we have: Centaur and Typol for Semantic Verification, xlinkit and Strudel. As Maintenance Oriented tools can be classified WebMaster, Centaur and Typol for Semantic Verification, the repairing framework based on xlinkit, CDET and its extension and Semantix to XML.

Data Driven VS Semantix Driven : For some tool is very important the classification

and storage of the data as the basis for further elaboration. Others instead focus more on the semantic relation between elements. To the first group of tools contains: CDET and Strudel. As Semantics Driven tools can be classified WebMaster, Centaur and Typol for Semantic Verification, xlinkit and Semantix to XML.

Web Focused VS General Data Focused : Some tools are explicitly devised to address Web Site consistence, therefore include link checking. Others tools have been created with a more general use in mind and then adapted for web page checking. In the first class of systems we consider: WebMaster, Centaur and Typol for Semantic Verification, xlinkit, Strudel. As General Data Focused tools can be classified CDET and Semantix to XML.

3

WebVerdi-M

This Chapter aims to give some insights of the WebVerdi-M framework, which is the starting brick of with this work. Section 3.1 gives an overview of the tool. Section 3.2 reminds the essentials of the Term Rewriting Systems. Section 3.3 introduces some technical concepts needed to understand the system. Section 3.4 explains the idea of partial rewriting. Section 3.5 illustrates the repairing algorithm and Section 3.6 summarizes our results and describes future work directions.

3.1 System Overview

WebVerdi-M is a methodology for semi-automatically repairing faulty Web sites integrated with a verification technique [6]. Starting from a categorization of the kinds of errors that can be found during the Web verification activities, a stepwise transformation procedure that achieves correctness and completeness of the Web site w.r.t. its formal specification while respecting the structure of the document (e.g. the schema of an XML document) is formulated. An interesting aspect of the approach is the ability to detect whether two rules expressing a requirement for the Web site are incompatible.

In previous work on GVERDI [4, 5], a rewriting-like approach to Web site specification and verification has been presented. This methodology allows to specify the integrity conditions for the Web sites and then diagnose errors by computing the requirements not fulfilled by a given Web site, that is, by finding out incorrect/forbidden patterns and missing/incomplete Web pages.

This technique has been complemented with a tool-independent technique for semi-automatically repairing the errors found during that verification phase. First, the kinds of errors that can be found in a Web site w.r.t. a Web Specification have been formalized. Then, the *repair actions* than can be performed to repair each kind of error have been classified. Since different repair actions can be executed to repair a given error, this methodology is tuned to deliver a set of correct and complete repair actions to choose between. The repair methodology is formulated in two phases. First, all the necessary actions to make the Web site *correct* are performed. Once correctness of the Web site has been achieved, the user is given the option to execute all the necessary actions to make it *complete*. Moreover, this methodology allows to detect any eventual incompatibility among the rules of the Web specification as well as to manage the problems that might arise from the interaction of repair actions.

3.2 Term Rewriting Systems

3.2.1 Description

In mathematics, computer science and logic, rewriting covers a wide range of potentially non-deterministic methods of replacing sub-terms of a formula with other terms. What is considered rewriting systems or term rewriting systems consists in its most basic form, of

a set of terms, plus relations on how to transform these terms.

Term rewriting can be non-deterministic. One rule to rewrite a term could be applied in many different ways to that term. Rewriting systems then do not provide an algorithm for changing one term to another, but a set of possible substitutions that could be applied. When combined with an appropriate algorithm, however, rewrite systems can be viewed as computer programs, and several declarative programming languages are based on term rewriting.

3.2.2 Properties

Terms which cannot be written any further are called normal forms. The potential existence or uniqueness of normal forms can be used to classify and describe certain rewriting systems. There are rewriting systems which do not have normal forms.

The property where terms can be rewritten regardless of the choice of rewriting rule to obtain the same normal form is known as confluence. The property of confluence is linked with the property of having unique normal forms.

3.3 Preliminaries

A finite set of symbols is called *alphabet*. Given the alphabet A, A^* denotes the set of all finite sequences of elements over A. Syntactic equality between objects is represented by $\equiv$.

By $\mathcal{V}$ is denoted a countably infinite set of variables and Σ denotes a set of function symbols, or *signature*. Varyadic signatures as in [16] are considered (i.e., signatures in which symbols have an unbounded arity, that is, they may be followed by an arbitrary number of arguments). $\tau(\Sigma, \mathcal{V})$ and $\tau(\Sigma)$ denote the *non-ground term algebra* and the *term algebra* built on $\Sigma \cup \mathcal{V}$ and Σ. Terms are viewed as labelled trees in the usual way. Positions are represented by sequences of natural numbers denoting an access path in a term. The empty sequence Λ denotes the root position. By notation $w_1.w_2$, is denoted the concatenation of position w_1 and position w_2. Positions are ordered by the prefix ordering, that is, given the positions w_1, w_2, $w_1 \leq w_2$ if there exists a position x such that $w_1.x = w_2$. Given $S \subseteq \Sigma \cup \mathcal{V}$, $O_S(t)$ denotes the set of positions of a term t which are rooted by symbols in S. Moreover, for any position x, $\{x\}.O_S(t) = \{x.w \mid w \in O_S(t)\}$. $t_{|u}$ is the sub-term at the position u of t. $t[r]_u$ is the term t with the sub-term rooted at the

position u replaced by r. Given a term t, we say that t is *ground*, if no variables occur in t.

A *substitution* $\sigma \equiv \{X_1/t_1, X_2/t_2, \ldots\}$ is a mapping from the set of variables $\mathcal{V}$ into the set of terms $\tau(\Sigma, \mathcal{V})$ satisfying the following conditions: (i) $X_i \neq X_j$, whenever $i \neq j$, (ii) $X_i\sigma = t_i$, $i = 1, ..n$, and (iii) $X\sigma = X$, for any $X \in \mathcal{V} \setminus \{X_1, \ldots, X_n\}$. By $Var(s)$ we denote the set of variables occurring in the syntactic object s.

Term rewriting systems provide an adequate computational model for functional languages. In the sequel, is followed the standard framework of term rewriting (see [10, 29]). A *Term Rewriting System* is a pair (Σ, R), where Σ is a signature and R is a finite set of reduction (or rewrite) rules of the form $\lambda \to \rho$, λ, $\rho \in \tau(\Sigma, \mathcal{V})$, $\lambda \notin \mathcal{V}$ and $Var(\rho) \subseteq Var(\lambda)$. Often just R will be used instead of (Σ, R). Sometimes, the signature of a Term Rewriting System (Σ, R) will be denoted by Σ_R.

A rewrite step is the application of a rewrite rule to an expression. A term s *rewrites* to a term t via $r \in R$, $s \to_r t$ (or $s \to_R t$), if there exist a position $u \in O_\Sigma(s)$, $r \equiv \lambda \to \rho$, and a substitution σ such that $s_{|u} \equiv \lambda\sigma$ and $t \equiv s[\rho\sigma]_u$. When no confusion can arise, any subscript (i.e. $s \to t$) will be omitted. A term s is a *irreducible form* (or *normal form*) w.r.t. R, if there is no term t s.t. $s \to_R t$. t is the irreducible form of s w.r.t. R (in symbols $s \to^!_R t$) if $s \to^*_R t$ and t is irreducible.

A Term Rewriting System R is *terminating*, if there exists no infinite rewrite sequence $t_1 \to_R t_2 \to_R \ldots$ A Term Rewriting System R is *confluent* if, for all terms s, t_1, t_2, such that $s \to^*_R t_1$ and $s \to^*_R t_2$, there exists a term t s.t. $t_1 \to^*_R t$ and $t_2 \to^*_R t$. When R is terminating and confluent, it is called *canonical*. In canonical Term Rewriting Systems, each input term t can be univocally reduced to a unique *irreducible form*.

Let $s = t$ be an equation, the equation $s = t$ *holds* in a canonical Term Rewriting System R, if there exists an irreducible form $z \in \tau(\Sigma, \mathcal{V})$ w.r.t. R such that $s \to^!_R z$ and $t \to^!_R z$.

3.4 Rewriting-based Web Verification

In this section, the formal verification methodology proposed in [4], which is able to detect forbidden/erroneous as well as missing information in a Web site is briefly recalled. By executing a Web specification on a given Web site, possible discrepancies between the Web site and the properties stated in the Web specification are recognize and exactly

```
(1) members(member(name(mario),surname(rossi),status(professor)),
                   member(name(franca),surname(bianchi),status(technician)),
                   member(name(giulio),surname(verdi),status(student)),
                   member(name(ugo),surname(blu),status(professor))),
(2) hpage(fullname(mariorossi),phone(3333),status(professor),
              hobbies(hobby(reading),hobby(gardening))),
(3) hpage(fullname(francabianchi),status(technician),phone(5555),
              links(link(url(www.google.com),urlname(google)),
              link(url(www.sexycalculus.com),urlname(FormalMethods))),
(4) hpage(fullname(annagialli),status(professor),phone(4444),
              teaching(course(algebra))),
(5) pubs(pub(name(ugo),surname(blu),title(blah1),blink(year(2003))),
          pub(name(anna),surname(gialli),title(blah2),year(2002))),
(6) projects(project(pname(A1),grant1(1000),grant2(200),
                        total(1100),coordinator(fullname(mariorossi))),
                project(pname(B1),grant1(2000),grant2(1000),
                        projectleader(surname(gialli),name(anna)),
                            total(3000)))}
```

Figure 3.1: An example of a Web site for a research group

located.

In this framework, a *Web page* is either an XML [50] or an XHTML [52] document well-formed. XHTML/XML documents can be encoded as Herbrand terms.

3.4.1 Denotation of Web sites

Consider two alphabets T and $\mathcal{T}ag$. The set T^* will be denoted by $\mathcal{T}ext$. An object $\mathtt{t} \in \mathcal{T}ag$ is called *tag* element, while an element $\mathtt{w} \in \mathcal{T}ext$ is called *text* element. Since Web pages are provided with a tree-like structure, they can be straightforwardly translated into ordinary terms of a given term algebra $\tau(\mathcal{T}ext \cup \mathcal{T}ag)$. Note that XML/XHTML tag attributes can be considered as common tagged elements, and hence translated in the same way. A *Web site* is a finite collection of ground terms $\{\mathtt{p}_1 \dots \mathtt{p}_n\}$. In Figure 3.1, a Web site W of a research group is presented, which contains information about group members affiliation, scientific publications, research projects, teaching and personal data.

In the following, terms of the non-ground term algebra $\tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$, which may contain variables will be considered. An element $\mathtt{s} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$ is called *Web page template*. In this methodology, Web page templates are used for specifying properties on Web sites as described in the following section.

3.4.2 Web specification language

A Web specification is a triple (I_N, I_M, R), where R, I_N, and I_M are finite set of rules. The set R contains the definition of some auxiliary functions which the user would like to provide in order to ease some common operations, such as string processing, arithmetic, boolean operators, etc. It is formalized as a term rewriting system, which is handled by standard rewriting [29].

The set I_N describes constraints for detecting erroneous Web pages (*correctness rules*). A correctness rule has the following form: $\mathtt{l} \rightharpoonup \mathtt{error} \mid \mathtt{C}$, with $\mathtt{Var(C)} \subseteq \mathtt{Var(l)}$, where l is a term, error is a reserved constant, and C is a (possibly empty) finite sequence containing membership tests (e.g. $\mathtt{X} \in \mathtt{rexp}$) w.r.t. a given regular language[1], and/or equations over terms. For the sake of expressiveness, is allowed to write inequalities of the form $\mathtt{s} \neq \mathtt{t}$ in C. Such inequalities are just syntactic sugar for $(\mathtt{s} = \mathtt{t}) = \mathtt{false}$. When C is empty, $\mathtt{l} \rightharpoonup \mathtt{error}$ is used. The meaning of a correctness rule $\mathtt{l} \rightharpoonup \mathtt{error} \mid \mathtt{C}$, where $\mathtt{C} \equiv (\mathtt{X_1\ in\ rexp_1}, \ldots, \mathtt{X_n\ in\ rexp_n}, \mathtt{s_1} = \mathtt{t_1} \ldots \mathtt{s_m} = \mathtt{t_m})$, is the following. C *holds* for substitution σ, if (i) each structured text $\mathtt{X_i}\sigma$, $\mathtt{i} = 1, \ldots, \mathtt{n}$, is contained in the language of the corresponding regular expression $\mathtt{rexp_i}$; (ii) each instantiated equation $(\mathtt{s_i} = \mathtt{t_i})\sigma$, $\mathtt{i} = 1, \ldots, \mathtt{m}$, holds in R.

The Web page p is considered incorrect if an instance $\mathtt{l}\sigma$ of l is *recognized* within p, and C holds for σ.

The third set of rules I_M specifies some properties for detecting incomplete/missing Web pages (*completeness rules*). A completeness rule is defined as $\mathtt{l} \rightharpoonup \mathtt{r}\ \langle \mathtt{q} \rangle$, where l and r are terms and $\mathtt{q} \in \{\mathtt{E}, \mathtt{A}\}$. Completeness rules of a Web specification formalize the requirement that some information must be included in all or some pages of the Web site. The attributes $\langle \mathtt{A} \rangle$ and $\langle \mathtt{E} \rangle$ are used to distinguish "universal" from "existential" rules. Right-hand sides of completeness rules can contain functions, which are defined in R. Intuitively, the interpretation of a universal rule $\mathtt{l} \rightharpoonup \mathtt{r}\ \langle \mathtt{A} \rangle$ (respectively, an existential rule $\mathtt{l} \rightharpoonup \mathtt{r}\ \langle \mathtt{E} \rangle$) w.r.t. a Web site W is as follows: if (an instance of) l is recognized in W, also (an instance of) the irreducible form of r must be recognized in *all* (respectively, *some*) of the Web pages which embed (an instance of) r.

Sometimes, may be interesting to check a given completeness property only on a subset of the whole Web site. For this purpose, some symbols in the right-hand sides of

[1] Regular languages are represented by means of the usual Unix-like regular expressions syntax.

the rules are marked by means of the constant symbol $\sharp$. Marking information of a given rule r is used to select the subset of the Web site in which the condition formalized by r should be checked. More specifically, rule r is executed on all and only the Web pages embedding the marking information. A detailed example follows.

Example 3.4.1. *Consider the Web specification which consists of the following completeness and correctness rules along with a term rewriting system defining the string concatenation function* $++$*, the arithmetic operators* $+$ *and* $*$ *on natural numbers and the relational operator* $\leq$*.*

$\mathtt{member(name(X),surname(Y))} \rightharpoonup \sharp\mathtt{hpage(fullname(X ++ Y),status)}\ \langle \mathtt{E} \rangle$
$\mathtt{hpage(status(professor))} \rightharpoonup \sharp\mathtt{hpage(}\sharp\mathtt{status(}\sharp\mathtt{professor),teaching)}\ \langle \mathtt{A} \rangle$
$\mathtt{pubs(pub(name(X),surname(Y)))} \rightharpoonup \sharp\mathtt{members(member(name(X),surname(Y)))}\ \langle \mathtt{E} \rangle$
$\mathtt{courselink(url(X),urlname(Y))} \rightharpoonup \sharp\mathtt{cpage(title(Y))}\ \langle \mathtt{E} \rangle$
$\mathtt{hpage(X)} \rightharpoonup \mathtt{error} \mid \mathtt{X\ in}$ `[:TextTag:]* sex [:TextTag:]*`
$\mathtt{blink(X)} \rightharpoonup \mathtt{error}$
$\mathtt{project(grant1(X),grant2(Y),total(Z))} \rightharpoonup \mathtt{error} \mid \mathtt{X} + \mathtt{Y} \neq \mathtt{Z}$
$\mathtt{project(grant1(X),grant2(Y))} \rightharpoonup \mathtt{error} \mid \mathtt{X} \neq \mathtt{Y} * 2$
$\mathtt{total(Z)} \rightharpoonup \mathtt{error} \mid \mathtt{Z} \geq 500000 = \mathtt{true}$

This Web specification models some required properties for the Web site of Figure 3.1. First rule formalizes the following property: if there is a Web page containing a member list, then for each member, a home page should exist which contains (at least) the full name and the status of this member. The full name is computed by concatenating the name and the surname strings by means of the ++ function. The marking information establishes that the property must be checked only on home pages (i.e., pages containing the tag "hpage"). Second rule states that, whenever a home page of a professor is recognized, that page must also include some teaching information. The rule is universal, since it must hold for each professor home page. Such home pages are selected by exploiting the marks which identify professor home pages. Third rule specifies that, whenever there exists a Web page containing information about scientific publications, each author of a publication should be a member of the research group. In this case, the check refers only to the property in the Web page containing the group member list. The fourth rule formalizes that, for each link to a course, a page describing that course must exist. The fifth rule forbids sexual contents from being published in the home pages of the group members. This is enforced by requiring that the word `sex` *does not occur in any home page by using the regular expression* `[:TextTag:]* sex [:TextTag:]*`*, which identifies the regular language of all the strings built over* $(\mathcal{T}ext \cup \mathcal{T}ag)$ *containing word* `sex`*. The sixth rule is provided with the aim of improving accessibility for people with*

disabilities. It simply states that blinking text is forbidden in the whole Web site. The last three rules respectively state that, for each research project, the total project budget must be equal to the sum of the grants, the first grant should be the double of the second one, and the total budget is less than 500000 euros.

In this methodology, diagnoses are carried out by running Web specifications on Web sites. This is mechanized by means of *partial rewriting*, a rewriting technique which is obtained by replacing the traditional pattern-matching of term rewriting with a new mechanism based on *page* (tree) *simulation* (cf. [4]).

3.4.3 Simulation and partial rewriting

Partial rewriting extracts "some pieces of information" from a page, pieces them together and then rewrites the glued term. The assembling is done by means of tree simulation,which recognizes the structure and the labeling of a given term (Web page template) inside a particular page of the Web site.

Simulations have been used in a number of works dealing with querying, transformation and similarity checks of semistructured data (cf. [1, 21, 12]). To keep the framework simple, a semantic change/load for labels haven't been considered; this in fact would have required to introduce ontologies, which are outside the scope of the work.

The notion of simulation, $\trianglelefteq$, is an adaptation of Kruskal's *embedding* (or "syntactically simpler") relation [13] where we ignore the usual *diving* rule[2] [30].

Definition 3.4.1 (simulation). *The* simulation *relation*

$$\trianglelefteq \subseteq \tau(\mathcal{T}ext \cup \mathcal{T}ag) \times \tau(\mathcal{T}ext \cup \mathcal{T}ag)$$

on Web pages is the least relation satisfying the rule:

> $\mathtt{f}(\mathtt{t}_1, \ldots, \mathtt{t}_\mathtt{m}) \trianglelefteq \mathtt{g}(\mathtt{s}_1, \ldots \mathtt{s}_\mathtt{n})$ *iff* $\mathtt{f} \equiv \mathtt{g}$ *and* $\mathtt{t}_\mathtt{i} \trianglelefteq \mathtt{s}_{\pi(\mathtt{i})}$, *for* $\mathtt{i} = 1, \ldots \mathtt{m}$, *and some injective function* $\pi : \{1, \ldots \mathtt{m}\} \rightarrow \{1, \ldots, \mathtt{n}\}$.

Given two Web pages $\mathtt{s}_1$ and $\mathtt{s}_2$, if $\mathtt{s}_1 \trianglelefteq \mathtt{s}_2$ then $\mathtt{s}_1$ *simulates* (or *is embedded* or *recognized* into) $\mathtt{s}_2$. Also $\mathtt{s}_2$ *embeds* $\mathtt{s}_1$. Note that, in Definition 3.4.1, for the case when

[2] The diving rule allows one to "strike out" a part of the term at the right-hand side of the relation $\trianglelefteq$. Formally, $s \trianglelefteq f(t_1, \ldots t_n)$, if $s \trianglelefteq t_i$, for some i.

m is 0 we have $\mathtt{c} \trianglelefteq \mathtt{c}$ for each constant symbol c. Also note that $\mathtt{s}_1 \not\trianglelefteq \mathtt{s}_2$ if either $\mathtt{s}_1$ or $\mathtt{s}_2$ contain variables. This definition by can be illustrated means of a rather intuitive example.

Example 3.4.2. *Consider the following Web pages (called* $\mathtt{p}_1$ *and* $\mathtt{p}_2$*, respectively):*

```
hpage(surname,status(professor),name,teaching)
hpage(name(mario),surname(rossi),status(professor),
      teaching(course(logic1),course(logic2)),
      hobbies(hobby(reading),hobby(gardening)))
```

The structure of $\mathtt{p}_1$ *can be recognized inside the structure of* $\mathtt{p}_2$*, hence* $\mathtt{p}_1 \trianglelefteq \mathtt{p}_2$*, while* $\mathtt{p}_2 \not\trianglelefteq \mathtt{p}_1$.

Now is possible to introduce the *partial rewrite* relation between Web page templates. W.l.o.g., are disregarded conditions and/or quantifiers from the Web specification rules. Roughly speaking, given a Web specification rule $\mathtt{l} \rightharpoonup \mathtt{r}$, partial rewriting allows to extract from, a given Web page s, a subpart of s which is simulated by a ground instance of l, and to replace s by a reduced, ground instance of r. Let $\mathtt{s}, \mathtt{t} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$. Then, s *partially rewrites* to t via rule $\mathtt{l} \rightharpoonup \mathtt{r}$ and substitution σ iff there exists a position $u \in O_{\mathcal{T}ag}(\mathtt{s})$ such that (i) $\mathtt{l}\sigma \trianglelefteq \mathtt{s}_{|u}$, and (ii) $\mathtt{t} = \mathit{Reduce}(\mathtt{r}\sigma, R)$, where function $\mathit{Reduce}(x, R)$ computes, by standard term rewriting, the irreducible form of x in R. Note that the context of the selected reducible expression $\mathtt{s}_{|u}$ is disregarded after each rewrite step. By notation $\mathtt{s} \rightharpoonup_I \mathtt{t}$, is denoted that s is partially rewritten to t using some rule belonging to the set I.

3.4.4 Error diagnoses

In order to diagnose correctness as well as completeness errors, the method presented in [4] is followed. The kind of errors which can be found in a Web site are classified in terms of the different outputs delivered by the verification technique when is fed with a Web site specification. In Section 3.5 this information will be exploited to develop the repairing/correction methodology. Lets start by characterizing correctness errors.

Definition 3.4.2 (correctness error). *Let* W *be a Web site and* (I_M, I_N, R) *be a Web specification. Then, the quadruple* $(\mathtt{p}, w, \mathtt{l}, \sigma)$ *is a* correctness error evidence *iff* $\mathtt{p} \in \mathtt{W}$*,* $w \in O_{\mathcal{T}ag}(\mathtt{p})$*, and* $\mathtt{l}\sigma$ *is an instance of the left-hand side* l *of a correctness rule belonging to* I_N *such that* $\mathtt{l}\sigma \trianglelefteq \mathtt{p}_{|w}$.

Given a correctness error evidence $(\mathtt{p}, w, \mathtt{l}, \sigma)$, $\mathtt{l}\sigma$ represents the erroneous information which is embedded in a sub-term of the Web page $\mathtt{p}$, namely $\mathtt{p}_{|w}$. Therefore, by analyzing such an evidence, is exactly know where the faulty information is located inside the Web page.

The set of all correctness error evidences of a Web site $\mathtt{W}$ w.r.t. a set of correctness rules I_N is denoted by $E_N(\mathtt{W})$. When no confusion can arise, just E_N is used.

As for completeness errors three classes of errors can be distinguished. Given a Web site $\mathtt{W}$, there are:

Missing Web pages. The Web site $\mathtt{W}$ lacks one or more Web pages;

Universal completeness error. The Web site $\mathtt{W}$ fails to fulfil the requirement that a piece of information must occur in *all* the Web pages of a given subset of $\mathtt{W}$.

Existential completeness error. The Web site $\mathtt{W}$ fails to fulfil the requirement that a piece of information must occur in *some* Web page of $\mathtt{W}$.

In the framework, all the three kinds of completeness errors can be detected by partially rewriting Web pages to some expression $\mathtt{r}$ by means of the rules of I_M and then checking whether $\mathtt{r}$ does not occur in a suitable subset of the Web site.

Definition 3.4.3 (Missing Web page). *Let* $\mathtt{W}$ *be a Web site and* (I_M, I_N, R) *be a Web specification. Then the pair* $(\mathtt{r}, \mathtt{W})$ *is a* missing *Web page error evidence if there exists* $\mathtt{p} \in W$ *s.t.* $\mathtt{p} \rightharpoonup^{+}_{I_M} \mathtt{r}$ *and* $\mathtt{r} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ *does not belong to* $\mathtt{W}$.

When a missing Web page error is detected, the evidence $(\mathtt{r}, \mathtt{W})$ signals the expression $\mathtt{r}$ that does not appear in the whole Web site $\mathtt{W}$. In order to formalize existential as well as universal completeness errors, the following auxiliary definition is introduced.

Definition 3.4.4. *Let* $\mathtt{P}$ *be a set of terms in* $\tau(\mathcal{T}ext \cup \mathcal{T}ag)$ *and* $\mathtt{r} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$. $\mathtt{P}$ *is* universally *(resp.* existentially*)* complete *w.r.t.* $\mathtt{r}$ *iff for each* $\mathtt{p} \in \mathtt{P}$ *(resp. for some* $\mathtt{p} \in \mathtt{P}$*), there exists* $w \in O_{\mathcal{T}ag}(\mathtt{p})$ *s.t.* $\mathtt{r} \trianglelefteq \mathtt{p}_{|w}$.

Definition 3.4.5 (Universal completeness error). *Let* $\mathtt{W}$ *be a Web site and* (I_M, I_N, R) *be a Web specification. Then the triple* $(\mathtt{r}, \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}, \mathtt{A})$ *is a* universal completeness error evidence*, if there exists* $\mathtt{p} \in \mathtt{W}$ *s.t.* $\mathtt{p} \rightharpoonup^{+}_{I_M} \mathtt{r}$ *and* $\{\mathtt{p}_1, \ldots, \mathtt{p}_n\}$ *is not universally complete w.r.t.* $\mathtt{r}$*,* $\mathtt{p}_i \in \mathtt{W}$*,* $i = 1, \ldots, n$.

Definition 3.4.6 (Existential completeness error). *Let* $\mathtt{W}$ *be a Web site and* (I_M, I_N, R) *be a Web specification. Then the triple* $(\mathtt{r}, \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}, \mathtt{E})$ *is an* existential completeness error evidence*, if there exists* $\mathtt{p} \in \mathtt{W}$ *s.t.* $\mathtt{p} \rightharpoonup^{+}_{I_M} \mathtt{r}$ *and* $\{\mathtt{p}_1, \ldots, \mathtt{p}_n\}$ *is not existentially complete w.r.t.* $\mathtt{r}$*,* $\mathtt{p}_i \in \mathtt{W}$*,* $i = 1, \ldots, n$*.*

Note that Definition 3.4.5 as well as Definition 3.4.6 precisely locate where completeness errors occur, since they provide the set of faulty Web pages, where the required information should be contained.

With $E_M(\mathtt{W})$ is denoted the set containing all the completeness error evidences w.r.t. I_M for a Web site $\mathtt{W}$ (missing Web pages as well as universal/existential completeness errors evidences). When no confusion can arise, just E_M is used.

The verification methodology of [4] generates the sets of correctness and completeness error evidences E_N and E_M mentioned above for a given Web site w.r.t. the input Web specification.

Definition 3.4.7 (Web site correctness). *Given a Web specification* (I_M, I_N, R)*, a Web site* $\mathtt{W}$ *is* correct *w.r.t.* (I_M, I_N, R) *iff the set* E_N *of correctness error evidences w.r.t.* I_N *is empty.*

Definition 3.4.8 (Web site completeness). *Given a Web specification* (I_M, I_N, R)*, a Web site* $\mathtt{W}$ *is* complete *w.r.t.* (I_M, I_N, R) *iff the set* E_M *of completeness error evidences w.r.t.* I_M *is empty.*

3.5 Repairing a faulty Web site

Given a faulty Web site $\mathtt{W}$ and the sets of errors $\mathtt{E}_N$ and $\mathtt{E}_M$, the goal is to modify the given Web site by adding, changing, and removing information in order to produce a Web site that is correct and complete w.r.t. the considered Web specification. For this purpose, in correspondence with the error categories a catalogue of *repair actions* which can be applied to the faulty Web site is introduced. Therefore fixing a Web site consists in selecting a set of suitable repair actions that are automatically generated, and executing them in order to remove inconsistencies and wrong data from the Web site.

The primitive repair actions considered are the following.

change($\mathtt{p}$**,w,**$\mathtt{t}$**)** replaces the sub-term $\mathtt{p}|_w$ in $\mathtt{p}$ with the term $\mathtt{t}$ and returns the modified Web page.

insert(p,**w**,t) modifies the term p by adding the term t into $p_{|w}$ and returns the modified Web page.

add(p,W) adds the Web page p to the Web site W and returns the Web page p.

delete(p,**t**) deletes all the occurrences of the term t in the Web page p and returns the modified Web page.

Note that it is possible that a particular error could be repaired by means of different actions. For instance, a correctness error can be fixed by deleting the incorrect/forbidden information, or by changing the data which rise that error. Similarly, a completeness error can be fixed by either 1) inserting the missing information, or 2) deleting all the data in the Web site that caused that error.

Note that modifying or inserting arbitrary information may cause the appearance of new correctness errors. In order to avoid this, the data considered for insertion should be *safe* w.r.t. the Web specification, i.e. they cannot fire any correctness rule. For this purpose, the following definition is introduced.

Definition 3.5.1. *Let* (I_M, I_N, R) *be a Web specification and* $\mathtt{p} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ *be a Web page. Then,* p *is* safe *w.r.t.* I_N*, iff for each* $w \in O_{\mathcal{T}ag}(\mathtt{p})$ *and* $(\mathtt{l} \rightharpoonup \mathtt{r} \mid \mathtt{C}) \in I_N$*, either*

- *there is no* σ *s.t.* $\mathtt{l}\sigma \trianglelefteq \mathtt{p}_{|w}$ *or*

- $\mathtt{l}\sigma \trianglelefteq \mathtt{p}_{|w}$*, but* $\mathtt{C}\sigma$ *does not hold.*

Example 3.5.1. *Let* p *be the Web page* (6) *(which is called* p *in this example) of the Web site given in Figure 3.1, Consider the Web specification* W *in Example 3.4.1 and the following terms:*

$$\begin{aligned} t_1 \equiv \mathtt{project}(&\mathtt{pname}(\mathtt{A1}), \mathtt{grant1}(\mathtt{1000}), \mathtt{grant2}(\mathtt{200}),\\ &\mathtt{coordinator}(\mathtt{fullname}(\mathtt{mariorossi})),\\ &\mathtt{total}(\mathtt{1200}))) \\ t_2 \equiv \mathtt{project}(&\mathtt{pname}(\mathtt{A1}), \mathtt{grant1}(\mathtt{1000}), \mathtt{grant2}(\mathtt{500}),\\ &\mathtt{coordinator}(\mathtt{fullname}(\mathtt{mariorossi})),\\ &\mathtt{total}(\mathtt{1500}))) \end{aligned}$$

Now consider the following Web pages which are obtained by executing two distinct change actions on page p*:*

$$\mathtt{p}_1 \equiv \mathbf{change}(\mathtt{p}, 1, t_1) \qquad \mathtt{p}_2 \equiv \mathbf{change}(\mathtt{p}, 1, t_2)$$

Then, $\mathtt{p}_1$ *is not safe w.r.t.* I_N*, whereas* $\mathtt{p}_2$ *is.*

In the following, a repairing methodology which gets rid of both, correctness and completeness errors is developed. First, are eliminated correctness errors. Some repair actions are automatically inferred and run in order to remove the wrong information from the Web site. After this phase, the website will be correct but still could be incomplete. At this point, other repair actions are synthesized and executed in order to provide Web site completeness.

3.5.1 Fixing correctness errors

Throughout this section are considered a given Web site W, a Web specification (I_M, I_N, R) and the set $E_N \neq \emptyset$ of the correctness error evidences w.r.t. I_N for W. The goal is to modify W in order to generate a new Web site which is correct w.r.t. (I_M, I_N, R). Whenever a correctness error is found, a possible repair action is chosen and executed in order to remove the erroneous information, provided that it does not introduce any new bug.

Given $\mathtt{e} = (\mathtt{p}, w, \mathtt{l}, \sigma) \in E_N$, e can be repaired in two distinct ways: 1) remove the wrong content $\mathtt{l}\sigma$ from the Web page p (specifically, from $\mathtt{p}_{|w}$), or 2) change $\mathtt{l}\sigma$ into a piece of correct information. Hence, it is possible to choose between the following repair strategies.

"Correctness through Deletion" strategy

In this case all the occurrences of the sub-term $\mathtt{p}_{|w}$ of the Web page p containing the wrong information $\mathtt{l}\sigma$ are removed applying the repair action **delete**$(\mathtt{p}, \mathtt{p}_{|w})$[3]

Example 3.5.2. *Consider the Web site in Figure 3.1 and the Web specification in Example 3.4.1. The term* $\mathtt{l}\sigma \equiv \mathtt{p}_{|1.4} \equiv$ blink(year(2003)) *embedded in the Web page* (5) *of* W *(which is also called* p *in this example) generates a correctness error evidence* $(\mathtt{p}, 1.4, \mathtt{l}, \sigma)$ *w.r.t. the rule* blink(x) $\rightharpoonup$ error *and hence a delete action will remove from* p *the sub-term* blink(year(2003)).

"Correctness through Change" strategy.

Given a correctness error $\mathtt{e} = (\mathtt{p}, w, \mathtt{l}\sigma) \in E_N$, the sub-term $\mathtt{p}_{|w}$ of the Web page p is replaced with a new term t input by the user. The new term t must fulfill some conditions

[3]Note that, instead of removing the whole sub-term $\mathtt{p}_{|w}$, it would be also possible to provide a more precise though also time-expensive implementation of the **delete** action which only gets rid of the part $\mathtt{l}\sigma$ of $\mathtt{p}_{|w}$ which is responsible for the correctness error.

which are automatically provided and checked in order to guarantee the correction of the inserted information.

Roughly speaking, first is ensured that (i) t does not embed sub-terms which might fire some correction rule (*local correctness property*). Next, is guaranteed that t, within the context surrounding it, will not cause any new correctness error (*global correctness property*).

Local correctness property. Conditional and unconditional correctness rules are handled separately. For conditional rules the following problem should be addressed.

Lets consider the correctness error evidence $\mathtt{e} = (\mathtt{p}, w, \mathtt{l}, \sigma) \in E_N$ and the associated repair action $\mathbf{change}(\mathtt{p}, w, t)$ and the set of conditions

$$CS_{\mathtt{e}} \equiv \{\neg \mathtt{C} \mid \exists \quad (\mathtt{l} \rightharpoonup \mathtt{r} \mid \mathtt{C}) \in I_N,\ \text{a position } w',\ \text{a substitution } \sigma \text{ s.t. } \mathtt{l}\sigma \trianglelefteq \mathtt{p}_{|w.w'}\}$$

$CS_{\mathtt{e}}$ is the *constraint satisfaction problem* associated with $\mathtt{e}$. $CS_{\mathtt{e}}$ is obtained by collecting and negating all the conditions of those correction rules which detect correctness errors in $\mathtt{p}_{|w}$. Such collection of constraints, that can be solved manually or automatically by means of an appropriate constraint solver [9], can be used to provide suitable values for the term t to be inserted. $CS_{\mathtt{e}}$ is *satisfiable* iff there exists at least one assignment of values for the variables occurring in $CS_{\mathtt{e}}$ that satisfies all the constraints. $Sol(CS_{\mathtt{e}})$ is the set of all the assignments that verify the constraints in $CS_{\mathtt{e}}$. The restriction of $Sol(CS_{\mathtt{e}})$ to the variables occurring in σ is denoted by $Sol(CS_{\mathtt{e}})_{|\sigma}$. Lets see an example.

Example 3.5.3. *Consider the Web site* W *in Figure 3.1 and the Web specification of Example 3.4.1. The following sub-term of Web page* (6)

```
project(pname(A1),grant1(1000),grant2(200),total(1100),
            coordinator(fullname(mariorossi)))
```

causes a correctness error e *w.r.t. the rule*

$$\mathtt{project(grant1(X), grant2(Y), total(Z))} \rightharpoonup \mathtt{error} \mathtt{X} + \mathtt{Y} \neq \mathtt{Z}.$$

The error can be fixed by changing the values of the grants and the total amount, according to the solution of the constraint satisfaction problem $CS_{\mathtt{e}}$ that follows:

$$X + Y = Z, X = Y) * 2, Z < 500000$$

The constraints in CS_{e} come from the conditions of the rules 7*,* 8 *and* 9*. An admissible solution might be*

$$\{\mathtt{X}/\mathtt{1000}, \mathtt{Y}/\mathtt{500}, \mathtt{Z}/\mathtt{1500}\} \in Sol(CS)$$

and the term t to be inserted might be

```
project(pname(A1),grant1(1000),grant2(500),
          coordinator(fullname(mariorossi)),total(1500))
```

which does not contain any incorrect data.

Sometimes CS_{e} might be not solvable. This implies that the Web specification is inconsistent, since there are two or more correctness rules conflicting (e.g. $I_N = \{l \rightharpoonup \mathtt{error} \mid c, l \rightharpoonup \mathtt{error} \mid \neg c\}$, and thus the user is asked to fix the Web specification before proceeding.

Lets now consider unconditional rules. Example 3.5.3 shows how to get rid of a correction error just by changing some values which occur into a piece of wrong information. However, sometimes might be needed to change not only the values of the variables but also the structure of the term containing the erroneous data. In this case, it might happen that we introduce a "forbidden" structure which can fire some unconditional correctness rule. Note that conditional rules cannot introduce any incorrect data in t, since is assumed that t has been chosen according to the solution of the constraint satisfaction problem CS_{e}. Therefore, in order to ensure correctness, the following *structural correctness check* on the structure of term t must be performed only for unconditional rules.

$$\forall\, \mathtt{l} \rightharpoonup \mathtt{r} \in I_N, w \in O_{\mathcal{T}ag}(t), \text{substitution } \sigma, \mathtt{l}\sigma \not\trianglelefteq t_{|w}. \tag{3.1}$$

The structural check (3.1) defined above ensures that no unconditional correctness rule can be triggered on t. This is achieved by checking that no left-hand side of an unconditional correctness rule is embedded into t.

Example 3.5.4. *Consider again Example 3.5.3 and the following terms t_1 and t_2. Both t_1 and t_2 modify the values and the structure of the faulty Web page.*

```
project(pname(A1),grant1(1000),grant2(500),projectleader(surname(gialli),
          name(anna)),blink(total(1500)))
project(pname(A1),grant1(1000),grant2(500),projectleader(surname(gialli),
          name(anna)),bold(total(1500)))
```

If the wrong information is replaced by term t_1, then the structural correctness check (3.1) would fail, since the term `blink(total(1200))` *will fire rule* 6 *of the Web specification, while it would succeed by using t_2.*

There is now to formalize the local condition for the "correctness through change" strategy.

Definition 3.5.2. *Given* $\mathtt{e} = (\mathtt{p}, w, \mathtt{l}, \sigma) \in E_N$ *and a repair action* $\mathbf{change}(\mathtt{p}, w, t)$*, we say that* $\mathbf{change}(\mathtt{p}, w, t)$ obeys the local correctness property *iff*

- *for each conditional rule* $(\mathtt{l} \rightharpoonup \mathtt{r} \mid \mathtt{C}) \in I_N$*,* $\mathtt{C} \neq \emptyset$*, substitution* σ' *and position* w'*, if* $\mathtt{l}\sigma' \trianglelefteq t_{|w.w'}$ *then*
  1. $\sigma' \in Sol(CS_{\mathtt{e}})_{|\sigma'}$*, when* $\neg\mathtt{C} \in CS_{\mathtt{e}}$*;*
  2. $\mathtt{C}\sigma'$ *does not hold, when* $\neg\mathtt{C} \notin CS_{\mathtt{e}}$*.*
- *for unconditional rules, the structural correctness check (3.1) succeeds.*

The local correctness property guarantees that a change action is "locally safe", in other words the term t which replaces the wrong information is safe w.r.t. I_N as stated by the following proposition.

Proposition 3.5.1. *Let* (I_M, I_N, R) *be a Web specification,* $\mathtt{e} = (\mathtt{p}, w, \mathtt{l}, \sigma) \in E_N$ *and* $\mathbf{change}(\mathtt{p}, w, t)$ *be a repair action. If* $p' \equiv \mathbf{change}(\mathtt{p}, w, t)$ *obeys the local correctness property, then* $p'_{|w} \equiv t$ *is safe w.r.t.* I_N*.*

Note that it is possible to repair several errors simultaneously by applying one single change action, since a change action affecting a term p may eventually fix all the correctness errors which occur in all the sub-terms of p. More formally, the following result holds.

Proposition 3.5.2. *Let* (I_M, I_N, R) *be a Web specification,* $\mathtt{e_i} = (\mathtt{p}, w_i, \mathtt{l}_i, \sigma_i) \in E_N$*,* $i = 1, \ldots, n$*,* $w_1 \leq w_2 \leq \ldots \leq w_n$*, and* $p' \equiv \mathbf{change}(\mathtt{p}, w_1, t)$ *be a repair action that obeys the local correctness property. Then,* $p'_{|w_1} \equiv t$ *is safe w.r.t.* I_N*.*

Now we proceed with the **Global correctness property.** Whenever some wrong data is fixed by executing a repair action $\mathbf{change}(\mathtt{p}, w, t)$, it is not enough to ensure that the term t to be introduced is locally safe (local correctness property), is also needed to consider t within the context that surrounds it in p. If such a global condition is ignored, some subtle correctness errors might arise as witnessed by the following example.

Example 3.5.5. *Consider the Web page* $\mathtt{p} \equiv \mathtt{f(g(a),b,h(c))}$*, and the following correctness rule set*

$$I_N \equiv \{(\mathtt{r1})\ \mathtt{f(g(b))} \rightharpoonup \mathtt{error}, (\mathtt{r2})\ \mathtt{g(a)} \rightharpoonup \mathtt{error}\}.$$

The Web page $\mathtt{p}$ *contains a correctness error according to rule* $(\mathtt{r2})$*. The Web page* $\mathtt{f(g(b),b,h(c))}$ *is obtained from* p *by executing, for instance, the repair action*

$$\mathbf{change}(\mathtt{f(g(a),b,h(c))}, 1, \mathtt{g(b)}).$$

Although the term $\mathtt{g(b)}$ *is safe w.r.t.* I_N *(i.e. it does not introduce any new correctness error), the replacement of* $\mathtt{g(a)}$ *with* $\mathtt{g(b)}$ *in* $\mathtt{p}$ *produces a new correctness error which is recognizable by rule* $(\mathtt{r1})$*.*

In order to avoid such kinds of undesirable repairs, the following global correctness property is defined, which simply prevents a new term t from firing any correctness rule when inserted in the Web page to be fixed.

The following definition is auxiliary. Let $\mathtt{s}, \mathtt{t} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ s.t. $\mathtt{s} \trianglelefteq \mathtt{t}$. The set $Emb_{\mathtt{s}}(\mathtt{t})$ is the set of all the positions in $\mathtt{t}$ which embed some symbol of $\mathtt{s}$. For instance, consider the terms $f(k, g(c))$, and $f(b, g(c), k)$. Then, $f(k, g(c)) \trianglelefteq f(b, g(c), k)$ and $Emb_{f(k,g(c))}(f(b, g(c), k)) = \{\Lambda, 2, 2.1, 3\}$.

Definition 3.5.3. *Let* (I_M, I_N, R) *be a Web specification,* $\mathtt{p}' \equiv \mathbf{change}(\mathtt{p}, w, t)$ *be a repair action producing the Web page* $\mathtt{p}'$*. Then,* $\mathbf{change}(\mathtt{p}, w, t)$ obeys the global correctness property *if, for each correctness error evidence* $\mathtt{e} = (\mathtt{p}', w', \mathtt{l}, \sigma)$ *w.r.t.* I_N *such that* $w' \leq w$*,*

$$\{w\}.O_{\mathcal{T}ag}(t) \cap \{w'\}.O_{\mathcal{T}ag}(Emb_{\mathtt{l}\sigma}(p'_{|w'})) = \emptyset$$

The idea behind Definition 3.5.3 is that any eventual error e in the new page $\mathtt{p}' \equiv \mathbf{change}(\mathtt{p}, w, t)$, obtained by inserting term t within $\mathtt{p}$, is not a consequence of this change but already present in a different sub-term of $\mathtt{p}$. For this purpose, the set of positions of the wrong information $\mathtt{l}\sigma$ cannot "overlap" the considered term t.

Example 3.5.6. *Consider again Example 3.5.5. The repair action*

$$\mathbf{change}(\mathtt{f(g(a),b,h(c))}, 1, \mathtt{g(b)})$$

does not obey the global correctness property. Indeed, it generates a Web page $\mathtt{f(g(b),b,h(c))}$ *containing a correctness error.*

The execution of a change action which obeys the global as well as the local correctness property, decreases the number of correctness errors of the original Web site as stated by the following proposition.

Proposition 3.5.3. *Let (I_M, I_N, R) be a Web specification and* W *be a Web site. Let $E_N(\mathtt{W})$ be the set of correctness error evidences w.r.t. I_N of* W*, and $(\mathtt{p}, w, \mathtt{l}, \sigma) \in E_N$. Given a repair action* $\mathbf{change}(\mathtt{p}, w, t)$*, which obeys the local as well as the global correctness property, we have that*

$$| E_N(\mathtt{W}') | < | E_N(\mathtt{W}) |$$

where $\mathtt{W}' \equiv \mathtt{W} \setminus \{\mathtt{p}\} \cup \{\mathbf{change}(\mathtt{p}, w, t)\}$.

3.5.2 Fixing completeness errors

In this section, we address the problem of repairing an incomplete Web site W. Without loss of generality, is assumed that W is an incomplete but correct Web site w.r.t. a given Web specification (I_M, I_N, R). Such an assumption will allows to design a repair methodology which "completes" the Web site and does not introduce any incorrect information.

Let $E_M(\mathtt{W})$ be the set of completeness error evidences risen by I_M for the Web site W. Any completeness error evidence belonging to $E_M(\mathtt{W})$ can be repaired following distinct strategies and thus by applying distinct repair actions. On the one hand, is possible to think to add the needed data, whenever a Web page or a piece of information in a Web page is missing. On the other hand, all the information that caused the error might be removed to get rid of the bug. In both cases, should be ensured that the execution of the chosen repair action does not introduce any new correctness/completeness error to guarantee the termination and the soundness of our methodology.

"Completeness through Insertion" strategy.

There are two distinct kinds of repair actions, namely **add**(p,W) and **insert**$(\mathtt{p}, w, t)$, according to the kind of completeness error to fix. The former action adds a new Web page p to a Web site W and thus will be employed whenever the system has to fix a given missing Web page error. The latter allows to add a new piece of information t to (a sub-term of) an incomplete Web page p, and therefore is suitable to repair universal as well as existential completeness errors. More specifically, the insertion repair strategy works as follows.

Missing Web page errors. Given a missing Web page error evidence (r,W), we fix the bug by adding a Web page p, which embeds the missing expression r, to the Web site W. Hence, the Web site W will be "enlarged" by effect of the following **add** action

$$\mathtt{W} = \mathtt{W} \cup \{\mathbf{add}(\mathtt{p}, \mathtt{W})\}$$

where $\mathtt{r} \trianglelefteq \mathtt{p}_{|w}$ for some $w \in O_{\mathcal{T}ag}(\mathtt{p})$.

Example 3.5.7. *Let* W *be the Web site of Figure 3.1. Consider the following missing Web page error evidence (*r,W*), where* $\mathtt{r} \equiv (\mathtt{hp}(\mathtt{fullname}(\mathtt{giulioverdi}), \mathtt{status}))$. *It is fixed it by adding, for instance, the following Web page* p *to the Web site* W*.*

$$\mathtt{p} \equiv \mathtt{hp}(\mathtt{fullname}(\mathtt{giulioverdi}), \mathtt{status}(\mathtt{student}), \mathtt{hobby}(\mathtt{ski})).$$

Existential completeness errors. Given an existential completeness error evidence (r,$\{\mathtt{p}_1, \mathtt{p}_2, \ldots, \mathtt{p}_n\}$,E), the bug is fixed by inserting a term t, that embeds the missing expression r, into an arbitrary page $\mathtt{p}_i$, $i = 1, \ldots, n$. The position of the new piece of information t in $\mathtt{p}_i$ is typically provided by the user, who must supply a position in $\mathtt{p}_i$ where t must be attached. The **insert** action will transform the Web site W in the following way:

$$\mathtt{W} = \mathtt{W} \setminus \{\mathtt{p}_i\} \cup \{\mathbf{insert}(\mathtt{p}_i, w, t)\}$$

where $\mathtt{r} \trianglelefteq \mathtt{p}_{i|w}$ for some $w \in O_{\mathcal{T}ag}(\mathtt{p})$.

Universal completeness errors. Given a universal completeness error evidence (r,$\{\mathtt{p}_1, \mathtt{p}_2, \ldots, \mathtt{p}_n\}$,A), the bug is fixed by inserting a term t_i, that embeds the missing expression r, into every Web page $\mathtt{p}_i$, $i = 1, \ldots, n$ not embedding r. The position of the new piece of information t_i in each $\mathtt{p}_i$ is typically provided by the user, who must supply a position w_i in $\mathtt{p}_i$ where t_i must be attached. In this case, we will execute a sequence of **insert** actions, exactly one for each incomplete Web page $\mathtt{p}_i$. Therefore, the Web site W will be transformed in the following way. For each $\mathtt{p}_i \in \{\mathtt{p}_1, \mathtt{p}_2, \ldots, \mathtt{p}_n\}$ such that $\mathtt{r} \not\trianglelefteq \mathtt{p}_{i|w_j}$ for each $w_j \in O_{\mathcal{T}ag}(\mathtt{p}_i)$

$$\mathtt{W} = \mathtt{W} \setminus \{\mathtt{p}_i\} \cup \{\mathbf{insert}(\mathtt{p}_i, w_i, t_i)\}$$

where $\mathtt{r} \trianglelefteq \mathtt{p}_{|w_i}$ for some $w_i \in O_{\mathcal{T}ag}(\mathtt{p}_i)$.

Both the **add** action and the **insert** action introduce new information in the Web site which might be potentially dangerous, since it may contain erroneous as well as incomplete data. It is therefore important to constrain the kind of information a user can add.

In order to preserve correctness, the user is compelled to insert only safe information in the sense of Definition 3.5.1. Hence, the new data being added by the execution of some repair action cannot fire a correctness rule subsequently.

Proposition 3.5.4. *Let* (I_M, I_N, R) *be a Web specification and* $\mathtt{W}$ *be a correct Web site w.r.t.* (I_M, I_N, R)*. Let* $\mathtt{p}_1 \equiv \mathbf{insert}(\mathtt{p}, w, t)$ *and* $\mathtt{p}_2 \equiv \mathbf{add}(\mathtt{p}_2, \mathtt{W})$*.*

- *If* $\mathtt{p}_1$ *is safe w.r.t.* I_N*, then* $\mathtt{W} \setminus \{\mathtt{p}\} \cup \{\mathtt{p}_1\}$ *is correct w.r.t.* (I_M, I_N, R)*.*
- *If* $\mathtt{p}_2$ *is safe w.r.t.* I_N*, then* $\mathtt{W} \cup \{\mathtt{p}_2\}$ *is correct w.r.t.* (I_M, I_N, R)*.*

Additionally, repair actions shouldn't introduce new completeness errors, that is, are fixed all and only the initial set of completeness errors of the Web site $\mathtt{W}$, namely $E_M(\mathtt{W})$. Given a completeness error evidence `(e,W)`, is used the notation `e(r)` to make evident the unsatisfied requirement `r` signaled by e.

Definition 3.5.4. *Let* (I_M, I_N, R) *be a Web specification and* $\mathtt{W}$ *be a Web site w.r.t.* (I_M, I_N, R)*. Let* $E_M(\mathtt{W})$ *be the set of completeness errors of* $\mathtt{W}$ *risen by* $I_M(\mathtt{W})$*.*

- *the repair action* $\mathtt{p}_1 \equiv \mathbf{insert}(\mathtt{p}, w, t)$ *is* acceptable *w.r.t.* (I_M, I_N, R) *and* $\mathtt{W}$ *iff*
  1. $\mathtt{p}_1$ *is safe w.r.t.* (I_M, I_N, R)*;*
  2. $\mathtt{r} \trianglelefteq t_{|w}$*,* $w \in O_{\mathcal{T}ag}(t)$*, for some* $\mathtt{e}(\mathtt{r}) \in E_M(\mathtt{W})$*;*
  3. *if* $\mathtt{W}' \equiv \mathtt{W} \setminus \{\mathtt{p}\} \cup \{\mathtt{p}_1\}$*, then* $E_M(\mathtt{W}') \subset E_M(\mathtt{W})$*.*
- *the repair action* $\mathtt{p}_2 \equiv \mathbf{add}(\mathtt{p}_2, \mathtt{W})$ *is* acceptable *w.r.t.* (I_M, I_N, R) *and* $\mathtt{W}$ *iff*
  1. $\mathtt{p}_2$ *is safe w.r.t.* (I_M, I_N, R)*;*
  2. $\mathtt{r} \trianglelefteq \mathtt{p}_{2|w}$*,* $w \in O_{\mathcal{T}ag}(\mathtt{p}_2)$*, for some* $\mathtt{e}(\mathtt{r}) \in E_M(\mathtt{W})$*;*
  3. *if* $\mathtt{W}' \equiv \mathtt{W} \cup \{\mathtt{p}_2\}$*, then* $E_M(\mathtt{W}') \subset E_M(\mathtt{W})$*.*

Definition 3.5.4 guarantees that the information which is added by **insert** and **add** actions is correct and does not yield any new completeness error. More precisely, the number of completeness errors decreases by effect of the execution of such repair actions.

Example 3.5.8. *Consider the Web specification of Example 3.4.1, the Web site* $\mathtt{W}$ *of Figure 3.1 and the universal completeness error evidence*

```
(hp(status(professor),teaching),
 {hpage(fullname(mariorossi),phone(3333),
    status(professor),hobbies(hobby(reading),hobby(gardening))),
  hpage(fullname(annagialli),status(professor),phone(4444),
           teaching(course(Algebra)))},A)
```

To fix the error, some information should be added to Web page (2), while Web page (4) is complete w.r.t. the requirement $\mathtt{hp(status(professor), teaching)}$. *Consider the pieces of information*

$$t_1 \equiv \mathtt{teaching(course(title(logic), syllabus(blah)))}$$
$$t_2 \equiv \mathtt{teaching(courselink(url(www.mycourse.com),urlname(Logic)))}.$$

If term t_1 *is introduced, the corresponding* **insert** *action is acceptable. However, inserting term* t_2 *would produce a new completeness error (i.e. a broken link error).*

"Completeness through Deletion" strategy.

When dealing with completeness errors, sometimes it is more convenient to delete incomplete data instead of completing them. In particular, this option can be very useful, whenever is required to eliminate out-of-date information as illustrated in Example 3.5.9. The main idea of the deletion strategy is to remove all the information in the Web site that caused a given completeness error. The strategy is independent of the kind of completeness error which are handled, since the missing information is computed in the same way for all the three kinds of errors by partially rewriting the original Web pages of the Web site. In other words, given the missing expression `r` of a completeness error evidence `e(r)` (that is, a missing page error $(\mathtt{r}, \mathtt{W})$ or an existential completeness error $(\mathtt{r}, \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}, \mathtt{E})$) or a universal completeness error $(\mathtt{r}, \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}, \mathtt{A})$)), there exists a Web page $\mathtt{p} \in \mathtt{W}$ such that $\mathtt{p} \rightharpoonup^{+} \mathtt{r}$. Therefore all the terms occurring in the partial rewrite sequences that lead to a missing expression `r` are eliminated from the Web pages.

More formally, given a Web specification (I_M, I_N, R), a Web site `W` and a completeness error evidence `e(r)`, the Web site `W` will change in the following way.

For each $t_1 \rightharpoonup t_2 \rightharpoonup \ldots \rightharpoonup \mathtt{r}$, where $t_1 \trianglelefteq \mathtt{p}_{|w}, w \in O_{\mathcal{T}ag}(\mathtt{p}), \mathtt{p} \in \mathtt{W}$

$$\begin{aligned} \mathtt{W} \equiv & \{\mathtt{p} \in \mathtt{W} \mid t_i \ntrianglelefteq \mathtt{p}_{|w}, \forall\, w \in O_{\mathcal{T}ag}(\mathtt{p}), i = 1, \ldots n\} \cup \\ & \{\mathbf{delete}(\mathtt{p}, t_i) \mid \mathtt{p} \in \mathtt{W}, t_i \trianglelefteq \mathtt{p}_{|w}, w \in O_{\mathcal{T}ag}(\mathtt{p}) \\ & i = 1, \ldots n\} \end{aligned}$$

Example 3.5.9. *Consider the Web specification of Example 3.4.1, the Web site* `W` *of Figure 3.1 and the missing Web page error evidence* $(\mathtt{hpage(fullname(ugoblu), status)}, \mathtt{W})$,

which can be detected in W *by using the completeness rules in* I_M. *The missing information is obtained by means of the following partial rewrite sequence:*

$$\begin{array}{l}\texttt{pub}(\texttt{name}(\texttt{ugo}), \texttt{surname}(\texttt{blu}), \texttt{title}(\texttt{blah1}),\\ \texttt{blink}(\texttt{year}(2003))) \rightharpoonup\\ \texttt{member}(\texttt{name}(\texttt{ugo}), \texttt{surname}(\texttt{blu})) \rightharpoonup\\ \texttt{hpage}(\texttt{fullname}(\texttt{ugoblu}), \texttt{status})\end{array}$$

By choosing the Deletion strategy, all the information regarding the group membership and the publications of Ugo Blu *are deleted from the Web site.*

As in the case of the insertion strategy, the effects of the execution of the repair actions are delicate. More precisely, the execution of any **delete** action shouldn't introduce new completeness errors. For this purpose, the following notion of **acceptable** delete action is considered.

Definition 3.5.5. *Let* (I_M, I_N, R) *be a Web specification and* W *be a Web site w.r.t.* (I_M, I_N, R). *Let* $E_M(\texttt{W})$ *be the set of completeness errors of* W *risen by* $I_M(\texttt{W})$. *The repair action* $\texttt{p}_1 \equiv \mathbf{delete}(\texttt{p}, t)$ *is* acceptable *w.r.t.* (I_M, I_N, R) *and* W *iff*

$$\textit{if}\ \texttt{W}' \equiv \texttt{W} \setminus \{\texttt{p}\} \cup \{\texttt{p}_1\},\ \textit{then}\ E_M(\texttt{W}') \subset E_M(\texttt{W}).$$

3.6 Accomplishments

A semi-automatic methodology for repairing Web Sites has been presented which has a number of advantages over other potential approaches and hence can be used as a useful complement to them:

1. In contrast to the active database Web management techniques, is able to predict whether a repair action can cause new errors to appear and assist the user in reformulating the action;

2. By solving the constraint satisfaction problem associated to the conditions in Web specification rules, is able to detect whether two rules expressing a requirement for the Web site are incompatible. Moreover, it is routine to modify our method to suggest fixes for the site specification when the constraint does not hold;

3. It smoothly integrates on top of existing rewriting-based web verification frameworks such as [4, 5], which offer the expressiveness and computational power of functions and allow one to avoid the encumbrances of DTDs and XML rule languages.

The system as it is cannot guarantee a complete and correct Web Site w.r.t. the specification as intended in logic. There is no way to specify in the system the informations which should be included in the page, for the absence of negation from the framework.

4

Undemanded Data Discovering

This Chapter is structured as follows. Section 4.1 defines the concept of undemanded data w.r.t. a specification and the reasons why its detection is valuable. Section 4.2 explains how the idea of detecting undemanded data' has been originated. Section 4.3 introduces the formal definition of the operators needed for the discovering. Section 4.4 presents the algorithm used to integrate this analysis within the WebVerdi-M framework. Finally, Section 4.5 studies a complete example of undemanded data detection.

4.1 Undemanded Data Concept

As described in Chapter 3, the verification of Web Sites performed by WebVerdi-M framework [6] aims to identify two categories of errors: "the data which is there but is incorrect" and "the data which is not there and is supposed to be there". There is another category of errors which is not considered: "the informations which is not supposed to be there at all".

The reason why this category was not considered is a consequence of the initial design choice of WebVerdi-M. The system was devised of with the idea of being able to express only rules involving the data present in the website. The concept of *undemanded data* has been introduced still believing in the effectiveness of this approach. Two are the reasons why this approach still stands. First because of the ease for the user to specify only the information he is interested in and not all the ones he is not interested at all. Second, because of the explosion of number of rules determined by the specification of all the elements which should not be present, which obviously are much more the ones that should be.

Considering the above, the definition of *undemanded data* should be: "information exceeding what is necessary, where necessary is what is included in the specification". The drawback is that it would be actually meaningless to check every single piece of data which is not included in the specification, because they are not designed to include all elements that could be present, but only some relations between them. Also on a more practical level is not feasible to investigate the infinite amount of information that could be included into a web page.

To proceed with the formalization of the concept of "extra information" we need to give the Definition 4.1.1.

Definition 4.1.1 (difference). *We say that data belonging to a web site* differs *w.r.t. a specification when it is not included by any completeness rule of the specification.*

In order to allow both to focus on the information interesting for the user, as well as delimit the research space, the Definitions 4.1.2 and have been formulated.

Definition 4.1.2 (undemanded data). *Some data belonging to a website is defined as* undemanded *w.r.t. a specification when* differs *from the given specification.*

Different terminology cultures could have been used to define this type of data as redundant, incoherent, inconsistent, irrelevant, extra, spurious, meaningless, unrelated, unnecessary, unneeded and so on. To avoid possible misunderstanding caused by the strong semantic meaning which associate each of them to others fields of computer science, our choice fell on *undemanded*.

By contrast the information which do not comply to the specification will be identified as demanded data w.r.t. to it.

There are two possible scenarios in which is useful to detect undemanded data:

i) Website Creation: it is important to publish only information needed, avoiding over expose the user to an enormous amount of data. Irrelevant data in fact disturb the navigation experience, resulting in a Web site where information is either disorganized or hard to find. This will usually leave users confused or frustrated and will affect their decision to revisit the site [3];

ii) Maintenance Processes: it is fundamental to detect outdated information which derives from previous states of the system. The change of some attribute associated to the data might lead to outdated information even if still respecting the completeness and correctness specification.

This approach somehow introduces the idea of correctness and completeness how is interpreted in logic: not only what is specified should be in the system, but also what is not specified should not be there. Still remains the difference that, due to the semantical complexity of the relations between pieces of information, not all undemanded data has not the right to exists. In fact might be all-right to have some undemanded data. For this reason a system detecting undemanded data can spot only possible errors, to be proved.

It is interesting to note how also in other fields of computer science, similar approaches have been developed to address the management of not required data on one side and of information coherency on the other.

The first approach is addressed in the Relational Database theory with the Normalization process to eliminate redundancy [58, 23], which is very similar to the given definition of undemanded data. Somehow the process of normalizing the database structure is comparable conceptually to our website verification.

The second issue is addressed in Distributed Systems, where specifications are used to describe how the data should be shaped. The way of enforcing uniformity between the

data structure and the given description is the chosen consistency model [57, 2].

Also in Micro-Architecture design there are mechanisms (memory coherence protocols [60]) to ensure that in all place the data is logically connected.

In all the fields of computer science is important to guarantee that the data presented accomplish certain properties.

4.2 Genesis of the Idea

The idea to develop a undemandedness test came out considering to extract information from the Web Site to be verified. For instance supposing that verifying the Web Site of a university in which all professors have in their homepage the courses they teach. Somebody could be tempted to affirm that everybody who specifies in the homepage the courses they teach, is a professor. This hypothesis might be right or wrong, depending on the context: some courses might be taught by assistants in some university, but in other strictly only professors teach. While in the first case nothing can be done, in the second the pattern "everybody that in the homepage specify the courses they teach should be a professor" can be enforced.

Such a reasoning process goes along the lines of the abductive reasoning [27]. Abductive reasoning in logic, is the process of generation of hypotheses to explain observations or conclusions and reasoning to the best explanations. In other words, it is the reasoning process that starts from a set of facts and derives their most likely explanations. It is worthy to mention the difference between deduction and abduction, is a matter of the direction in which a rule like "a entails b" is used for inference:

Deduction Allows deriving b as a consequence of a, deduction is the process of deriving the consequences of what is known;

Abduction Allows deriving a as an explanation of b by allowing the precondition a of "a entails b", to be derived from the consequence b, abduction is the process of explaining what is known. In a large extension abduction could be seen as the reverse of deduction.

In our context from a set of precondition (specifications) and a set of facts (website) we can derive hypothesis that can be approved or rejected. This work does not pretend to

integrated abduction in a functional programming framework, but rather of using the ideas behind the abductive reasoning to derive relevant hypothesis regarding the data included into the website.

4.3 Integration with WebVerdi-M

The integration of the undemanded data concept w.r.t. a specification, within the WebVerdi-M framework as described in Chapter 3 is not obvious. In the context of Web Site verification the "required information" is, by definition, the completeness rules. These part of specification in fact define which data should be present of the Web Site to be complete as formalized by definition 3.4.4.

The algorithm can only formulate hypothesis on which data could be undemanded w.r.t. the specification, not classifying the data itself. This limit is due to the incapacity of understand the semantic meaning of the data. What can be done instead is to learn from the user classification of the data and learn from it for future verifications.

Hypothesis of possible undemandedness errors are then created in order be presented to the user and learn from his classification. This hypothesis should be regarding the data mentioned in the completeness specification. A set of rule is then obtained from the completeness specification allowing to formulate the hypothesis.

The idea behind this integration is that from the completeness specification can be derived that, if in the Web Site is present any expression simulating a right part of a rule `r`, without its respective left counterpart `l`, this particular expression is not required, unless a rule of the type $\mathtt{r} \rightharpoonup \mathtt{l}$ is included in the specification. The formalization of this process require the introduction of an operator with will be called Reverting Operator and will be described in Section 4.3.1.

Once rules have been derived, they can be used to find possible undemandedness errors in the website using the regular WebVerdi-M execution. Since not all the undemanded data w.r.t. a specification is actually erroneous, the user will be asked to classify the data found by the algorithm.

From the user classification, the algorithm can decide if the rules has to be remembered and added to the original completeness specification set accumulating knowledge, or just discarded. Being the process of aggregating extra rules to the initial specification set not so obvious, Section 4.4 is dedicated to it.

4.3.1 Reverting Operator

The reverting rule operation is defined as the operation that from a $\mathtt{l} \rightharpoonup \mathtt{r}$ rule derive a new rule $\mathtt{l}' \rightharpoonup \mathtt{r}'$, in which $\mathtt{l}'$ is derived by $\mathtt{r}$ and $\mathtt{r}'$ derived by $\mathtt{l}$ considering the following:

Filtering sign $\sharp$: With respect to the meaning of filtering given by the $\sharp$, is clear that in $\mathtt{l}'$ cannot be used because the left side of rewriting rules doesn'tt allow so (and wont have any sense since $\mathtt{l}'$ is already a pattern matching expression). In the creation of $\mathtt{r}'$, $\sharp$ has to be put in front of all terms which are not variables, to enforce the fact that $\mathtt{l}$ perform a pattern matching operation.

Example 4.3.1. *All filtering signs of the right side of the rules have been ignored, while the new right side uses them:*

Rule:$project(id(X)) \rightharpoonup \sharp ppage(\sharp id(X))\langle E\rangle$
Rule after application of Reverting Operator:
$ppage(id(X)) \rightharpoonup \sharp project(\sharp id(X))\langle E\rangle$

The original rule mean thats is required that every page with the construction $ppage(id))$ *has as argument the* X *value, correspondent to the value of the variable in right part* $\mathtt{r}$*. This is represented putting the function in the left part* $\mathtt{l}'$ *of the new rule. Instead to represent the exact pattern-match of the left part of the original rule* $\mathtt{l}$*, in the right part of the new rule* $\mathtt{r}'$ *the sign* $\sharp$ *is used when the function* $project(id(X))$ *is present.*

Quantifiers: The quantifiers are not touched, keeping their meaning in the rules also after the application of the Reverting Operator. In fact if the rule obtained by the reversion is a meaningful rule, the same quantifiers comply to it.

Example 4.3.2. *Existential quantifiers in this rule are kept in the reverting process:*

Rule: $project(id(X), name(Y), link(Z)) \rightharpoonup$
$\sharp project\text{-}list(\sharp item(\sharp id(X), \sharp name(Y), \sharp link(Z)))\langle E\rangle$
Rule after application of Reverting Operator:
$project\text{-}list(item(id(X), name(Y), link(Z))) \rightharpoonup$
$\sharp project(\sharp id(X), \sharp name(Y), \sharp link(Z))\langle E\rangle$

Given the original rule stating that if a project exists it should be included in the project list, also its reverted version stating that if there is a project in the project list it should be mentioned somewhere in the Web Site, is of the same existential type.

Extra variables occurring in the $\mathtt{l}$ part: The extra variable in the $\mathtt{l}$ part is not to be considered for two reasons. First because $\mathtt{l}(X)$ is more general than $\mathtt{l}(X, Y)$ so would only enforces the presence of one argument, when two are already present for sure. Second because if there are no constrains on the extra variable on the right, the variable it doesn't need to assume any special values as long as the terms which contains it exists.

Example 4.3.3. *The variable Z could be omitted after the application of the Reverting Operator:*

Rule: $person(name(X), surname(Y), link(Z)) \rightharpoonup \sharp hpage(\sharp id(X + +'_' + +Y))\langle E\rangle$
Rule after application of Reverting Operator:
$$hpage(id(X + +'_' + +Y)) \rightharpoonup \sharp person(\sharp name(X), \sharp surname(Y), \sharp link)\langle E\rangle$$

When reverting the rule the variable Z is not considered because it does not enforce any constraints beside the presence of the term $link$.

4.3.2 Reverting Operator Algorithm

All the above is formalized into the following algorithm:

Algorithm 1 Reverting Operator Algorithm

1: **procedure** REVERTING-OPERATOR($\mathtt{l} \rightharpoonup \mathtt{r}\langle q\rangle$)
2: $\quad \mathtt{l}' \leftarrow \mathtt{r}$
3: $\quad \mathtt{r}' \leftarrow \mathtt{l}$
4: $\quad$ **for** all $X \in Var(\mathtt{r}') \wedge X \notin Var(\mathtt{l}')$ **do**
5: $\quad\quad \mathtt{r}' \leftarrow eraseSymbol(\mathtt{r}', X)$
6: $\quad$ **end for**
7: $\quad \mathtt{l}' \leftarrow eraseSymbol(\mathtt{l}', \sharp)$
8: $\quad \mathtt{r}' \leftarrow putSharp(\mathtt{r}')$
9: $\quad Output \leftarrow \{\mathtt{l}' \rightharpoonup \mathtt{r}'\langle q\rangle\}$
10: **end procedure**

Some auxiliary function are below defined:

- $eraseSymbol(f(g(X)), Y)$ to delete all the occurrences of Y in $f(g(x))$

  Example 4.3.4. *This function eliminates all the occurrences of the given symbol or variable from the given function:*

  $eraseSymbol(f(\sharp g(X)), \sharp) = f(g(X))$
  $eraseSymbol(f(g(X)), X) = f(g)$

- $putSharp(f(g(X)))$ to add in front of all not ground terms the $\sharp$ symbol.

  Example 4.3.5. *This function adds the filtering symbol following the guidelines above:*

  $putSharp(f(g(X))) = \sharp f(\sharp g(X))$

4.4 Testing Algorithm

The net result of the execution of the algorithm is to add rules to the completeness specification to be able to catch undemandedness errors which haven't been considered by the user. If for some reasons is not possible to add such rules to the initial specification, the algorithm returns the set of errors which will be detected by such rule. The user can then process this errors as he prefers.

The algorithm is able to learn from the user input and refine the specification given. This is accomplished through an operation of marking the completeness rules, as described below. If the user wish, the algorithm could be used without using the learning features as well.

The testing algorithm is composed by six steps: Preparing, Filtering, Reverting, Checking, Searching and Learning.

1. **Preparing Step**: Only some of the rules belonging to the completeness specification are used for the testing process. This is due to two reasons, considering multiple sequential execution of the algorithm. First to avoid the reverted version of the rules to be reverted once more. It will not only mean wasting resources, but

because of the reverting operator, will introduce a new rule very similar but not exactly the same of a rule already present. Second because has already been learned that some rules do not need to be reverted. This isn't true all the times. The user might decide to not use the knowledge accumulated, for instance consequentially to a heavy edit of the website. See formal details of this step in Algorithm 2.

Algorithm 2 Preparing Step Details

1: **procedure** PREPARING($I, Mode$)
2: **case** $Mode$ **of**
3: default **:**
4: $I_{out} \leftarrow$ all rules without $_rev$, $_val$-$[number]$ or $_dem$
5: clean **:**
6: $I_{out} \leftarrow$ all rules without $_rev$ or $_val$-$[number]$
7: **end case**
8: $Output \leftarrow clearMarks(I_{out})$
9: **end procedure**

2. **Filtering Step**: Between all rules obtained from the previous step not all are interesting. A non interesting rule is a rule which its right part `r` simulates its left part `l`, because after the reverting step it will always be verified, see Example 4.4.1.

Example 4.4.1. *The left right side of left side of the rule embeds the right side:*

$person \rightharpoonup \sharp person(name, surname, link)\langle A \rangle$

If reverted this rule will require each term $person(name, surname, link)$ *to include the term* $person$*, which is obviously always true.*

Beside considering only the conceptually interesting rules, this procedure allows to ensure that the set obtained rules keep the same properties of the original specifications, avoiding the right-hand side `r` of a rule simulates its left-hand side `l`. In fact [11] proof that whenever a left-hand side `l` of a rule is simulated by (a sub-term of) the right-hand side `r` of a (possibly different) rule and no variables in the substructure of `r` which is recognized by simulation is located at positions deeper than all the positions of the variables in `l`, the algorithm is bounded. The details of the filtering step are illustrated in Algorithm 3.

Algorithm 3 Filtering Step Details

```
1: procedure FILTERING(I)
2:     for all r ∈ I do
3:         if l ⋬ r then
4:             I_out ← I_out ∪ r
5:         end if
6:     end for
7:     Output ← I_out
8: end procedure
```

3. **Reverting Step**: The rules obtained by the previous step are then reverted using the Reverting Operator as described in Section 4.3.1.

4. **Checking Step**: The aim of this step is to check if the initial completeness specification united with the set of reverted rules obtained by the previous step are sufficient to verify each page of the website. This is done putting together all pages which embeds a least a left part `l` of a rule. The details of this operation are illustrated in Algorithm 4.

   This test detects all undemanded information w.r.t. a specification, in all pages to which at least one of the rewriting rules apply. It s not possible to say anything about the pages to which none of the rules apply. The only conclusion that could do be drawn about the pages not touched by any rule is that are undemanded or that the specification given is incomplete. To have the whole Web Site undemandedness free, there should be at least one rule applying to each page.

   In case the specification given does not touch some of the pages, the user is notified in order to decide how to proceed. Possible choices are:

   - Stop to add some specification, so all pages will be touched by the specification;
   - Continue, if he consider fine to check only some pages.

5. **Searching Step**: The reverted rules as obtained by the Reverting Step are here used as input for the WebVerdi-M Tool, in order to obtain all possible undemandedness errors. The execution of the verification is done by executing different instances of

Algorithm 4 Checking Step Details

```
 1: procedure COMPLETESPECIFICATION(W, I)
 2:     P ← all pages of W which none I apply
 3:     for all p ∈ P do
 4:         case AskUser() of
 5:             stop :
 6:                 result ← false
 7:             continue :
 8:                 result ← true
 9:         end case
10:     end for
11:     Output ← result, W
12: end procedure
```

Verdi-M for every reverted rule, then merging the list of errors obtained by each execution. Using different instances of the algorithm guarantees the termination of the algorithm, because the set of reverted rules might not be bounded.

All error types (missing pages, universal or existential) are treated similarly in this phase, differing only for the repairing strategy.

6. **Learning Step**: Each possible undemandedness error evidence is here subjected to the user validation. The user has the chance of choosing different possibilities: confirming that every evidence like the one identified is an error, confirming that the particular evidence identified is an error or affirming that all evidences like the one identified is not erroneous. The case of affirming that only the particular evidence identified is not erroneous has not being considered being of difficult formalization in the form algorithm as been exposed. Anyhow it can be easily expressed by the combination of the other cases which instead can be expressed: specifying all cases in which is erroneous beside the correct one is sufficient.

   In order to apply the knowledge just learned in the future Web Site Verification, the users' classification is than incorporated into the completeness specification. The learning process happens differently for every of the possibilities the user can choose.

   To treat the first case, confirming that every evidence like the one identified is an error, the reverted rule which originates the error evidence is included in the completeness specification by adding to it. The problem is that the operation of adding

a rewriting rule to the completeness specification might break the properties of the initial set. In particular there is a fundamental property of that set which is the property of being bounded [11]. To ensure that it is kept after the adding operation, a checking procedure on the rule to be added is needed. This check is illustrated in Algorithm 5.

Algorithm 5 Checking To Perform Learning Step

1: **procedure** SPECKEEPSBOUNDED(I, r_a)
2: $Output \leftarrow true$
3: **for all** $\{r_s \in I\}$ **do**
4: **if** $(\texttt{l} \text{ of } r_a) \trianglelefteq (\texttt{r} \text{ of } r_s)$ **then**
5: **if** depth $Var(\texttt{r} \text{ of } r_s) >$ depth $Var(\texttt{l} \text{ of } r_a)$ **then**
6: $Output \leftarrow false$
7: **end if**
8: **end if**
9: **end for**
10: **end procedure**

If the condition is verified, the rule is finally added and marked, as well as the original rule. This marking operation is done for the same reasons already explained in the Preparing Step.

If the condition is not verified, the Testing Algorithm will return to the user only the list of undemandedness error, but without learning from it. Nevertheless the user can still fix the errors with some repairing operations.

Concerning the learning process of the possibility that only a particular evidence identified is an error, is sufficient to add to the initial specification the reverted rule originating the error, using the particular substitution σ which originates the error. Differently from the previous case is not needed to check if the completeness specification keeps bounded after the addition, because the rule added as consequence of the substitution σ does not contains variables, respecting the boundary condition proved in [11]. The rule added in this circumstance is not considered in another execution of the algorithm because its reverted version, and more general version, is part of the original specification.

The original rule instead is not marked, differently from the previous case, because during a new execution of the algorithm the user might want to give a different evaluation of the error evidences, not having specified a general rule.

When the user affirms that all evidences like the one identified are not erroneous, the rule will be marked in order to remember that is not necessary to do any deductions regarding undemandedness from it.

All this marking rules operations are done with the use of the function $markRule(_[string], r, I)$, which simply denote the addition of the mark $_[string]$ in front of a rule, see Example 4.4.2.

Example 4.4.2. *This function add a mark in front of a rule:*

Specifications I *before the operation:*
$r1$: $person \rightharpoonup \sharp person(name, surname, link)\langle A\rangle$
$r2$:$ppage(id(Y), owner(person(link(X)))) \rightharpoonup$
$\sharp hpage(\sharp id(X), \sharp projects(\sharp project(\sharp id(Y))))\langle E\rangle$
Operation: $markRule(_dem, r2, I)$
Specifications I *after the operation:*
$r1$: $person \rightharpoonup \sharp person(name, surname, link)\langle A\rangle$
$r2_dem$:$ppage(id(Y), owner(person(link(X)))) \rightharpoonup$
$\sharp hpage(\sharp id(X), \sharp projects(\sharp project(\sharp id(Y))))\langle E\rangle$

In the pseudo-code used there are auxiliary function which are described below.

The operator $clearMarks(I)$ simply denote the deletion of any sign $_[string]$ in front any rule included in the specifications I.

Example 4.4.3. *This function clear all marks from the rules in the specifications*

Specifications I *before the operation:*
$r1_dem$:$ppage(id(Y), owner(person(link(X)))) \rightharpoonup$
$\sharp hpage(\sharp id(X), \sharp projects(\sharp project(\sharp id(Y))))\langle E\rangle$
$r2_pr$:$project(id(X), name(Y), link(Z)) \rightharpoonup$
$\sharp project\text{-}list(\sharp item(\sharp id(X), \sharp name(Y), \sharp link(Z)))\langle E\rangle$
Operation: $clearMarks(I)$
Specifications I *after the operation:*

$r1$:$ppage(id(Y), owner(person(link(X)))) \rightharpoonup$
$\sharp hpage(\sharp id(X), \sharp projects(\sharp project(\sharp id(Y))))\langle E\rangle$
$r2$:$project(id(X), name(Y), link(Z)) \rightharpoonup$
$\sharp project\text{-}list(\sharp item(\sharp id(X), \sharp name(Y), \sharp link(Z)))\langle E\rangle$

The algorithm described is illustrated in Algorithm 6.

Algorithm 6 Complete Undemandedness Test Algorithm

1: **procedure** UNDEMANDEDNESSTEST($\mathtt{W}, (I_N, I_M, R), Mode$)
2: $I'_M \leftarrow prepare(I_M, Mode)$
3: $I''_M \leftarrow filter(I'_M)$
4: $I_{MR} \leftarrow reverting(I''_M)$
5: $I_{All} \leftarrow I_M \cup I_{MR}$
6: $result, \mathtt{W} \leftarrow CompleteSpecification(\mathtt{W}, I_{All})$
7: **if** $result = true$ **then**
8: $E_U \leftarrow \emptyset$
9: **for all** $r \in I_{MR}$ **do**
10: $E_M \leftarrow Verdi\text{-}M(\mathtt{W}, (\emptyset, \{r\}, R))$
11: **for all** $e = (r, C, \sigma) \in E_M$ **do**
12: **case** AskUser() **of**
13: undemanded-this-time **:**
14: $I_M \leftarrow I_M \cup \{r_\sigma\}$
15: $markRule(_val\text{-}[number], r_\sigma, I_M)$
16: undemanded-always **:**
17: **if** $specKeepsBounded(I_M, r)$ **then**
18: $I_M \leftarrow I_M \cup \{r\}$
19: $markRule(_rev, r, I_M)$
20: $markRule(_or, r_{original}, I_M)$
21: **else**
22: $E_U \leftarrow E_U \cup \{e\}$
23: **end if**
24: demanded-always **:**
25: $markRule(_dem, r, I_M)$
26: **end case**
27: **end for**
28: **end for**
29: $Output \leftarrow \{E_U, I_M\}$
30: **end if**
31: **end procedure**

To be noted that the $\cup$ operation is meant to be used as the union of sets, so no repeated elements can results after such operation.

4.5 A Complete Example

This example shows what happens in each steps of the algorithm described above, applying a specification to the web site included in Appendix A. To be noted is that the web site comply to a regular completeness check given the Specification below.

By steps:

1. **Preparing Step**: Considering the first time the algorithm is executed the preparation operation does not modify the specification given:

```
1) person(name(X),surname(Y),link(Z)) -> #hpage(name(X),surname(Y)) <E>
2) person() -> #person(name(),surname(),link()) <A>
3) hpage() -> #hpage (id(),name(),surname(),tel(),team(),projects()) <A>
4) person(name(X),surname(Y),link(Z)) ->
   #item(#name(X),#surname(Y),link(Z)) <E>
5) hpage(name(X),surname(T),team(person(name(Y),surname(Z)))) ->
   #hpage(#name(Y),#surname(Z),#team(#person(#name(X),#surname(T)))) <E>
6) project (id(X)) -> #ppage(#id(X)) <E>
7) project() -> #project(id(),name(),link()) <A>
8) ppage() -> #ppage(id(),name(),owner(),budget()) <A>
9) ppage(id(Y),owner(person(link(X)))) ->
   #hpage(#id(X),#projects(#project(#id(Y)))) <E>
10) owner() -> #owner(person()) <A>
11) project(id(X),name(Y),link(Z)) ->
    #item(#id(X),#name(Y),#link(Z)) <E>
```

2. **Filtering Step**: The filtering steps refine the specification given to only the rules interesting for reversion:

```
1) person(name(X),surname(Y),link(Z)) -> #hpage(name(X),surname(Y)) <E>
4) person(name(X),surname(Y),link(Z)) ->
   #item(#name(X),#surname(Y),link(Z)) <E>
6) project (id(X)) -> #ppage(#id(X)) <E>
9) ppage(id(Y),owner(person(link(X)))) ->
   #hpage(#id(X),#projects(#project(#id(Y)))) <E>
11) project(id(X),name(Y),link(Z)) ->
   (#item(#id(X),#name(Y),#link(Z)) <E>
```

3. **Reverting Step**: The reversion operation on the rules obtained by the previous step returns what is represented:

```
1_rev) hpage(name(X),surname(Y)) -> #person(#name(X),#surname(Y),#link()) <E>
4_rev) item(name(X),surname(Y),link(Z))  ->
        #person(#name(X),#surname(Y),#link(Z)) <E>
6_rev) ppage(id(X))  -> #project (#id(X)) <E>
9_rev) hpage(id(X),projects(project(id(Y)))) ->
        #ppage(#id(Y),#owner(#person(#link(X))))  <E>
11_rev) item(id(X),name(Y),link(Z)) ->
         #project(#id(X),#name(Y),#link(Z)) <E>
```

4. **Checking Step**: The checking will be performed using the set of rules from the previous step and the original specification. The user won't be notified because some rule can be applied to each page, therefore does not have the choice of stopping or continue.

5. **Searching Step**: Assuming the user decided to continue from the previous steps, all reverted rules will be used with the WebVerdi-M engine, to create a list of errors. This list of error will be:

```
<collectionErrors WebSite=" ... " Specification=" ... ">
   <errorUndemandeness>
      <ruleCompleteness Name="1_rev"> ... </ruleCompleteness>
      <pages> ... </pages>
      <sigma>
         <sust>
            <var> X </var>
            <value> 'Maria' </value>
         </sust>
         <sust>
            <var> Y </var>
            <value>'Alpuente'</value>
         </sust>
      </sigma>
      <type> E </type>
   </errorUndemandeness>

   <errorUndemandeness>
      <ruleCompleteness Name="1_rev"> ... </ruleCompleteness>
      <pages> ... </pages>
      <sigma>
         <sust>
            <var> X </var>
            <value> 'Sonia' </value>
         </sust>
         <sust>
            <var> Y </var>
            <value> 'Flores' </value>
         </sust>
```

```
      </sigma>
      <type> E </type>
   </errorUndemandeness>

   <errorUndemandeness>
      <ruleCompleteness Name="4_rev"> ... </ruleCompleteness>
      <pages> ... </pages>
      <sigma>
         <sust>
            <var> X </var>
            <value> 'Beatriz' </value>
         </sust>
         <sust>
            <var> Y </var>
            <value> 'Alarcon' </value>
         </sust>
         <sust>
            <var> Z </var>
            <value> 'Beatriz_Alarcon' </value>
         </sust>
      </sigma>
      <type> E </type>
   </errorUndemandeness>

   <errorUndemandeness>
      <ruleCompleteness Name="4_rev"> ... </ruleCompleteness>
      <pages> ... </pages>
      <sigma>
         <sust>
            <var> X </var>
            <value> 'Maria' </value>
         </sust>
         <sust>
            <var> Y </var>
            <value> 'Alpuente' </value>
         </sust>
         <sust>
            <var> Z </var>
            <value> 'Maria_Alpuente' </value>
         </sust>
      </sigma>
      <type> E </type>
   </errorUndemandeness>

   <errorUndemandeness>
      <ruleCompleteness Name="9_rev"> ... </ruleCompleteness>
      <pages> ... </pages>
      <sigma>
         <sust>
            <var> X </var>
            <value> 'Maria_Alpuente' </value>
```

```
      </sust>
      <sust>
         <var> Y </var>
         <value> '1' </value>
      </sust>
   </sigma>
   <type> E </type>
</errorUndemandeness>

<errorUndemandeness>
   <ruleCompleteness Name="9_rev"> ... </ruleCompleteness>
   <pages> ... </pages>
   <sigma>
      <sust>
         <var> X </var>
         <value> 'Sonia_Flores' </value>
      </sust>
      <sust>
         <var> Y </var>
         <value> '1' </value>
      </sust>
   </sigma>
   <type> E </type>
</errorUndemandeness>

</collectionErrors>
```


6. **Learning Step**: Each of the errors will be notified by the user which will decide what to do with those. Lets assume the following:

   - Only the fact that $SoniaP$ have a page but is not mentioned anywhere in the site is considered erroneous, so the action $addValue$ is suggested for rule 1r with the substitution $X/Sonia$, $Y/Flores$. Note that for the other error originated no action is required;

   - All the error evidences originated by rule 4_ rev are considered erroneous. Is not fine to have in the people list individuals which are not mentioned anywhere else in the Web Site, therefore such a rule is asked to be added with the action $addRule$;

   - Is fine to not consider undemandedness errors the evidences originated by rule 9_ rev. It is understandable that if you work on a project, you might not be the owner. This condition is marked be the action $notAnError$.

This is expressed here:

```
<collectionPairErrorAction WebSite=" ... " Specification=" ... ">
   <pairErrorAction>
      <errorUndemandeness>
         <r><ruleCompleteness Name="9_rev"> ... </ruleCompleteness></r>
         <pages> ... </pages>
         <sigma>
            <sust>
               <var> X </var>
               <value> 'Sonia_Flores' </value>
            </sust>
            <sust>
               <var> Y </var>
               <value> '1' </value>
            </sust>
         </sigma>
         <type> E </type>
      </errorUndemandeness>
      <action>notAnError</action>
   </pairErrorAction>

   <pairErrorAction>
      <errorUndemandeness>
         <r><ruleCompleteness Name="1_rev"> ... </ruleCompleteness></r>
         <pages> ... </pages>
         <sigma>
            <sust>
               <var> X </var>
               <value> 'Maria' </value>
            </sust>
            <sust>
               <var> Y </var>
               <value> 'Alpuente' </value>
            </sust>
         </sigma>
         <type> E </type>
      </errorUndemandeness>
      <action>noActionRequired</action>
   </pairErrorAction>

   <pairErrorAction>
      <errorUndemandeness>
         <r><ruleCompleteness Name="1_rev"> ... </ruleCompleteness></r>
         <pages> ... </pages>
         <sigma>
            <sust>
               <var> X </var>
               <value> 'Sonia' </value>
            </sust>
            <sust>
               <var> Y </var>
```

```
            <value> 'Flores' </value>
         </sust>
      </sigma>
      <type> E </type>
   </errorUndemandeness>
   <action>addValue</action>
</pairErrorAction>

<pairErrorAction>
   <errorUndemandeness>
      <r><ruleCompleteness Name="9_rev"> ... </ruleCompleteness></r>
      <pages> ... </pages>
      <sigma>
         <sust>
            <var> X </var>
            <value> 'Maria_Alpuente' </value>
         </sust>
         <sust>
            <var> Y </var>
            <value> '1' </value>
         </sust>
      </sigma>
      <type> E </type>
   </errorUndemandeness>
   <action>notAnError</action>
</pairErrorAction>

<pairErrorAction>
   <errorUndemandeness>
      <r><ruleCompleteness Name="4_rev"> ...</ruleCompleteness></r>
      <pages> ... </pages>
      <sigma>
         <sust>
            <var> X </var>
            <value> 'Beatriz' </value>
         </sust>
         <sust>
            <var> Y </var>
            <value> Alarcon </value>
         </sust>
         <sust>
            <var> Z </var>
            <value> 'Beatriz_Alarcon' </value>
         </sust>
      </sigma>
      <type> E </type>
   </errorUndemandeness>
   <action>addRule</action>
</pairErrorAction>

<pairErrorAction>
```

```
      <errorUndemandeness>
         <r><ruleCompleteness Name="4_rev"> ... </ruleCompleteness></r>
         <pages> ... </pages>
         <sigma>
            <sust>
               <var> X </var>
               <value> 'Maria' </value>
            </sust>
            <sust>
               <var> Y </var>
               <value> 'Alpuente' </value>
            </sust>
            <sust>
               <var> Z </var>
               <value> 'Maria_Alpuente' </value>
            </sust>
         </sigma>
         <type> E </type>
      </errorUndemandeness>
      <action>addRule</action>
   </pairErrorAction>

</collectionPairErrorAction>
```

It will determine the modification of the specification as represented here:

```
1)          person(name(X),surname(Y),link(Z)) ->
            #hpage(name(X),surname(Y)) <E>
1_rval-0) hpage(name('Sonia'),surname('Flores')) ->
              #person(#name('Sonia'),#surname('Flores'),#link()) <E>
2)          person() -> #person(name(),surname(),link()) <A>
3)          hpage() -> #hpage (id(),name(),surname(),tel(),team(),projects()) <A>
4_or)       person(name(X),surname(Y),link(Z)) ->
             #people-list(#item(#name(X),#surname(Y),link(Z))) <E>
4_rev)      peoplelist(item(name(X), surname(Y),link(Z))) ->
              #person(#name(X),#surname(Y),#link(Z)) <E>
5)          hpage(name(X),surname(T),team(person(name(Y),surname(Z)))) ->
            #hpage(#name(Y),#surname(Z),
                       #team(#person(#name(X),#surname(T)))) <E>
6)          project (id(X)) -> #ppage(#id(X)) <E>
7)          project() -> #project(id(),name(),link()) <A>
8)          ppage() -> #ppage(id(),name(),owner(),budget()) <A>
9_dem)    ppage(id(Y),owner(person(link(X)))) ->
             #hpage(#id(X),#projects(#project(#id(Y)))) <E>
10)       owner() -> #owner(person()) <A>
11)       project(id(X),name(Y),link(Z)) ->
            #projectlist(#item(#id(X),#name(Y),#link(Z))) <E>
```

In particular:

- Rule 1_ rval-0 has been added to the specification with the substitution which originates the error, causing the extinction of all errors originated. Note that rule 1 is not marked as original;
- Rule 4_ rev has been added because it really identifies real undemandedness errors and its introduction keeps the specification bounded. This will also cause the extinction of all errors originated by it and the marking of rule 4 as original;
- Rule 9_ rev is not added because it does not originates any real error. Rule 9 is instead marked as final and the errors originated by rule 9_ rev extinguished.

The collection of errors returned by this operation is the only error which is asked to not resolve, the other are in fact resolved by modifying the specification. This happens with the genaration of a new specification with the id returned to the client as follows:

```
<collectionErrors WebSite=" ... " Specification=" ... ">
   <errorUndemandeness>
      <r><ruleCompleteness Name="1_rev"> ... </ruleCompleteness></r>
      <pages> ... </pages>
      <sigma>
         <sust>
            <var> X </var>
            <value> 'Maria' </value>
         </sust>
         <sust>
            <var> Y </var>
            <value> 'Alpuente' </value>
         </sust>
      </sigma>
      <type> E </type>
   </errorUndemandeness>
</collectionErrors>
```

Is immediate that running now a completeness check with the changed specification will allow to detect the errors identified as so. An output of that can be seen here:

```
<collectionErrors>
   <errorCompleteness>
      <r><ruleCompleteness Name="4_rev"> ... </ruleCompleteness></r>
      <pages> ... </pages>
      <sigma>
         <sust>
```

```
          <var>X</var>
          <value>'Beatriz'</value>
        </sust>
        <sust>
           <var>Y</var>
           <value> 'Alarcon' </value>
        </sust>
        <sust>
           <var>Z</var>
           <value>'Beatriz_Alarcon'</value>
        </sust>
     </sigma>
     <type>E</type>
  </errorCompleteness>

  <errorCompleteness>
     <r><ruleCompleteness Name="4_rev"> ... </ruleCompleteness></r>
     <pages> ... </pages>
     <sigma>
        <sust>
           <var>X</var>
           <value>'Maria'</value>
        </sust>
        <sust>
           <var>Y</var>
           <value>'Alpuente'</value>
        </sust>
        <sust>
           <var>Z</var>
           <value>'Maria_Alpuente'</value>
        </sust>
     </sigma>
     <type>E</type>
  </errorCompleteness>

  <errorCompleteness>
     <r><ruleCompleteness Name="1_rval-0">...</ruleCompleteness></r>
     <pages> ... </pages>
     <sigma>
        <sust>
           <var>P</var>
           <value>P</value>
        </sust>
     </sigma>
     <type>E</type>
  </errorCompleteness>
</collectionErrors>
```

5

Implementation

This Chapter is structured as follows. Section 5.1 illustrates SOAP Web Services Java-based. Section 5.2 describes the software structure. Section 5.3 introduces to functional programming and to the Maude language. Section 5.4 formalizes the API of the Web Service created.

5.1 Web Services

The Web has been a phenomenal success at enabling simple computer/human interactions at Internet scale [14]. The original HTTP [44] and HTML [51] protocol stack used by today's Web browsers has proved to be a cost-effective way to project user interfaces onto a wide array of devices. A key factor in the success of HTTP and HTML was their relative simplicity: both are primarily text-based and can be implemented using a variety of operating systems and programming environments.

Web services take many of the ideas and principles of the Web and apply them to computer/computer interactions. Like the World Wide Web, Web services communicate using a set of foundation protocols that share a common architecture and are meant to be realized in a variety of independently developed and deployed systems. Like the World Wide Web, Web services protocols owe much to the text-based heritage of the Internet and are designed to layer as cleanly as possible without undue dependencies within the protocol stack.

A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML (specifically WSDL [53]). A Web service enhances interoperability among software applications. A Web service is also self-describing and can be published, located, and invoked across the Web. Hence, Web services facilitate the development of distributed applications by adopting a loosely coupled Web programming model. Systems developed in terms of Web services are language independent and platform independent. Additionally, they are easily scalable and extensible by establishing connections to new Web services when necessary.

The core principles that have driven the design and implementation of the Web service architecture protocols are as follows:

Message orientation: only messages are used to communicate between services;

Protocol composability: an infrastructure based on protocol building blocks that may be used in nearly any combination is dominant;

Autonomous services: independent endpoints in terms of design, deployment, management, versioning and security are used;

Managed transparency: the control of which aspects of an endpoint are visible to external services is strict;

Protocol-based integration: the restriction of cross-application coupling to wire artifacts only is used.

Commonly all Web service interaction is performed by exchanging SOAP messages, see Section 5.1.1.

To provide for a robust development and operational environment, services are described using machine-readable meta-data to enable interoperability. Web service meta-data is used to describe the message interchange formats the service can support, and the valid message exchange patterns of a service. Meta-data is also used to describe the capabilities and requirements of a service. Message interchange formats and message exchange patterns are expressed in WSDL. WSDL specifies what a request message must contain and what the response message will look like in unambiguous notation. In addition to describing message contents, WSDL may define where the service is available and what communications protocol is used to talk to the service. This means that the WSDL file can specify the base elements required to write a program to interact with a Web service.

5.1.1 Simple Object Access Protocol

SOAP provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. SOAP is designed to reduce as much as possible the engineering cost of integrating applications built on different platforms, with the assumption that the lowest-cost technology has the best chance of gaining universal acceptance. A SOAP message is an XML document information item that contains three elements: <Envelope >, <Header> and <Body>.

The *E*nvelope is the root element of the SOAP message and contains an optional *H*eader element and a mandatory *B*ody element. The *H*eader element is a generic mechanism for adding features to a SOAP message in a decentralized manner. Each child element of *H*eader is called a header block, and SOAP defines several well-known attributes that can be used to indicate who should deal with a header block (role) and whether processing it is optional or mandatory. When present, the *H*eader element is always the first child element of the *E*nvelope. The *B*ody element is always the last child element of the *E*nvelope, and is a container for the payload intended for the ultimate recipient of the message. SOAP itself defines no built-in header blocks and only one payload, which is

the *F*ault element used for reporting errors.

The messaging flexibility provided by SOAP allows services to communicate using a variety of message exchange patterns, satisfying the requirements of distributed applications.

SOAP is defined independently of the underlying messaging transport mechanism in use. It allows the use of many alternative transports for message exchange and allowing both synchronous and asynchronous message transfer and processing.

5.1.2 Java Implementation

The Java Platform provides the APIs and tools to design, develop, test and deploy Web services and clients that fully inter-operate with other Web Services and clients running on any platform. This full interoperability is possible because application data is translated "behind-the-scenes" to a standardized XML-based data stream.

The framework also include a functionality names Java Web Start [47] which allows application software for the Java Platform to be started directly from the Internet using a web browser. Web Start applications do not run inside the browser and the sandbox in which they run does not have to be as restricted.

5.2 Software Structure

The application has been structured as a SOAP Web Service [55]. The main reason behind this choice is the advantage of using the Service Oriented Architecture paradigm [61, 25]. In this paradigm services are distributed, autonomous, and independent. They are realized using standard protocols, in order to build networks of collaborating applications. This style of architecture allows one to reuse at the macro level (service), rather than micro level (object).

WebVerdi-M as a service-oriented architecture allows to access the core verification engine Verdi-M as a reusable entity. This implementation is an extension of the work [6] which has the following characteristics. Refer to `http://www.dsic.upv.es/users/elp/WebVerd`
.

WebVerdi-M can be divided into two layers: front-end and back-end. The back-end layer provides web services to support the front-end layer. This architecture allow clients on the network to invoke the Web service functionality through the available interfaces.

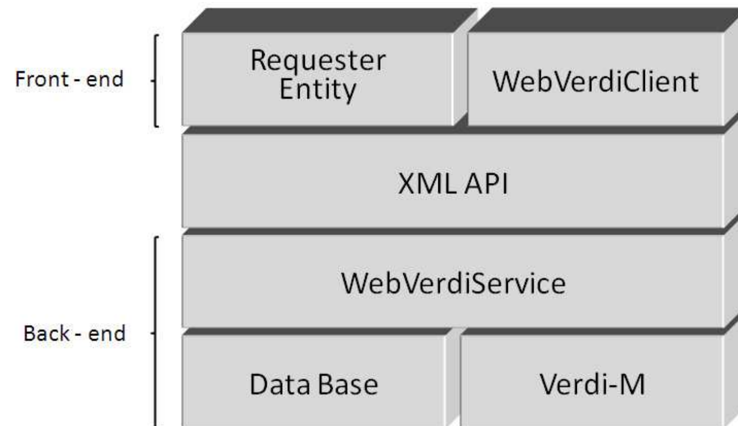

Figure 5.1: Components of WebVerdi-M

The tool consists of the following components: Web service WebVerdiService, Web client WebVerdiClient, core engine Verdi-M, XML API, and database DB. Figure 5.1 illustrates the overall architecture of the system.

5.2.1 WebVerdiService

The sever has been structured as a Web Service and its control parts is implemented in Java 1.4 [32], in order to use the large number of implementations of the Web Service standards available as the TriActive JDO persistence package [45]. Persistence is used to store both web pages and specification locally to the sever in a MySQL database [34].

TriActive JDO is an open source implementation of Sun's JDO specification [33], designed to support transparent persistence using any JDBC-compliant database. TriActive JDO allows to generate schema, meaning it takes user-written Java classes and automates the tasks required to transparently persist objects to a database.

All the elaboration is performed on the server side by modules written in Maude [37, 36]. This flavor of functional programming has been used because of its meta-level functionality, which together with the properties of associativity and commutativity, allows a simple and elegant implementation of the simulation algorithm.

The web service exports eleven operations that are network-accessible through standardized XML messaging. These operations are: store a specification, retrieve a specification, remove a specification store a Web site, remove a Web site, retrieve a Web site, add Web page to a Web site, check correctness, check completeness, check undemand-

edness and classify undemandedness evidences. The Web service acts as a single access point to the core engine Verdi-M which implements the Web verification methodology in Maude. Following the standards, the architecture is also platform and language independent so as to be accessible via scripting environment as well as via client applications across multiple platforms.

5.2.2 XML API

In order for successful communications to occur, both the WebVerdiService and WebVerdiClient (or any user) must agree to a common format for the messages being delivered so that they can be properly interpreted at each end.The WebVerdiService Web service is developed by defining an API that encompasses the executable library of the core engine. This is achieved by making use of Oracle JDeveloper [], including the generation of WSDL for making the API available. The OC4J Server (the web server integrated in Oracle JDeveloper) handles all procedures common to Web service development. Synthesized error symptoms are also encoded as XML documents in order to be transferred from the WebVerdiService Web service to client applications as an XML response by means of the SOAP protocol.

5.2.3 Verdi-M

Verdi-M is the most important part of the tool. Here is where the verification methodology is implemented. This component is implemented in Maude language and is independent of the other system components.

Four Maude modules have been added to the original engine described in Chapter 3. These modules are invoked as separate processes when needed by the Java main thread. These modules, using the functionalities of the core, allow to:

Filter Rules: select among a set of rules the ones to be reverted;

Revert Rules: construct the reverted version of a rule;

Count Rules Usage: obtain the list of pages to which a set of rules apply within a Web Site;

Join Specifications: join two set of rules to create another bounded set, given that one of them is bounded.

With respect to the algorithm illustrated in Section 4.4, most of the steps are implemented combining Java and Maude functionalities. In particular:

1. **Preparing Step**: The operation executed in this step are implemented in Java, being only a filtering of rules based on the characters at the beginning of the line;

2. **Filtering Step**: The filtering operation is realized with the Maude module mentioned above, named "Filter Rules". Java creates a new process which proceed to the elaboration of the specification given, returning only the rules which does not simulates with their right part `r`, their left part `l`;

3. **Reverting Step**: The reverting operation is executed as well with a Maude module. This module has been named "Revert Rules" and given a set of rules, returns its reverted version;

4. **Checking Step**: The checking operation uses the Maude module "Count Rules Usage" which returns the list of pages touched by a given specification. This compared by Java with the list of pages of the Web Site allows to tell to the user if the specification check the Web Site as a whole;

5. **Searching Step**: The search of the errors is implemented using the WebVerdi-M Maude function of *checkCompletitud*. A different processes is create by Java for each of the reverted rule and their output is joined together creating a common list of undemandedness errors;

6. **Learning Step**: The learning operation happens when the user specify that an error evidence is erroneous in its generalized form or in that particular substitution of σ. While in the second case the rule with the σ substitution will be simply added by Java to the specification, in the first case before doing this addition, the Maude module "Join Specifications" is used to obtain a bounded set of rules. The rules that couldn't be added can be inferred founding the difference between the sets before the join and its result. With the help of a Java function then, only the errors which are originated by rules which couldn't be added are kept in the error list to be returned to the user for manual solving.

Overall the flow execution of the elaboration is Java controlled.

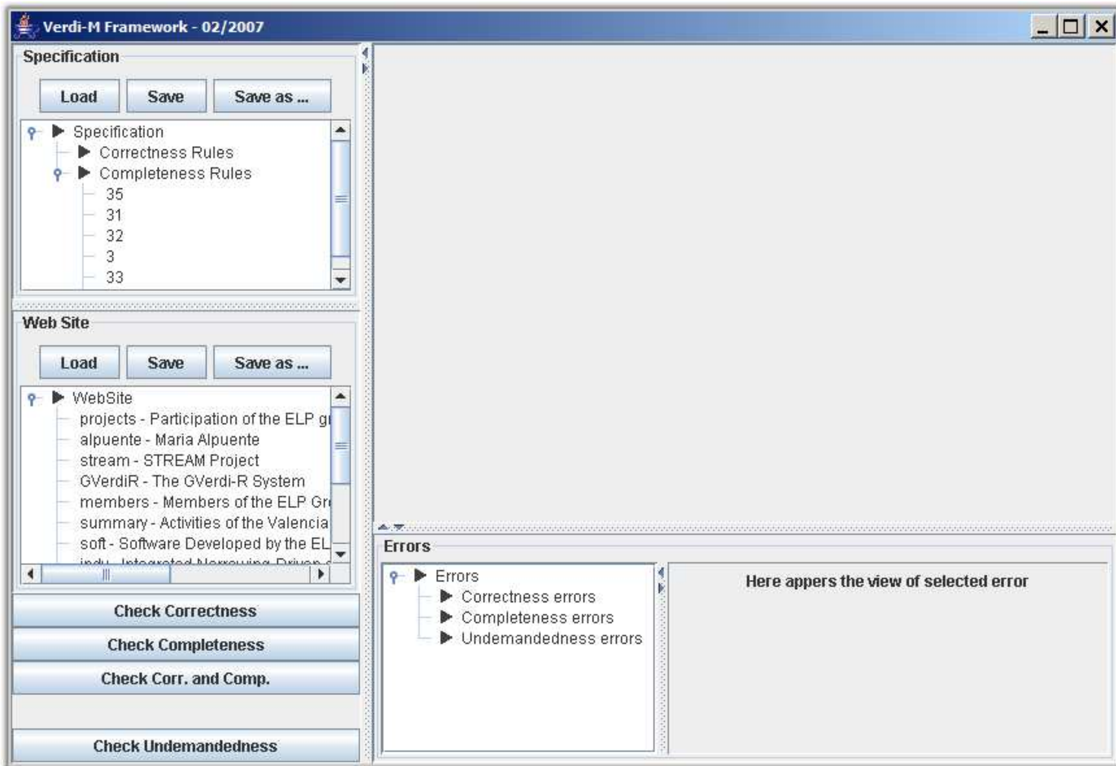

Figure 5.2: WebVerdiClient Snapshot

5.2.4 WebVerdiClient

The client consist of a Java graphical interface which allows to use the functionalities offered by the Web Server. The client is using the API specified in Section 5.4 to interact with it through a network connection and uses the Java Web Start functionalities to execute. The main goal was to provide an *intuitive* and *friendly* interface for the user.

WebVerdiClient is provided with a versatile, new graphical interface that offers three complementary views for both the specification rules and the pages of the considered Web site: the first one is based on the typical idea of accessing contents by using folders trees and is particularly useful for beginners; the second one is based on XML, and the third one is based on term algebra syntax. The tool provides all translations among the three views.

A snapshot of WebVerdiClient is shown in Figure 5.2.

Any other client using the API of the Web Server could be used.

5.2.5 DB

The WebVerdiService Web service needs to transmit abundant XML data over the Web to and from client applications. The common behavior of a user when using the tool is to modify the default rules provided for the Web specification and then verify a particular Web site. After modifying the Web specification, it would be necessary to send back to the service the considered specification as well as the whole Web site to verify. After the application invokes the WebVerdiService Web service with these two elements, synthesized errors are progressively generated and transferred to the client application. The standard Web service architecture requires client applications to wait until all data are received and then errors are sent, which could cause significant time lags in the application. In order to avoid this overhead and to provide better performance to the user, we use a local *MySQL07* data base where the Web site and Web errors are temporarily stored at the server side.

5.3 Maude

5.3.1 Functional Programming

Functional programming is a programming paradigm that conceives computation as the evaluation of mathematical functions, avoiding states and mutable data. This programming style emphasizes the application of functions, in contrast with imperative programming, which emphasizes changes in state and the execution of sequential command.[24]

Functional programming uses the notion of higher-order functions: functions are higher-order when they can take other functions as arguments and/or return functions as results. Where a traditional imperative program might use a loop to traverse a list, a functional style would often use a higher-order function, map, that takes as argument a function and a list, applies the function to each element of the list and returns a list of the results.

Since pure functions do not modify state, no data may be changed by parallel function calls, purely functional programs have no side effects. These functions are therefore thread-safe, which allow interpreters and compilers to use call-by-future evaluation [59]. In call-by-future evaluation the function's argument may be evaluated in parallel with the function body. The two threads of execution synchronize when the argument is needed

in the evaluation of the function body. If the argument is never used, the argument thread may be killed.

Referential transparency is enforced: if two expressions have equal values, then one can be substituted for the other in any larger expression without affecting the result of the computation.

Iteration is usually accomplished via recursion. Recursive functions invoke themselves, allowing an operation to be performed over and over. Recursion may require maintaining a stack, but tail recursion can be recognized and optimized by a compiler into the same code used to implement iteration in imperative languages.

The most significant difference with imperative programming is the avoidance of side effects, disallowing side effects completely. Providing in this way referential transparency, makes it easier to verify, optimize and parallelize programs, as well as to write automated tools to perform those tasks.

Functional programming languages have automatic memory management with garbage collection.

5.3.2 The Maude Language

Maude is a high-performance reflective language supporting both equational and rewriting logic programming, which is particularly suitable for developing domain-specific applications [40, 18]. As a matter of fact, it has a *clear* and *well-understood* semantics, which eases the development of complex systems. On the other hand, it is *expressive* enough to implement a wide range of applications, ranging from deterministic systems to highly concurrent ones. In addiction, the Maude language is not only intended for system prototyping, but it has to be considered as a real programming language with competitive performance.

The Maude programming language uses rewriting rules, as the others functional languages such as Haskell [26], ML [46], Scheme [28] and Lisp [22]. The rewriting logic allows to define complex computational modules as for concurrent or object programming.

The development of Maude originates from an international initiative which objective is to design a common platform for the investigation, teaching and application of the declarative languages.

A system written in Maude is composed by different modules. Each module includes

declarations of types and symbols, together with the equations which describes the logic of some of the symbol or of the functions calls. Symbols and equations defined in a functional module have a deterministic behavior which execution ends, as in the Example 5.3.1. Each functional module is defined within the words $fmod$ and $endfm$.

Example 5.3.1. *Factorial function in* Maude*:*

```
fmod FACT is
   INT .
   op _! : Int -> Int .
   var N : Int .
   --- factorial of N=0 if 1
   eq 0 ! = 1 .
   --- factorial of N>0 is N * (factorial of N-1)
   eq N ! = (N - 1)! * N [owise] .
endfm
```

The $Fmod$ declaration provide the deterministic behaviour of the module

Maude can use both the prefix and mixfix notation for the operations. The prefix notation is used normally in programming languages such as C++ and Java. The prefix notation is constituted by the name of the operator, followed by its arguments enclosed in parenthesis and separated by commas. To call the operator "+" on the variables "x" and "y" with the prefix notation this form is used: "+(x,y)". Obviously is easier to use it in operations like "bounce(ball3)".

The alternative is the mixfix notation, which in Maude can be declared specifying the places for the variables. While for the prefix notation the sum operator is defined as op "+", in the mixfix notion would be op "_ +_ ", see Example 5.3.2.

Once declared an operation en prefix or mixfix notation with the reserved word "op" followed by the name of the operation and ":" can be specified the name(s) of the type of variables used for the operation. Is than used the symbol "$->$" and the name of the type of the resulting variable, than a full stop.

Example 5.3.2. *Prefix versus mixfix notation:*

```
op + : Nat Nat -> Nat .
op _+_ : Nat Nat -> Nat .
```

In the rest of the section, we recall some of the most important features of the Maude language which have been conveniently exploited for the optimized implementation of the web site verification engine.

Equational attributes

A Maude program consists of functional *modules*, which define a typed signature Σ, a set of typed variables, and a set of equations implementing the functions in the signature Σ. Equational attributes are a means of declaring certain kinds of equational axioms for a particular binary operator of Σ which Maude uses efficiently in a built-in way. Some of the attributes supported are: `assoc` (associativity), `comm` (commutativity), `id:<identity name>`. Employing equational attributes not only avoids termination problems and leads to much more efficient term evaluation, but it also us allows to define more "flexible" data structures. As an example, let us describe how we model (part of) the internal representation of XML documents in our system.

The chosen representation slightly modifies the data structure provided by the Haskell HXML Library [19] by adding commutativity to the standard XML tree-like data representation. In other words, in our setting, the order of the children of a tree node is not relevant: e.g., $f(a,b)$ is "equivalent" to $f(b,a)$.

```
fmod TREE-XML is
sort XMLNode .
op RTNode : -> XMLNode .                    -- Root (doc) information item
op ELNode _ _ : String AttList -> XMLNode . -- Element information item
op TXNode _ : String -> XMLNode .           -- Text information items
--- ...  definitions of the other XMLNode types omitted ...
sorts XMLTreeList XMLTreeSeq XMLTree .
op Tree (_) _ : XMLNode XMLTreeList - > XMLTree .
subsort XMLTree < XMLTreeSeq .
op _,_ : XMLTreeSeq XMLTreeSeq -> XMLTreeSeq    [comm assoc id:null] .
op null : -> XMLTreeSeq .
op [_] : XMLTreeSeq -> XMLTreeList .
op [] : -> XMLTreeList .
endfm
```

In the previous module, the XMLTreeSeq constructor `_,_` is given the equational attributes `comm assoc id:null`, which allow us to get rid of parentheses and disregard the ordering among`XML` nodes within the list. The significance of this optimization will be clear when we consider rewriting XML trees with AC pattern matching.

As mentioned Maude include the possibility of specifying symbols with algebraic proprieties as associativity, commutativity and identity element, which simplify a lot the process of writing software.

For instance, lets see more in details the definition of XMLTreeListBasic from the Example 5.3.3.

Example 5.3.3. *XMLTreeListBasic:*

```
op _,_ : XMLTreeListBasic XMLTreeListBasic ->
            XMLTreeListBasic [comm assoc id: null].
```

Lets consider the three attributes separately. So lets analyze the associativity:

```
op _,_ : XMLTreeListBasic XMLTreeListBasic ->
            XMLTreeListBasic [assoc]
```

Specifying as associative the "," operator, which concatenates XMLTreeListBasic, allows to ignore the parenthesis when the arguments are written. Note that the arguments should be of the same type to be able to specify the associativity of the operator. With respect to the definition the Maude interpreter consider identical the arguments in Example 5.3.4.

Example 5.3.4. *XMLTreeListBasic examples:*

```
Tree (RTNode) [] , ( Tree (TXNode Hello) [] , null )
(Tree (RTNode) [] , Tree (TXNode Hello) []) , null
```

Is then possible to specify an identity element:

```
op null : -> XMLTreeListBasic .
op _,_ : XMLTreeListBasic XMLTreeListBasic ->
            XMLTreeListBasic [assoc id:null]
```

With respect to the definition of an identity element the Maude interpreter consider identical the arguments in Example 5.3.5.

Example 5.3.5. *XMLTreeListBasic examples:*

```
Tree (RTNode) [] , Tree (TXNode Hello) [] , null
Tree (RTNode) [] , Tree (TXNode Hello) []
null , Tree (RTNode) [] , null , Tree (TXNode Hello) [] , null
```

The last facility is to add the commutative propriety to the list:

```
op _,_ : XMLTreeListBasic XMLTreeListBasic ->
            XMLTreeListBasic [comm assoc id: null].
```

With respect to the definition of commutation the Maude interpreter, consider identical the arguments in Example 5.3.6.

Example 5.3.6. *XMLTreeListBasic examples:*

```
Tree (RTNode) [] , Tree (TXNode Hello) [] , null
Tree (TXNode Hello) [] , Tree (RTNode) []
null , Tree (RTNode) [] , null , null , Tree (TXNode Hello) []
```

AC pattern matching

The evaluation mechanism of Maude is based on rewriting modulo an equational theory E (i.e. a set of equational axioms), which is accomplished by performing *pattern matching modulo* the equational theory E. More precisely, given an equational theory *E*, a term *t* and a term *u*, we say that *t matches u modulo E* (or that *t E-matches u*) if there is a substitution σ such that $\sigma(t) =_E u$, that is, $\sigma(t)$ and *u* are equal modulo the equational theory *E*. When E contains axioms for associativity and commutativity of operators, we talk about *AC pattern matching*. AC pattern matching is a powerful matching mechanism, which we employ to inspect and extract the partial structure of a term. That is, we use it directly to implement the notion of homeomorphic embedding of Definition 3.4.1. Let us see Example 5.3.7.

Example 5.3.7. *Let us define an associative and commutative binary infix operator* $\star$ *along with the constants* a, b, c, d*. Then* $a \star d$ *AC-matches (a part of) the term* $t \equiv a \star b \star c \star d$ *since* t *is equivalent — modulo associativity and commutativity— to the term* $t' \equiv (a \star d) \star (b \star c)$*, and, hence,* $a \star d$ *trivially matches the first sub-term of* t'*. Thus, we are able to recognize the structure* $a \star d$ *as a substructure of the term* $a \star b \star c \star d$*.*

Meta-programming

Maude is based on rewriting logic [31], which is reflective in a precise mathematical way. In other words, there is a finitely presented rewrite theory $\mathcal{U}$ that is universal in the sense that we can represent in $\mathcal{U}$ (as a data) any finitely presented rewrite theory $\mathcal{R}$ (including $\mathcal{U}$ itself), and then mimic in $\mathcal{U}$ the behavior of $\mathcal{R}$. Roughly speaking, there exists a universal Maude program that is able to "reproduce" the computations of any other Maude program (including itself). This leads to novel meta-programming capabilities that can greatly increase software re-usability and adaptability, which have been exploited in our context to implement the semantics of correctness as well as completeness rules (e.g. implementing the homeomorphic embedding algorithm, evaluating conditions of conditional rules, etc.). Namely, during the partial rewriting process, functional modules are dynamically created and run by using the meta-reduction facilities of the language.

5.4 API

With respect to the API as reported in Appendix B the following addition has been introduced.

5.4.1 Data Representation

Web Pages

To identify a set of pages is used the notation of Example 5.4.1.

Example 5.4.1. *Example of page set:*

```
<pages>
   <pid>p1</pid>
   <pid> ... </pid>
   <pid>pn</pid>
</pages>
```

This XML element it is used to give to the client the list of pages to which some rewriting rule apply, as result of the method **pagesControlled**.

Errors

Possible undemandedness errors are defined depending on the type of error generated:

- Missing Page Errors are defined by the n-pla (r,W,sigma):

  r: Rule which generated the error;

  W: Web Site;

  sigma: the substitution(s) in `l` which produce(s) the error.

- Universal/Existential Errors are defined by the n-pla (r,P,sigma):

  r: Rule which generated the error;

  P: The set of identifiers of pages which does not comply to the rule;

  sigma: the substitution(s) in `l` which produce(s) the error.

Is to be noted that the rules used to catch undemandedness errors have almost the same XML representation of the completeness rules. The only difference between is in the undemandedness rules both the original rule and the reverted rule are included.

As for completeness rules a distinction is needed between different type of errors. This is accomplished using the special attribute <type></type>. The values of this attributes will be M (Missing Page), A (Universal), E (Existential). Its XML representation is as in Example 5.4.2.

Example 5.4.2. *Example of undemandedness error:*

```
<errorUndemandedness>
   <r> ... </r>
   <pages>
      <pid>p1</pid>
      <pid> ... </pid>
      <pid>pn</pid>
   </pages>
   <sigma> ... </sigma>
   <type> ... </type>
</errorUndemandedness>
```

A collection of possible undemandedness errors includes the identifiers of the web site and of the Specification to which refers. It is represented in XML as in Example 5.4.3.

Example 5.4.3. *Example of collection of errors:*

```
<collectionErrors WebSite=" ... " Specification=" ..." >
   <errorUndemandedness> ... </errorUndemandedness>
   <errorUndemandedness> ... </errorUndemandedness>
   <errorUndemandedness> ... </errorUndemandedness>
   ...
   <errorUndemandedness> ... </errorUndemandedness>
</collectionErrors>
```

This XML element it is used to give to the client the list of possible undemandedness error first, then the list of errors which couldn't be solved adding rules to the initial specification. More in details as output of the methods **checkUndemandedness** and **doActions**.

Actions

An action is how the system will deal with a specific error. To deal with undemandedness error four instances of actions are introduced:

addRule: To add as a new rule the rule originating the error, to the specification to ensure that any error of the same type won't be repeated;

addValue: To add as a new rule the rule originating the error with the value of its particular instance, to the specification to ensure that particular error won't be repeated;

notAnError: To let the system know that evidence is not actually an error, ensuring that any of the evidences of the same type won't be detected as error again;

noActionRequired: If no action is requested to solve a particular error.

Its XML representation is as in Example 5.4.4.

Example 5.4.4. *Example of action:*

```
<action>
   <type> ... </type>
</action>
```

To repair errors, each of those is paired to an action. In the case of repairing each undemandedness error without adding a rule to the initial specification, a specific action for every of the errors is needed. It includes the identifiers of the web site and of the Specification to which refers.

Its XML representation is as in Example 5.4.5.

Example 5.4.5. *Example of collection of PairErrorActions:*

```
<collectionPairErrorAction WebSite=" ... " Specification=" ..." >
   <pairErrorAction>
      <errorUndemandedness> ... </errorUndemandedness>
      <action> ... </action>
   </pairErrorAction>
   ...
   <pairErrorAction>
      <errorUndemandedness> ... </errorUndemandedness>
      <action> ... </action>
   </pairErrorAction>
</collectionPairErrorAction>
```

This XML element it is used to give to the server the list of repairing actions required by the user. Its one of the input of the method **doActions**.

5.4.2 Methods

To decouple the actions on the specifications from the client execution, some functions have been added (storeSpecifications, recoverySpecifications and removeSpecifications). The introduction of these three new functions impacts also on all previous methods definition (see B.2) which uses the SPEC parameter, determining the use of the idSpec instead (checkCorrectness, checkCompleteness and changeCS).

Methods Details

storeSpecification(specification)
Store a Specification locally to the server. **Input:** specification: The specification to store. **Output:** The identifier of the specification stored in the server.
retrieveSpecification (idSpec)
Load a specification. **Input:** idSpec: The identifier of the specification. **Output:** The specification if the identifiers exists.
removeSpecification(idSpec)
Delete the specification stored in the server. **Input:** idSpec: the identifier of the specification as stored in the server. **Output:** True if the specification has been successfully deleted, False otherwise.
checkUndemandedness (idWS, idSpec, Mode)
Return a collection of possible undemandedness errors of a Web Site w.r.t. the specification. **Input:** idWS: The identifier of the Web Site as stored in the server. idSpec: The identifier of the specification as stored in the server. Mode: Specify the type of execution (default, clean). **Output:** Collection of possible undemandedness errors which includes idWS and idSpec.
doActions (proposedPairErrorActionCollection)
Return a collection of undemandedness errors still to be repaired. **Input:** idWS: The identifier of the Web Site as stored in the server.

idSpec: The identifier of the specification as stored in the server.
proposedPairErrorActionCollection: List of proposed PairErrorAction.

Output:
Return a collection undemandedness errors still to be repaired which includes idWS and idSpec if all pages are checked with the current specification, else the list of pages not checked.

6

Case Study

This Chapter is structured as follows. Section 6.1 defines the scenario in which the new WebVerdi-M is applied and Section 6.2 outlines the outcome of its application.

6.1 Scenario Description

6.1.1 Web Site

To prove our theoretical results in practice we decided to use our tool in a real scenario. The real scenario is constituted by the web site of the ELP research group (Extensions of Logic Programming) research group – http://www.dsic.upv.es/users/elp/elp.html.

Its structure is the most common nowadays on the Internet: a web site realized in plain HTML, which supposedly should respect HTML 4.0 Transitional [49] standards, but in fact cannot be validate as so by the W3C Markup Validation Service [56].

6.1.2 Preparation

To be able to run the WebVerdi-M framework upon the server two issues has to be tackled: the XML representation of the content and the formalization of a specification.

XML Representation Generation

The only way to generate the XML needed for the undemandedness analysis is the manual creation of such a document for the time being. No automatic tool in fact is capable to give meaning to the information contained in an HTML document and associate it to an XML tag.

For sake of simplification we than only considered fourteen pages of such web site, here is the complete list, identified by page name:

elp: the main index file;

pres2: page presenting the research group;

summary: summary of the activities of the group;

papers: paper published by the group;

members: member list;

alpuente: a page of one of the members of the group;

gvidal: a page of one of the members of the group;

projects: the projects of the group;

```
3)   link(id(X),name(Y),type('Internal')) -> #p(#id(X),#title(Y)) <E>
18) coordinators(person(name(X),surname(Y),
                                   university('Technical U. Valencia UPV'))) ->
     #stafflist(person(name(X),surname(Y))) <E>
31) project(link(id(X),type('Internal')),details(Y)) ->
     #p(#id(X),#content(#extendedproject(#acronym(Y)))) <E>
32) project(link(id(X),type('Internal')),id(Y)) ->
     #p(#id(X),#content(#extendedproject(#code(Y)))) <E>
33) project(link(id(X),type('Internal')),coordinators(Y)) ->
     #p(#id(X),#content(#extendedproject(#coordinators(Y)))) <E>
35) address(name(X),surname(Y)) -> #stafflist(person(name(X),surname(Y))) <E>
```

Figure 6.1: Specification

stream: a page of a project of the group;

trend: a page of a project of the group;

soft: the software produced by the group;

indy: a page of a software produced by the group;

GVerdi: a page of a software produced by the group;

GVerdiR: a page of a software produced by the group.

Specification Generation

A large completeness specification has been created to enforce consistency of the data of the site. Among all the completeness rules are shown in Figure 6.1 only the one which than will impact on the undemandedness check.

To be noted is a completeness check with the given rules won't find any error.

6.2 Undemandedness Analysis Outcome

The undemandedness check returns the list of possible undemandedness errors which actually are all considered errors, therefore marked as such as shown in Figure 6.2. In contrast to the initial completeness check didn't identify any error, the undemandedness analysis recognized 20 possible error instances.

Being able to add all new rules to the initial specification, new completeness check will identify what has been considered undemandedness errors, as shown in Figure 6.3.

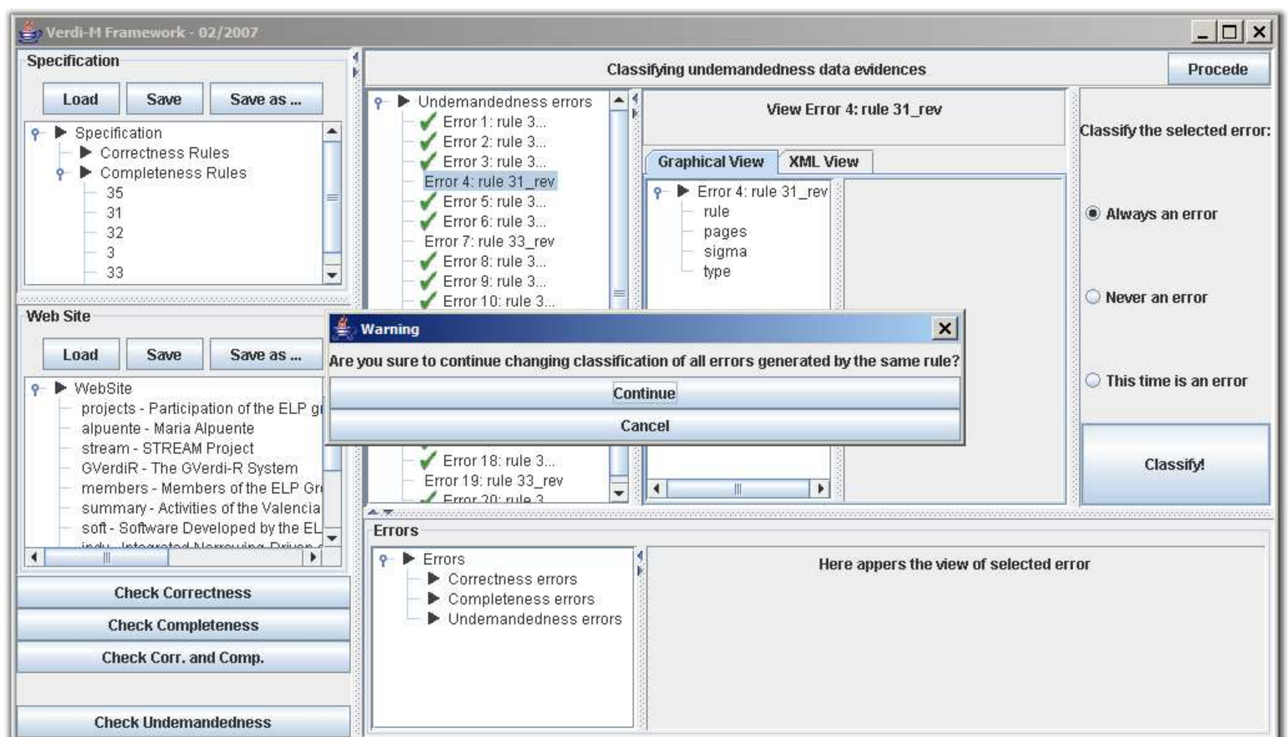

Figure 6.2: Options for undemandedness check

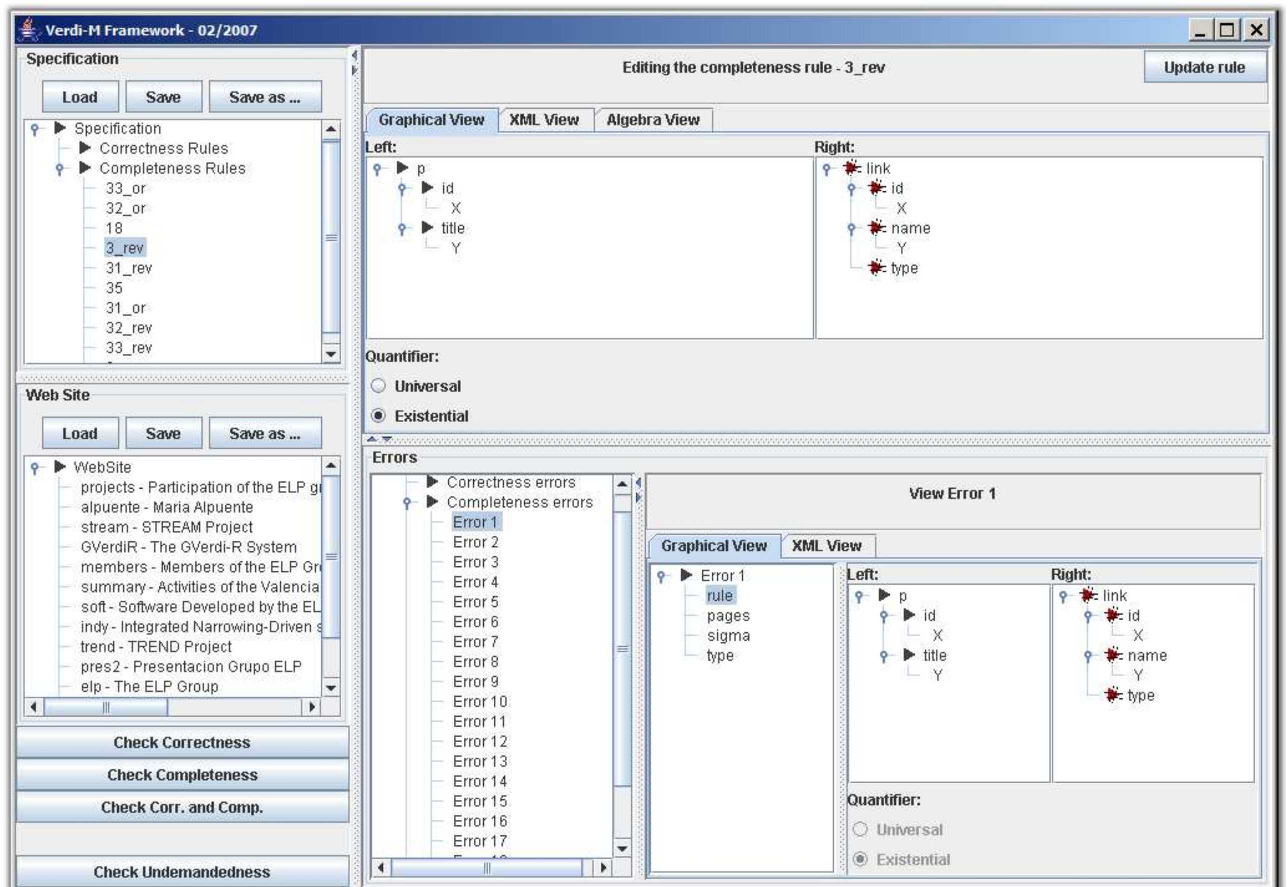

Figure 6.3: Options for undemandedness check

7
Conclusions

This Chapter is structured as follows. Section 7.1 illustrate the achievements of the approach followed and Section 7.2 outline possible future research directions.

7.1 Achievements

In the literature on Web management, Web sites verification has mainly a syntactic focus with a particular concern for the accessibility and usability perspective [7, 8]. The formal, semantic verification of Web sites is still an open issue.

This work addresses in particular the management of "extra information" formalizer as *undemanded data*, integrating the WebVerdi-M framework with the needed tools for such analysis. This is achieved exploiting Maude's capabilities such as associative commutative pattern matching and meta-programming.

From both the example used in Section 4.5 and more importantly the case study illustrated in Chapter 6 is clear how our undemandedness analysis play an important role in the detection of inconsistent and outdated data.

Extending WebVerdi-M meant to design of a new block of API calls which expanded the service-oriented architecture of the framework, as well as integrating the WebVerdi-Client with a new graphical interface to manage possible undemandedness errors.

This methodology to detect undemanded data w.r.t. a specification in Web Sites has a number of positive effect:

1. Being able of localizing such data, our tool is a first step toward the management of the design, construction and maintenance of Web with an engineering prospective;

2. Webmaster are enabled to offer a consistent data-set, which results in a web site that communicates a sense of order, unity and ease of use. These qualities are very much appreciated by visitors, especially if they find the information they need. This will encourage repeat visitors to the web site and pulling in new ones;

3. Coherently updated data will ensure the web site to communicate competence and professionalism which enhances the owner image, being the "virtual face" to the stakeholders;

4. The capability to learn patterns of undemandedness endowed in our algorithm allows to reuse the information assimilated in future undemandedness analysis. Nevertheless in can be inhibited if the user consider it superfluous during a particular execution.

7.2 Future Research Directions

In this Section are described possible future investigation directions within the WebVerdi-M framework [39].

7.2.1 Automatic HTML Translation to XML

In Section 6.1 is evident the practical difficulties to apply such an approach to a real web site. Nowadays there is no tool which allow the automatic translation from plain HTML to the XML form needed to apply the WebVerdi-M framework. This limits the application of the tool to web site specifically designed for XML, which anyhow includes most of the new web applications. To use the framework with older web site without such a manual approach, an automatic tool could be built.

7.2.2 Strategies to Optimize Completeness Repairing

In Section 3.5 is presented how to optimize repairing strategy for correctness errors. A similar analysis could be done for completeness errors. A main object could be to order them to apply than repairing actions. A first approximation could be to order the rules which originates them. This order could respect the type of dependency between the left and the right part of the rules. Will be then necessary to identify how to process the quantifiers and how those impacts on the processing order. Another issue is the impact of the repair of an error upon the other errors of the web site.

7.2.3 Repairing Strategies Upon Node Labels

In Section 3.5 has been illustrated how is possible to eliminate errors from a web site using repairing actions, applied in the form of a strategy. The general schema of those strategies is to consider a term of the web site tree and apply to it an action (for example *change* or *delete*), which will affect all the sub-terms of the considered term. In an ideal strategy instead actions could be applied at the root of the tree and eliminate all the errors of that type. This ideal schema has the lamentable effect to eliminate irregardless all the information of such a type. To be able to avoid this side-effect, could be possible to define repairing actions which would affect only the nodes where the error is produced or actions which uses as arguments only a subset of sub-terms.

7.2.4 Extension of the Specification Language

The undemandedness analysis described in Chapter 4 brought to the consideration of the limits of the actual specification language. A extension which would increase the flexibility and the expressive power of the language would be the incorporating of first order logic as well as the inclusion a conditional processing, which would allow to define rules such as:

$$(l_1 \rightharpoonup r_1) \text{ and } (l_2 \rightharpoonup r_2) \rightharpoonup r_3$$

$$\text{if } (l_1 \rightharpoonup r_1) \text{ them } (l_2 \rightharpoonup r_2)$$

7.2.5 Analysis of the Filtering Language for XML

The language developed within the framework WebVerdi-M to filter the XML/XHTML documents has been designed to be as simple and immediate as possible, but has an drawback: is that is "yet another language" to be learned. To overcome this would be possible create an automatic tool to translate a to obtain the filtering expression from an XQuery [54] expression. XQuery in fact is definitely more powerful. To achieve so is necessary to define the subset of the XQuery language equivalent to the formalism used in WebVerdi-M, in order to realize a mapping between the two.

Example Data

In this section is included part of the code which is mentioned in Section 4.5.

A.1 Web Site XML

```
<website>
   <page>
      <id>p1</id>
      <name>Home</name>
      <data>
         <npage>
            <title>ELP Homepage</title>
            <content>
               <text> This is ELP homepage.
You can find all the people and the project of the research group. People:
               </text>
               <peoplelist>
                  <item>
                     <name>Daniel</name>
                     <surname>Romero</surname>
                     <link>Daniel_Romero</link>
                  </item>
                  <item>
                     <name>Matteo</name>
                     <surname>Zanella</surname>
                     <link>Matteo_Zanella</link>
                  </item>
                  <item>
                     <name>Maria</name>
                     <surname>Alpuente</surname>
                     <link>Maria_Alpuente</link>
                  </item>
                  <item>
                     <name>Beat</name>
                     <surname>A</surname>
                     <link>Beat_A</link>
                  </item>
```

```
            </peoplelist>
            <text>Projects: </text>
            <projectlists>
               <projectlist>
                  <item>
                     <id>1</id>
                     <name>Project 1</name>
                     <link>1</link>
                  </item>
                  <item>
                     <id>2</id>
                     <name>Project 2</name>
                     <link>2</link>
                  </item>
                  <item>
                     <id>3</id>
                     <name>Project 3</name>
                     <link>3</link>
                  </item>
                  <item>
                     <id>4</id>
                     <name>Project 4</name>
                     <link>4</link>
                  </item>
                  <item>
                     <id>5</id>
                     <name>Project 5</name>
                     <link>5</link>
                  </item>
               </projectlist>
            </projectlists>
         </content>
      </npage>
   </data>
</page>

<page>
   <id>p2</id>
   <name>Daniel Home</name>
   <data>
      <hpage>
         <id>Daniel_Romero</id>
         <name>Daniel</name>
         <surname>Romero</surname>
         <tel>1234</tel>
         <team>
            <person>
               <name>Matteo</name>
               <surname>Zanella</surname>
               <link>Matteo_Zanella</link>
            </person>
```

```
         </team>
         <projects>
            <project>
               <id>4</id>
               <name>Project 4</name>
               <link>4</link>
            </project>
            <project>
               <id>1</id>
               <name>Project 1</name>
               <link>1</link>
            </project>
            <project>
               <id>3</id>
               <name>Project 3</name>
               <link>3</link>
            </project>
            <project>
               <id>5</id>
               <name>Project 5</name>
               <link>5</link>
            </project>
         </projects>
      </hpage>
   </data>
</page>

<page>
   <id>p3</id>
   <name>Teo Home</name>
   <data>
      <hpage>
         <id>Matteo_Zanella</id>
         <name>Matteo</name>
         <surname>Zanella</surname>
         <tel>1234</tel>
         <team>
            <person>
            <name>Daniel</name>
            <surname>Romero</surname>
            <link>Daniel_Romero</link>
         </person>
         </team>
         <projects>
            <project>
               <id>2</id>
               <name>Project 2</name>
               <link>2</link>
            </project>
         </projects>
      </hpage>
```

```
   </data>
</page>

<page>
   <id>p4</id>
   <name>Maria Home</name>
   <data>
      <hpage>
         <id>Maria_Alpuente</id>
         <name>Maria</name>
         <surname>Alpuente</surname>
         <tel>1234</tel>
         <team>Blu</team>
         <projects>
            <project>
               <id>1</id>
               <name>Project 1</name>
               <link>1</link>
            </project>
         </projects>
      </hpage>
   </data>
</page>

<page>
   <id>p5</id>
   <name>Sonia Home</name>
   <data>
      <hpage>
         <id>Sonia_P</id>
         <name>Sonia</name>
         <surname>P</surname>
         <tel>1234</tel>
         <team>Red</team>
         <projects>
            <project>
               <id>1</id>
               <name>Project 1</name>
               <link>1</link>
            </project>
         </projects>
      </hpage>
   </data>
</page>

<page>
   <id>p6</id>
   <name>P1 Home</name>
   <data>
      <ppage>
         <id>1</id>
```

```
            <name>Project 1</name>
            <owner>
               <person>
                  <name>Daniel</name>
                  <surname>Romero</surname>
                  <link>Daniel_Romero</link>
               </person>
            </owner>
            <budget>2</budget>
         </ppage>
      </data>
   </page>

   <page>
      <id>p7</id>
      <name>P2 Home</name>
      <data>
         <ppage>
            <id>2</id>
            <name>Project 2</name>
            <owner>
               <person>
                  <name>Matteo</name>
                  <surname>Zanella</surname>
                  <link>Matteo_Zanella</link>
               </person>
            </owner>
            <budget>2</budget>
         </ppage>
      </data>
   </page>

   <page>
      <id>p8</id>
      <name>P3 Home</name>
      <data>
         <ppage>
            <id>3</id>
            <name>Project 3</name>
            <owner>
               <person>
                  <name>Daniel</name>
                  <surname>Romero</surname>
                  <link>Daniel_Romero</link>
               </person>
            </owner>
            <budget>2</budget>
         </ppage>
      </data>
   </page>
```

```
   <page>
      <id>p9</id>
      <name>P5 Home</name>
      <data>
         <ppage>
            <id>5</id>
            <name>Project 5</name>
            <owner>
               <person>
                  <name>Daniel</name>
                  <surname>Romero</surname>
                  <link>Daniel_Romero</link>
               </person>
            </owner>
            <budget>2</budget>
         </ppage>
      </data>
   </page>

   <page>
      <id>p10</id>
      <name>P4 Home</name>
      <data>
         <ppage>
            <id>4</id>
            <name>Project 4</name>
            <owner>
               <person>
                  <name>Daniel</name>
                  <surname>Romero</surname>
                  <link>Daniel_Romero</link>
               </person>
            </owner>
            <budget>244</budget>
         </ppage>
      </data>
   </page>

</website>
```

B

Old API

In this section are illustrated the set of methods which allows to interact with the Web Service and the how the data should be encapsulated in order to interact with it. It is divided in two sections: data representation and methods........

B.1 Data Representation

B.1.1 Web Site

A website is composed by a set of pages, which is defined by the triple (id,name,data) where:

id: is the identifies of the page;

name: is the name of the page;

data: is the XML content of the page.

An example of page is Example B.1.1.

Example B.1.1. *Example of page set:*

```
<page>
   <id>p1</id>
   <name>biblio.htm</name>
   <data>
      <biblioteca>
         <libro ndoc="99231">
            <isbn>8437607000</isbn>
            <autor>Rojas, Fernando de</autor>
            <titulo>La Celestina</titulo>
         </libro>
         <libro ndoc="158290">
            <isbn>8403870485</isbn>
```

```
            <autor>Homero</autor>
            <titulo>Iliada</titulo>
         </libro>
         <libro ndoc="181227">
            <isbn>8466401040</isbn>
            <autor>Kafka, Franz</autor>
            <titulo>La metamorfosi</titulo>
         </libro>
      </biblioteca>
   </data>
</page>
```

A Web Site will be then represented by a collection of web pages, which in XML would look like Example B.1.2.

Example B.1.2. *Example of web site:*

```
<webSite>
   <page> ... </page>
   <page> ... </page>
   <page> ... </page>
   ...
   <page> ... </page>
</webSite >
```

B.1.2 Rules

There are different kinds of rules, which has different characteristics. Different XML representations are therefore needed.

Correctness rules

A correctness rule is defined by the triple (l,r,C), where:

l: is the left part of the rule;

r: is the right part of the rule;

C: is the condition if defined.

An attribute is added to identify the rule among the specification set.

Its XML representation is a in Example B.1.3.

Example B.1.3. *Example of correctness rule:*

```
<ruleCorrectness Name="...">
   <left> ... </left>
   <right> ... </right>
   <condition> ... </condition>
</ruleCorrectness>
```

Completeness rules

A completeness rule is defined by the triple (l,r,q), where:

l: is the left part of the rule;

r: is the right part of the rule;

q is the quantifier which can be E (Existential) o A (Universal).

An attribute is added to identify the rule among the specification set.

Its XML representation is as in Example B.1.4.

Example B.1.4. *Example of completness rule:*

```
<ruleCompletness Name="...">
   <left> ... </left>
   <right> ... </rigth>
   <quantifier> ... </quantifier>
</ruleCompletness>
```

To mark terms the attribute Mark has to be added to the element $< attrib >$ which precedes it.

Defining parts of rules

The following definitions can apply to both left and right part of both correctness and completeness.

To be able to port to XML the structure of our language of specification which allow the use of forms as $f(X, Y)$, is needed to introduce the way of including more then one variable. This is achieved with the use of the element $< atrib >< /atrib >$. See Example B.1.5.

Example B.1.5. *Example of* $< atrib >$ *element:*

```
<f>
   <atrib>X</atrib>
   <atrib>Y</atrib>
</f>
```

If the function include another function, the element won't change. For instance in the function $f(g(X, Y))$ where f, g are function and X, Y are variables the corresponding XML is as in Example B.1.6.

Example B.1.6. *Example of* $< atrib >$ *element:*

```
<f>
   <atrib>
      <g>
         <atrib>X</atrib>
         <atrib>Y</atrib>
      </g>
   </atrib>
</f>
```

Following the above is applied to the rules of correctness and completeness.

Given the correctness rule $f(X,Y) \rightharpoonup error$ which means “A page is erroneous if f have two arguments, its XML representation will be as in Example B.1.7.

Example B.1.7. *Full example of a correctness rule:*

```
<ruleCorrectness Name="Rule 1">
   <left>
      <atrib>
         <f>
            <atrib>X</atrib>
            <atrib>Y</atrib>
         </f>
      </atrib>
   </left>
   <right> </rigth>
   <condition> </condition>
</ruleCorrectness>
```

Given the completeness rule $f(g(X)) \rightharpoonup h(g(X)))\langle E\rangle$ which means “If $f(g(X))$ exists, then should exist at least one page containing $h(g(X))$, its XML representation will be as in Example B.1.8.

Example B.1.8. *Full example of a completeness rule:*

```
<ruleCompletness Name="Rule 1">
   <left>
      <atrib>
         <f>
            <atrib>
               <g>
                  <atrib>X</atrib>
               </g>
            </atrib>
         </f>
      </atrib>
   </left>
   <right>
      <atrib Mark=true>
```

```
         <h>
            <atrib Mark=true>
               <g>
                  <atrib>X</atrib>
               </g>
            </atrib>
         </h>
      </atrib>
   </rigth>
   <quantifier>E</quantifier>
</ruleCompletness>
```

Note that to keep the coherency marking a term is done using the appropriate attribute in the $< atrib >$ element as in the example above. This is the reason why both elements $< left >$ and $< rigth >$ have as first element $< atrib >$.

A specification is a collection of rules (completeness and correctness). Its XML representation is as in Example B.1.9.

Example B.1.9. *Specification example:*

```
<specification>
   <ruleCorrectness Name=xxx> ... </ruleCorrectness>
   <ruleCompletness Name=xxx> ... </ruleCompletness>
   <ruleCompletness Name=xxx> ... </ruleCompletness>
   ...
   <ruleCorrectness Name=xxx> ... </ruleCorrectness>
</specification>
```

B.1.3 Errors

Correction errors are defined by the n-pla (pid,w,l,sigma,C), where:

pid: is the page identifier of the page containing errors;

w: is the position within the page where the error is located, this position is defined by a vector of integers as [1, 4, 5], in XML is equivalent to a string in the form "1.4.5;

l: the left part of the rules which produce the error;

sigma: the substitution(s) in `l` which produce(s) the error;

C: the condition which produces the error.

Its XML representation is as in Example B.1.10.

Example B.1.10. *Correctness error example:*

```
<errorCorrectness>
   <pid>p1</pid>
   <w>1.2</w>
   <l>
      <autor>X</autor>
   </l>
   <sigma>
      <sust>
          <var>X</var>
          <value>Rojas, Fernando de</value>
      </sust>
   </sigma>
   <condition>X=Rojas, Fernando de</condition>
</errorCorrectness>
```

The completeness error are defined depending on the type of error generated:

- Missing Page Errors are defined by the triple (r,W,sigma):

  r: Rules which generated the error;

  W: Web Site;

  sigma: the substitution(s) in `l` which produce(s) the error.

- Universal/Existential Errors are defined by the triple (r,P,sigma):

  r: Rules which generated the error;

  P: The set of identifiers of pages which does not comply to the rule;

  sigma: the substitution(s) in `l` which produce(s) the error.

Being the Web Site a set of pages, to be able to distinguish between different type of error an attribute is needed. The values of this attributes will be M (Missing Page), A (Universal), E (Existential). Its XML representation is as in Example B.1.11.

Example B.1.11. *Completeness error example:*

```
<errorCompletness>
   <r> ... </r>
   <pages>
      <pid>p1</pid>
      <pid> ... </pid>
      <pid>pn</pid>
   </pages>
   <sigma> ... </sigma>
   <type> ... </type>
</errorCompletness>
```

A collection of errors is a set of errors of completeness or correctness which is represented in XML as in Example B.1.12.

Example B.1.12. *Error collection example:*

```
<collectionErrors>
   <errorCorrectness> ... </errorCorrectness>
   <errorCorrectness> ... </errorCorrectness>
   <errorCorrectness> ... </errorCorrectness>
   ...
   <errorCorrectness> ... </errorCorrectness>
</collectionErrors>

   ...

<collectionErrors>
   <errorCompletness> ... </errorCompletness>
   <errorCompletness> ... </errorCompletness>
   <errorCompletness> ... </errorCompletness>
   ...
   <errorCompletness> ... </errorCompletness>
</collectionErrors>
```

B.1.4 Actions

An action is the primitive to repair the website. There are four of those:

change (pid,w,t): Change the sub-term of the page pid in the position w for the term t;

insert(pid,w,t): Add to the page pid in the position w the term t;

delete(pid,t): Delete the term t from the page pid;

add(p, idWS): Add the page p tot he website idWS.

Its XML representation is as in Figure B.1.13.

Example B.1.13. *Action example:*

```
<action>
   <type>change</type>
   <pid>p1</pid>
   <w>1.2</w>
   <t>
      <autor>Perez, Pepito</autor>
   </t>
</action>
```

B.1.5 PairErrorAction

To repair errors, each of those is paired to an action. In the case of repairing an error generated by a Universal rule, is needed an action of every of the pages not complying the rule.

Its XML representation is as in Example B.1.14.

Example B.1.14. *PairErrorAction example:*

```
<pairErrorAction>
   <errorCorrectness> ... </errorCorrectness>
   <action> ... </action>
</pairErrorAction>

   ...

<pairErrorAction>
   <errorComplettness> ... </errorComplettness>
   <action> ... </action>
</pairErrorAction>
```

B.2 Methods

To optimize the data transfer some functions are added and defined below (storeWebSite, recoveryPage, recoveryWebSite and removeWebSite).

B.2.1 Methods Details

storeWebSite (WebSite)
Store a Web Site locally to the server running the engine. **Input:** WebSite: The Web Site to store. **Output:** The identifiers of the Web Site stored in the server.

retrievePage(idP, idWS)
Load a page of the a specified Web Site. **Input:** idP: The identifier of the page. idWS: The identifier of the Web Site as stored in the server.

Output:
The page of the website, if the identifiers exists.

retrieveWebSite(idWS)

Load the Web Site stored in the server.

Input:
idWS: the identifiers of the website as stored in the server.

Output:
The Web Site, if the identifier exists.

removeWebSite(idWS)

Delete the Web Site stored in the server.

Input:
idWS: the Identifiers of the Web Site as stored in the server.
Output:
True if the website has been successfully deleted, False otherwise.

checkCorrectness(idWS SPEC)

Returns a collection of correctness errors of a Web Site, in respect of the given specification.

Input:
idWS: The identifier of the Web Site as stored in the server.
SPEC: The specification to validate B.1.2.

Output:
A collection of correctness errors as B.1.3.

checkCompletness(idWS SPEC)

Returns a collection of completeness errors of a Web Site, in respect to the given specification.

Input:
idWS: The identifier of the Web Site as stored in the server.
SPEC: The specification to validate B.1.2.

Output:
A collection of completeness errors as B.1.3.

fixErrorCorrectnessByDelete (errorCorrecness idWS)

Solve completeness error with deletion action.

Input:
errorCorrecness: error to be fixed.
idWS: The identifier of the Web Site as stored in the server.

Output:
True in case the error was successfully deleted, False otherwise.

changeCS (errorCorrecness SPEC idWS)

Solve automatically the correctness error with a Constraint Solver.

Input:
errorCorrecness: error to be fixed.
SPEC: The specification to validate B.1.2.
idWS: The identifier of the Web Site as stored in the server.

Output:
True in case the error was successfully solved, False otherwise.

fixErrorCompletness (pairErrorAction SPEC idWS)

Solve a completeness error.

Input:
pairErrorAction: pair error-action B.1.5.
SPEC: The specification to validate B.1.2.
idWS: The identifier of the Web Site as stored in the server.

Output:
True in case the error was successfully solved, False otherwise.

fixErrorCompletnessByStrategyM (idWS pairEA_Set)

Solve a completeness error using a Minimal strategy.

Input:
idWS: The identifier of the Web Site as stored in the server.
pairEA_Set: The set of couples ¡error,action¿ B.1.5.

Output:
True in case the error was successfully solved, False otherwise.

fixErrorCompletnessByStrategyMNO (idWS pairEA_Set)

Solve a completeness error using a Minimal non-overlapping strategy.

Input:
idWS: The identifier of the Web Site as stored in the server.
pairEA_ Set: The set of couples ¡error,action¿ B.1.5.

Output:
True in case the error was successfully solved, False otherwise.

Bibliography

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web. From Relations to Semistructured Data and XML.* Morgan Kaufmann, 2000.

[2] S. Adve and K. Gharachorloo. Shared Memory Consistency Models. *IEEE Computer*, 29(12):66–76, 1996.

[3] A. Alegre. Developing Coherent Information Architectures. *Sustainable Development Communications Network*, 2001.

[4] M. Alpuente, D. Ballis, and M. Falaschi. Automated Verification of Web Sites Using Partial Rewriting. *Software Tools for Technology Transfer*, 2004.

[5] M. Alpuente, D. Ballis, and M. Falaschi. A Rewriting-based Framework for Web Sites Verification. *ENTCS*, 124(1), 2005. Proc. of 1st Int'l Workshop on Ruled-Based Programming (RULE'04).

[6] M. Alpuente, D. Ballis, M. Falaschi, P. Ojeda, and D. Romero. Web Verdi-M. In *3rd Annual Conference ICT for EU-India Cross-Cultural Dissemination. Udine, Italy*, Dec. 2006. Available at: `http://euindia.dimi.uniud.it/`.

[7] M. Alpuente, S. Escobar, and M. Falaschi (Eds.). *Automated Specification and Verification of Web Sites, 1st Int'l Workshop WWV'05*, volume 157(2). Elsevier, 2006.

[8] M. Alpuente, S. Escobar, and M. Falaschi (Eds.). *Automated Specification and Verification of Web Systems, 2nd Int'l Workshop WWV'06*. IEEE Computer Society Press, 2007.

[9] K. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[10] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[11] D. Ballis. *Rule-based Software Verification and Correction*. PhD thesis, University of Udine and Technical University of Valencia, 2005.

[12] E. Bertino, M. Mesiti, and G. Guerrin. A Matching Algorithm for Measuring the Structural Similarity between an XML Document and a DTD and its Applications. *Information Systems*, 29(1):23–46, 2004.

[13] M. Bezem. *TeReSe, Term Rewriting Systems*, chapter Mathematical background (Appendix A). Cambridge University Press, 2003.

[14] L. Cabrera, C. Kurt, and D. Box. An introduction to the web services architecture and its specifications. *MSDN Library*, 2004.

[15] L. Capra, W. Emmerich, A. Finkelstein, and C. Nentwich. XLINKIT: a Consistency Checking and Smart Link Generation Service. *ACM Transactions on Internet Technology*, 2(2):151–185, 2002.

[16] N. Dershowitz and D. Plaisted. Rewriting. *Handbook of Automated Reasoning*, 1:535–610, 2001.

[17] T. Despeyroux and B. Trousse. Semantic verification of web sites using natural semantics, 2000.

[18] S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL model checker and its implementation. In *Model Checking Software: Proc. 10 th Intl. SPIN Workshop*, volume 2648 of *LNCS*, pages 230–234. Springer, 2003.

[19] Joe English. The HXML Haskell Library, 2002. Available at: `http://www.flightlab.com/ joe/hxml/`.

[20] M. Fernandez, D.Florescu, A.Levy, and D. Suciu. Web-Site Management: The STRUDEL Approach. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, pages 14–20, 1998.

[21] M. F. Fernandez and D. Suciu. Optimizing Regular Path Expressions Using Graph Schemas. In *Proc. of Int'l Conference on Data Engineering (ICDE'98)*, pages 14–23, 1998.

[22] P. Grahamr. *ANSI Common Lisp*. Prentice-Hall, 1995.

[23] M. Hillyer. An introduction to database normalization. *MySQL Developer Zone*, 2006. Available at: `http://dev.mysql.com`.

[24] P. Hudak. Conception, evolution, and application of functional programming languages. *ACM Computing Surveys*, 1989.

[25] IBM. Service Oriented Architecture, 2007. Available at: `http://www-306.ibm.com/software/solutions/soa`.

[26] S. Jones. Introduction to Haskell, 2007. Available at: `http://www.haskell.org`.

[27] A.C. Kakas and L.A. Kowalski. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1993.

[28] R. Kelsey, W. Clinger, and J. Rees. Revised Report on the Algorithmic Language Scheme. *ACM SIGPLAN Notices*, 1998.

[29] J.W. Klop. Term Rewriting Systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume I, pages 1–112. Oxford University Press, 1992.

[30] M. Leuschel. Homeomorphic Embedding for Online Termination of Symbolic Methods. In T. Æ. Mogensen, D. A. Schmidt, and I. H. Sudborough, editors, *The Essence of Computation*, volume 2566 of *Lecture Notes in Computer Science*, pages 379–403. Springer, 2002.

[31] N. Martí-Oliet and J. Meseguer. Rewriting Logic: Roadmap and Bibliography. *Theoretical Computer Science*, 285(2):121–154, 2002.

[32] Sun Microsystems. JavaTM 2 SDK, Standard Edition Documentation, Version 1.4.2, 2003. Available at: `http://java.sun.com`.

[33] Sun Microsystems. JSR 12: JavaTM Data Objects (JDO) Specification, 2006. Available at: `http://www.jcp.org/en/jsr/detail?id=12`.

[34] MySQLAB. MySQL, 2007. Available at: `http://www.mysql.com`.

[35] C. Nentwich, W. Emmerich, and A. Finkelstein. Consistency Management with Repair Actions. In *Proc. of the 25th International Conference on Software Engineering (ICSE'03)*. IEEE Computer Society, 2003.

[36] Department of Computer Science Univeristy of Illinois at Urbana-Champaign. Maude manual and examples. Available at: `http://maude.cs.uiuc.edu`.

[37] Department of Computer Science Univeristy of Illinois at Urbana-Champaign. Maude primer and examples. Available at: `http://maude.cs.uiuc.edu`.

[38] G. Psaila and S. Crespi-Reghizzi. Adding Semantics to XML. In *Second Workshop on Attribute Grammars and their applications*, 1999.

[39] D. Romero and M. Alpuente. Estrategias de Reparacion Automatica de Sistemas Web, 2007. Trabajo es realizado en el marco del programa ALFA LERNet AML/19.0902/97/0666/II-0472-FA.

[40] J. Meseguer S. Escobar, C. Meadows. A Rewriting-Based Inference System for the NRL Protocol Analyzer and its Meta-Logical Properties. *Theoretical Computer Science*, 367(1-2):162–202, 2006.

[41] J. Scheffczyk, U. Borghoff, P. Rödig, and L. Schmitz. Consistent Document Engineering: formalizing type-safe consistency rules for heterogeneous repositories. In *Proc. of the 2003 ACM Symposium on Document Engineering (DocEng '03)*, pages 140–149. ACM Press, 2003.

[42] J. Scheffczyk, U. Borghoff, P. Rödig, and L. Schmitz. Managing inconsistent repositories via prioritized repairs. In *Proc. of the 2004 ACM Symposium on Document Engineering (DocEng '04)*, pages 137–146. ACM Press, 2004.

[43] J. Scheffczyk, U. Borghoff, P. Rödig, and L. Schmitz. S-DAGs: Towards efficient document repair generation. In *Proc. 2nd Int. Conf. on Computing, Communications and Control Technologies*, volume 2, pages 308–313, 2004.

[44] The Internet Society. Hypertext Transfer Protocol HTTP/1.1 (RFC 2616), 1999. Available at: `http://www.w3.org`.

[45] SourceForge. TriActive JDO, 2005. Available at: `http://tjdo.sourceforge.net`.

[46] R. Stansifer. *ML Primer*. Prentice-Hall, 1992.

[47] Inc. Sun Microsystems. Java Web Start, 2007. Available at: http://java.sun.com/products/javawebstart.

[48] F. van Harmelen and J.van der Meer. Webmaster: Knowledge-based verification of web-pages. In *Proceedings of the Twelfth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, 1999.

[49] World Wide Web Consortium (W3C). HyperText Markup Language (HTML) 4.01, 1997. Available at: http://www.w3.org.

[50] World Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.0, second edition, 1999. Available at: http://www.w3.org.

[51] World Wide Web Consortium (W3C). HTML 4.01 Specification, 1999. Available at: http://www.w3.org.

[52] World Wide Web Consortium (W3C). Extensible HyperText Markup Language (XHTML), 2000. Available at: http://www.w3.org.

[53] World Wide Web Consortium (W3C). Web Service Description Language, 2001. Available at: http://www.w3.org.

[54] World Wide Web Consortium (W3C). XQuery: A Query Language for XML, 2001. Available at: http://www.w3.org.

[55] World Wide Web Consortium (W3C). SOAP Version 1.2, 2003. Available at: http://www.w3.org.

[56] World Wide Web Consortium (W3C). Markup Validation Service v0.7.4, 2007. Available at: http://validator.w3.org.

[57] Wikipedia. Consistency Model, 2007. Available at: http://en.wikipedia.org.

[58] Wikipedia. Database Normalization, 2007. Available at: http://en.wikipedia.org.

[59] Wikipedia. Evaluation Strategies, 2007. Available at: http://en.wikipedia.org.

[60] Wikipedia. Memory Coherence, 2007. Available at: http://en.wikipedia.org.

[61] Wikipedia. Service Oriented Architecture, 2007. Available at: http://en.wikipedia.org.