María Alpuente, Santiago Escobar, Moreno Falaschi (Eds.)

Automated Specification and Verification of Web Sites

First International Workshop, WWV'05

March 14-15, 2005. Valencia, Spain

Dpto. de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Preface

This report contains the pre-proceedings of the *International Workshop on Automated Specification and Verification of Web Sites* (WWV'05), held in Valencia (Spain) during March 14-15, 2005. WWV'05 provided a common forum for researchers from the communities of Rule-based programming, Automated Software Engineering, and Web-oriented research, in order to facilitate the cross-fertilization and the advancement of hybrid methods that combine the three areas. This workshop is part of the activities of the EU-India project ALA/95/23/2003/077-054.

The Program Committee of WWV'05 collected three reviews for each paper and held an electronic discussion during February 2005. Finally, the Program Committee selected 10 regular papers, 2 position papers, and 6 system descriptions or works in progress. In addition to the selected papers, the scientific program included two invited lectures by Anthony Finkelstein from the University College London, UK, and Shriram Krishnamurthi from Brown University, USA. We would like to thank them for having accepted our invitation.

We would also like to thank all the members of the Program Committee and all the referees for their careful work in the review and selection process. Many thanks to all authors who submitted papers and to all conference participants. We gratefully acknowledge all the institutions and corporations who have supported this event. Finally, we express our gratitude to all members of the local Organizing Committee whose work has made the workshop possible.

Valencia
March 2005

María Alpuente, Santiago Escobar, Moreno Falaschi
WWV'05 Editors

Organization

Program Committee

María Alpuente	Technical University of Valencia, Spain
Sarat Babu	CDAC Hyderabad, India
Demis Ballis	University of Udine, Italy
Gilles Barthe	INRIA Sophia-Antipolis, France
Thierry Despeyroux	INRIA Sophia-Antipolis, France
Wolfgang Emmerich	University College London, UK
Santiago Escobar	Technical University of Valencia, Spain
Moreno Falaschi	University of Siena, Italy
María del Mar Gallardo	Technical University of Málaga, Spain
Furio Honsell	University of Udine, Italy
Giorgio Levi	University of Pisa, Italy
Jan Maluszynski	Linköping University, Sweden
Massimo Marchiori	MIT Lab for Computer Science, USA
Tiziana Margaria	University of Göttingen, Germany

Organizing Committee

María Alpuente	Cèsar Ferri	Javier Oliver
Demis Ballis	Javier García-Vivó	María José Ramírez
Santiago Escobar	José Hernández	Josep Silva
Vicent Estruch	Salvador Lucas	Alicia Villanueva

Additional Referees

Antonio Cisternino	Salvador Lucas	Anne-Marie Vercoustre
Marco Comini	Vincenzo Della Mea	Ivan Scagnetto
Paolo Coppola	Pedro Merino	Francesca Scozzari
Fabio Gadducci	Stefano Mizzaro	Laura Semini
Luca di Gaspero	Ulf Nilsson	Angelo Troina
Martin Karusseit	Mariela Pavlova	Alicia Villanueva
Marina Lenisa	Harald Raffelt	Mariemma I. Yague

Sponsoring Institutions

Departamento de Sistemas Informáticos y Computación (DSIC)
Universidad Politécnica de Valencia (UPV)
Generalitat Valenciana
Ministerio de Ciencia y Tecnología
CologNET: A Network of Excellence in Computational Logic
EU-India Economic Cross-Cultural Programme

Table of Contents

Accessibility Evaluation

XML transformation and optimization

Rule-based approaches to Web site analysis and verification

Model-checking and Static Analysis applied to the Web

Business Data Validation: lessons from practice

Anthony Finkelstein

University College London
Department of Computer Science
Gower Street
London WC1E 6BT, UK

Abstract. All modern businesses store many kinds of data distributed throughout their organization. Data is in different formats and has complex interdependencies (customer records, invoices, financial trade details, sales figures, compliance filings, etc.) Inconsistent data in poorly integrated systems, human errors in manual exception processes, the failure to comply with industry standards and the inability to enforce business rules create operational errors and large, daily losses for business. By checking complex, distributed data against business rules, reference data and industry standards, solutions for data intensive businesses are provided which allow companies to (i) validate the integrity of data across repositories in scattered locations, (ii) draw distributed information together, and (iii) focus knowledge quickly and efficiently on the job in hand.

Web Verification: Perspective and Challenges

Shriram Krishnamurthi [1]

Computer Science Department
Brown University
Providence, RI, USA

Abstract

The Web poses novel and interesting problems for both programming language design and verification—and their intersection. This paper provides a personal outline of one thread of work on this topic.

Key words: Web applications, temporal verification, access control, program analysis

1 What is a Web Site?

The term "Web site" contains a hidden ambiguity. Is a site a static entity, to be viewed as a program source, or is it a dynamic entity, to be viewed through the lens of user behavior? This distinction significantly impacts what it means to analyze and verify a Web site. All the traditional trade-offs between static and dynamic analyses apply: a static analysis can quantify over all program behaviors, but will usually be less specific; a dynamic analysis can only offer guarantees relative to the behaviors it has examined, but the additional contextual information can often yield more informative answers. This distinction potentially matters more on the Web, due to the nature of Web interactions.

2 Web Interactions

In a console or even a GUI application, a user cannot choose to go back or forward, to clone a window and submit responses from both clones, and so on. These user interaction capabilities distinguish Web applications from many other kinds of interactive programs. Indeed, many Web sites are notorious for

[1] This work is partially funded by NSF grants CCR-0305949 and CCF-0447509.

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

their poor handling of such interactions. For instance, on some travel Web sites, viewing the list of hotels, viewing one choice in a new window, examining a second in another new window, then switching back to make a reservation for the first hotel, will result in a reservation at the second hotel [5].

A Web application must not only be sensitive to the possibility of these actions, it must often detect them without help from the browser (which does not report every user action). Furthermore, the availability of rendering platforms is likely to spark innovation, meaning the set of browsers—and, consequently, of interaction behaviors—will grow over time. This makes Web site analysis especially exciting and challenging.

3 Web Verification

My work has focused on static analyses of Web programs. Specifically, I have studied Web applications from two complementary perspectives. All this work has been driven by a desire to build a robust application that has value in its own right, in addition to serving as a generator of research problems.

3.1 A Driving Application

The concrete application is Continue [7,9], a Web-based application for conference paper management. Continue is similar in spirit to several programs in this genre (though it has had considerably more investment into its user interface quality), so familiarity with one of those applications is sufficient for understanding it at a high level. It has several useful features not found in many other conference applications, covering soliciting sub-reviews, helping chairs with assignments, and changing user identity. My goal is to create an application that is not only usable, but has also been verified along as many dimensions as necessary for sufficient reliability. After all, when a community has the creation of papers—as an expression of research—as a primary goal, the safe handling of those papers should be considered mission-critical!

3.2 Temporal Behavior

Web applications must satisfy many temporal specifications. For instance, on a travel reservation site, a user expects that the hotel they reserve is the same hotel that they selected—even if they chose to investigate other hotels along the way. In a virtual bookstore, a user might have a "shopping cart" into which they place their selections. They—or, at least, the store!—would want that every book placed in the cart is purchased upon final check-out. (In fact, the actual property would be more subtle: the books purchased must be all those that were placed in the cart *and not subsequently removed*, creating an additional level of temporal quantification.) There are several similar expectations of "reasonable" behavior in Continue.

The statement of temporal properties naturally suggests the use of a model checker [1]. This proves to be somewhat complex in practice. A naive use of model checking will not, for instance, capture some of the interaction-induced errors mentioned in section 2. Why not? Because the natural model that one would construct from the Web application fails to capture the many behaviors that users can perform through the browser; colloquially speaking, nowhere in the source code of a Web application does it say, "Here the user clicks the Back button".

The problem of building accurate models is further hampered by the problem of accounting for the many kinds of Web interactions. Not only do browsers offer a plethora of choices, even the popular browsers have different feature sets—and this doesn't account for the possibility of additional features in the future.

To support the many interaction features of browsers, we employ prior work that presents a core model of Web interactions [5]. This model presents a small set of Web primitives that are sufficient for modeling all the known Web interaction forms, and should cover many new ones as well. Given this abstraction, we have been studying the problem of building a model checker that can handle the subtleties of the Web [10]. Note that this is not a model checker *only* for Web-specific interactions, but rather one that will *also* account for Web interactions: that is, if the program violates a property independently of any Web interactions, the checker will find those also.

The property language in this work is subtle and, therefore, interesting. Specifically, properties need to be able to refer to elements of Web pages. To index these elements, we refrain both from parsing HTML (a thankless activity!) and from using static distance coordinates (which would be too brittle). Instead, we expect the developer to tag individual page elements using Cascading Style Sheet (CSS) tags. These are not only part of most developers' vocabulary, often the developer has already tagged interesting page elements to highlight visually. While such ideas are not scientifically deep, I believe they are essential for successfully deploying formal methods.

3.3 Information Safety and Visibility

Verifying a program for temporal behavior isn't enough. In a conference server, it is at least as important to ensure both the safety and availability of information: e.g., program committee members can see reviews they should see, and can't see ones that they shouldn't. These properties generally fall under the rubric of *access control*. Once we discovered actual information access bugs in Continue (which have since been fixed!), we embarked on a study of access control policy specification and verification.

Access control has gained fresh popularity owing to the widespread availability of information on the Web. In particular, because many Web applications are interfaces over databases and provide the same data in different

circumstances to different users, access control has increasingly become *role-based*. Industrial standards such as XACML [3], which are effectively rule-based languages, are being employed to describe—and their evaluation engines are being used to enforce—such policies.

Our work in this area [2] has focused on two problems for a restricted (but nevertheless useful) subset of XACML. First, naturally, is the question of whether a policy meets some set of properties; this is the traditional verification question. The second is more intriguing. Given the simplicity of these policy languages, it is easy to patch problems and quickly check that the new policy does what it was intended—on a specific input. The patch may, however, have both exposed private data, or made necessary information unavailable. This danger is exacerbated by the declarative nature of these policy languages, for changes can have very non-local impact. As a result, a simple syntactic difference is no longer sufficient; users require some form of *semantic* differencing. This is the second problem our work addresses.

It is worth noting that the problems surrounding information access—especially the danger of leakage—make a compelling case for static analyses: no reviewer wants to hear that their confidential comments were leaked due to a lack of good test cases. Wading through false positives is certainly onerous; to be effective, this cost must be kept minimal. Nevertheless, this is an instance where the universally quantified guarantees that a static analysis can provide are worth reasonable costs.

4 The Structure of Web Programs

Focusing on Web *programs* (as static entities) raises an interesting subtlety. To obey the stateless nature of the Web, the structure of Web applications has traditionally been "inverted", resembling programs written in continuation-passing style [4,8,12]. Furthermore, important information is communicated using hidden fields and other channels that are traditionally ignored by static analyses. A traditional static analysis would, therefore, approximate a great deal of the important information (particularly the values referred to in properties). The resulting models would simply not be useful for further analysis.

Not surprisingly, the same problems that affect analyses also plague developers. There has thus recently been a trend towards using continuation-based primitives in the source program, which can be handled either by a specialized server [6,12] or on a traditional server after compilation [11]. This means, for instance, that lexical bindings remain as such in the source, rather than being transformed into hidden fields or other external storage. By avoiding this program inversion, this form of source program is therefore a better input to an existing program analysis.

5 Some Research Problems

There are numerous open research problems in this area. What follows is only a small and eclectic sampling.

Some of the most interesting ones have to do with access control. For instance, most of the access control verification work deals solely with the policy. But to be truly effective, it must also take into account the program's behavior relative to the policy. (In an extreme case, if an application were to rigorously consult a policy engine but always ignore its response, no amount of policy verification would be useful. While this particular behavior may appear extreme, it is not inconceivable during testing, and a lack of good test suites will fail to uncover all the places where this prototype failed to grow from a script into a program.)

Access control policies also need to address the temporal behavior of these applications. While some research has studied temporal policies, it is unclear how well these results apply to the less structured world of the Web, where a program potentially has several entry points and users can engage in complicated interactions that the program cannot prevent.

One other important aspect of Web applications is that they are increasingly no longer "on the Web". A growing number of Web sites now make extensive use of client-side scripting languages, especially JavaScript, to implement numerous user operations. In particular, whereas the use of JavaScript tended to be limited to echoing browser operations or performing consistency checks before transmitting data over the wire, now a non-trivial part of the application source is downloaded with the page. This creates a challenge and opportunity for cross-language analyses.

Acknowledgments

I thank the several co-authors with whom I've had the pleasure of conducting this research. I especially thank Pete Hopkins, who helped transform Continue from a very good prototype into a true product. Even his bugs are more interesting than most people's programs.

References

[1] Clarke, E., O. Grumberg and D. Peled, "Model Checking," MIT Press, 2000.

[2] Fisler, K., S. Krishnamurthi, L. A. Meyerovich and M. C. Tschantz, *Verification and change-impact analysis of access-control policies*, in: *International Conference on Software Engineering*, 2005.

[3] Godik, S. and T. M. (editors), *eXtensible Access Control Markup Language, version 1.1* (2003).

[4] Graham, P., *Beating the averages* (2001), `http://www.paulgraham.com/avg.html`.

[5] Graunke, P. T., R. B. Findler, S. Krishnamurthi and M. Felleisen, *Modeling Web interactions*, in: *European Symposium on Programming*, 2003, pp. 238–252.

[6] Graunke, P. T., S. Krishnamurthi, S. van der Hoeven and M. Felleisen, *Programming the Web with high-level programming languages*, in: *European Symposium on Programming*, 2001, pp. 122–136.

[7] Hopkins, P. W., *Enabling complex UI in Web applications with send/suspend/dispatch*, in: *Scheme Workshop*, 2003.

[8] Hughes, J., *Generalising monads to arrows*, Science of Computer Programming **37** (2000), pp. 67–111.

[9] Krishnamurthi, S., *The* Continue *server*, in: *Symposium on the Practical Aspects of Declarative Languages*, 2003, pp. 2–16.

[10] Licata, D. R. and S. Krishnamurthi, *Verifying interactive Web programs*, in: *IEEE International Symposium on Automated Software Engineering*, 2004, pp. 164–173.

[11] Matthews, J., R. B. Findler, P. T. Graunke, S. Krishnamurthi and M. Felleisen, *Automatically restructuring programs for the Web*, Automated Software Engineering: An International Journal (2003).

[12] Queinnec, C., *The influence of browsers on evaluators or, continuations to program web servers*, in: *ACM SIGPLAN International Conference on Functional Programming*, 2000, pp. 23–33.

Feature-based Modelling of a Complex, Online-Reconfigurable Decision Support Service

Martin Karusseit and Tiziana Margaria [1,2]

Universität Göttingen, Germany and Universität Dortmund, Germany

Abstract

In this paper, we show how the concepts of components, features and services are used today in the Online Conference System (OCS) in order to marry the modelling of functionally complex, online reconfigurable internet services at the application level with the needs of a model-driven development amenable to analyze and verify the models. Characteristic of the approach is the coarse-grained approach to modelling and design of features and services, which guarantees the scalability to capture large complex systems. The interplay of the different features and components is realized via a coordination- based approach, which is an easily understandable modelling paradigm of system-wide business processes, and thus adequate for the needs of industrial application developers.

1 Features as Modelling Entities

The concrete application scenario considered in this paper is an example of developing complex, collaborative, online reconfigurable internet services. Such services combine heterogeneous architectures with black/grey-box implementation, which is one of the typical difficulties of large, incrementally developed systems, in particular concerning the continuous redesign and modifications arising during the lifetime of the systems [21]. In order to provide an understandable and manageable high-level model of the system, we design the whole application by defining and enforcing entities of complex behavior called *features*, which are superposed on a base system and coordinated within the system under development. The challenge is precisely how to handle this superposition and coordination in an understandable, well partitioned, and manageable way.

[1] eMail:martin.karusseit@cs.uni-dortmund.de

[2] eMail:margaria@cs.uni-goettingen.de

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

In our applications the *user-level flow of control* is of central importance: this level is realized via coordination graphs, called in our environment Service Logic Graphs (SLGs). They establish a specific modelling level which allows a direct modelling of these control aspects on their own, at the feature level, without being overwhelmed by more detailed implementation concerns like, e.g., data structures, architectures, deadlocks, and load balancing. These concerns are hidden from the application logic designer and are taken care of at a different level, during the object-oriented, component-based development of the single functionalities (which may well have a global impact on the system), capturing individual user-level requirements. Particularly well-suited for our approach are therefore applications where the user-level flow of control frequently needs adaptation or updating, as it is the case when addressing user-specific workflows or situation-specific processes. Besides the design of a number of internet services, typically role-based, client-server applications like the Online Conference Service [10,12] that we are here using as a running illustrative example, we have also successfully addressed the modelling and test of (web-enabled) CTI applications, as presented in [4].

Our understanding of the feature concept can be well explained along the similarities and differences wrt. the definitions of feature and of feature-oriented description given in the literature. We learned to appreciate the concept and the use of features in the context of Intelligent Networks [6,7,20], but our notion of features is more general than e.g. what defined in [3], in order to also capture a more general class of services like online, financial, monitoring, reporting, and intelligence services:

Definition 1.1 [Feature]

(i) A *feature* is a piece of (optional) functionality built on top of a base system.
(ii) It is *monotonic*, in the sense that each feature *extends* the base system by an *increment* of functionality.
(iii) The description of each feature *may* consider or require other features, additionally to the base system.
(iv) It is defined from an *external* point of view, i.e., by the viewpoint of *users* and/or *providers* of *services*.
(v) Its granularity is determined by *marketing* or *provisioning* purposes.

Differently from the IN setting, where the base system was the switch, offering POTS functionality, and the features were comparatively small extensions of that behaviour, we have (e.g. in CSCW-oriented internet services like the OCS) a *lean* base service, that deals with session, user, and role-rights management, and a rich collection of features.

In the traditional telephony setting, features are understood as modifiers of the base service [3], which are basically executed sequentially,each of them departing and returning to the base service (the so called "daisy" or sequential model of execution [16]). In web-based applications, the proportion between base system and features is more extreme: web services have a minimal skeleton service, and are almost completely constituted by features.

The literature is rich of approaches for the modelling and verification of such feature-based systems: for example, advanced compositional model checking techniques have been proposed in [9,2], which deal efficiently with the specification of properties of feature-based systems. Their goal is to be able to partition both the functionality and also the requirements, and to achieve automatic property composition at checking time.

In order to account for complex evolutions of services, we allow a *multi-level organization* of features, whereby more specialistic features are built upon the availability of other, more basic, functionalities.

In order to keep this structure manageable and the behaviours easily understandable, we restrict us to *monotonic* features, which are guaranteed to *add* behaviour. Restricting behaviour, which is also done via features in other contexts (e.g. in a feature-based design in [5]) and similarly in aspect-oriented design [8]), is done in an orthogonal way in our setting, via constraints at the requirements level. Redefinition of behaviour via features, which is considered e.g. in [5], with a clear influence of object-oriented design practices, is not allowed in our setting. Attempts to define and analyze interactions in presence of redefining features have clearly shown that it is very hard to deal with such a feature model, and that it is preferable to avoid it.

Additionally, we distinguish between features as implementations and properties of feature behaviours. Both together yield the *feature-oriented description* of services enforced in our work.

Definition 1.2 [Feature-oriented Description]

(i) A *feature-oriented service description* of a complex service specifies the behaviours of a *base system* and a set of *optional features*.
(ii) The behaviour of each feature and of the base system are given by means of Service Logic Graphs (SLGs) [7].
(iii) The realization of each SLG bases on a library of *reusable components* called Service Independent Building-Blocks (SIBs).
(iv) The feature-oriented service description includes also a set of *abstract requirements* that ensure that the intended purposes are *met*.
(v) *Interactions* between features are regulated *explicitely* and are usually expressed via *constraints*.
(vi) Any *feature composition* is allowed that does not violate the constraints.

In contrast to the proposal by [3], we distinguish the description of the feature's behaviour from that of the legal use of a feature. Restrictions of behaviours are in fact expressed at a different level, i.e. at the requirements level (via temporal logic constraints), and they are part of an aspect-oriented description of properties that we want to be able to check automatically, using formal verification methods.

In the following Sections, we first introduce our concrete example: the Online Conference Service (OCS) (Sect. 2), subsequently we describe in detail the

adopted feature-oriented description technique and illustrate the interplay of the described concepts in Sect. 3 using a specific portion of the OCS. Finally Sect. 4 contains our conclusions.

2 Application: The Online Conference Service (OCS)

The OCS (Online Conference Service) (see [10,11] for a description of the service and of its method of development) proactively helps Authors, Program Committee Chairs, Program Committee Members, and Reviewers to cooperate efficiently during their collaborative handling of the composition of a conference program. It is customizable and flexibly reconfigurable online at any time for each role, for each conference, and for each user. The OCS has been successfully used for over 35 computer science conferences, and many of the ETAPS Conferences. In 2004 and 2005 it served them all, with 6 instances of the service running in parallel.

The service's capabilities are grouped in *features*, which are assigned to specific *roles*. In the OCS, a single user may cover many roles (e.g., PC Members may submit papers and thus be simultaneously Authors), and can switch between them at any time during a working session. A fine granular roles and rights management system takes care of the adequate administration of the context, role and user-specific permissions and restrictions. The roles cooperate during the lifetime of a PC's operations and use the OCS capabilities, which are provisioned at the feature level. Through the cooperation of its features, the OCS provides timely, transparent, and secure handling of the papers and of the related submission, review, report and decision management tasks.

2.1 Feature Description

Features are assigned to the roles, and can be fine-granularly tuned for conference-specific policies. E.g., some conferences practice blind reviewing, meaning that certain fields of the article submission form are not published to the reviewers and secret between Author and PC Chair. In this paper we focus on the principal features and on the discussion of their implications for feature-based service development. The following features are illustrative of the size and granularity adopted in the OCS, while the full collection is shown in Fig. 2.

Article Management: Over 30% of the service activity consistently concerns this feature. The central page corresponding to this feature is the Article overview page (Fig. 1(bottom)), which also contains links to activities like report submission or paper delegation that go beyond just providing access to the article and article managements pages.

Delegation Management: here the PC Chair delegates papers to appropriate PC Members and it supports PC members in their dialogue with their subreviewers. It manages the PC Members and Reviewers tasklists. The delegation process is iterative as PC members/subreviewers might refuse a

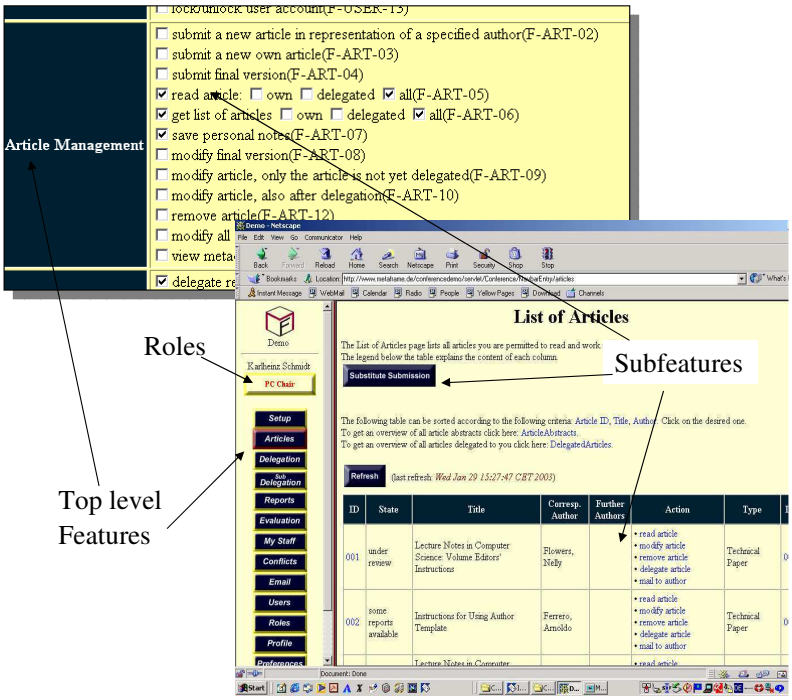


Fig. 1. Role-based Feature Management in the OCS

task, e.g., due to conflicts of interest and/or lack of expertise.

Role Management: it allows the PC Chair to define, modify, reconfigure, cancel roles at any time during the OCS operation. These capabilities are very powerful, and they are responsible for our need of checking the rights at runtime. Fig. 1(top) shows the fine granular feature-based definition of the article management for the role PC Chair. These capabilities also exceed a typical RBAC role-based access implementation [15]: this way, there is no fixed role definition, in particular there is no role hierarchy: it is almost never the case that a role includes and/or monotonically extends the capabilities of "underlying" roles. On the contrary, different roles are prevalently orthogonal to each other in the sense that they bring different access rights.

Setup Management: it enables the responsible administrator and/or the PC Chair to configure the service before it goes public. It also allows online reconfigurations (e.g. setting global deadlines, assigning further responsibilities, establishing newsgroups) while the service is running.

As shown in Fig. 1, the features interact: by configuring differently the role PC Member (Feature Role Management, Fig. 1(top)) a PC Chair can at any moment grant and revoke coarse and fine granular access rights to the whole Article Management feature (or to portions of it) to single users or to user groups. We address the challenge to guarantee that these dynamically defined possible service behaviours obey all our requirements.

2.2 Property Description

Security and confidentiality precautions have been taken to ensure proper handling of privacy and intellectual property sensitive information. In particular,

- the service can be accessed only by registered users,
- users can freely register only for the role Author,
- the roles Reviewer, PC Member, PC Chair are sensitive, and conferred to users by the administrator only,
- users in sensitive roles are granted well-defined access rights to paper information,
- users in sensitive roles agree to treat all data they access within the service as confidential.

We need to be able to check these service-wide properties in a service architecture organized in a hierarchical feature structure. The following sections explain how the needs of the development of internet services like the OCS are taken care of by our application development environment.

3 Designing the OCS as a Feature Based System

As characteristic for the application development by means of our *Agent Building Center* (ABC) framework [11,18], the application definition layer is structured by means of a hierarchical use of features, which are realized in terms of components and (possibly nested) macros, each with its own SLG. Concretely, the ABC supports at the component level

- a basic granularity of components in term of *SIBs* which offer atomic functionalities and are organized in application-specific collections. These building blocks are identified on a functional basis, understandable to application experts, and usually encompass a number of 'classical' programming units (be they procedures, classes, modules, or functions).
- and a structuring mechanism via *macros*, which allow developers to build higher-order components that can be consistently reused as if they were basic components. Consistency is in fact a central concern in terms of analyzability and diagnosis via model checking - as explained in Sect. 3.4.

Application development consists then of the behaviour-oriented combination of SIBs and macros at a *coarse*-granular level.

The design of the OCS, a complex application whose SLG has currently approximately 2500 nodes and 3500 edges, reflects the typical feature-based organization of the application logic in the ABC [11]. As shown in Fig. 2, the global, application-level SLG is quite simple:

- it contains at the top level the logic for the service initialization (init-service) and the *base service*, which is a skeleton service that provides generic inter-

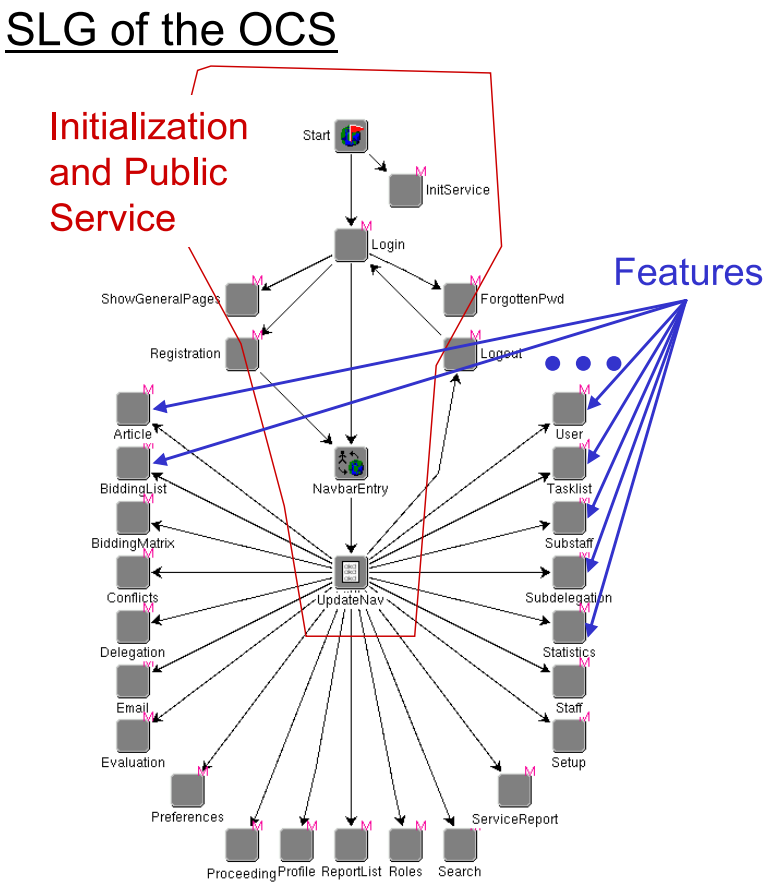


Fig. 2. The Application SLG of the OCS: Base and Public Service, and Features

net login and session management services, and the public functionalities (those accessible without being a registered user), and

- it coordinates the calls to and the interferences between the single *features*.

As we have seen in the feature description, features influence each other, thus one of the aims of service validation via model checking and testing is exactly the discovery and control of the so called *feature interactions*.

3.1 Feature-based Design

As shown in Fig. 2, each feature is implemented as a macro, thus it has an own Service Logic Graph that defines all the services and the behaviours possible under that feature. Fig. 3 shows, e.g. the SLG that implements the Article Management top-level feature. Top-level features typically provide a number of services to the users. In the case of the OCS, the depicted version offers in addition to the Article, Delegation, and Setup Management features already briefly introduced in Sect. 2 also services for the management of Roles, Users, and Staff, as well as e.g. a feature for performing the PC Members' Bidding for papers. This structure becomes immediately evident through the SLG, and it is also explicitly made publicly available over the GUI, as we can see in the

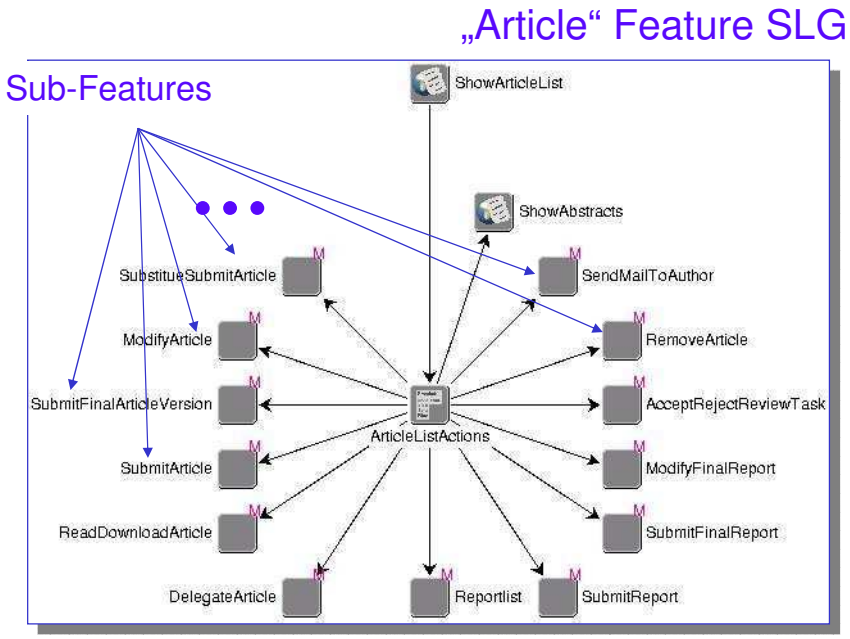


Fig. 3. SLG of the Article Management Feature: Hierarchical Feature Structure

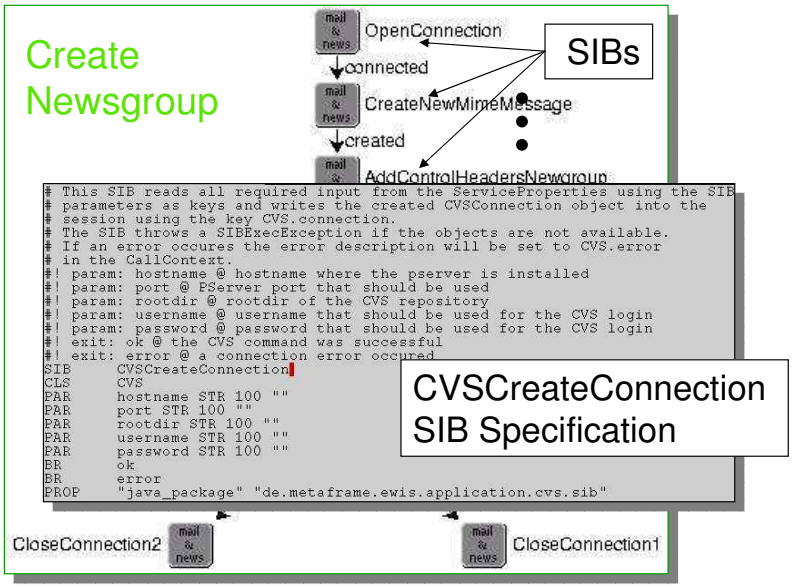


Fig. 4. SIB Occurrences in a SLG, and SIB Specification

navigation bar on the left side of the screen shots of Fig. 1.

Fig. A.1 (in Appendix) shows an excerpt of the features and subfeatures of the OCS. We see that several subfeatures occur in more than one feature, thus can be accessed under a variety of conditions. Altogether, the OCS has over 100 features. New releases of the OCS usually do not touch the basis service but involve the addition or major redesign of top-level features.

3.2 Hierarchy of Features

According to the needs of the application, features can be structured in finer-granular (sub-)features, which are themselves also implemented by means of SLGs. Similar to the structure at the application level, the SLG of the Article Management feature, shown in Fig. 3,

- contains itself a workflow, here quite simple since it provides only navigation capabilities, and
- coordinates the calls to and the interferences among a number of *finer granular features*, which can be themselves substructured according to the same mechanisms.

In our example, the Article Management feature deals both with the management of articles, as evident from subfeatures like SubmitArticle, ModifyArticle, SubmitFinalArticleVersion, and with article-related tasks that reside in other features, like Reportlist or DelegateArticle, which are part of the features Role and Delegation respectively.

To illustrate a complete top-down SLG-based refinement structure, we examine the SubmitArticle subfeature, reported in Fig. 5, which is technically again implemented as a macro. We reach in this SLG the refinement level where the actual business logic is described: embedded in the context of several checks and of error-handling logic,

(i) the ShowSubmitArticle SIB prepares and displays the webpage for the submission action,

(ii) ShowConfirmArticle allows the user to confirm the submission after checking the correctness of the metadata (like title, article, authors, abstract),

(iii) then the actual upload in the database and in the CVS versioning system is performed, and finally

(iv) the ShowSubmitArticleAcknowledgement SIB notifies the submitter of the successful execution.

The SLG also makes use of three macros, CVS_Checkin, mail_notification, and CreateNewsgroup (see Fig. 4). These macros embed reusable pieces of business logic which are relevant to the application designers, but not to the users. Accordingly, they do not deserve the status of a feature.

In the ABC, features are enabled and published to the end-users on their finer granularity, according to a complex, personalized role-right-context management. As an example, only users with a PC Chair role are able to submit articles in the name of another user. The design of the sub-structure of features is driven exactly by the needs of distinguishing behaviours according to different contexts. Sub-features in fact usually arise by refinement of features as a consequence of the refinement of the configuration features and of the role-rights management system. This way we enable a very precise fine-tuning of the access to sensitive information and to protected actions.

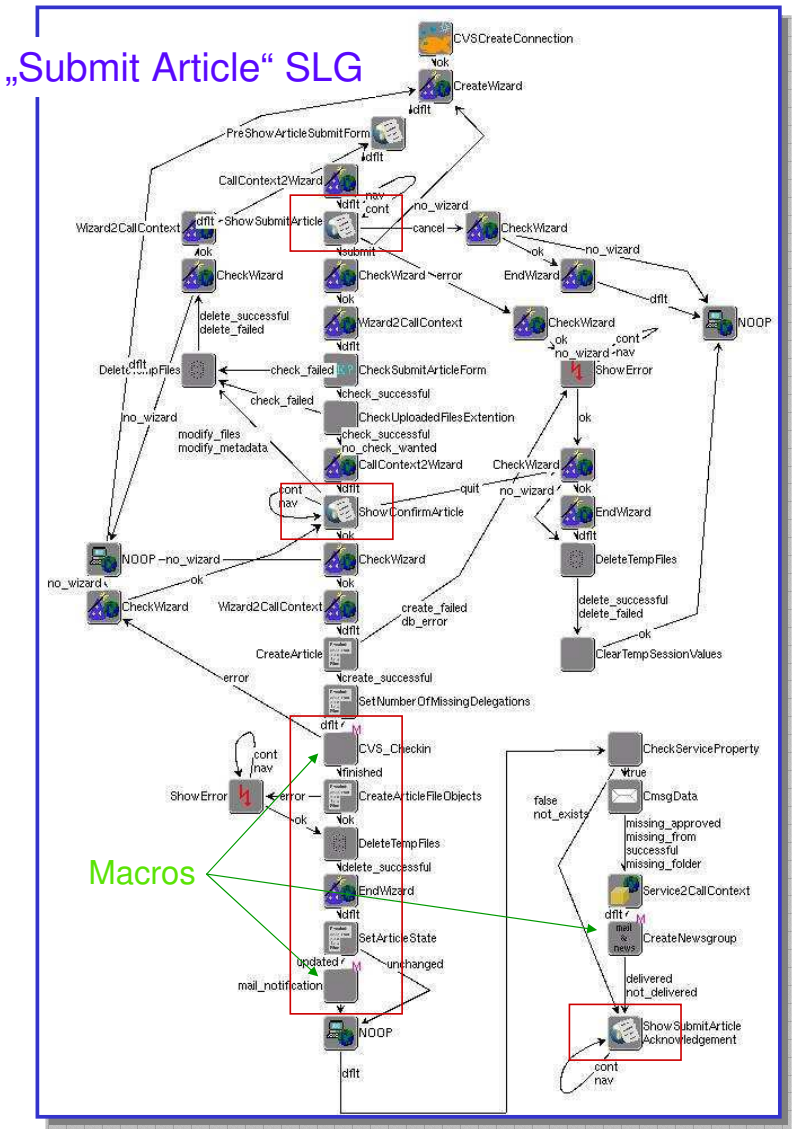


Fig. 5. A Hierarchical Macro: The Submit Article Feature

3.3 Organizing the User/Role Management

Once an internet service is online, it is continuously navigated in parallel by a cohort of *agents* that execute its global service logic on behalf of a user, within the limits imposed by the roles/rights of the user they are associated with.

The SLG of an application defines the *space of potential behaviours* that agents can assume, and each agent's behaviour is defined implicitly as the currently valid projection onto this potential, filtered via

(i) the roles-and-rights management system, which defines dynamic, reconfigurable projections on the behaviours defined in the SLG, and

(ii) the current global status of the application, including the data space, the configuration, and certain event- and time-dependent permissions.

This has consequences on the design of the user and role management, and on its interaction with the feature-based model of the service functionality. From the point of view of the user and role management, features are seen as a collection of functionalities of the service which can be switched on and off for single roles and for single users. The service functionalities have unique names, whose naming scheme is quite simple:

`F-<FeatureCategory>-<SubfeatureID>.<Filter>`

- The `FeatureCategory` is the name of a feature at the modelling level, implemented as an own SLG in the service,
- The `SubfeatureID` specifies a subfeature of the feature at the modelling level, that is implemented either as an own SLG in the service, or as a functionality of a SIB.
- The `Filter` suffix is optional and allows steering the fine granular right management: it restricts the access at runtime to the capabilities of the business objects underlying the features.

The user and role management are themselves implemented by means of features: Roles and Users, as seen in Fig. 1 are typically accessible to the Administrator and the PC Chair.

From the User/Role management's point of view, the Article Management feature is itself managed in the FeatureCategory `ART`. The right to submit an article in the OCS is called permission `F-ART-03`: the single permissions of a FeatureCategory are numbered, thus uniquely named. In case of access to the subfeature `SubmitArticle` (see Fig. 1(top)), it is first checked whether the calling agent (implemented as a process) is granted the permission `F-ART-03`. Only then the access is allowed.

Some subfeatures, like the permission to read an article (`F-ART-05`), have finer granular variants which are administered through filters. The permission `F-ART-05` says that the subservice that provides access to the content of a submission can be executed, but it does not specify on which articles. This is managed through filters, which distinguish the access only to the own articles (`F-ART-05.own`), only to the ones the user should review (`F-ART-05.delegated`) or to all the articles (`F-ART-05.all`).

This User/Role management mechanism exploits these fine granular permissions to create at need personalized views, limiting e.g. for a user the scope of access to certain resources (documents or functionalities). A role is defined via a set of permissions, and it is reconfigurable online at any time by users which have the corresponding rights on the feature `Roles`. This concerns the modification of the current roles, but also the definition of new roles (e.g. to deal with exceptional cases. An example of exception elegantly dealt with this way was the definition of a *Substitute PC Chair* role, where a PC member acted as PC Chair for articles submitted by the PC Chair to the conference he was chairing, which should obviously be treated completely independently.

This way we grant a very high flexibility of the service usage.

3.4 Model Checking-Based High-Level Validation

The correctness and consistency of the application design enjoys fully automatic support: throughout the behavior-oriented development process, the ABC offers access to mechanisms for the verification of libraries of constraints by means of model checking. The model checker individually checks hundreds of typically very small and application- and purpose-specific constraints over the flow graph structure. This allows concise and comprehensible diagnostic information in case of a constraint violation since the feedback is provided on the SLG, i.e. at the application level rather than on the code.

The ABC contains an iterative model checker based on the techniques of [17], recently extended to a game based model checker [14]: it is optimized for dealing with the large numbers of constraints which are characteristic for our approach, in order to allow verification in real time. Concretely, the algorithm verifies whether a given model (a flattened SLG, where the hierarchy information in form of macros has been expanded) satisfies properties expressed in a user friendly, natural language-like macro language [13]. Internally, the logic is mapped to the modal mu-calculus with parameterized atomic propositions and modalities.

Example 1. The general OCS policies already mentioned in Sect. 3 as well as conference-specific policies inherently define a loose specification of the service at the service logic level, which can be directly formulated as properties of the OCS in our model checking logic. For example, the access control policy is a primary source of constraints like "*A user can modify the defined roles only after having successfully registered as Administrator*", expressed as

¬(modify-roles) **unless** user-login [Role=Admin]

as a global constraint on the SLG of the whole application. This example illustrates the slightly indirect way of expressing the intended constraint. It says,"*A user cannot modify the defined roles unless (s)he has successfully registered as Administrator*". Additionally the example shows a parameterized atomic proposition: user-login [Role=Admin] is parameterized in the possible roles a user might have, and [Role=Admin] does not only require a user-login to appear, but also that the role matches, in this case administrator.

All the properties mentioned earlier in Sect. 2 are requirements expressible in this logic, and they are instances of the classes of safety and consistency requirements identified in [1] to be characteristic of Computer Supported Collaborative Work platforms. Being able to automatically verify such properties via model checking is a clear advantage of the ABC, and it is essential in applications like the OCS where the role-dependency is much more dynamic than in standard RBAC applications.

A previous version of the OCS, which was not organized in features, had

been already checked wrt. temporal logic properties like the one above [19] This global approach became impractical due to the growing size of the web service, to the increased importance of the Setup feature, which allows almost complete reconfigurability at any time, and to the transition to distributed development and maintenance, which are distributed feature-wise within a team of people. At this point, it became central to be able to partition also the verification feature-wise. This allows us e.g. to keep the properties readable, since we do not need to add large numbers of conjuncts just to isolate specific portions of the global graph, very often coincident with the features.

Meanwhile we use a slightly enhanced variant of CTL, where we have both *forward* and *backward* modalities. This is common e.g. in program analysis, and turns out to be useful also in our application. Examples of such operator pairs are $AF\_F(\phi)$ and $AF\_B(\phi)$, the well known *always finally* forward and backward CTL operators. We use often also *until* operators, useful to describe "layered" regions of properties: $ASU\_F(\phi, \psi)$ (resp. $AWU\_F(\phi, \psi)$) mean ϕ *strong forward-until* ψ (resp. ϕ *weak forward-until* or *unless* ψ). Thereby, the primed SIB names, like 'ShowFrameSetFiles, are the atomic propositions of the logic. Given the large alphabet of SIB and branch names it is convenient to use edge modalities with sets, as e.g. in $[\sim \{ok\}]\phi$, meaning that ϕ is true in each successor state reachable via an edge not labelled ok.

Apart from a number of simpler constraints that just enforce some forward or backward sequence of SIBs (useful e.g. in conjunction with macros, to enforce a certain well-formedness of reusal), most properties express reachability or a certain loose ordering of functionalities.

Example 2. In the ForgottenPwd feature, e.g., we would like that once the page with the form for answering the private question has been shown (done by the SIB ShowFrameSetFiles), the user-entered data should always be checked for correctness and completeness SIB CheckReqParam [3]. This is expressed as

$$'ShowFrameSetFiles => [\{ok\}]AF\_F('CheckReqParam)$$

Example 3. Once this parameter check fails, the user should return to the page with the input form. The SIB CheckReqParam is in this case exited along the branch missing or exists_empty:

$$'CheckReqParam => [\{missing, exists\_empty\}]AF\_F('ShowFrameSetFiles)$$

Example 4. The password question should only be shown once a valid e-mail address has been input. The constraint

$$'ShowPwdQuestion =>$$

$$(AF\_B('CheckEmailAddr) \wedge AWU\_B(\sim' CheckEmailAddr, ![\{successful\}]!T))$$

meaning that every occurrence of ShowPwdQuestion is preceded by a CheckEmailAddr) and that that CheckEmailAddr) has been exited along a successful

[3] We would like to ensure this before forwarding the data to the persistency layer.

branch. Here we rely on the uniqueness of the successful edge within the feature. In the general case we would need additional constraints like

$AG\_F(< \{successful\} > T =>' CheckEmailAddr)$

to delimit the scope more precisely.

Example 5. The notification page that an e-mail with the new password has been sent should not be shown before it was really sent out without an explicit acknowledgement by the user:

$'Service2CallContext => ASU\_F(\ 'ShowPwdAck,' SendMimeMessage)$

Here we see that, as soon as the service logic becomes a bit more complex, the intuitiveness of the constraints is also quickly impaired: in order to check properties of the service logic, we need to refer to technical SIBs like Service2CallContext. We also see that sometimes the "minimality" of constraints is not obvious: here we use *until* instead of *next* because in the graph there are self-loops.

An example of non satisfied constraints concerned the treatment of back browse branches in some areas of the OCS like the Report management feature, where several successive modifications of forms are possible in sequence. In order to check the existence and correctness of these (quite large) loops, we have decided to model the navigation structure of these OCS portions at the SLG level. However, due to the reusal of previously available subfeatures, some of the navigation options were still implemented at the GUI level, thus we were able to detect e.g. a number of missing back branches in the SLG. This was not a functional error, but an inconsistency in the modelling style.

4 Conclusions

We are not aware of any feature-based design approach similar in its intent to our goals, in particular concerning the simplicity at the modelling level. The closest approaches we know of typically require far more knowledge at the application level (at least programming expertise) and/or lack systematic support by means of formal methods, and therefore are inadequate for the scenarios and users we address.

The impact of our approach on the efficiency of the design and documentation has been proven dramatic in industrial application scenarios: our industrial partners reported a performance gain in time-to-market of the applications of a factor between 3 and 5. The reason for the reported gain was in particular the early error detection, due to the tightened involvement of the application expert into the development cycle. More generally, we see the current approach as an instance of Model Driven Application Development, where heterogeneous models allow the individual, but interdependent modelling of complementary aspects. And indeed, features constitute a specific category of

such aspects, adequate for the structuring of complex applications according to complementary views and to support elegant and powerful approaches to proving correctness and compatibility of complex behaviours.

References

[1] T. Ahmed, A. Tripathi: *Static Verification of Security Requirements in Role Based CSCW Systems*, Proc. 8th Symp. on Access Control Models and Technologies, Como (I), ACM Press, pp.196-203, 2003.

[2] C. Blundell, K. Fisler, S. Krishnamurthi, P. Van Hentenryck: *Parameterized Interfaces for Open System Verification of Product Lines* Proc. ASE 2004, IEEE International Symposium on Automated Software Engineering.

[3] J. Bredereke: *On Feature Orientation and Requirements Encapsulation*, volume "Components, Features, and Agents", LNCS 2975, 2004, Springer Verlag.

[4] A. Hagerer, T. Margaria, O. Niese, B. Steffen, G. Brune, H.-D. Ide: *An Efficient Regression Testing of CTI Systems: Testing a complex Call-Center Solution*, Annual Review of Communic., Vol. 55, Int. Engin. Consortium, Chicago, 2001.

[5] H. Harris, M. Ryan: *Theoretical Foundations of Updating Systems*. ASE 2003, 18th IEEE Int. Conf. on Aut. Software Engineering, IEEE-CS Press, 2003.

[6] ITU: *General recommendations on telephone switching and signaling intelligent network: Introduction to intelligent network capability set 1*, Recommendation Q.1211, Telecommunication Standardization Sector of ITU, Geneva, Mar. 1993.

[7] ITU-T: *Recommendation Q.1203. "Intelligent Network - Global Functional Plane Architecture"*, Oct. 1992.

[8] S. Katz, Y. Gil: *Aspects and Superimpositions*, Proc.ECOOP 1999, LNCS N.1743, Springer Verlag.

[9] H.C. Li, S. Krishnamurthi, K. Fisler: *Verifying Cross-Cutting Features as Open Systems*, Proc. FSE-10, ACM SIGSOFT Int. Symp. on the Foundations of Software Engineering, 2002.

[10] B. Lindner, T. Margaria, B. Steffen: *Ein personalisierter Internetdienst für wissenschaftliche Begutachtungsprozesse*, GI-VOI-BITKOM-OCG-TeleTrusT Konferenz Elektronische Geschäfts-prozesse (eBusiness Processes), Universität Klagenfurt, September 2001, http://syssec.uni-klu.ac.at/EBP2001/ .

[11] T. Margaria: *Components, Features, and Agents in the ABC*, in "Components, Features, and Agents", LNCS 2975, pp.154-174, 2004, Springer Verlag.

[12] T. Margaria, M. Karusseit: *Community Usage of the Online Conference Service: an Experience Report from three CS Conferences*, 2nd IFIP Conf. on "e-commerce, e-business, e-government" (I3E 2002), Lisboa (P), Oct. 2002, in "Towards the Knowledge Society", Kluwer, pp.497-511.

[13] T. Margaria, B. Steffen: *Lightweight Coarse-grained Coordination: A Scalable System-Level Approach*, in STTT, Int. Journal on Software Tools for Technology Transfer, Vol.5, N.2-3, pp. 107 - 123, Springer-Verlag, March 2004.

[14] M. Müller-Olm, H. Yoo: *MetaGame: An Animation Tool for Model-Checking Games*, Proc. TACAS 2004, LNCS 2988, pp. 163-167, 2004, Springer Verlag.

[15] R. Sandhu, E. Coyne, H. Feinstein, C. Youman: *Role-Based Access Control Models*, IEEE Computer, 29(2):38-47, Feb. 1996.

[16] M. Shaw, D. Garlan: *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, 1996.

[17] B. Steffen, A. Claßen, M. Klein, J. Knoop, T. Margaria: *The Fixpoint Analysis Machine*, (invited paper) CONCUR'95, Pittsburgh (USA), August 1995, LNCS 962, Springer Verlag.

[18] B. Steffen, T. Margaria: *METAFrame in Practice: Intelligent Network Service Design*, In *Correct System Design – Issues, Methods and Perspectives*, LNCS 1710, Springer Verlag, 1999, pp. 390-415.

[19] B. Steffen, T. Margaria, V. Braun: *Coarse Granular Model Checking in Practice*, Proc.8th Intern. SPIN Workshop on Model Checking Software, satellite to ICSE 2001, Toronto (Canada), May 2001, LNCS 2057, pp. 304-312, Springer Verlag.

[20] B. Steffen, T. Margaria, V. Braun, N. Kalt: Hierarchical service definition, Annual Rev. of Communication, IEC, Chicago, 1997, pp. 847-856.

[21] H. Weber: *Continuous Engineering of Information and Communication Infrastructures*, Proc. Int. Conf. on Fundamental Approaches to Software Engineering (FASE'99), Amsterdam, LNCS N. 1577, Springer Verlag, pp. 22-29.

A Hierarchical Feature Structure

Main Features	Sub-Features	Article	Bidding	Delegation	Report	Tasklist
Article	SubstituteSubmitArticle					
	SubstituteSubmitArticle					
	ModifyArticle					
	SubmitFinalArticleVersion					
	SubmitArticle					
	ReadDownloadArticle		x			
	DelegateArticle			x		
	Reportlist				x	
	SubmitReport					x
	SubmitFinalReport					
	ModifyFinalReport				x	
	AcceptRejectReviewTask					x
	RemoveArticle					
	SendMailtoAuthor					
	ShowAbstracts					
Bidding						
-Biddinglist	Bid					
	DisplayPdf					
	ReadDownloadArticle	x		x		
-BiddingMatrix	ReadOnlyBiddingMatrix					
	EditBiddingMatrix					
	DisplayPdf					
	DelegateArticle	x		x		
Delegation	DelegateArticle	x				
SubDelegation	DisPlayPdf					
	ReadReport					
	RemoveDelegation					
	ReadDownloadArticle	x	x			
	ExtendDecisionTaskDeadline					
Email	MgmtEmailTextMacros					
	MgmtEmailTemplates					
	ComposeAndSendEmail					
Report	ModifyFinalReport					
	ModifyReport					
	ReadReport					
	RemoveReport					
	ReadDownloadArticle					
Roles	DefineNewRole					
	ModifyRole					
	RemoveRole					
Tasklist	SubmitReport					
	DelegateArticle					
	ReadDownloadArticle		x			
	SubmitReport	x				
	AcceptRejectReviewTask					
	Bid					
...	...	...				

Fig. A.1. Hierarchical Feature Structure and Feature Reusal

Generating commercial web applications from precise requirements and formal specifications

David Crocker [1]

Escher Technologies Ltd.
Aldershot, United Kingdom

John H. Warren [2]

Precision Design Technology Ltd.
Maidenhead, United Kingdom

Abstract

We present a new model-based approach that we are using to build commercial web-based applications. The user requirements together with a data model are formally specified in a graphical notation using the CREATIV toolset. The specification may be checked by animation before being automatically translated to *Perfect* notation. The *Perfect Developer* toolset uses automated reasoning to generate formal proofs of correctness. It then generates C++ or Java code which, in conjunction with an application framework also written in *Perfect*, forms the complete application including the HTML user interface. The whole process provides a rapid turnaround from new requirements to a formally-verified application.

Key words: formal specification, formal verification, web applications

1 Introduction

A recent survey of 1027 information technology projects [1] found that only 12.7% were deemed successful. Poor requirements were considered to be a contributory factor in 76% of the failing projects. These figures indicate that a better approach to IT system development is needed and that any new approach must include better definition of requirements. We present such an approach.

[1] Email: dcrocker@eschertech.com
[2] Email: john.warren@precisiondesign.co.uk

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

2 Defining Requirements with CREATIV

One important specification dichotomy is that while user understanding requires the use of natural language, correctness requires the precision of a mathematical approach. The CREATIV approach offers an improved process for requirement specification that aids user understanding by using natural language while ensuring consistency and functional correctness by using concealed mathematical logic.

The underlying approach is model-based and axiomatic [2]. A proof in an axiomatic theory is a finite sequence of statements in the theory in which each statement either is an axiom or derives from an earlier statement by applying a rule of reasoning. The CREATIV reasoning process uses five axioms and one rule of inference, which reads informally as: IF a candidate entry satisfies its preconditions THEN add the entry to the specification AND include the consequential closure that results from its addition. The tool imposes some of the preconditions and the application itself imposes others. The reasoning occurs in a metamodel that represents the requirements process, represented in first order predicate calculus; the metamodel construction is also axiomatic, and uses identical axioms and rules of reasoning.

Starting from the axioms and the rule of inference, we can prove a succession of theorems at a very low level. There are several hundred such theorems, which are used as lemmas to support higher level proofs. We then present the reasoning system, specified in its own notation, for proof in exactly the same way. We can then present the application specification similarly. This layered organisation of the approach has an important advantage: we can express reasoning about specification behaviour (animation) as one more layer in the reasoning process and one that uses the same approach and rule of inference.

The tool can translate a correct specification into other languages, such as *Perfect* or Z, because there is one well-defined meaning associated with each theorem. When we translate into *Perfect*, which has its own more powerful theorem prover, we can also generate a large number of hypotheses, by rule, that express properties that the application code should possess. Users may also state further hypotheses, arising from their own knowledge. We expect all these hypotheses to be provable; proof failures almost always indicate important specification errors.

Translation to *Perfect* also allows immediate generation of Java or C++, so the production of web-based systems can be automated once the specification is complete. We guarantee that any proven property will be present in the generated system.

In order to construct a specification using this approach, it is necessary to construct the business model (diagram) and the supporting information (text). While analysts can enter information in any order, it is usual to develop a part of the diagram first, then to add the supporting text, using form-filling dialogues, and finally to check this partial specification or model. Analysts can

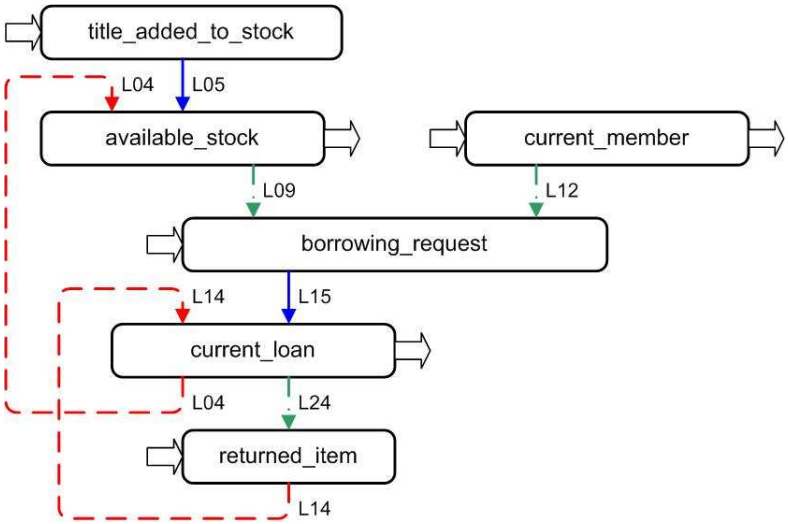


Fig. 1. Library model

then correct any errors and extend the partial model by repeating the above process. The CREATIV workbench supports this process and retains all the entered material in an underlying database. Analysts can make new entries in the database and change or delete existing entries. At user request, the workbench checks that all the entered facts are provable and self-consistent, by translating the database content into a formal notation, attempting mathematical proof, and reporting any proof failures. Since this is at user request, specifications can be temporarily inconsistent during their development; this makes those constructs that involve iteration and circularity easier to define.

We define a specification as an assembly of five structured collections of information; of these, three contain the data, functions, and events of the application. Of the other two, concepts conjoin data, function, logic, and event material to provide a view of the data within a class; constraints (or business rules) establish how data moves between processes within the system. Each constraint defines a data flow line, the specific data transfer (defined by one or more predicates), and the type of flow that occurs. Each destination concept includes a logical expression that must be satisfied before any flow can occur.

Each collection (data, function, event, concept, and constraint) contains one or more relational tabulations of facts relevant to the requirement specification. The entry of these facts constitutes the specification process. The CREATIV specification workbench guides users through this entry process with no reference to any mathematical notation. The totality of this information allows the derivation of a formal specification that will support detailed reasoning. In particular, we can predict the behaviour of the specification to arbitrary sequences of input data; this not only allows users to test the correctness of the specification but also provides acceptance test data and expected results.

The notation is similar to the Business Activity Diagram of UML but provides more detail. An example is shown in Fig. 1, which represents a lending library. The library acquires books (concepts that accept external events are marked with an input arrow), each entered as *title_added_to_stock*. For each such entry, the L05 constraint transfers selected attribute values to represent *available_stock*, that is, items available for loan. We similarly enter each *current_member*, who must be traceable if he is to borrow books, though in this case there is no consequential inference. The L12 broken line indicates read-only access to this data.

Any person can make a *borrowing_request* for any book but only members can borrow books (L12) and members can only borrow available books (L09). Requests by non-members are unsatisfiable, as are requests for non-present books. If the logic "L09 AND L12" is satisfied (both values are true) then the *borrowing_request* is accepted. An entry is added to *current_loan* to reflect the loan (L15) and as a further consequence, the item is deleted from *available_stock* (L04), indicating that the item is no longer available. The dashed line indicates a logical inversion.

Upon completion of the loan, the borrower returns the book and this event is entered in *returned_item*. Provided that the entry matches a borrowed item (as required by L24), the matching entry is deleted from *current_loan* (L14) and reinstated in *available_stock* (L04), since the deletion is now inverted. If we enter a small number of books and members, we can test the model specification by submitting borrowing request and returned item events (both sound and faulty).

This simple model is inadequate as a library model; but because it is easy to understand, it is easy to enquire about its deficiencies. For example: how do we limit the number of loans to one borrower? How do we represent members who leave the library?

The CREATIV tool can reason about the model to verify that the specification is self-consistent. Next, we can animate the specification in order to demonstrate the behaviour of the model to users of the proposed system, allowing specification correctness to be tested. It is entirely usual for initial versions of a specification to contain errors; the analyst corrects these until the users agree the correctness of the specified behaviour.

Documentation is generated automatically, by direct deduction from the specification facts. We can produce a structured English specification, UML documentation such as interaction diagrams, and various additional elements such as checklists and tabulations.

3 Generating and Verifying the Application with *Perfect Developer*

Perfect Developer [3], [4] is an object-oriented formal tool which provides for the definition of state-based specifications and related functional requirements.

It also has facilities for manual or automatic refinement to a lower-level specification, from which C++ or Java code can be generated. Included is an automated reasoning engine which attempts to prove that the specifications are complete and consistent, the specifications meet the requirements, and the refinements preserve the observable behaviour apart from resource usage.

The *Perfect* specifications generated by CREATIV are designed to work in conjunction with an application framework that has been written directly in *Perfect*. These specifications include a model of the relational database tables needed to represent the data, and a large number of operations specifying how the state is to be changed in response to various events. Also generated is a partial data dictionary, for use in generating HTML screens to interact with the user.

The application framework provides the base classes from which classes in the CREATIV-generated specification are derived. It automates the generation of HTML pages for user interaction and the parsing of returned form data. A small supporting library allows it to be built as a web application using the CGI interface. We have not yet made much effort to achieve full formal verification of the framework, so at present only 92% of the 1108 proof obligations that it gives rise to are discharged automatically. Most of the failures relate to proof obligations from library components used to perform file I/O and match regular expressions, rather than from the framework itself.

4 Case Study

As a commercial example of this approach, we have specified part of a web-enabled database system for a UK government department. We have currently specified about 10% of the system; this appears as 1340 specification clauses and translates at present to 35,799 lines of *Perfect*. The subsequent code generation produces 62,224 lines of Java. *Perfect Developer* also generates 9,819 proof obligations relating to the CREATIV-generated files, all of which are proven automatically. The entire generation process (axiomatic specification proof, translation to *Perfect*, and construction of Java code) takes less than 30 minutes on a laptop machine of modest speed (750 MHz). Complete proof by *Perfect Developer* of the generated obligations requires about 4 hours 20 minutes on the same machine (averaging about 1.6 seconds per proof).

5 Related Work

Automatic code generation for large parts of web applications from semi-formal or informal specifications (such as UML) is widely used. An approach to the construction of web applications from formal specifications is outlined in [5]. The use of formal specifications for testing web applications has been proposed by a number of authors including [6].

6 Conclusions and Further Work

We have shown that it is possible and practical formally to specify and verify a substantial part of a web-based commercial application, and to generate code from the specifications. We contend that the generated system is inherently immune to buffer overflow attacks. This will be proven formally when we achieve complete verification of the application framework and associated library.

There are some limitations to the present version of the workbench. Both reasoning systems are founded on predicate calculus and are unable to reason about temporal properties, non-functional properties or concurrency. CREATIV uses the relational model for information storage; this may not be the best representation for some applications.

In the future we intend to specify formally a subset of the W3C HTML 4.01 definition in *Perfect*. This will allow us to prove not only that the application always generates well-formed HTML pages, but also that the system is immune to cross-site scripting attacks.

References

[1] Taylor A, *IT Projects Sink or Swim*. In "BCS Review 2001", British Computer Society. Available at `http://www.bcs.org/review/2001/articles/itservices/projects.htm`.

[2] Warren J.H and Oldman R.D, *A Rigorous Specification Technique for High Quality Software*. In "Proceedings of the Twelfth Safety-Critical Systems Symposium" (ed. F.Redmill and T.Anderson) 43-65, Springer-Verlag (London) (2004). ISBN 1-85233-800-8.

[3] Crocker D, *Safe Object-Oriented Software: The Verified Design-By-Contract Paradigm*. In "Proceedings of the Twelfth Safety-Critical Systems Symposium" (ed. F.Redmill and T.Anderson) 19-41, Springer-Verlag (London) (2004). ISBN 1-85233-800-8 (also available via `http://www.eschertech.com`).

[4] Crocker D and Carlton J, *A High Productivity Tool for Formally Verified Software Development*. To be published in the International Journal of Software Tools for Technology Transfer, Special Section on Formal Methods 2003.

[5] Fons J, Pelechano V et al., *Extending an OO Method to Develop Web Applications*. The Twelfth International World Wide Web Conference, Budapest, Hungary.
At `http://www2003.org/cdrom/papers/poster/p329/p329-fons.htm`.

[6] Xiaoping Jia and Hongming Liu, *Rigorous and Automatic Testing of Web Applications*.
At `http://jordan.cs.depaul.edu/research/web-test-paper.htm`.

What Kind of Verification of Formal Navigation Modelling for Reliable and Usable Web Applications?

Marco Winckler [1] Eric Barboni [2] Philippe Palanque [3]
Christelle Farenc [4]

LIIHS-IRIT
University Paul Sabatier
Toulouse, France

Abstract

In this paper we introduce briefly a notation dedicated to model navigation of Web applications and some strategies that we plan to employ to assess models built with such as a notation. Our aim with this kind of evaluation is to ensure (prior to implementation) that important users tasks can (or cannot) be performed with the system under construction.

Key words: Model-based Web developpement, Model-based verification, Formal description techniques, Usability evaluation.

1 Navigation modeling with the SWC notation

Model-based design is a relatively recent field over the Web but it is growing fast due the demanding need of modelling support to build more and more complex Web applications [7]. In recent years, some models for the development of Web applications have been proposed such as the OO-H method [5] and the WebML approach [3]. These models may increase productivity but they often lead to ambiguous descriptions of requirements or impose a particular kind of implementation. However, even if more appropriate modelling techniques exist, the use of modelling methods alone it is not enough to ensure the usability and reliability of Web applications. For web applications, usability evaluation is not only important for identifying problems in the early

[1] Email: winckler@irit.fr
[2] Email: barboni@irit.fr
[3] Email: palanque@irit.fr
[4] Email: farenc@irit.fr

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

phases of the development process but also for managing the frequent updates performed over web applications.

In order to overcome the limitations of informal models, we have proposed the StateWebCharts (SWC) [9] notation for describing navigation over Web application. SWC is a formal notation that extends StateCharts [6] by adding appropriate semantics for states and transitions, includes notions like dialogue initiative control and client and transient activities. A detailed description of this notation can be found in [9]. More generally, Web pages (static or dynamically generated) are associated to SWC states. Thus, each state describes which objects users can see when navigating over the applications. Links are represented by transitions between states. Data can be transferred from a state to another through the transitions. Persistent data can be held by a special kind of state called transient state (see [9] for details).

The operational semantic for a SWC state is: current states and their containers are visible for users while non-current states are hidden. Users can only navigate outgoing relationships (represented by the means of transitions in the model) from current states. When a user selects a transition the system leaves the source state which becomes inactive letting the target state to be the next active state in the configuration.

2 Strategies for model-based evaluation of navigation

The purpose of a navigation modelling is to create navigable paths among units of an information space. Hereafter we present some strategies to evaluate the usability of models using the set of basic information concerning states, relationships and events provided by the SWC notation.

Static verification refers to analysis without model execution. Several methods have been proposed as described in [2] for supporting model-based evaluation such as model checking (typically used for software testing, it is used to check intrinsic elements of a model) [8]. This category covers the following issues: a) Model consistency; b) Unreachable states; c) Dangling links; d) Redundancy of links; e) Unreachable paths; f) Shortest path; g) Compulsory path; and h) Data dependency.

The principle of dynamic verification is that the model must be executed or simulated in order to perform the verification. The simulation, step by step under the control of the designer, can be used as a kind of walkthrough method for inspecting both navigation specifications and their corresponding Web applications. In this case, the designer 'navigates' the model as a real user would do. This task can be eased by the appropriated tool support. However, walkthrough over model is not the unique way for performing dynamic verification. The navigation models can be accessed to measure the coverage of a given test suit or a collection of test cases. The test suit consists of a set of directives which are used by automated tools for inspecting the model [1]; for example: a) every state is visited at least once in some test case; b) every

relationship from every state is followed at least once; every path in the site is followed in at least one test case.

The verification of properties is made up by inspecting the models in order to verify if the model is compatible or not with some predefined behavioural rules (as we only consider in this paper properties concerned by navigation) called properties. They can be expressed by using Temporal Logic or any other abstract declarative formal description technique. Properties can be very generic/abstract or very concrete such as "do not allow more than seven relationships going out from a state". This kind of property might be verified automatically over the model and even embedded into editor tools but this heavily depends on the expressive power of the notation and the complexity of the models. Until now, only few works [4] have been devoted to verify ergonomic rules as properties of models.

Assessment of navigation model using user scenarios aims at exploiting two complementary models and to cross-check their compatibility. Task models are aimed at describing not only how users perform their task but also when and why those tasks are performed (in order to achieve which goal). Task models are typically a result of a detailed specification of functional requirements of the interface describing user tasks with the application. The basis of such an assessment is to extract scenarios from tasks models and to play them over navigation models [10]. Contrarily to other works that analyze navigation paths, this procedure allows designers comparing navigation paths to real tasks, which is supposed to give deeper insights about the usability of the web application's user interface.

3 Discussion and future work

Several methods have been proposed as described in for supporting model-based evaluation [8][4][2]. These techniques have been available and usefully employed for a long time in the field of formal methods for communication protocols or software engineering. Besides, despite the first paper published in this field by [8] their actual use for Web applications remains very seldom and limited.

In this work we have introduced a notation devoted to the navigation modelling and we have discussed strategies for evaluation of such as models. The SWC notation provides all the set of basic information concerning states, relationships and events which is required to deal with the evaluation described above. The edition of SWC models is supported by the SWCEditor which also allows the simulation and verification of models. Our ongoing work consists in developing and integrating analysis tools to our prototype in order to support a cumbersome and resource demanding manual process.

References

[1] Beizer, B., "Software testing techniques (2nd ed.)," Van Nostrand Reinhold Co., 1990.

[2] Campos, J. C. and M. Harrison, *Formally verifying interactive systems : a review*, in: *Design, Specification and Verification of Interactive Systems '97* (1997), pp. 109–124.

[3] Ceri, S., P. Fraternali and A. Bongio, *Web modeling language (webml): a modeling language for designing web sites*, in: *Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking* (2000), pp. 137–157.

[4] Farenc, C., P. Palanque and R. Bastide, *Embedding ergonomic rules as generic requirements in the development process of interactive software.*, in: *INTERACT 99 7th International Conference on Human-Computer Interaction*, 1999, pp. 408–416.

[5] Gomez, J., C. Cachero and O. Pastor, *Extending a conceptual modelling approach to web application design*, in: *CAiSE '00: Proceedings of the 12th International Conference on Advanced Information Systems Engineering* (2000), pp. 79–93.

[6] Harel, D., *Statecharts: A visual formalism for computer system.*, Sci. Comput. Program. **8** (1987), pp. 231–274.

[7] Murugesan, S. and Y. Deshpande, editors, "Web Engineering, Software Engineering and Web Application Development," Lecture Notes in Computer Science **2016**, Springer, 2001.

[8] Ricca, F. and P. Tonella, *Analysis and testing of web applications*, in: *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering* (2001), pp. 25–34.

[9] Winckler, M. and P. Palanque, *Statewebcharts: A formal description technique dedicated to navigation modelling of web applications.*, in: *DSV-IS*, 2003, pp. 61–76.

[10] Winckler, M., P. Palanque, C. Farenc and M. S. Pimenta, *Task-based assessment of web navigation design*, in: *TAMODIA '02: Proceedings of the First International Workshop on Task Models and Diagrams for User Interface Design* (2002), pp. 161–169.

How Recent is a Web Document?

Bo Hu [1]

Universität der Bundeswehr München
Department of Business Administration
Munich, Germany

Florian Lauck [2]

Universität der Bundeswehr München
Department of Business Administration
Munich, Germany

Jan Scheffczyk [3]

Universität der Bundeswehr München
Institute for Software Technology
Munich, Germany

Abstract

One of the most important aspects of a Web document is its *up-to-dateness* or recency. Up-to-dateness is particularly relevant to Web documents because they usually contain content origining from different sources and being refreshed at different dates. Whether a Web document is relevant for a reader depends on the history of its contents and so-called external factors, i.e., the up-to-dateness of semantically related documents.

In this paper, we approach *automatic management* of up-to-dateness of Web documents that are managed by an XML-centric Web content management system. First, the *freshness* for a single document is generated, taking into account its change history. A document metric estimates the distance between different versions of a document. Second, *up-to-dateness* of a document is determined based on its own history and the historical evolutions of semantically related documents.

Key words: Web site management, up-to-dateness, content management systems, document metric, semantic links

[1] Email: bo.hu@unibw.de
[2] Email: florian.lauck@unibw.de
[3] Email: jan.scheffczyk@unibw.de

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

1 Introduction

The WWW has been designed for dynamic information from the very beginning [1]. *Up-to-dateness* [4] is one of the most significant characteristics of Web documents, because a Web site typically contains numerous Web pages origining from different sources and evolving at different rates. Unlike books in a traditional library, Web pages continue to change even after they are initially published by their authors [2]. In this paper we distinguish between *freshness*, which depends on the history of a single document, and *up-to-dateness*, which also takes into account semantically related documents.

Many works have explored measures of Web documents "from a search engine perspective" [18]. It has been found out that usually Web documents change trivially or in their markup only [6]. On the other hand, news pages containing "breaking news" change their content frequently and significantly.

This update heterogeneity severely impacts Web content management systems (WCMS), which should alleviate the continual maintenance of Web documents. More often than not, WCMS pretend an increased "freshness" of a Web page that changed gradually only. Worse, this notion of freshness is not application specific. Changes may be of syntactic or semantic nature. Syntactic changes can reflect editing efforts or improve readability (although the semantics is not changed). Semantic changes can increase the relevance of a document to specific purposes. We find related areas for semantic methods in text classification, retrieval, and summarization [19,13,9,17]. Since the date of the "last-modifying" is used for filtering and sorting, it is just fair to authors and readers if the WCMS generates the *freshness* of a Web document automatically using an algorithm that takes into account the *degree* of changes w.r.t. the application at hand.

Due to the importance of Web sites and the increasingly complex and collaborative Web publishing process, versioning of Web documents is an essential feature of WCMS [11]. Since the history of each document is available in such a system, a history-aware metric of changes can be implemented. This metric is essential if the freshness of a document should be estimated automatically or some versions should be vacuumed to free space [5].

In this paper, we present an approach to calculate the freshness of a document automatically based on its complete history. An important parameter is a *document metric*, which measures how much a document has been changed. This metric may be of syntactic or semantic nature and can be tuned to specific applications. We have implemented our approach in our WCMS [10], in which an XML structure represents a whole Web site, where each leaf stands for a Web page, containing further XHTML markup. For test purposes we also have implemented a document metric (based on XML nodes in this context) to estimate the syntactic distance between two versions of a Web page. By our metric not only the plain text information but also the XHTML markups

[4] Also called "recency" or "freshness."

Pattern	Change Frequency	Change Extent	Change Content	Usage
News page	hourly	large	text / markup	commercial
Home page	monthly / yearly	small	text / markup	private
Boards	minutely / hourly	large	text	private
Online stores	minutely / hourly	large	text	commercial
Enterprise site	monthly / yearly	small	text / markup	commercial
WCMS	minutely / hourly	medium	text	private / comm.

Table 1
Change patterns for Web documents

are compared to each other. In this way the real human resource usage for a Web document can be reflected. Although our metric is purely syntactic, we have achieved surprisingly good results in measuring editing efforts in student works. Since XSLT pre- and post-processing are involved, the metric can be easily adapted to special situations for creating and updating the document. Particularly, we have applied our approach to Chinese Web documents.

Whether a Web document is relevant for a reader depends not only on the document's own history but also on so-called external factors, i.e., the historical evolution of semantically related documents. This proves useful, e.g., for news pages that change frequently. In our setting, semantic relations [8,12] cover aspects like "is translation of," "provides background information," "is essential part of," or "is recommended reading." Therefore, we also calculate the *up-to-dateness* of a document w.r.t. the historic evolution of semantically related documents.

The aim of our approach is to provide a language- and topic-independent algorithm that determines real up-to-dateness of documents in a WCMS. In addition, old versions of a document without significant contribution to up-to-dateness of the current version (or any version in the future) might be vacuumed to free space. The contribution of this paper is a flexible approach to calculate the up-to-dateness of documents based on their own history and on the history of semantically related documents. The major enabling factors are version control and explicit semantic links. The most significant parameter is a document metric, which can be tuned to specific applications.

From here, we proceed as follows. In Sect. 2 we introduce the running example for this paper by which we illustrate our approach. We address the freshness of a single document in Sect. 3. Sect. 4 describes the implementation of our approach in our XML-centric WCMS. In Sect. 5 we approach up-to-dateness of a document w.r.t. semantically related documents. We conclude this paper and sketch directions for future research in Sect. 6.

Version 1:
Ukrainians Hit Polls to Elect President
KIEV, Ukraine - Rival candidates Viktor Yushchenko and Viktor Yanukovych faced off Sunday in a repeat election triggered by a fraudulent runoff vote and massive protests that resulted in an unprecedented third round in Ukraine's fiercely waged presidential contest.
...

Version 2:
Ukraine Elects President in Runoff Vote
KIEV, Ukraine - Rival candidates Viktor Yushchenko and Viktor Yanukovych faced off Sunday in a repeat election triggered by a fraudulent runoff vote and massive protests that resulted in an unprecedented third round in Ukraine's fiercely waged presidential contest.
...

Version 8:
Ukraine Holds Presidential Vote a 3rd Time
KIEV, Ukraine - Rival candidates Viktor Yushchenko and Viktor Yanukovych faced off Sunday in a repeat election that all sides hoped would resolve Ukraine's fiercely waged presdential contest after fraud wrecked one vote and prompted massive protests that deeply divided the nation.
...

Version 9:
Exit Polls Give Yushchenko the Presidency
KIEV, Ukraine - Three exit polls projected Ukrainian opposition leader Viktor Yushchenko the winner by a commanding margin over Prime Minister Viktor Yanukovych in Sunday's fiercely fought presidential rematch.
...

Fig. 1. Version history of an article about the Ukrainian President's Vote

2 News Pages — A Challenge for Up-to-dateness

In order to illustrate the viability of our approach, we introduce this compelling and vivid example, which will be referred to in the following sections. This running example was taken right out of practice and hence reveals large actuality. In our experiments we found typical patterns regarding the modifications of Web documents (see Tab. 1). For this paper, we choose the news page pattern, which is particularly suitable because it is characterized by an hourly change combined with a large extent of textual alteration. In addition, news pages show extensive semantic interrelation. The usage of our approach is not limited to news pages. Up-to-dateness of any document in a WCMS under version control can be estimated. In a collaborative authoring environment even the partial contribution of each author to a document section can be calculated as well.

The President's Vote in the Ukraine on December, 26 2004 represents a perfect example within the news page pattern. This day constituted of a large amount of breaking news worldwide, concerning forecasts, results, background information, and opinions. Starting hourly from 12.00 CET to 21.00 CET the sources of four different news pages (CNN.com, MSNBC.com, YahooNews.com, USAToday.com, derstandard.at) were downloaded and saved to a database, in order to later apply our approach. Each download represents a different version of a news page. Commercial banners and scripts were eliminated before using our metric.

Fig. 1 illustrates the version history of the first paragraph of the YahooNews.com breaking news Web page about the Ukrainian President's Vote. [5] Changes between versions range from correcting typos or changing the layout towards dramatic changes in content. For example, we see a slight change between versions one and two, whereas between versions eight and nine the article was rewritten almost completely. Clearly, the freshness of this news page should represent these facts.

3 Freshness of a Single Document

Up-to-dateness of a document depends on two aspects: its own freshness and the historical evolution of semantically related documents. In this section, we only deal with the freshness of a single document, which can be seen as a document property based on the historical development. Up-to-dateness w.r.t. semantically related documents is dealt with in Sect. 5.

3.1 Approaching Freshness

For a document with only a single version the freshness is given by the time t_1 at which the document was saved. If a document has multiple versions $1, \ldots, n$, its freshness t^* might be expressed as a weighted average time stamp of all versions:

$$t^* := \frac{\sum_{i=1}^{n} t_i \cdot c_i}{\sum_{i=1}^{n} c_i}$$

where c_i, the *contribution* of the version i to the end version n, is still to be determined. A common and trivial way is to set

$$c_i = \begin{cases} 1 \text{ if } i = n \\ 0 \text{ otherwise} \end{cases}$$

so we have the "last-modified" time stamp, which ignores the contributions of all past versions.

To take into account the contributions of all versions, one must know how much a document has been changed from one version to another one, which we call the *distance* $D_{i,j}$ between versions i and j of a document. Since all Web pages are under version control in our WCMS, such distances can be easily employed. Recall that our approach is parametric in the document metric D. For example, we can calculate the syntactic distance between two versions of an XML document by analyzing the modifications of XML nodes using a so-called *XML Diff Language Diffgram*. Currently, we achieve surprisingly good results by simply defining $D_{i,j}$ as the number of necessary XML node

[5] URL story.news.yahoo.com/news?tmpl=story&u=/ap/20041226/ap_on_re_eu/ukraine_election&e=1&ncid=

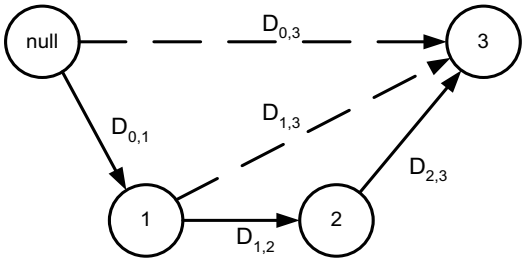


Fig. 2. Distances between different versions of a document

modifications to change the content of version i to that of the version j. For the moment, we have chosen this simple metric because originally we wanted to measure the editing effort of students. Moreover, we want to find out further applications that benefit from such a simple approach already. See Sect. 4.2 for further details about the implementation. Of course, this simple metric can be easily replaced by a semantic metric, depending on the specific application [9,17].

Based on the metric D we find two possible definitions of the contribution of a given version to the current version of a document. The basic idea is illustrated in Fig. 2, where nodes represent the versions of a Web document and edges represent the distance. The empty document, from which the first version origins, is denoted as null. The contribution of version 2, for example, may be defined as $D_{1,3} - D_{2,3}$ or "how much the distance to the end version 3 has decreased from version 1 to version 2." Alternatively, the contribution can be defined as $D_{1,2}$ or "how much the document has been changed from version 1 to version 2". Notice that the contributions of some versions may be negative in the first case.

In the first case we have

$$\bar{c}_i := D_{i-1,n} - D_{i,n}$$

where $D_{0,i}$ is the distance between the empty document to version i. Then the *effective freshness* of a document with n versions is given by (since $D_{n,n} = 0$):

$$\begin{aligned}
\bar{t}_n &= \frac{\sum_{i=1}^{n} t_i \cdot (D_{i-1,n} - D_{i,n})}{\sum_{i=1}^{n} (D_{i-1,n} - D_{i,n})} \\
&= \frac{t_1 \cdot \sum_{i=1}^{n} (D_{i-1,n} - D_{i,n}) + \sum_{i=1}^{n} (t_i - t_1) \cdot (D_{i-1,n} - D_{i,n})}{\sum_{i=1}^{n} D_{i-1,n} - \sum_{i=1}^{n} D_{i,n}} \\
&= \frac{t_1 D_{0,n} + \sum_{i=2}^{n} t_i D_{i-1,n} - \sum_{i=1}^{n-1} t_i \cdot D_{i,n} - t_n D_{n,n}}{D_{0,n} - D_{n,n}} \\
&= t_1 + \sum_{i=2}^{n} (t_i - t_1) \cdot \frac{D_{i-1,n} - D_{i,n}}{D_{0,n}}
\end{aligned}$$

If a document only has a single version ($n = 1$) the effective freshness is t_1, as expected. Each additional version may increase the effective freshness,

depending on the time difference between the new version and the first version, and depending on how much the content has been changed comparing with *all* past versions. Using this algorithm when a new version of a Web document is added to the WCMS, a comparison to each past version must be carried out. At the moment, we are investigating whether we could relieve this restriction to the last "x" versions.

In the second case we have

$$\tilde{c}_i := D_{i-1,i}.$$

Then the *incremental freshness* of a document with n versions is given by:

$$\begin{aligned}\tilde{t}_n &= \frac{\sum_{i=1}^{n} t_i(D_{i-1,i})}{\sum_{i=1}^{n} D_{i-1,i}} \\ &= \frac{t_n D_{n-1,n} + \sum_{i=1}^{n-1} t_i D_{i-1,i}}{D_{n-1,n} + \sum_{i=1}^{n-1} D_{i-1,i}} \\ &= \frac{t_n \cdot D_{n-1,n} + \tilde{t}_{n-1} \cdot \sum_{i=1}^{n-1} D_{i-1,i}}{D_{n-1,n} + \sum_{i=1}^{n-1} D_{i-1,i}}\end{aligned}$$

Notice that $\tilde{t}_n$ can be calculated incrementally using $\tilde{t}_{n-1}$ and the accumulated $\sum_{i=1}^{n-1} D_{i-1,i}$. If a document only has a single version ($n = 1$), the incremental freshness yields t_1, as expected. Each additional version increases the incremental freshness, depending on the time difference between the new version and the first version, and depending on how much the content has been changed compared to the *previous* version. A comparison only to the previous version is necessary, which reduces computational complexity substantially.

Unfortunately as pointed out by [3]: "...resemblance is not transitive, ...for instance consecutive versions of a paper might well be 'roughly the same', but version 100 is probably quite different from version 1." Or in the reverse case: If extensive changes made to a version have been undone completely in the following version, no real increase of freshness is achieved whilst a comparison between the consecutive versions might pretend a significant increase. Therefore, we expect that $\bar{t}_n$ resembles our idea of freshness better than $\tilde{t}_n$ at the cost of additional computation.

3.2 Freshness of a News Page

When we apply our metrics to our example news page it turns out that only the time consuming and complex calculation of the effective freshness $\bar{t}_n$ yields useful results.

For each of the ten versions of the breaking news about the Ukrainian President's Vote, which we found at YahooNews.com we recorded the last-modified

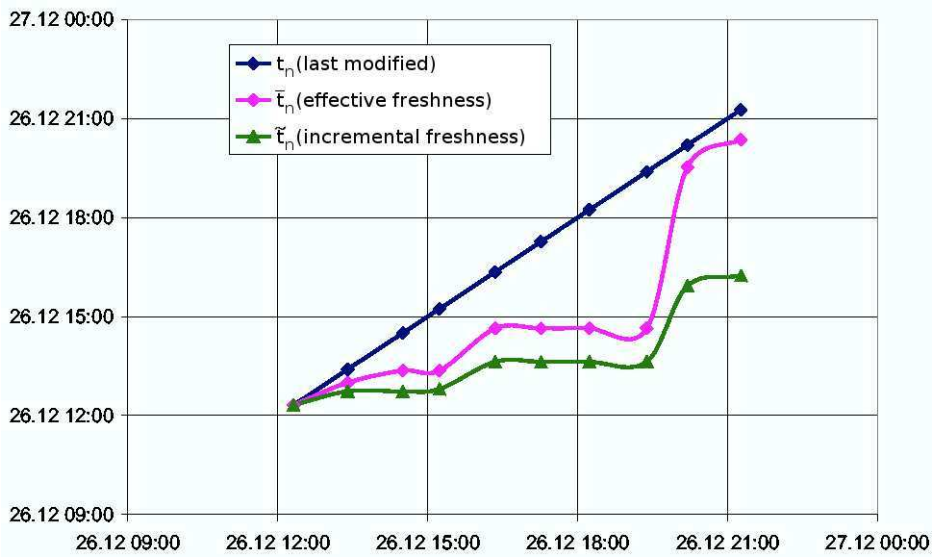


Fig. 3. Recorded and calculated time stamps for our example news page

time stamp t_n, and calculated the effective freshness $\bar{t}_n$ and the incremental freshness $\tilde{t}_n$, discussed in Sect. 3.1. As shown in Fig. 3 the graphs representing $\bar{t}_n$ and $\tilde{t}_n$, respectively, are below the graph representing t_n. The graph of $\bar{t}_n$ makes a jump towards t_n at the ninth version, which is caused by many changes made to that version. This jump is also visible in the graph of $\tilde{t}_n$, but it does not reveal the significance of the content change.

We did the same test for the top news about the same topic at another news site[6]. The difference between $\bar{t}_n$ and $\tilde{t}_n$ there is even more significant since the document has been changed completely many times.

As a matter of fact, if a document has been changed completely the effective freshness should be set to the last-modified time stamp, as the calculation of $\bar{t}_n$ delivers.

4 An Up-to-dateness Aware WCMS

We have implemented our approach into our XML-centric WCMS, which supports versioning of XML contents [10]. An external sub-system — XMLDiff [15] including pre- and post-processing — has been set up for the estimation of distances between the different versions of a Web document and to calculate the effective freshness. Due to our open architecture, other (semantics based) document metrics can be easily "plugged in."

4.1 XML-centric WCMS

The WCMS has been developed by the authors based on Windows Active Server Pages technology. Its architecture is XML centric regarding information

[6] URL derstandard.at/druck/?id=1901315

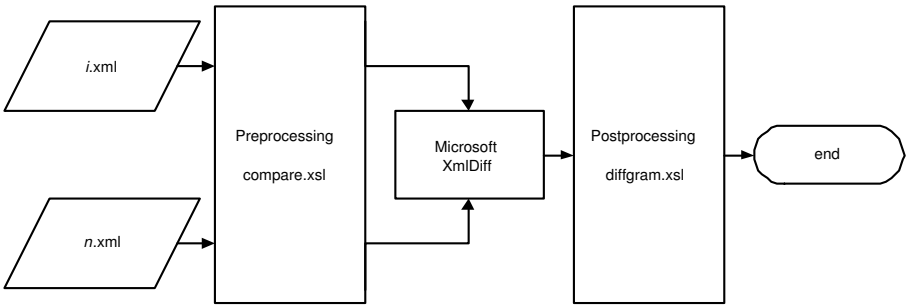


Fig. 4. XMLDiff including pre- and post-processing

processing and data storage. The WCMS uses the Open Source HTML editor RichTextEdit [7], which provides a WYSIWYG user interface and converts HTML information into valid XHTML. The WCMS makes available all the necessary XML data for a Web page and supplies XSLT templates that should be used for generating HTML information from these data.

When a client requests an HTML page, the WCMS responds with XML information from the database. An XML-capable browser translates this information into presentable HTML using the associated XSLT template, which it fetches from the server in a second request. For non-XML browsers the XML information can be translated to HTML on the server side as well.

Because of its XML-centric and simple data model on the database layer, the WCMS is flexible regarding extensions. In most cases, extensions can be realized by making adaptions in XSLT. Besides the current version of each Web document all past versions of the same document, their authors, and "last-modified" time stamps can also be retrieved.

4.2 A Simple Syntactic Document Metric based on XMLDiff

Fig. 4 gives an overview of the sub-system for comparing different versions of a Web document, i.e., determining the difference $D_{i,n}$ between versions i and n. The sub-system is implemented in PHP and can be triggered manually. It retrieves all versions of a given Web document from the WCMS and pre-processes them via XSLT. This transformation accomplishes several goals.

Firstly, the WCMS holds internal metadata that should not be compared. For example, there are the URL of the separate image server or the login name of the current user. Since our WCMS is an experimental system there are even debugging and profiling entries included. The pre-processing simply removes these (in this case) superfluous metadata.

Secondly, it takes different effort to add or change different XHTML nodes. The pre-processing compensates these differences by adding or removing nodes. For example, a paragraph may have more or less words. During the pre-processing each word is transformed to a node, such that text modifications can be detected more exactly. For example, Chinese texts are divided into single "characters" (see Fig. 5). We do so because Chinese word boundaries

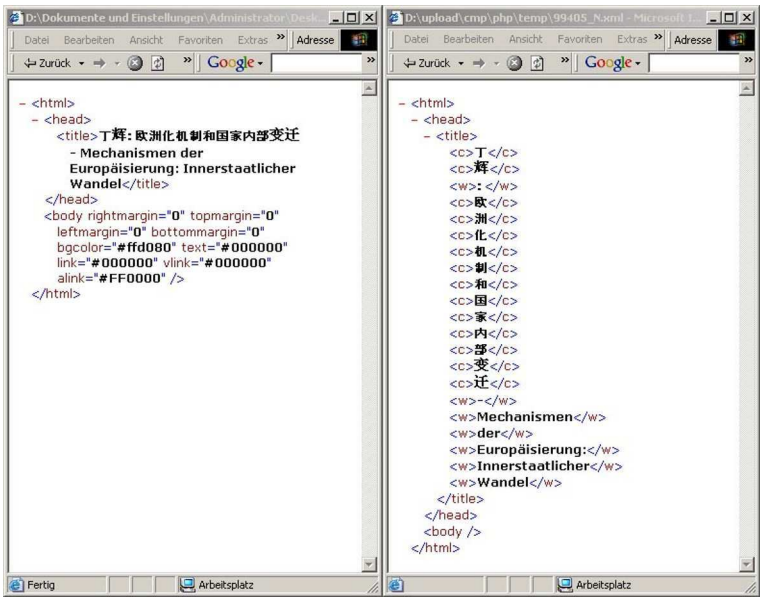


Fig. 5. Pre-processing Chinese texts

are not marked using white spaces and cannot be detected using traditional document metrics.

Finally, the HTML editor we use may generate additional entries pretending more additions or changes. For example, when the width of a table column has been adjusted, changes might be detected in each row of the table. The pre-processing suppresses these additional changes.

The pre-processed documents are compared using Microsoft's XMLDiff [15], which represents the changes using XDL, a proprietary XML-based language for describing differences between two XML documents. XDL Diffgrams contain information regarding additions, changes, or removals of document content, or content being moved within the XML tree. During the XSLT post-processing, the entries in an XDL Diffgram are compiled and the number of modifications is calculated. Recall that XMLDiff is just a parameter, similar to other (maybe semantics based) document metrics [13].

4.3 Measuring Editing Effort by a Syntactic Document Metric

Our WCMS represents a realistic testing environment as it is used by students to write their final theses and term papers and, therefore, contains many different documents. In order to examine our approach, we have tested our syntactic metric on several students' works as well as on generated test cases. Indeed, this was our original idea. The examined students works were all from the area of business administration. Henceforward, they were characterized by a relatively similar structure especially concerning the relation of text to non-text parts, e.g., tables, links, and pictures.

To be able to draw general conclusions based on the empirical results of the examined students works, the main focus lay on several test cases that

have been generated. The purpose of these tests was to determine whether the efforts in writing corresponded to the effective change estimated by our metric. The test cases have been generated based upon the variety of different functions available in the built-in text editor RichTextEdit. To cover the whole process of document writing with all its components, the work on objects (e.g., tables, links, or pictures) and the document style was taken into account. XMLDiff distinguishes four kinds of changes (additions, changes, removals, and moving of nodes) within its comparison between the two XML files, so these different types of changes were tested.

Each test case represented a different action, which was derived from the combination of a function with one of the four possible changes (e.g., "add table" or "delete picture"). The average time to perform each action (e.g., delete a picture) was measured using a stopwatch in several independent runs to receive the temporary effort. In the next step the metric was applied on a document where this specific action had been realized and the resulting value (XML node) was related to the average temporary effort measured.

Our metric calculates with nodes, each word representing an individual node. If new words have been added to a document, our metric presents these changes throughout its result, which is a figure of nodes. For text writing (add text), each node measured by our metric corresponds to 1.92 seconds of real editing effort. Since removing a single word or an associated group of words represents little real time effort only (by average 2 seconds per removal), this action is treated by the system as a single node, which leads to the average figure of 2 seconds per node for text removal. The real editing effort of all different actions, using objects like tables, pictures etc. were analyzed, too, and expressed by the average value of real time effort per node. These values ranged from 2 to 6 seconds per node. In the students works, objects like tables or pictures were relatively rare in relation to the text parts. Therefore, the small deviation of these values from the values of text creation (1.92 seconds) and text removal (2 seconds) have almost no impact on the overall real editing effort of a whole document.

By using this easily adjustable XML-centric system including the pre- and post-processing, the influence of each action and object could be treated and, therefore, lead to a discretional adjustment of the real editing effort value. In dependence on the application area of the text (e.g., business administration or computer science) or the individual author, by using the real editing effort statistics, the real time effort could be personalized or altered w.r.t. the type of text.

5 Determining Up-to-dateness of Multiple Documents

Web pages do not exist on their own. Instead, they are semantically related to each other. Due to these semantic relations, changes in the up-to-dateness of one Web page "somehow" influences the up-to-dateness of related Web pages.

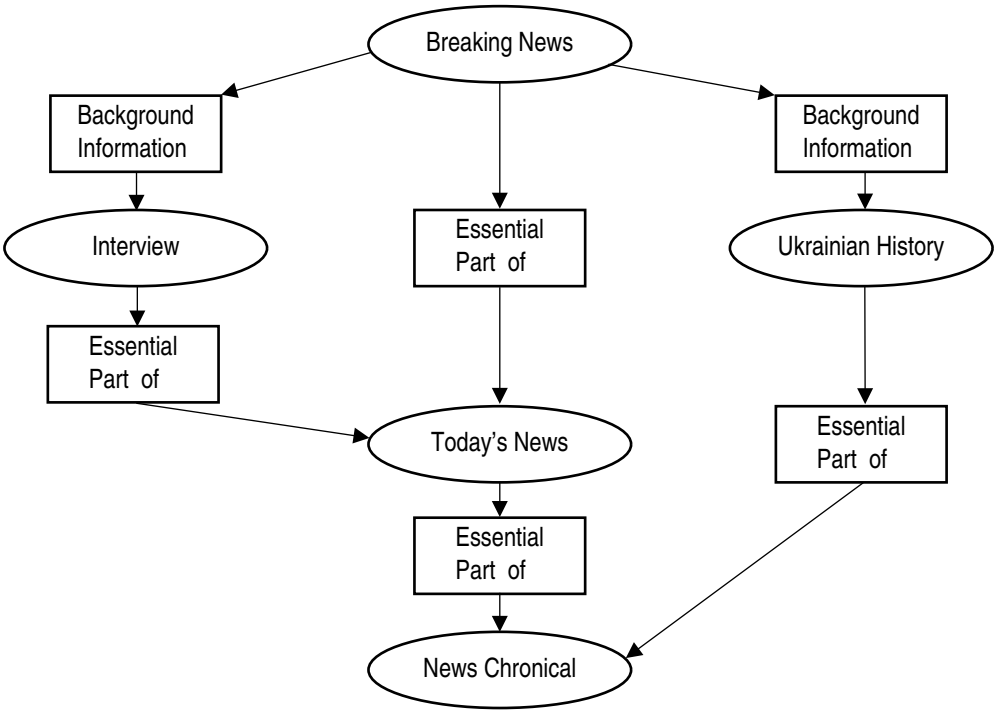


Fig. 6. Semantic structure of our example Web site

Consider our running example again: Our breaking news page is hosted at YahooNews.com, which also contains interviews and background information. For the purposes of this paper, suppose that there are an interview and an historical article about the Ukraine at YahooNews.com revealing further background information. In addition, the breaking news page and the interview are part of a "Today's News" summary, which itself is part of a news chronical. Of course, the historical article is part of the chronical, too.

5.1 Semantic Relations

Clearly, we would expect that up-to-dateness of the chronical is influenced by the up-to-dateness of the breaking news page. This is because the chronical is (indirectly) semantically related to the breaking news page.

We use the hypertext model as basis for representing semantic relations, similarly to [8]. Semantic relations are represented via *semantic links*, which are treated as first-class entities connecting source documents with target documents. A semantic link can connect multiple source documents to multiple target documents.[7] The semantic structure of a Web site (see Fig. 6) is represented through a bipartite graph, where ellipses represent documents and rectangles represent semantic links. Source and target documents, re-

[7] Notice that this is different from embedded links in HTML pages. Treating links as first-class entities gives us much more flexibility, because it supports multiple link structures on the same documents without altering the documents themselves.

spectively, of a semantic link are denoted by directed edges. Our idea is to propagate changes in the up-to-dateness of the source documents to the target documents. The example contains bidirectional links only; our approach is, however, independent of link arity.

Currently, semantic links are added by hand. Often, they are in reverse direction to the usual link structures in Web pages; e.g., the "Today's News" page links to the breaking news page and the interview. In the future, we plan to derive semantic links, e.g., based on embedded HTML links or semantic similarity via latent semantic linking techniques [14].

Semantic links are typed by the number of source and target documents, respectively. A semantic link of type $\tau_{n,m}$ has n source documents and m target documents. [8] This type is associated to an up-to-dateness propagation function $[\![\tau_{n,m}]\!] : \mathbb{R}^n \to \mathbb{R}^m$, which given up-to-dateness changes in the source documents calculates up-to-dateness changes of the target documents. On an update of a source document d, we can propagate its up-to-dateness changes along the edges of the semantic graph. Notice that we can reach a target document d' at different paths origining from d, each of which may require to change the up-to-dateness of d' differently. Also, circles in the semantic graph are quite usual for Web sites and are, therefore, permitted in our setting.

5.2 Propagating Up-to-dateness Changes

Consider that a document d has been updated to such an extent that its freshness changes. Then we immediately update d's up-to-dateness according to the change of its freshness, which naturally determines d's up-to-dateness change $\Delta(d)$. Up-to-dateness changes of other documents are set to zero. Our propagation algorithm traverses the semantic graph, where it regards d as root. Each edge that has been traversed is marked as "processed." Each document node is marked as "processed" if all incoming edges have been marked as "processed." That way we avoid running into circles. We traverse the semantic graph as follows:

- Processing a document node d:
  If all incoming edges are marked as "processed," then update d's up-to-dateness according to its up-to-dateness change $\Delta(d)$, and process all semantic links emanating from d and mark their edges as "processed." Otherwise, process any semantic link targeting d whose edge has not been processed already. Processing a semantic link will update d's up-to-dateness change $\Delta(d)$ and return to d to further process incoming links or emanating links.
- Processing a semantic link node l of type $\tau_{n,m}$:
  First, process all source documents of l that have not been processed already and mark their edges as "processed." Processing these documents

[8] Due to the simplicity of our example document metric D, we do not need to further type source and target documents. Of course, this may change if we employ a more sophisticated document metric.

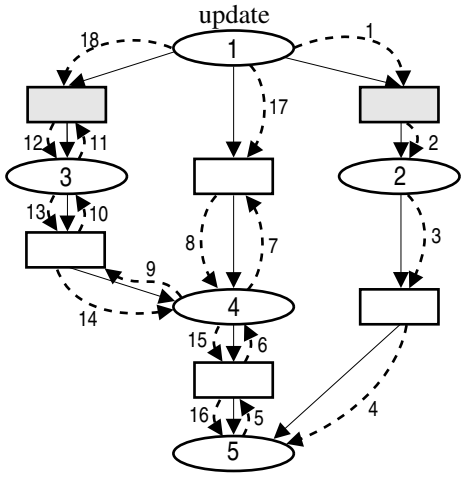


Fig. 7. Up-to-dateness propagation of our example breaking news page

will determine their up-to-dateness changes. Second, apply l's update function $[\![\tau_{n,m}]\!]$ to the up-to-dateness changes $\Delta(d_i)$ of the source documents d_i ($i \in \{1, \ldots, n\}$). This results in m up-to-dateness changes $\Delta_l(d'_j)$ for the target documents d'_j ($j \in \{1, \ldots, m\}$). Update the (already calculated) up-to-dateness changes $\Delta(d'_j)$ of the target documents d'_j to the maximum of $\Delta_l(d'_j)$ and $\Delta(d'_j)$. Third, process all target documents d'_j of l and mark their edges as "processed."

Our algorithm is non-deterministic because we do not make any assumptions about the order in which nodes are processed. Since up-to-dateness changes are updated to the maximum, however, our algorithm always yields the same result, no matter in which order nodes are processed. [9] Notice that up-to-dateness resembles freshness of "lonely" documents, which are not semantically related to other documents, and as long as no semantically related document has been changed. Otherwise, the up-to-dateness of a document may differ from its freshness.

5.3 Propagating Up-to-dateness of our Example Web Site

For example, consider an update of our example breaking news page. Fig. 7 shows a possible traversal of our algorithm through our example Web site. Ellipses, rectangles, and edges represent the semantic graph as shown in Fig. 6. "Background Information" links are marked grey, in order to distinguish them from "Essential Part of" links. Dotted arrows show how our algorithm traverses the graph; they are numbered according to the order in which nodes are processed. Document nodes are numbered according to the order in which they are *marked* as "processed;" i.e., the order in which their new up-to-dateness is determined.

[9] Instead of the maximum, any commutative and associative function can be used here.

From version eight to nine the effective freshness of the breaking news page jumps up by five hours (see Fig 3). Given the up-to-dateness propagation functions

$$[\![\text{Background Information}]\!](src) = src \cdot 0.5 \qquad [\![\text{Essential Part of}]\!](src) = src \cdot 0.2$$

our algorithm yields the following up-to-dateness changes:

$$\begin{aligned} \Delta(\text{Breaking News}) &= 5.0 \text{ hrs} & \Delta(\text{Interview}) &= 2.5 \text{ hrs} \\ \Delta(\text{Today's News}) &= 1.0 \text{ hrs} & \Delta(\text{Ukrainian History}) &= 2.5 \text{ hrs} \\ \Delta(\text{News Chronical}) &= 0.5 \text{ hrs} \end{aligned}$$

That is the up-to-dateness of the chronical increases by half an hour.

6 Conclusions and Outlook

This paper describes our approach towards automatic management of up-to-dateness of documents that are managed by an XML-centric WCMS. We introduce two measures: the freshness of a document is based on its own history only; the up-to-dateness of a document also employs semantic relations to other documents. *Freshness* of a multiversioned Web document is calculated w.r.t. a document metric, which detects changes between document versions. Currently we use a syntactic metric based on the modifications of XML-nodes, which roughly reflects editing effort. Due to our open architecture, this metric can be easily replaced by an application-specific semantic metric. Since pre- and post-processing using XSLT are included, the document metric can be easily adapted to different human-machine-interfaces (HMI) or user groups. *Up-to-dateness* is based on semantic relations between documents. Changes in up-to-dateness are propagated along these relations.

In the future, we plan to extend our WCMS by recording the document process time automatically. Not only in this way more data on the real effort for document processing by different users should be collected for further validations of the document metrics. We believe that such metrics are a key success factor for managing document processes as one of the crucial parts of business process management (BPM). We also want to learn more document authoring patterns for which content management systems and collaborative authoring systems can be optimized. Monitoring the deviation between freshness and up-to-dateness of a document might also contribute to authoring patterns. For comparing XML documents we consider to use other tools like [4] and metrics that are based on the semantics of documents like [13,14]. Also, we will implement a plug-in mechanism into our WCMS that supports easy integration of user-defined document metrics. For easier maintenance we currently stick to a central WCMS. It would be interesting to see how our approach scales to distributed Web sites, which use persistent URLs to address Web documents.

In addition, we plan to integrate our up-to-dateness approach with automated consistency management as offered by CDET [20] or xlinkit [16]. Clearly, up-to-dateness changes give rise to consistency constraints based on semantic links. For example, consider a German translation of our breaking news page, which should be as up to date as the English original.

In summary, we think that our approach provides a good basis to manage real up-to-dateness of Web documents beyond the simple "last-modifying" time stamp, which has proven insufficient for many purposes.

References

[1] T. Berners-Lee. Information management: A proposal, 1989. www.w3.org/History/1989/proposal.html.

[2] B. E. Brewington and G. Cybenko. How dynamic is the Web? In *Proc. of the 9th Int. World Wide Web Conf.*, Amsterdam, The Netherlands, 2000. W3C.

[3] A. Broder, S. Glassman, M. Manasse, and G. Zweig. Syntactic clustering of the Web. In *Proc. of the 6th Int. World Wide Web Conf.*, pages 391–404, Santa Clara, CA, 1997. W3C.

[4] G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in XML documents. In *Proc. of the 18th Int. Conf. on Data Engineering*, San Jose, CA, 2002. IEEE CS Press.

[5] C. Dyreson, H.-L. Lin, and Y. Wang. Managing versions of Web documents in a transaction-time Web server. In *Proc. of the 13th Int. World Wide Web Conf.*, pages 422–432, New York, NY, 2004. W3C.

[6] D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A large-scale study of the evolution of Web pages. In *Proc. of the 12th Int. World Wide Web Conf.*, pages 669–678, Budapest, Hungary, 2003. W3C.

[7] A. France et al. RichTextEdit, 2003. www.richtext.org.uk/.

[8] S. C. Gupta, T. N. Nguyen, and E. V. Munson. The software concordance: Using a uniform document model to integrate program analysis and hypermedia. In *Proc. of 10th Asia-Pacific Software Engineering Conf.*, pages 164 – 173, Chiang Mai, Thailand, 2003. IEEE CS Press.

[9] J. Hand. *Feasibility of using citations as document summaries.* PhD thesis, Drexel University, 2003.

[10] B. Hu and F. Lauck. Prototype of a Web and XML based collaborative authoring system. In *Proc. of the Int. Conf. on Computing, Communications and Control Technologies*, volume IV, pages 79–84, Austin, TX, 2004. IIIS.

[11] A. Kirby, P. Rayson, T. Rodden, I. Sommerville, and A. Dix. Versioning the Web. In *7th Int. Wkshp. on Software Configuration Management*, pages 163–173, Boston, MA, 1997.

[12] B. Krieg-Brückner, A. Lindow, C. Lüth, A. Mahnke, and G. Russell. Semantic interrelation of documents via an ontology. In *DeLFI 2004: Die e-Learning Fachtagung Informatik, Tagung der Fachgruppe e-Learning der Gesellschaft für Informatik e.V.*, pages 271–282. Gesellschaft für Informatik e.V., 2004.

[13] N. Littlestone. Learning quickly when irrelevant attributes are abound: A new linear threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[14] A. Macedo, M. Pimentel, and J. Guerrero. Latent semantic linking over homogeneous repositories. In *Proc. of the 2001 ACM Symp. on Document engineering*, pages 144–151, Atlanta, GA, 2001. ACM Press.

[15] Microsoft Corp. XMLDiff 1.0. apps.gotdotnet.com/xmltools/xmldiff/, 2002.

[16] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein. xlinkit: a consistency checking and smart link generation service. *ACM Trans. Inter. Tech.*, 2(2):151–185, 2002.

[17] K. Nigam and M. Hurst. Towards a robust metric of opinion. In *AAAI Spring Symposium on Exploring Attitude and Affect in Text: Theories and Applications*, Palo Alto, CA, 2004. Stanford University.

[18] A. Ntoulas, J. Cho, and C. Olston. What's new on the Web? the evolution of the Web from a search engine perspective. In *Proc. of the 13th Int. World-Wide Web Conf.*, pages 1–12, New York, NY, 2004. W3C.

[19] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., 1986.

[20] J. Scheffczyk, U. M. Borghoff, P. Rödig, and L. Schmitz. Managing inconsistent repositories via prioritized repairs. In *Proc. of the 2004 ACM Symp. on Document Engineering*, pages 137–146, Milwaukee, WI, 2004. ACM Press. see also www2-data.informatik.unibw-muenchen.de/cde.html.

Validating Scripted Web-Pages

Dr R G Stone [1]

Department of Computer Science
Loughborough University
Leicestershire, LE11 3TU, England

Abstract

The validation of XML documents against a DTD is well understood and tools exist to accomplish this task. But the problem considered here is the validation of a generator of XML documents. The desired outcome is to establish for a particular generator that it is incapable of producing invalid output. Many (X)HTML web pages are generated from a document containing embedded scripts written in languages such as PHP. Existing tools can validate any particular instance of the XHTML generated from the document. Howevere there is no tool for validating the document itself, guaranteeing that all instances that might be generated are valid.

A prototype validating tool for scripted-documents has been developed which uses a notation developed to capture the generalised output from the document and a systematically augmented DTD.

Key words: VALIDATION, XHTML, WML, PHP, DTD.

1 Introduction

The validation of a static web-page against a DTD can be achieved by certain browsers (e.g. Internet Explorer[1]), by web-based services (such as that offered by W3C[2], WDG[3]) and by commercial products (such as the CSE HTML Validator[4]).

The problem of validating *generators* of web pages has been tackled by various researchers by constructing controlled environments where invalid output is not possible[5,6,7]. This has been done by controlled macro substitution or by the design and use of a special purpose language. This solves the problem neatly for those able and willing to adopt a new strategy but has nothing to offer for the legacy problem which is addressed here. Millions of web documents exist, scripted using languages like PHP[8], which are capable of generating

[1] Email: R.G.Stone@lboro.ac.uk

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

different XML pages each time they are browsed but there is no method by which the source document itself can be validated.

For presentation the examples used will be of PHP generating WML but the techniques used apply equally well to other scripting languages and other XML compliant languages, notably XHTML.

2 Embedded Scripting

A web-page containing server-side scripting must have the script executed before being passed to the browser. There are several server-side scripting languages (PHP[8], ASP[9], Perl[10], etc.). At its simplest, a server-side scripting language generates its output by echo or print commands. The scripted elements are often embedded among the marked-up text so the code to generate a minimal WML page using PHP could look like this

```
<wml>
<?php
      echo "<card>";
      echo "</card>";
?>
</wml>
```

In this and subsequent examples, the required `<?xml ...>` header and the `<!DOCTYPE wml ...>` header lines are omitted for brevity. Also note that PHP code is written inside 'brackets' which can be written

```
<?php ... ?>
```

and which can, in certain circumstances, be abbreviated to

```
<? ... ?>
```

3 Validation against a DTD

The context of this paper is where a script is used to deliver a page that is valid XML according to a Document Type Definition (DTD)[11]. A DTD describes the tags that can be used, their attributes and the content that the tags enclose. As an example, a simplified extract of the WML DTD[12] can be shown as

```
<!ELEMENT wml ( card+ )>
<!ELEMENT card ( p* )>
<!ELEMENT p ( #PCDATA )*>
```

This DTD notation can be read as follows. For a document to be a valid WML document there must be a single wml element which must contain at least one (+) `card` element. Each `card` element may contain zero or more (*) paragraph elements (`p`). Finally each paragraph element may contain an

arbitrary amount of 'Parsed Character Data' (meaning anything that is not a tagged element). The part of the DTD which defines attribute structure is not shown. The output of the script in the previous section would be

```
<wml><card></card></wml>
```

and this would therefore be acceptable and be taken to be exercising the right to have no paragraph elements (p).

4 Informal Validation of Scripted Web-Pages

Here is an example of a PHP script which contains a structured statement (a loop)

```
<wml>
<card>
<?
      while($i<$limit){
            echo "<p>";
            ...
            echo "</p>";
            $i++;
      }
?>
</card>
</wml>
```

We might argue informally that, whatever the value of `$limit`, the result of this script is valid WML because the while-loop, when executed, will always generate paragraph tags (`<p>`, `</p>`) in pairs and that the `<card>` tag accepts any number of such pairs (including none).

A more formal way of approaching this is to capture the output of the script using the star notation borrowed from regular expressions

```
<wml> <card> ( <p> ... </p> )* </card> </wml>
```

This may be read as describing the output as a wml tag containing a card tag which in turn contains zero or more paragraph tags. It is this output expression which is 'checked' against the WML DTD. The wml element contains exactly one card element (1 or more is allowed) and the card element contains zero or more paragraph elements (zero or more allowed). The idea of using regular expression notation to capture the generalised output from a script is developed further in what follows. However the notation is converted into XML style so that the result can still be validated by a DTD obtained by augmenting the original with extra rules. Hence

```
<wml> <card> ( <p>...</p> )* </card> </wml>
```

will become

```
<wml> <card> <p_list0> <p>...</p> </p_list0> </card> </wml>
```

Other invented tags like `<p_list0>` will eventually be needed and they will be referred to as *meta*-tags.

5 Generalised Output and Augmenting the DTD

A system is envisaged in which the scripted web-page is passed through a processor to obtain the generalised output expression and the generalised output expression is then validated against a DTD which has been obtained by augmenting the original DTD with rules involving the *meta*-tags. The various repetition and selection control structures in the scripting language will require appropriate *meta*-tags to describe their contribution to the generalised output expression. These are summarised in Table 1. The correspondence with the regular expression operators used in the DTD which is shown in the same table will provide the insight into how the DTD should be augmented to accept the meta-tags.

Continuing the example in the previous section, if a scripted while loop has produced

```
<p_list0> <p>...</p> </p_list0>
```

the DTD will need to be augmented to accept this as a replacement for

```
<p>...</p> <p>...</p> ...  <p>...</p>
```

For this example it would be sufficient to replace all occurrences of `p*` in the DTD with `(p*|p_list0)` and to add the definition

```
<!ELEMENT p_list0 ( p )>
```

Concept	RegExp	Program Control	Example Code	Meta-tag
0,1,2,...	*	while loop	`while()...`	`<t_list0>`
1,2,3,...	+	repeat loop	`do...while()`	`<t_list1>`
option	?	short conditional	`if()...`	`<t_option>`
choice	|	long conditional	`if()...else...`	`<t_choices>`

Table 1
A table of correspondences between regular expression operators, program control structures and meta-tags

However only the simplest case has been considered so far where a sequence of literal paragraph elements has been created entirely by a simple while loop. In the more general case a script may be written to generate a sequence of paragraph elements using any mixture of literal tags, loops and conditionals.

The following example is more realistic as it creates a sequence of paragraph elements via a sequence involving literals, a loop and a conditional:

```
<wml>
<card>
<?
      echo "<p>...</p>";
      while(...){
            echo "<p>...</p>";
      }
      if(...)echo "<p>...</p>";
?>
</card>
</wml>
```

In this case the generalised output expression will look like

```
<wml>
<card>
      <p>...</p>
      <p_list0>
            <p>...</p>
      </p_list0>
      <p_option>
            <p>...</p>
      </p_option>
</card>
</wml>
```

To express this generality the entity $p0$ is introduced so that $p*$ in the DTD is replaced by $(\%p0;)*$ with the definition

```
<!ENTITY % p0 (p|p_list0|p_list1|p_option|p_choices) >
```

Under this definition $(\%p0;)*$ means a sequence of zero or more elements each of which contributes zero or more paragraph elements.

This rule must be repeated for all tags (t), so that wherever$t*$ occurs in the DTD it is to be replaced by $\%t.star$; under the definitions

<!ENTITY % t.star (%t0;)* >

<!ENTITY % t0 (t|t_list0|t_list1|t_option|t_choices) >

Note that in (...|t|...)*, where the * applies to various alternatives including t, the t should also be replaced by the entity %t0.

6 The Augmented DTD

All of the changes to the DTD so far have been motivated by the occurrence of 'zero or more' tagged elements, including meta-tagged elements, in the output expression which are validated by substituting occurrences of $t*$ in

the DTD. Obviously it now remains to look at what other parts of the DTD might need augmenting. Repeat loops with their signature output of 'one or more' can be captured by the meta-tag *t_list*1 and would be expected to cause substitutions for $t+$ within the DTD. Short conditionals (no else part) with their signature 'optional' output can be captured by the meta-tag *t_option* and would be expected to cause substitutions for $t?$ within the DTD. Long conditionals (with an else part) have a signature 'alternative' output and can be captured by the meta-tags *t_choices* and *t_choice* like this

```
<t_choices><t_choice>...this...</t_choice>
    <t_choice>...or this...</t_choice></t_choices>
```

A long conditional would be expected to cause substitutions for any unadorned instances of t (that is an occurrence of t in the absence of any of the operators '*','+','?') because alternative choices for a single tag t are being offered.

The substitution for $t+$ in the DTD is more complicated than for $t*$ because it is necessary to ensure that at least one element with tag t is present. Before considering the substitution in detail, compare the following four entity definitions:

(i) Zero or more occurrences of elements t, `t0` (presented earlier)

```
<!ENTITY % t0 ( t|t_list0|t_list1|t_option|t_choices )>
```

(ii) One or more occurrences of elements t, `t1`

```
<!ENTITY % t1 ( t|t_choices|t_list1 )>
```

(iii) Zero or one occurrences of element t, `t01`

```
<!ENTITY % t01 ( t|t_option|t_choices) >
```

(iv) Exactly one element t, `t11`

```
<!ENTITY % t11 ( t|t_choices )>
```

It is now possible to replace $t+$ by the entity `t.plus` under the definition

```
<!ENTITY % t.plus ( (t_option|t_list0)*, %t1; , %t.star; ) >
```

This can be read as defining `t.plus` to be zero or more elements that cannot be relied upon to contain a t tag, followed by an element which definitely contains at least one t tag, followed by zero or more elements which will contribute zero or more t tags.

The substitution for $t?$ in the DTD is the entity `t01` with the definition already given. The substitution for t is the entity `t11` with the definition already given.

The substitutions to be made to the DTD are summarised in Table 2. To support these substitutions there are the new entities `t_star`, `t_plus`, `t0`, `t1`, `t01` and `t11` to be added as defined above and finally the new element rules describing the derived tags *t*_list0, *t*_list1, *t*_option, *t*_choices and *t*_choice for each tag t.

DTD phrase	replacement
$t*$	$\%t.star;$
$(...\|t\|...)*$	$(...\|\%t0;\|...)*$
$t+$	$\%t.plus;$
$(...\|t\|...)+$	$(...\|\%t1;\|...)+$
$t?$	$\%t01;$
t	$\%t11;$

Table 2
An table of replacements to be made in the DTD

```
<!ELEMENT t_list0 %t.star; >
<!ELEMENT t_list1 %t.plus; >
<!ELEMENT t_option %t01; >
<!ELEMENT t_choices (t_choice,t_choice) >
<!ELEMENT t_choice %t11; >
```

Note that the augmentation rules do not alter the meaning of the DTD when no meta-tags are present. For example if $t*$ is replaced by $t0*$ and $t0$ is defined to be $(t|t\_list0|t\_list1|t\_option|t\_choices)$ then, in the situation where no meta-tags (t_list0, t_list1, t_option, $t\_choices$) are present, the substitution degenerates back to $t*$.

In the prototype the process of augmenting the DTD is handled by a prolog program which reads the original DTD, generates the extra ELEMENT definitions and ENTITY definitions and outputs the augmented DTD. This is made easier in SWI-prolog[15] by using a pre-written module[16] to read the DTD.

7 The Script Processor

Earlier it was stated that the script validation system was constructed of two parts. The first part has to process the script, introduce the meta-tags and generate the generalised output expression. The second part validates the output expression against an augmented DTD. In the prototype the first part, the script processor, has itself been split into two stages. The script processor first generates an output expression using general meta-tags like $list0$, $list1$, $option$ and $choices$. A second stage inspects the output of the first and inserts the correct tags to change these to specific meta-tags like p_list0, $card\_option$.

In the current implementation the first stage of the script processor is written in C using LEX[13] and YACC[14] to parse the script and this stage

produces an output expression containing general meta-tags. For example

```
<wml> <card> <list0> <p>...</p> </list0> </card> </wml>
```

The second stage is written in prolog and produces specific meta-tags, for example

```
<wml> <card> <p_list0> <p>...</p> </p_list0> </card> </wml>
```

8 Current implementation

The current implementation for PHP scripts producing WML and XHTML works perfectly well on a large class of scripts. However, if it fails to validate a script, it is not necessarily the case that the script is capable of emitting invalid output. The weak point is the first stage where the meta-tags are inserted. The problem lies with assuming that a control structure in the script language will generate a complete tagged structure capable of being described by the meta-tags. This does not always happen. An example to illustrate this would be

```
echo "<p>";
echo "0";
while(...){
      echo "</p>";
      echo "<p>";
      echo "1";
}
echo "</p>";
```

For any particular execution this script will result in a sequence like

```
<p> 0 </p> <p> 1 </p> <p> 1 </p> <p> 1    ...    </p>
```

which is valid. However it will be given the following meta-tags

```
<p> 0    <list0> </p> <p> 1 </list0>   </p>
```

This expression, in which the tags are not properly nested, fails the second stage of the process (replacing general meta-tags with specific meta-tags) because the input stage assumes that the input is well-formed XML.

Work has begun to introduce an extra middle stage into the processor which uses rules along the lines of

```
ab(cab)*c  =>  abc(abc)*  =>  (abc)+
```

so that the example above can be manipulated to

```
<p> 0 </p> <list0>  <p> 1 </p> </list0>
```

The problem with this is that the starting expression is not valid XML precisely because the tags are not properly nested, so that the expression cannot be read and manipulated as an XML document. This means that the

manipulation has to be done by treating the expression merely as a linear mixture of starting tags, ending tags and non tag elements. This makes the processing harder but not intractable.

A more serious problem exists with the current code which replaces general meta-tags with specific meta-tags. At present, if the processor meets a opening `<list0>` tag it checks all the top-level tags up to the closing `</list0>` tag expecting them all to be of the same type (t say) so that the general tag `<list0>` can be changed to `<t_list0>`. This will not always be the case as in the following example

```
echo "<p>";
while(...){
      echo "<ul>...</ul>";
      echo "<br />";
}
echo "</p>";
```

The processor is presented with

```
<list0><ul>...</ul><br /></list0>
```

and cannot find a tag name t to change `<list0>` to `<t_list0>`. There are potential solutions to this. One is that with reference to the DTD it may be possible to change the scope of the `<list0>` tags thus:

```
<list0><ul>...</ul></list0> <list0><br /></list0>
```

Although this changes the meaning of the expression, if the DTD contains a rule along the lines of

```
<!ELEMENT p (...|ul|...|br|...)* >
```

the change will not alter the validity of the expression and so the validity check on the new expression will obtain the desired result. In practice it has been possible in many cases like this for the programmer to circumvent the issue by adding an enclosing `<span>` or `<div>` tag within the loop.

A further problem lies with the simplicity of the first stage of the processor. Because it is largely syntactic in nature it does not, and cannot, actually execute the script language. This means that if the script generates any tags by any other method than printing literals (for example by constructing them by string concatenation or obtaining them as part of a database lookup) then these tags will not be represented in the generalised output and consequently these tags will not be validated.

9 Summary

The concept of validating a scripted web-page rather than its output is thought to be novel and potentially very useful, at least for the large number of legacy sites which use this technology. A method has been found to validate such

scripts which depends on processing the script to provide a generalised output expression and then validating this against an augmented DTD. The method has been prototyped for PHP scripts generating WML and XHTML. The method is readily applicable to any other combination of procedural scripting language and XML-based output.

Although the method can validate a large class of scripts it has its limitations. The processor which produces the generalised output expression has to be able to recognise where the script is generating tags. The current prototype requires these to be literal text within an echo/print command and not 'hidden' by string manipulation operators or resulting from database lookup. The current prototype also requires control statements within the script to generate well-formed XML, although there are plans to extend the processor to accommodate non well-formed output in situations where special rules can be applied which are derived from regular expression equivalences.

References

[1] using Internet Explorer as a validator: http://www.w3schools.com/dtd/dtd_validation.asp.

[2] W3C validation service: http://validator.w3.org/.

[3] WDG validation service: http://www.htmlhelp.com/.

[4] CSE validator: http://www.htmlvalidator.com.

[5] JWIG: http://www.brics.dk/JWIG.

[6] CDuce: http://www.cduce.org/.

[7] XDuce: http://xduce.sourceforge.net/.

[8] PHP main web-site: http://www.php.net.

[9] ASP web reference: http://msdn.microsoft.com/asp/.

[10] PERL web reference: http://www.perl.com/.

[11] DTD web reference: http://www.w3schools.com/dtd/default.asp.

[12] WML DTD web reference: http://www.wapforum.org/DTD/wml_1_1.dtd.

[13] LEX, Unix Programmers Manual (see also web reference: http://dinosaur.compilertools.net/.

[14] YACC, Unix Programmers Manual (see also web reference: http://dinosaur.compilertools.net/.

[15] SWI-Prolog web reference: http://www.swi-prolog.org/.

[16] SWI-Prolog SGML/XML parser package web reference: http://www.swi-prolog.org/packages/sgml2pl.html.

Testing web applications in practice

Javier Jesús Gutiérrez, Maria José Escalona, Manuel Mejías, Jesús Torres
Department of Computer Languages and Systems.
University of Seville.
{javierj, escalona, risoto, jtorres}@lsi.us.es

ABSTRACT

Software testing process is gaining importance at same time that size and complexity of software are growing. The specifics characteristics of web applications, like client-server architecture, heterogeneous languages and technologies or massive concurrent access, makes hard adapting classic software testing practices and strategies to web applications. This work exposes an overview of testing processes and techniques applied to web development. This work also describes a practical example testing a simple web application using only open-source tools.

1. INTRODUCTION

Internet gives to developers a new and innovative way to build software. Internet also allows the access of millions of user to a web application [4]. Thus, problems in a web application can affect to millions of users, cause many costs to business [2] and destroy a commercial image.

The Business Internet Group of San Francisco undertook a study to capture, assess and quantify the integrity of web applications on 41 specific web sites. According to the report, of those 41 sites, 28 sites contained web application failures [14]. Thus, software testing acquires a vital importance in web application development.

First web applications had a simple design based on static HTML pages. Nowadays, web applications are much more complex and interactive with dynamic information and customized user interfaces. Design, support and test modern web applications have many challenges to developers and software engineers.

This work is organized as follow. Section 1 defines some basic concepts used in this work, shows the basic aspects of a client-server web application and introduces software testing process. Section 2 describes a simple web application used as example in this work. Section 3 describes how to make unit testing over a web application. Section 4 describes how to make integration testing over a web application. Section 5 resumes conclusions and future work.

1.1. Definitions

A web page is all kind of information that can be displayed into a browser window [2]. A web page uses to be composed by HTML code, generated statically or dynamically, or by client-side executable components, like Macromedia Flash modules or Java Applets.

A web site is a group of web pages, where the information of every page is semantically related and syntactically related by links among them. User access to a web site is made by HTTP requests. Client-side user interface uses to be a browser program running over a personal computer (PC, Mac, etc.).

A web application is built applying heterogeneous technologies like client-side scripting languages included into HTML, client-side components like Java applets, server-side scripting languages like PHP or PERL, server-side components like Java Servlets, web services, databases servers, etc. All these heterogeneous technologies have to work together, obtaining a multi-user, multiplatform application.

1.2. Client-Server Architecture in web applications

Functioning of a web application is similar to classic client-server application with thin client, as showed in Figure 1. When a user writes a web address in a browser, this one acts like a client, requesting a file to a server accessible by Internet. Server processes the request and sends the file. Client, at the time to receive the file, process its content and shows them.

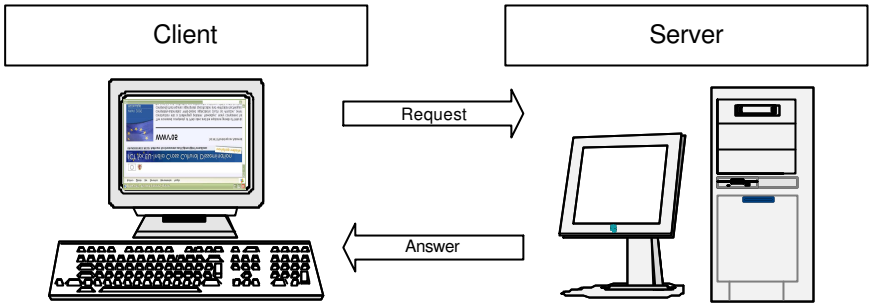


Figure 1. Static web application

First web applications were composed only by static web pages. They have not the possibility to modify their content depending by date, user, or number of requests. The user always received the same file with the same information into it. Actually, as showed in Figure 2, it is possible to build web pages dynamically, changing their information depending of many factors.

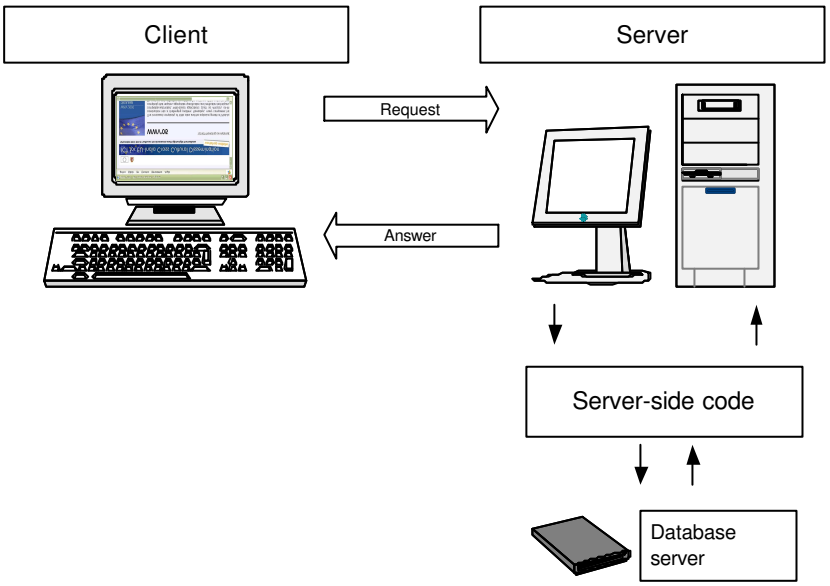


Figure 2. Dynamic web application..

Client-Server architecture in Figure 2 is the same that Client-Server architecture in Figure 1. Main different is that the server in figure 2 has two elements, one dedicated to receive requests and answer to them, and other dedicated to execute web application and generate HTML dynamically.

1.3 An overview of software testing process

Nowadays, test process has became in a vital task in development process of all kind of software systems [3]. It is needed to make a classification [1] of testing, the moment to apply them and their objectives before expose how to apply test process to web applications. Table 1 shows this classification.

Kinds of tests	Moment to apply	Description
Unit testing.	During building of software system.	Unit testing verifies design and functionality of every component of the system.
Integration testing.	During building of software system.	Integration testing verifies the union among system components through their interfaces and their functionality.
System testing.	After building of software system.	System testing verifies in depth the functionality of the system, as a black box, checking that all requirements have been implemented in the correctly.
Implantation testing.	During production environment implantation.	Implantation testing verifies the correct function of the system in the real production environment.
Acceptance testing.	After software system implantation.	Acceptance testing verifies that system has all requirements expected and satisfies the needs of the user.
Regression testing.	During modify Software system.	Regression testing verifies that changes in code do not generate unexpected errors.

Table 1. Testing classification.

Unit and integration testing verifies components of the system. System, implantation and acceptance testing verifies the entire system as a black box, from different points of view. This word is focused in unit and integration test only.

1.4. Related work

There are several works describing how to test a web application. For example, Liu [16] considers each web application component as an object and generates test cases based on data flow between those objects. Ricca [15] proposes a model based on the Unified Modeling Language (UML), to enable web application evolution analysis and test case generation. Wu [17] defines a generic analysis model that characterizes both static and dynamic aspects of web based applications. This technique is based on identifying atomic elements of dynamic web pages that have static structure and dynamic contents. Elbaum [18] explores the notion that user session data gathered as users operate web applications can be successfully employed in the testing of those applications.

This paper does not proposal a new web test model but a set of techniques to test web component. These techniques can be used in any web test model proposal to implemented test cases. These techniques can be applied to test both client-side and server-side components and they are useful to put test model in practice.

2. PRACTICAL CASE

This section describes a simple web application to insert customers in a database. In client-side, application is composed of three web pages: a form to insert customers, a web page with a message if insertion was possible and another web page with a message if insertion was not possible. In server-side, application is composed of a MySQL [8] database server and a PHP [9] insertion script. Figure 3 shows these components.

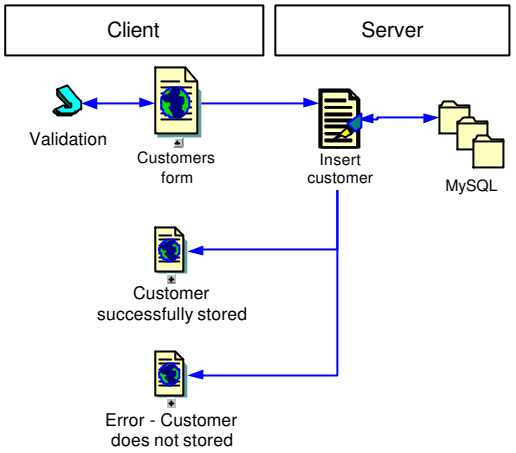


Figure 3. Insert customer web application components.

Figure 4 shows captures of HTML web pages of the application.

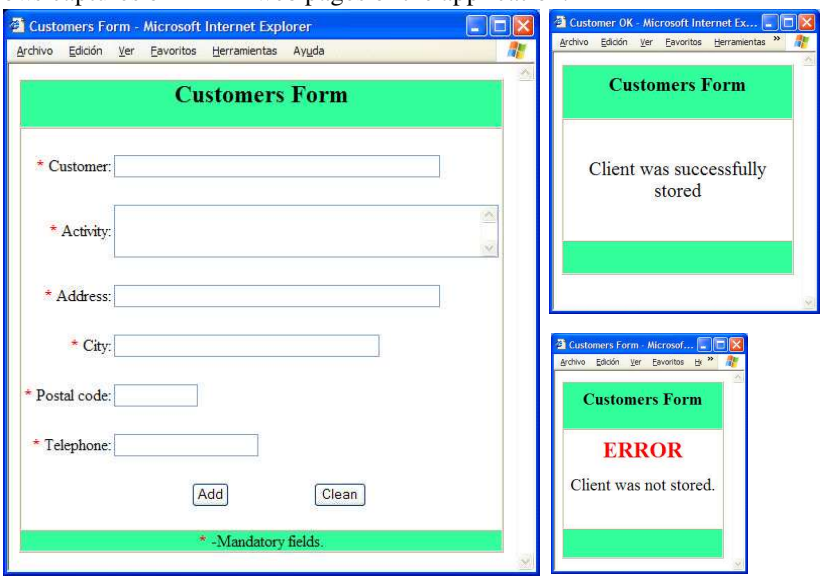


Figure 4. Insert customer form.

Validation code, written in JavaScript and included into HTML form, will verify that none obligatory field will be empty.

This application stores customers in a MySQL table. SQL code to create customers table are showed in Figure 5.

```
CREATE TABLE Customers(Id BIGINT(20)
  UNSIGNED NOT NULL AUTO_INCREMENT
  PRIMARY KEY,
Entity VARCHAR(50) NOT NUL        L,
Activity VARCHAR(250) NOT NULL,
Address VARCHAR(50) NOT NULL,
City VARCHAR(50) NOT NULL,
ZIP_Code VARCHAR(10) NOT NULL,
Telephone VARCHAR(50) NOT NULL,
Contact_person VARCHAR(50),
Contact_phone VARCHAR(10),
Observations VARCHAR(250) );
```

Figure 5. Customer SQL code.

To test this application, we will write, at first time, a set of unit tests to verify client-side components and server-side components. At second time, we will write an integration test to verify the correct working of customer form and insertion script together.

3. UNIT TESTING

Objective of unit testing is to verify the functionality of every component in isolation. To do this, we are going to divide components in two sets: client-side components (web pages, JavaScript code, etc.) and server-side components (server-scripts, databases, etc.) [1]. Each component set will have its own testing strategy and tools. Division in our example web application is showed in Figure 6.

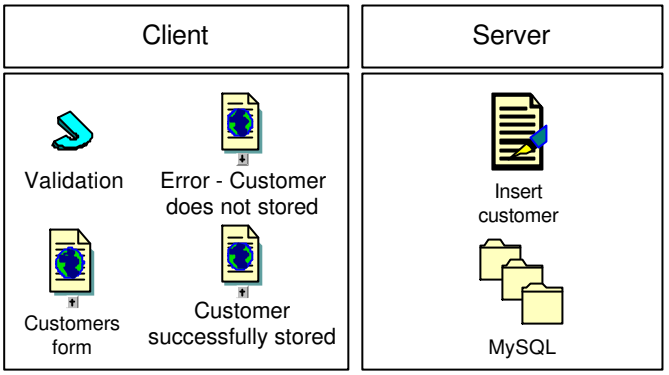


Figure 6. Client side and server side components.

Client-side components are downloaded and executed in client web browser. Server-side components are executed in server and their results are sent to client. Section 3.1 describes techniques and tools to test server-side components. Section 3.2 describes techniques and tools to test client-side components.

3.1. Server-side testing

The strategy to test server code is similar to strategy to develop unit testing in not-web applications. Main idea is to write a code fragment to execute the code under test with a set of test values and compares it result with expected results. In our sample web application we are going to write a test script in PHP.

There are many open-source tools to make easy and automatist this process. In general, all tools implements JUnit architecture [11]. A list of xUnit tools can be found in [7]. We have selected PEAR PHPUnit [13], among all PHP xUnit available.

The example web application has a function to insert a customer with one expected parameter with a table with all information about customer. This function returns a value that indicates if the client was or was not inserted. The prototype of this function is shown in Figure 7. Test case will invoke function in figure 7 with a test customer and will check result returned.

```
// $result is TRUE if customer was added into database
// and FALSE if does not.
function insertCustomer($customer)
{
      //...
      return $result;
}
```

Figure 7. InsertCustomer function prototype.

The unit test is divided in two actions. These actions are described in Table 4.

Step	Action	Verification
1	To call function “insertCustomer” with a test customer.	To verify that function result is TRUE.
2	To search test customer inserted in step 1 in customer database.	To verify that customer exists in database and its values are equals those values of test customer.

Table 4. Test steps.

The PHP scripts which implements this unit test is showed in Figure 8. First line includes PHPUnit libraries which offer functions similar to JUnit.

```
<?
  include_once('./PHPUnit/PHPUnit.php');
  include_once('./     InsertCustomerFunction       .php');

  class InsertCustomerTest extends PHPUnit_TestCase {

    var $  test_cust  omer ;
    function testInsert      Customer  () {
      $this   ->PHPUnit_TestCase("testInsertarUnCliente");
    }
    function setUp() {
      $this   ->test_customer   ['customer'] = "    test_customer    ";
      $this   ->test_customer   ['activity'] = "    test_activity    ";
      $this   ->test_customer   ['address'] = "    test_address    ";
      $this   ->test_customer   ['city '] = "  test_city   ";
      $this   ->test_customer   ['ZIP_Code  '] = "00000";
        $this ->test_customer   ['telephone  '] = "000  -00-00-00";
    }
    function testInsert      ACustomer  () {
      $  result = insert  Customer  ($this ->test_customer    );
      $this   ->assertTrue($result);
    }
    function test    CustomerInserted    (){
      $conn = mysql_connect("localhost", "", "")
      mysql_select_db("C       ustomer  ")
      $sql = "SELECT * FROM       Customer  WHERE  'custome  r'='".
            $this    ->test_customer   ['customer'   ]."'";
        $result  = mysql_query($sql)
      $  Customer   = mysql_fetch_array($resul) ;
      $this   ->assertEquals($this     ->test_customer   ['customer'   ],
                        $       Customer  ['customer'   ], " Different   customers   .");
      //   ...

      mysql_free_result($result);
    }
}
  echo "<HTML> <BODY> <HR> <BR>          Insert Customer Test      . <BR>";
  $suite  = new PHPUnit_TestSuite("InsertCustomerTest ");
  $result = PHPUnit::run($suite);
  echo $result      -> toStrin  g();
  echo "<HR>";
?>
```

Figure 8. InsertCustomer function unit test.

If “insertCustomer” function has no error, test script will write an output like figure 9. That output indicates that test was success.

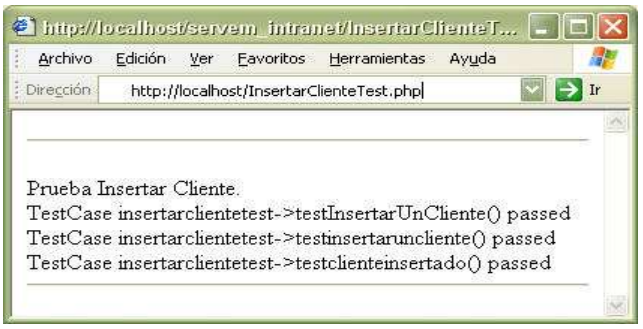


Figure 9. Successfully test.

3.2. Client-side testing

Objectives of dient-side components testing are to verify that HTML is correct and compliments standards [10] and to verify dynamic components into web pages. For example, we will verify that HTML satisfied HTML 4.01 Transitional standard and the JavaScript validation code in our example web application. It is important to fulfil HTML standards to guarantee that a web page is correctly visualized in different browsers.

Other unit test that can apply to client-side components are to verify that web pages are correctly visualized in different browsers, verify user interface usability, etc.

3.2.1. HTML web pages testing

A HTML web page contains the information that will be displayed and a set of tags that indicates how that information has to be displayed. Thus, we have to test that every web page in our web application example satisfied HTML standards proposed by W3C consortium. A HTML validation tools is available at W3C consortium web site [6]. We have used that tool to verify our pages. Results are resumed in Figure 10.

Validation form has an option to upload a web page to validate. In our example web application, when customer form is validated, some errors appeared:

Line 107, column 38: document type does not allow element "BODY" here
<body bgcolor="#FFFFFF" text="#000000" >
Line 108, column 75: there is no attribute "BORDERCOLOR"
... cellpadding="0" align="left" bordercolor= "#0066FF">

First error is because of before <body> tag must be </head> tag.
"Bordercolor" attrib is obsolete and does not complaint version 4.0 of HTML specification [10]. Style sheets must be used to define color.
Once corrected two errors, validation tool shows next message: "This Page Is Valid HTML 4.01 Transitional!"
The other two web pages have no errors.

Figure 10. Insert customers HTML form test.

To avoid testing of each web page in isolation and uploading pages one by one, we can use an option to write and URL and let application to verify all HTML pages in that URL. There are also, applications that connect with W3C web site.

3.2.2. Testing JavaScript code

The form to insert customers includes JavaScript code to avoid blank fields. It is necessary to test this code to verify that its functionality is the expected one and it is able to detect all invalid combinations possible.

Originally, JavaScript code was included into HTML web form. This one makes hard to test it. So, JavaScript code was refactorized and moved into its own script file called “validate.js”. Illustration 11 shows a fragment of validation code.

```
function Validador(      CustomerForm    ) {

    if ( CustomerForm    .customer   .value == "") {
        alert(" Field   \"Customer  \" is obligatory     .");
        CustomerForm    .customer   .focus();
        return (false);
    }

    // ....

    return (true);
}
```

Ilustración 11.Validate.js fragment code.

Code showed in Illustration 11 notices if an obligatory field is empty. Code displays a message window, showed in Figure 12 and Figure 13, and sets form focus in empty field.

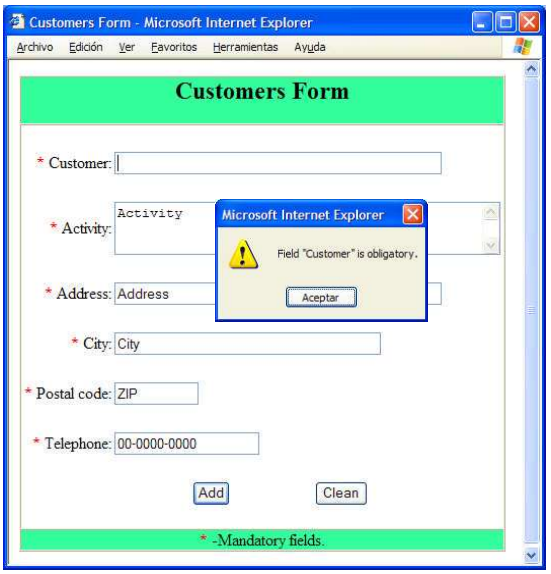


Figure 12. An error in customers form.

Figure 13. Alert window detail.

The first strategy to validate JavaScript code is to apply manual testing. Manual testing can be performed by a worker who will write all combinations possible and who will verifies that JavaScript code results are the expected results. This is the simplest solution, but worse too. Manual testing requires dedication in exclusive of one programmer, allows many errors because it is a very repetitive and bored task, and it is needed to make again all test every time customers form or JavaScript code changes.

Another strategy is to write a unit test case to test JavaScript code. Unit test case has to verify that functionality of “validate.js” is the expected functionality when applying a set of values, in the same way

that test written in section 3.1. To test "validate.js" in isolation, it is needed to write an object that represents form. A fragment of that code is showed in Figure 14.

```
function customer (value ) {
    this.value= value ;
    return(this);
}

function CustomerForm (e, a, d, l, c, t) {
     this. customer =new customer (e);
     this. activity =new activity (a);
     this. address =new address (d);
     this. city =new city (l);
     this.postal _code =new postal _code (c);
     this. telephone =new telephone (t);
     return (this);
}
```

Figure 14. JavaScript object representing customer form.

A test to verify validation code using the representation of the form is showed in Figure 15.

```
<SCRIPT LANGUAGE="JavaScript" SRC="validar.js"></SCRIPT>
<SCRIPT LANGUAGE="JavaScript" SRC="          customerForm    .js"></SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
  var form= new      CustomerForm  ("","a", "d", "l", "c", "t");
  if (Validador(form) == false) {
    alert("   Test failed   .");
  } else {
    alert("   Test valid   .");
  }
</SCRIPT>
```

Figure 15. Test of form customer JavaScript code.

It will be possible to write similar tests based on test described in Figure 15, changing the position of blank field in customer form creation. Test in Figure 15 verifies that function return expected value. However, this strategy is not good because. Test does not verify that alert message has expected text or the form focus will be in the empty field. Even more, test stops after executing "alert" instruction, so manual verification is still needed. Another problem is that test has to be loaded into the browser to be executed. This strategy has no advantages from manual test strategy. We still need a programmer who changes the code, execute the test and verifies that alert message is expected message.

Other strategies to improve JavaScript testing are to use a JavaScript interpreter instead executing test in a web browser or to use a mouse-record tool. Improvement of JavaScript testability is a future work.

This same process can also be done to verify other client-side scripts, like VBScript.

4. INTEGRATION TESTING

Once verified each components of the system with unit tests, it is time to verify that those components are able to work right among them. This one is the goal of integration tests.

In a web application, components in client-side communicate with components in server-side using HTTP protocol, instead of classic messages to methods or functions. This allows a very low coupling but makes useless classic xUnit tools to develop integration tests.

There are two main techniques to develop integration testing in a web application to develop tests that operates application through it HTML web interface. One technique is using stored macros, and replaying them to make a test. Macros have to be record again when interface or application changes, so they are not the best option. Another technique is using an API to analyze HTTP communication. This API allows to write a program that sends HTTP requests and receives and analyzes their answers. This second technique is more flexible, and allow to test in depth web pages, but they spend more development time. We are going to write test based on API to test our web example application.

There are many open-source tools that offer APIs to write integration tools. We have chosen HttpUnit [6] to write an example. HttpUnit it is written in Java, but we will show that it is possible to use it to test a PHP web application with a HTML user interface.

4.1. Writing an integration test

HttpUnit can request a web page in the same way than a web client. HttpUnit offers an interface to ask if received page includes certain elements and to interact with the elements of the page, by example navigating through a link. This tool offers almost all functionality of a web browser, like cookies control, header analysis, GET and POSTS, etc.

In this point, we are going to write a test with HttpUnit. This test will be a Java class that request customer form and, later, verifies that the web page received has expected elements.

Test goes to verify:

1. Connection with server and customer form requesting.
2. Verify title of received page to test if it is the expected web page.
3. Verify if web received page contains customer form.
4. Verify if form includes all expected fields and assign them test values.
5. Verify if, when pressing add button, web server answers with expected page.

Figure 16 shows java code of this test.

```
import net.sourceforge.jwebunit.WebTestCase;
import com.meterware.httpunit.*;
import com.meterware.servletunit.*;
import java.util.*;
import junit.framework.*;

public class Test CustomerForm extends TestCase {
    public Test CustomerForm () {
        super(" Test CustomerForm ");
    }
    public void testInsert Customer ()
      throws Exception
    {
      WebConversation wc = new WebConversation();
      WebResponse resp = wc.getR esponse("http://localhost/CustomForm.htm");
     Assert.assertEquals(resp.getTitle().compareTo(" Customer Form "), 0);
      WebForm form = resp.getFormWithName(" CustomerForm ");
      Assert.assertNotNull(form);
      form.setParameter(" customer ", " test_cus tomer ");
        form.setParameter(" activity ", " test_activity ");
      form.setParameter(" address ", " test_address ");
      form.setParameter(" city ", " test_city ");
      form.setParameter("postal _code ", "00000");
        form.setParameter("tele phone ", "00 -00-00");
      WebRequest req = form.getRequest("Submit");
      resp = wc.getResponse(req);
      String output = resp.getText();
      Assert.assertEquals( output.indexOf("Error"), -1 );
    }
}
```

Figure 16. Customer form integration test.

It will be possible to add an additional validation to verify that testing customer inserted is really stored into database. This validation can check that insertion script works right when it is called from HTML form, not only when it is directly called.

5. CONCLUSIONS

This work shows how it is possible to build a web application and apply it different testing process with open-source tools only. All techniques exposed in this work can be easily applied to other web development platforms like ASP.NET or JSP and Servlets. Some of tests showed in this work are development with Java over a PHP web application, which demonstrates the interoperability among technologies and languages that web engineering offers.

All tools used in this work are free for use and download through Internet and, also, their source code is accessible, even in commercials tools like MySQL (although free for use depends of license of application).

We have seen with our example web application, that it is very important to separate different components of a web application, presentation components like HTML and code components like PHP scripts, to facility testing process. A good separation among components improves development process, testing process and maintainability.

Future lines of investigation from this work are to investigate new tools and strategies to test web user interfaces built with HTML and JavaScript, study with examples the application of this techniques and practices in other development platforms like .NET and other components, like Flash user interfaces, and study strategies to test HTML dynamically generated.

REFERENCES

[1] Ash, L. 2003. *The Web Testing Companion: The Insider's Guide to Efficient and Effective Tests*. John Wiley & Sons, Hoboken, USA.

[2] Ye Wu, Jeff Offutt, Xiaochen Du. 2004. *Modeling and Testing of Dynamic Aspects of Web Applicationsy*. Submitted for journal publication

[3] M.J. Escalona, M. Mejías, J.J. Gutiérrez, J. Torres. 2004. *Métodos de Testing Sobre La Ingeniería De Requisitos Web de NDT*. IADIS WWW/Internet 2.004. 353-360.

[4] Jeff Offutt et-al. 2004. *Web Application Bypass Testing*. ISSRE '04

[5] Métrica v3. http://www.csi.map.es/csi/metrica3/

[6] HttpUnit. http://httpunit.sourceforge.net/

[7] Herramientas xUnit. http://www.xprogramming.com/software.htm

[8] MySQL. http://www.mysql.com

[9] PHP. http://www.php.net/

[10] HTML 4.01 Specification. http://www.w3.org/TR/html4/

[11] Junit. http://junit.org/

[12]W3C HTML Validator. http://validator.w3.org/

[13] PEAR PHPUnit. http://pear.php.net/package/PHPUnit/

[14] BIGSF. Government Web Application Integrity. The Business Internet Group of San Francisco, 2003.

[15] F. Ricca, P. Tonella. Analysis and testing of web applications. In Proceedings of the 23rd international conference on Software Engineering. IEEE Computer Society Press, 2001.

[16] S. Chun, J. Outt. Generating test cases for XML-based web application. In Proceedings of the 12th International Symposium on Software Reliability Engineering, pages 200–209, Hong Kong, November 2001.

[17] Y. Wu, D. Pan, M.H. Chen. Techniques for testing component-based software. In Proceedings of the 7th IEEE International Conference on Engineering of Complex Computer Systems, pages 15–23, Skövde, Sweden, June 2001.

[18] M. J. Harrold, M. L. Soa. Selecting data flow integration testing. IEEE Software, 8(2):58–65, March 1991.

Web Categorisation Using Distance-Based Decision Trees

V. Estruch [a] C. Ferri [a] J. Hernández-Orallo [a]
M.J. Ramírez-Quintana [a]

[a] *DSIC, Universidad Politécnica de Valencia, Camino de Vera s/n, Apdo. 22012, 46071 Valencia, Spain. Email: {vestruch, cferri,jorallo,mramirez}@dsic.upv.es.*

Abstract

In Web classification, web pages are assigned to pre-defined categories mainly according to their content (content mining). However, the structure of the web site might provide extra information about their category (structure mining). Traditionally, both approaches have been applied separately, or are dealt with techniques that do not generate a model, such as Bayesian techniques. Unfortunately, in some classification contexts, a comprehensible model becomes crucial. Thus, it would be interesting to apply rule-based techniques (rule learning, decision tree learning) for the web categorisation task. In this paper we outline how our general-purpose learning algorithm, the so called distance based decision tree learning algorithm (DBDT), could be used in web categorisation scenarios. This algorithm differs from traditional ones in the sense that the splitting criterion is defined by means of metric conditions ("is nearer than"). This change allows decision trees to handle structured attributes (lists, graphs, sets, etc.) along with the well-known nominal and numerical attributes. Generally speaking, these structured attributes will be employed to represent the content and the structure of the web-site.

1 Introduction

Etzioni [4] defined Web mining as the use of data mining techniques for extract information from Web documents and services. Given the large amount of documents available in the Web, one of the most common task performed on the Web is the classification of documents into one or more categories. For instance, this is essential in applications that have to catalog news articles, sort and filter electronic mail, recommend films or music or search information

[1] This work has been partially supported by ICT for EU-India Cross Cultural Dissemination Project ALA/95/23/2003/077-054 and Generalitat Valenciana under grant GV04B/477 and CICYT under grant TIN 2004-7943-C04-02.

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

about a topic (search engines). Although some authors distinguish classification from categorisation [2], for the sake of simplicity, in this paper we use both of them as synonyms since a categorisation problem can be solved by several classifiers. The simplest approach to the categorisation of Web documents is to take only the textual part of them into account (Text categorisation). The basic idea is to classify a document as of class c if certain words relevant to the c definition are present in the document.

However, Web documents are more than just plain text and the information contained in other parts like the hyper-links can also be relevant to the categorisation process. For instance, if we are classifying sports news, a more accurate classification can be obtained if our classifier considers that a piece of sports news contains words like team, play or stadium, or contains links to other sports news. Therefore, recent research solves this problem by merging ideas from Web content mining and Web structure mining. For instance, [7] appends the text of the links to the text of the target page. [1] considers the text of a Web page along with the text and the category of its neighbouring pages. Some other approaches are able to handle both the text components in the pages and the links among them, such as [2], [5], or [6].

In this paper, we study how the DBDT approach fits to the web classification problem. This method allows us to integrate both the Web content and the Web structure mining in a unique framework by using structured data types for representing each component or context feature (title, keywords, text, links, ...) found in the pages. This evidence is then used by the DBDT in that the splitting criterion is defined by means of metric conditions ("is nearer than") and handle structured attributes. We illustrate that the method is suitable for this kind of application by applying it to a simple example of Web classification and we briefly discuss about how the metric conditions can be expressed in an equivalent but more comprehensible form.

The paper is organised as follows. In Section 2 the DBDT algorithm is outlined. An illustrative example of our approach is shown in Section 3. Finally, Section 4 presents some conclusions.

2 Distance Based Decision Trees

In [3] we defined a learning method named Distance Based Decision Trees. This proposal is based on the use of *prototypes* and distances to define the partitions for the decision tree. Our decision tree inference strategy is a modification of the centre splitting method [8] consisting in to compute a set of attribute prototypes unlike the other one which takes all the attributes into account. Basically, for each attribute and for each class, a prototype (that value which minimises the sum of all the distances from it to the others) is calculated, considering only the values belonging to that attribute and that

[2] The classification is the process of inducing a model in that only one class is assigned to each document, whereas categorisation concerns with the situation in that a document can belong to more than one class.

class. Once this process is finished, an attribute is chosen in order to split the data set. The split proceeds by associating every instance to its closest attribute prototype. The splitting attribute is selected according to some of the well-known heuristic functions (gain ratio, GINI index, etc). For this purpose, a metric space is associated to every attribute. Note that the fact of handling all the attributes as whole entity, just as centre splitting does, turns the comprehensible model extraction into a harder task, even if the involved attributes are nominal or numerical.

The result of this adaptation of centre splitting is not very different from classical decision trees (see the algorithm below), when attributes are either nominal and numeric, but in our case, we are able to deal with data containing structured attributes such as sets, lists, or trees.

PROCEDURE DBDT(S, m); // Single Attribute Centre Splitting. Learns a decision tree based on attribute distances

INPUT: A training set S as a set of examples of the form: $(x_1, \ldots, x_n), n \geq 1$ where every attribute is nominal, numerical or structured. A metric space is associated to every attribute. m is the maximum # of children per node.

BEGIN

$C \leftarrow \{Class(e) : e \in S\}$ // C is the set of existing classes

If $|C| < 2$ **Then RETURN End If**

For each attribute x_j:

// Computes two (or more) centres for each class using attribute x_j

If $Values(x_j, S) < 2$ **Then CONTINUE End If** //next iteration

$ProtList \leftarrow$ Compute_Prototypes(x_j, S, m, C).

If $Size(ProtList) \leq 1$ **Then RETURN End If**

$Split_j \leftarrow \emptyset$ // Set of possible splits for attribute x_j

For i $\leftarrow$ 1 **to** $length(ProtList)$ // for all the prototypes

$\hat{S}_i \leftarrow \{e \in S : i =$ Attracts$(e, ProtList, x_j)\}$ // $\hat{S}_i$ contains the examples attracted by prototype i

$Split_j = Split_j \cup \hat{S}_i$ // We add a new child to this split

$i \leftarrow i + 1$;

End For

End For

$BestSplit = Argmax_{Split_j}(Optimality(Split_j))$ // GainRatio, MDL, ...

For each set S_k in $BestSplit$

DBDT(S_k, n) // go on with each child

End For

END

The auxiliary functions `Attracts` and `Compute_Prototypes` are inherent to the method. In a nutshell, the function `Attracts` just determines which prototype is assigned with a new example and, the function `Compute_Prototypes` obtains a set of prototypes for each attribute.

3 An illustrative example

The previous step, before running the DBDT algorithm, consists of deciding what sort of data types are going to be used, as well as their associated metric functions. Let us consider the following example. A user is interested in seek-

ing sports news from the Internet using a search engine. This search engine must "decide" automatically which available documents fit the search parameters. Thus, this task can be addressed as a two class classification problem. The information, extracted from an HTML document for this purpose, can be grouped in these three categories:

- **Structure**: it refers how the pages from a web site are connected one each others by means of hyper-links. Formally, it is represented as a graph.However, we will use a simpler approach but it is in its turn a very common proposal in the graph mining literature: we represent a graph as a set of ordered pairs where each pair encodes two linked pages. Concretely, each item in the ordered pair will store a set of key words. Also, for the sake of brevity, we use the well-known symmetric difference between sets as a metric function.
- **Content**: It deals with the information contained in a web page.Traditionally, this information is represented as a bag or a vector of words. In our example, we only consider one attribute, a set, reflecting the whole content (*Content*), and we use an adaptation of the symmetric difference between sets as a metric function.
- **Web use**: we mean by web use information the information derived from the HTTP connection to a web server All these data can be encoded by means of nominal or numerical attributes. For these types we can use the discrete metric or the absolute value difference, respectively. In our example, this attribute is referred by *Connections* and it contains the number of daily connections.

The next step is to infer a classifier by training a model from a processed dataset that contains collected information from some web pages, such as that included in Table 1.

Id.	Structure	Content	Conn.	Class
1	{([Olympics,games],[swim]),([swim],[win]), ([Olympics,games],[boxing]) , ([win],[medal])}	{(Olympics,30),(held,10) (summer,40)}	10	No
2	{([Olympics,games],[swim]),([swim],[win]), ([win],[medal])}	{(Olympics,15),(summer,20) (Athens,40)}	20	Yes
3	{([football],[Europe]),([Europe],[final]), ([final],[best,player])}	{(football,20),(champion,10)}	40	No
4	{([football],[match]),([match],[team,players]), ([football],[referees]),([match],[results])}	{(football,20),(Europe,10), (champion,12)}	40	Yes
5	{([football],[match]),([match],[team,players]), ([match],[scores])}	{(football,20),(Europe,10)}	40	Yes

Table 1
Information from a web server sample repository.

The set {([Olympics,games],[swim]),([swim],[win]),([win],[medal])} in the *Structure* attribute is interpreted in the following way. The first component of the list stands for words "Olympics" and "games" appear as keywords in a web page. This page links another one which has "swim" as its only key word. The reasoning is the same for the second and third components of the set.

If we apply the DBDT algorithm (using an accuracy-based heuristic), we find that the first attribute to be selected, as the first split, is *Connection*, being the values 40 (*Conn* value for the 4*th* instance) and 10 (*Conn* value for the 1*st* instance) the prototypes for the class "yes" and "no" respectively. Iterating the process, attributes *Structure* and *Content* are used to split the left and the right first level nodes, respectively. Finally, the new obtained nodes are pure and the process stops, getting the distance based decision tree (see figure below [3]).

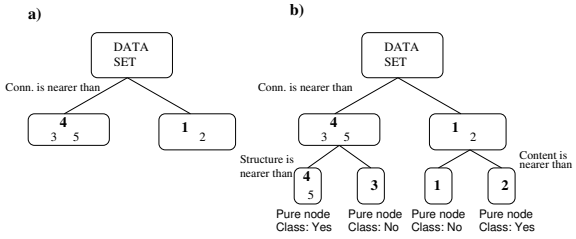


Fig. 1. a) Decision tree after the first split. b) Decision tree after finishing the process.

Id.	Structure	Content	Conn.	Class
	{([football],[match]),([match],[players]), ([match],[results])}	{(football,30),(held,10) (Europe,7)}	36	No

Table 2
Information from a web server sample repository.

Imagine now that a web site described as in Table 2 is stored in the list along with other web sites which are candidates to be shown to the customer. Before listing them directly we should classify the we site repository in order to filter not suitable information. First, we look inside the connection attribute. As the number of daily connections is closer to 40 than 10, the instance is hooked up to the left first-level node. Then, we repeat the same process for the structure attribute, in this case, the structure of this new web site is more similar to the structure of the fourth instance in the table than the third one. Then, this instance would be classified as sport news site, and, consequently, listed to the user.

Currently, we are thincking over how the metric conditions could be expressed into terms of patterns associated to the metric function (for instance, *belong to* could be a pattern for sets) [9], and obtain a transformed (and more comprehensible) model containing rules as this one: IF the word "football" appears in *Content* and the connections {([football],[match]), ([match],[team,players])} are found in *Structure* THEN this web-site is a sport media web-site.

[3] The numbers correspond to instance id, and the bold numbers stand for the prototype of each class for a particular partition.

4 Conclusions

In this paper, we have studied the feseability of DBDT proposal to tackle web categorisation problems. DBDT has been developed in Java (*www.dsic.upv.es/users/elp/soft/*) and tested for both, structured and non structured, well-known classification problems, showing a really interesting performance. For this reason, we consider that this algorithm could be applied for more concrete scenarios, such as categorisation web.

References

[1] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *SIGMOD Conference*, pages 307–318, 1998.

[2] M. Craven and S. Slattery. Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning*, 43(1/2):97–119, 2001.

[3] V. Estruch, C. Ferri, J. Hernández, and M. J. Ramírez. Distance-based decision tree learning for structured data types. Technical report, DSIC,UPV, 2004.

[4] O. Etzioni. The world-wide web: Quagmire or gold mine? *Communications of the ACM*, 39(11):65–68, 1996.

[5] Wen-Chen Hu. Webclass: Web document classification using modified decision trees.

[6] S. Slattery and T. M. Mitchell. Discovering test set regularities in relational domains. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, pages 895–902, 2000.

[7] A. Sun, E.-P. Lim, and W.-K. Ng. Web classification using support vector machine. In *Proceedings of the 4th Int. Workshop on Web Information and Data Management*, pages 96–99, November 2002.

[8] Chris Thornton. *Truth from Trash: How Learning Makes Sense.* The MIT Press, Cambridge, Massachusetts, 2000.

[9] J. Hernández V. Estruch, C.Ferri and M.J. Ramírez. Identifying generalisation patterns for distance-based methods. In *19th Internatinal Joint Conference on AI,IJCAI05, submitted.*

Web Accessibility Evaluation Tools: a survey and some improvements

Vicente Luque Centeno Carlos Delgado Kloos
Jesús Arias Fisteus Luis Álvarez Álvarez

Department of Telematic Engineering
Carlos III University of Madrid
vlc@it.uc3m.es

Abstract

Web Content Accessibility Guidelines (WCAG) from W3C consist of a set of 65 checkpoints or specifications that Web pages should accomplish in order to be accessible to people with disabilities or using alternative browsers. Many of these 65 checkpoints can only be checked by a human operator, thus implying a very high cost for full evaluation. However, some checkpoints can be automatically evaluated, thus spotting accessibility barriers in a very effective manner. Well known tools like Bobby, Tawdis or WebXACT evaluate Web accessibility by using a mixture of automated, manual and semiautomated evaluations. However, the automation degree of these Web evaluations is not the same for each of these tools. Since WCAG are not written in a formalized manner, these evaluators may have different "interpretations" of what these rules mean. As a result, different evaluation results might be obtained for a single page depending on which evaluation tool is being used.

Taking into account that a fully automated Web accessibility evaluation is not feasible because of the nature of WCAG, this paper evaluates the WCAG coverage degree of some well known Web accessibility evaluation tools spotting their weaknesses and providing a formalized specification for those checkpoints which have had fewer automation coverage in nowadays' tools.

Key words: Automated evaluation, WAI, WCAG, Web Accessibility, Web site evaluators

1. Introduction

Web Content Accessibility Guidelines (WCAG) 1.0 [1] were defined by W3C as a set of 14 guidelines and 65 checkpoints that Web pages should accomplish in order to be accessible to people with disabilities, people using alternative browsers, or even Web agents, as shown at [15]. However, they were defined

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

using high level terms not being focused on the underlying HTML format. As a result many of those checkpoints require human judgement and they don't provide an objective criteria to be followed. This implies that it is not possible to build a fully automated accessibility evaluator based only on cheap-to-be-evaluated automatically conditions.

In order to fully determine Web accessibility, some semiautomated and manual checkpoints should also be evaluated. Thus, we can classify checkpoints into the four following groups.

(i) **Objectively automated** rules are clearly defined and clearly specify a condition that nobody might reject. Checking whether a set of well defined mandatory elements and attributes are present within the HTML markup is a typical checkpoint within this category.

(ii) **Subjectively automated** rules specify fuzzy conditions that can be automated, but whose particular *non-fuzzy* interpretation might be accepted or rejected by different groups of people. Checking whether a text is *too long* is a subjective condition since it mentions a condition which is fuzzy to evaluate. Even though some people might agree that an alternative text longer than 150 characters might be rejected as too long, some other people might not think so. For these kind of subjective automatable conditions, W3C has defined a set of heuristics [2] that provide some help.

(iii) **Semi-automated** rules, which can not be evaluated automatically, but tool's assistance can focus user's interest on relevant markup.

(iv) **Manual** rules, which require human judgement. Both semi-automated and manual rules are very expensive to evaluate and should be kept to a minimum.

2. Comparison of Web Accessibility Evaluation Tools

One main problem of Web Accessibility evaluation tools is that they provide a poor coverage of WCAG. Their behaviour consists on addressing only a few accessibility barriers on Web pages, leaving other barriers as unprocessable. Thus they can not guarantee that accessibility is achieved, but they can guarantee that accessibility is faulty whenever a barrier is found. This weakness of nowadays' Web accessibility evaluation tools is mostly based on the nature of how WCAG were specified.

A major problem, however, is that we can easily find different evaluation results for a page depending on which tool is being used. The problem is that, while a checkpoint can be automated in one tool, it probably may be semi-automated or ignored in other tools. Even though considered in both tools, since each tool has its own *interpretation* a condition triggered in a tool may easily not be triggered in other tools.

Several Web Accessibility tools have been evaluated in this article in order

to determine their automation coverage of WCAG 1.0, and some formalized rules are given as a specification (and implementation) for these tools.

2.1. Tools being evaluated

Web accessibility evaluation tools selected for this comparison are Bobby [12], Tawdis (Taw) [13] and WebXACT [14] (from the same company of Bobby). We have chosen these because they are well-known and can be used online for free. We know of no other similar evaluation tools (except Torquemada [18], which, sadly, is no longer available) that provide such final results without human intervention.

2.2. Checkpoints Vs Checkpoints' conditions

WCAG are very heterogeneous. Many of them imply a condition that only human judgement may evaluate. Some of them imply a set of several lower level conditions. In order to properly evaluate, we defined the WCAG checkpoints' conditions, a set of 103 conditions that perform some more detailed evaluation that the original 65 WCAG conditions. A WCAG condition may imply 1 or several WCAG checkpoint's conditions. For example, checkpoint WCAG 4.1 (Clearly identify changes in the natural language) might imply checkpoint's conditions WCAG 4.1a (Use `xml:lang` attribute for pieces of text in a different language within the same document) and WCAG 4.1b (Use `hreflang` attribute on links pointing to documents in a different language).

2.3. Automated coverage of checkpoints evaluation

Table 1 indicates how the 65 WCAG checkpoints can be classified into the categories previously mentioned according to each tool. Both semi-automated and manual rules imply human evaluation cost, so our interest is focused on automated rules only. From a total of 14 possible objectively automated checkpoints, WebXACT detects 5 of them, Bobby only detects 4, and only a single one is detected by Taw. This means that, at least 14 - 5 = 9 objectively automatable checkpoints, some of them provided in the following sections, are not implemented by any analized tool. Subjectively automated rules are used in these tools as semi-automated, because the user has no ability to specify preferences for those subjective conditions, thus requiring human intervention.

Table 2 indicates the same as table 1 , but looking at our 103 WCAG checkpoints' conditions. The number of objectively automated is now significantly increased for WebXACT and Bobby, having 25 and 24 objectively automated conditions respectively, but this number is still far from the 43 possible conditions that can be evaluated in this category. This implies that at least 43 - 25 = 18 objectively automatable conditions are not fully evaluated by any of our tools. The rest is only usable for partial evaluation. Taw has a poor result of 10 rules from a set of 43.

	Theoretical	Bobby	Taw	WebXACT
Objectively automated	14	4	1	5
Subjectively automated	2	0	0	0
Semi-automated	31	33	20	33
Purely manual	18	28	44	27
Total	**65**	**65**	**65**	**65**

Table 1
Comparison of automation of WCAG checkpoints

	Theoretical	Bobby	Taw	WebXACT
Objectively automated	43	24	10	25
Subjectively automated	11	0	0	0
Semi-automated	21	30	18	30
Purely manual	28	49	75	48
Total	**103**	**103**	**103**	**103**

Table 2
Comparison of automation of WCAG checkpoints' conditions

3. Checkpoints having similar behaviours

From tables 1 and 2 , we determine that, even though our tools might have similar behaviours for some checkpoints, some other checkpoints clearly have different behaviours. Prior to see differences, table 3 shows some checkpoint conditions having the same automated behaviour on all evaluated tools. As expected, the most well known rule 1.1a (provide `alt` attribute for images), is within this set. The second column of table 2 provides a formalized XPath 1.0 [3] and XPath 2.0 [4] expression that can be used to address all the HTML elements breaking the rule within that page. For example, rule for 1.1a addresses all images that have no `alt` attribute, rule for 1.1b addresses all image buttons without an alternative text, rule for 1.1e addresses all framesets not providing a `noframes` section, ...

Rule 3.2: Validate document against a public grammar

According to [16], Web accessibility evaluators are not XHTML [7] validators. None of our selected tools provide support for validation against a well known DTD or XML Schema, because there are specific tools (like [11]) for this task and because accessibility is more than only XHTML validation (and because

WCAG #	Formalized rule	Bobby,WebXACT,Taw
1.1a	//img/[not(@alt)]	Auto Obj.
1.1b	//input[@type="image"][not(@alt)]	Auto Obj.
1.1e	//frameset[not(noframes)]	Auto Obj.
1.5	//area[not(@alt)]	Auto Obj.
7.2a	//blink	Auto Obj.
7.3a	//marquee	Auto Obj.
13.2b	//head[not(title)]	Auto Obj.

Table 3
Some rules with same behaviour in automated evaluators

their authors wanted to make accessibility still applyable for non validated documents). However, we must say that rule 3.2 from WCAG probably guarantees more accessibility than any other rule by itself, because several WCAG rules are partially based on statements that are automatically achieved whenever a document is validated against a DTD or XML Schema. In fact, we have found that, depending on the *flavour* of XHTML being used, accessibility might be easier (or more difficult) to be achieved. We have found that XHTML 2.0 [10] provides more accessibility features than XHTML Basic 1.0 [8], which itself provides more accessibility features than XHTML 1.1 [9].

Examples of WCAG checkpoints achieved when a proper DTD or XML Schema is chosen and validated against, are:

- Rule 3.3: Use style sheets to control layout and presentation
- Rule 3.6: Mark up lists and list items properly
- Rule 11.2: Avoid deprecated features of W3C technologies

4. Checkpoints having different results

Tables 4 and 5 provide a summary of the most important differences on WCAG's coverage in our evaluated tools. Each row corresponds to an (objectively or subjectively) fully automatable rule. However, WebXACT and Bobby only provide a fully automated evaluation on rules 4.3, 9.2a-c, 10.4a-d, 10.5, 12.4b and 13.1a, leaving the others as semi-automated or manual, despite they could be fully automated, as depicted in the second column. Rules marked up as "Pseudo" refer to rules that detected a problem only under particular conditions, but could not evaluate with the same accuracy as the formalized rule.

Table 4 includes W3C-standards-based rules for specifying when to use the `longdesc` attribute (rule 1.1c), when heading elements are used properly

WCAG #	Formalized rule	Bobby,WebXACT	Taw
1.1c	//img[*toolong_alt*(@alt)][not(@longdesc)]	Semi	Manual
3.5a	//h2[not(preceding::h1)]	Pseudo	Manual
3.5b	See fig 1	Pseudo	Manual
3.5c	See fig 2	Pseudo	Manual
3.5d	See fig 3	Pseudo	Manual
3.5e	See fig 4	Pseudo	Manual
4.3	//html[not(@xml:lang)]	Auto Obj.	Manual
6.4a	//*[@onmouseover != @onfocus]	Semi	Manual
6.4b	//*[@onmouseout != @onblur]	Semi	Manual
7.4, 7.5	//meta[@http-equiv="refresh"]	Semi	Manual
9.2a	//*[@onmousedown != @onkeydown]	Auto Obj.	Manual
9.2b	//*[@onmouseup != @onkeyup]	Auto Obj.	Manual
9.2c	//*[@onclick != @onkeypress]	Auto Obj.	Manual
10.4a	See fig 9	Auto Obj.	Manual
10.4b	//textarea[normalize-space(text())=""]	Auto Obj.	Manual
10.4c	See fig 10	Auto Obj.	Manual
10.4d	See fig 11	Auto Obj.	Manual
10.5	See fig 12	Auto Obj.	Manual
12.4b	See fig 16	Auto Obj.	Manual
13.1a	See fig 17	Auto Obj.	Manual

Table 4
Rules with different behaviour in automated evaluators: WebXACT and Bobby Vs Taw

(rules 3.5), whether document's idiom is specified (rule 4.3), when keyboard and mouse dependant handlers are not paired (rules 9.2), and the detection of form fields having no label (rule 12.4b) or ambiguous links (rule 13.1a), among others.

Table 5 indicates that frames should have a description (`title` attribute). However, neither Bobby nor Taw require so. This is the only improvement we have found on WebXACT if compared to Bobby.

On the other hand, table 6 shows a sadly common poor behaviour of all analized tools with respect to fully automatable rules, that, however, are

WCAG #	Formalized rule	WebXACT	Bobby,Taw
12.1	//frame[not(@title)]	Auto Obj.	Manual

Table 5
Rules with different behaviour in automated evaluators: WebXACT Vs Bobby and Taw

improperly treated as manual or semi-automated.

WCAG #	Formalized rule	Theory	Tools
3.5f	See fig 5	Auto Subj.	Manual
4.2a	See fig 6	Auto Subj.	Manual
6.3b	//a[starts-with(@href,"javascript:")]	Auto Obj.	Manual
6.3b	//area[starts-with(@href,"javascript:")]	Auto Obj.	Manual
9.4	See fig 7	Auto Obj.	Manual
9.5	See fig 8	Auto Obj.	Manual
10.1a	//*[@target="_blank"]	Auto Obj.	Semi
10.1a	//*[@target="_new"]	Auto Obj.	Semi
12.2	//frame[*toolong*(@title)][not(@longdesc)]	Auto Subj.	Semi
12.3a	See fig 13	Auto Subj.	Semi
12.3b	See fig 14	Auto Subj.	Semi
12.3c	//p[*toolong_p*(text())]	Auto Subj.	Manual
12.4a	See fig 15	Auto Obj.	Manual

Table 6
Rules with common poor behaviour in automated evaluators

Table 6 includes rules for the detection of Javascript being found on improper attributes (rule 6.3b), improper `tabindex` (rule 9.4), and improper `accesskey` shortcuts (rule 9.5). Table 6 also depicts how to restrict some popups and other windows appearances (rule 10.1) labels pointing to more than one form field (rule 12.4a), and many others that are detailed in the following figures (referred from tables 4 , 5 and 6). It is quite surprising that none of these very simple checkings were detected by any analized tool, though they easily could. Only manual or semi-automated support was provided for these fully automatable rules.

Rule 3.5: Use headings properly

Figure 1 specifies an XQuery 1.0 [5] addressing expression for all non properly used `h3` elements. It is necessary that a `h1` and `h2` properly precede any `h3` element. So, this rule will be broken for those `h3` having no prior `h1` or `h2` or having the nearest `h2` prior the nearest `h1`.

```
//h3[let $h3:=self::h3 return
let $h2:=$h3/preceding::h2[last()] return
let $h1:=$h3/preceding::h1[last()] return
$h1=() or $h2=() or $h1>>$h2]
```

Fig. 1. XQuery 1.0 expression for h3 elements breaking WCAG **3.5b**

Figures 2 , 3 and 4 provide addressing expressions for improper `h4`, `h5` and `h6` elements respectively, in a similar way as rule from figure 1 .

```
//h4[let $h4:=self::h4 return
let $h3:=$h4/preceding::h3[last()] return
let $h2:=$h4/preceding::h2[last()] return
let $h1:=$h4/preceding::h1[last()] return
$h1=() or $h2=() or $h3=() or
$h1>>$h2 or $h2>>$h3 or $h3>>$h4]
```

Fig. 2. XQuery 1.0 expression for h4 elements breaking WCAG **3.5c**

```
//h5[let $h5:=self::h5 return
let $h4:=$h5/preceding::h4[last()] return
let $h3:=$h5/preceding::h3[last()] return
let $h2:=$h5/preceding::h2[last()] return
let $h1:=$h5/preceding::h1[last()] return
$h1=() or $h2=() or $h3=() or
$h4=() or $h1>>$h2 or $h2>>$h3 or
$h3>>$h4 or $h4>>$h5]
```

Fig. 3. XQuery 1.0 expression for h6 elements breaking WCAG **3.5d**

Header elements should not be too long, as defined in figure 5 . If a header appears to be too long, it should be considered as a paragraph or its size should be reduced.

Rule 4.2: Use proper abbreviations and acronyms

Abbreviations and acronyms should not be used inconsistently. They should have unique definitions. Figure 6 detects all abbreviations and acronyms that provide more than a single definition for a given text. Rule 4.2a, however, is not enough to guarantee rule 4.2, since it detects improper marked up acronyms and can not detect whether all possible abbreviations and acronyms are currently marked up.

```
//h6[let $h6:=self::h6 return
let $h5:=$h6/preceding::h5[last()] return
let $h4:=$h6/preceding::h4[last()] return
let $h3:=$h6/preceding::h3[last()] return
let $h2:=$h6/preceding::h2[last()] return
let $h1:=$h6/preceding::h1[last()] return
$h1=() or $h2=() or $h3=() or
$h4=() or $h5=() or $h1>>$h2 or
$h2>>$h3 or $h3>>$h4 or
$h4>>$h5 or $h5>>$h6]
```

Fig. 4. XQuery 1.0 expression for h6 elements breaking WCAG **3.5e**

```
(//h1|//h2|//h3|//h4|//h5|//h6)[toolong_h(text())]
```

Fig. 5. XQuery 1.0 expression for header elements breaking WCAG **3.5f**

```
(//abbr | //acronym)[let $a:=self::node() return
count((//abbr | //acronym)[text() = $a/text()]) != 1]
```

Fig. 6. XQuery expression for abbr and acronym elements breaking WCAG **4.2a**

Rule 9.4: Specify a proper tab order

Whenever a tab order being different from the default is being specified, `tabindex` attributes should be used consistently. Figure 7 spots all elements that have a `tabindex` attribute which is not a proper number or which is shared by several elements (it should be used uniquely).

```
//*[@tabindex][let $n:=self::node()/@tabindex return
not(isnumber($n)) or count(//*[@tabindex=$n]) != 1 or
number($n)<1 or number($n)>count(//*[@tabindex])]
```

Fig. 7. XQuery expression for elements breaking WCAG **9.4**

Rule 9.5: Provide proper keyboard shortcuts

Whenever keyboard shortcuts are being specified, `accesskey` attributes should be used consistently. Figure 8 spots all elements that have an `accesskey` attribute which is not a proper character or which is shared by several elements (it should be used uniquely).

```
//*[@accesskey][let $c:=self::node()/@accesskey return
not(ischar($c)) or count(//*[@accesskey=$c]) != 1]
```

Fig. 8. XQuery expression for elements breaking WCAG **9.5**

Rule 10.4: Include default, place-holding characters in form fields

Editable (not hidden) form fields different from text areas (i.e, text fields, radio or checkbox buttons) that have null or no default values are detected by the XQuery addressing expression from figure 9 . Empty text areas are detected by the expression of 10.4b from table 6 .

```
//input[@type!="hidden"][not(@value) or @value=""]
```

Fig. 9. XQuery expression for input elements breaking WCAG **10.4a**

Select boxes having no selected option by default are detected by expression from figure 10 . Figure 11 addresses sets of radio buttons where no option has been checked by default. Though not explicitly declared on WCAG, these last two rules might also be considered as an extension of rule 10.4.

```
//select[not(@multiple)][not(.//option[@selected])]
```

Fig. 10. XQuery expression for select elements breaking WCAG **10.4c**

```
//input[@type="radio"][let $n:=self::input/@name return
not(ancestor::form//input[@name=$n][@checked])]
```

Fig. 11. XQuery expression for radio buttons breaking WCAG **10.4d**

Rule 10.5: Include non-link, printable characters between adjacent links

Figure 12 addresses all consecutive links that have nothing more that white spaces between them. If so, several consecutive links might be interpreted as a single big link. For this rule we use a combination of both XQuery and XPointer [6].

```
//a[position() < last()][let $a:=self::a return
normalize-space((end-point($a)/range-to($a/following::a))/text())=""]
```

Fig. 12. XPointer + XQuery based rule for WCAG **10.5**

Rule 12.3: Divide large blocks of information into more manageable groups

The `fieldset` elements is highly recommended for forms that have several form fields. It can be used to group several of them which are semantically related within a same group, thus providing more manageable groups of form fields. Though it could be a subjectively automated rule (*toomany_inputs* should be defined with some subjective criteria), both Bobby, WebXACT and Taw treat this rule as semi-automated, asking a human evaluator whenever a form addressable by expression from figure 13 is found. The same applies for rule 12.3b from figure 14 (for non grouped options within a select).

```
//form[toomany_inputs(.//input)][not(.//fieldset)]
```

Fig. 13. XQuery expression for forms breaking WCAG **12.3a**

```
//select[toomany_options(option)][not(optgroup)]
```

Fig. 14. XQuery expression for select elements breaking WCAG **12.3b**

Rule 12.4: Associate labels explicitly with their controls

This rule implies two different conditions. Figure 15 addresses all labels which are not properly used, i.e., that point to more than a single control (rule 12.4a). Vice-versa, all form fields should have no more than a single label (rule 12.4b). Otherwise, they can be detected by expression from figure 16 . If both rules 12.4a and 12.4b are followed, there exists a bijective mapping between labels and form fields, as desirable, so no confusion can be obtained when mapping labels to controls or asking for control's labels.

```
//label[let $l:=self::label return
count((//select|//input|//textarea)[@id=$l/@for]) != 1]
```

Fig. 15. XQuery expression for labels breaking WCAG **12.4a**

```
(//select|//textarea|//input[@type="text" or @type="password" or
@type="radio" or @type="checkbox"])[let $ff:=self::node() return
count(//label[@for=$ff/@id]) != 1]
```

Fig. 16. XQuery expression for form fields breaking WCAG **12.4b**

Rule 13.1: Clearly identify the target of each link

It is not possible to automatically determine whether the text of a link can be understood by itself when read out of context (which is treated in manual rule 13.1b). However, we still can determine if two links are used in a confusing manner. Figure 17 contains a XQuery expression for finding all links that, having same text and descriptions, point to different targets.

```
(//a | //area)[let $a:=self::node() return
(//a | //area)[@title = $a/@title and
text() = $a/text() and @href != $a/@href] != ()]
```

Fig. 17. XQuery expression for links breaking WCAG **13.1a**

5. Conclusions and future work

As a result of this evaluation, we can determine that it is easier to detect accessibility barriers using Bobby or WebXACT than using Taw. However, none of these tools provide as much coverage as we could desire. In fact, they are far away from having a good automated coverage of WCAG. Thus, we are

now working on a Web evaluator tool that will have a better coverage and, since it will be implemented with the XQuery and XPath expressions showed in this article (among some other similar expressions not appearing in this article), we hope it will not be very expensive to build.

Normalization is also an important focus, so that we don't have multiple rulesets for accessibility depending on which country we are. We hope that a formalization attempt like ours may help in that direction. Comparison of WCAG 1.0 and Bobby's implementation of U.S. Section 508 Guidelines [17] has lead us to conclude that both rules have a lot of common rules and minor changes should be added to WCAG 1.0 to include this normative.

There are few tools freely available for accessibility evaluation. Torquemada is no longer operative and WebXACT is offered by Bobby's developers, something which explains why WebXACT only offers a minor enhancement from Bobby, the most well known tool in this area. Anyway, different misinterpretations of W3C's WCAG will still occur on future implementations as long as there is no common formalized interpretation of these rules. Our proposal is to give some light in this process. That's why we have redefined the set of 65 WCAG into a set of 103 rules which are more focused on low level details than those of WCAG.

6. Acknowledgements

The work reported in this paper has been partially funded by the projects INFOFLEX *TIC2003-07208* and SIEMPRE *TIC2002-03635* of the Spanish Ministry of Science and Research.

References

[1] W3C *Web Content Accessibility Guidelines 1.0*
`www.w3.org/TR/WCAG10`

[2] W3C *Techniques For Accessibility Evaluation And Repair Tools W3C Working Draft, 26 April 2000*
`www.w3.org/TR/AERT`

[3] W3C *XML Path Language (XPath) Version 1.0 W3C Recommendation 16 November 1999*
`www.w3.org/TR/xpath`

[4] W3C *XML Path Language (XPath) 2.0 W3C Working Draft 29 October 2004*
`www.w3.org/TR/xpath20`

[5] W3C *XQuery 1.0: An XML Query Language W3C Working Draft 29 October 2004*
`www.w3.org/TR/xquery`

[6] W3C *XML Pointer Language (XPointer), W3C Working Draft 16 August 2002*
www.w3.org/TR/xptr

[7] W3C *XHTML 1.0 ^{TM} The Extensible HyperText Markup Language (Second Edition), A Reformulation of HTML 4 in XML 1.0, W3C Recommendation 26 January 2000, revised 1 August 2002*
www.w3.org/TR/xhtml1

[8] W3C *XHTML Basic W3C Recommendation 19 December 2000*
www.w3.org/TR/xhtml-basic

[9] W3C *XHTML ^{TM} 1.1 - Module-based XHTML, W3C Recommendation 31 May 2001*
www.w3.org/TR/xhtml11

[10] W3C *XHTML ^{TM} 2.0, W3C Working Draft 22 July 2004*
www.w3.org/TR/xhtml2

[11] W3C *Markup Validation Service*
validator.w3.org

[12] Watchfire *Bobby Accessibility tool*
bobby.watchfire.com/bobby/html/en/index.jsp

[13] CEAPAT, Fundación CTIC, Spanish Ministry of Employment and Social Affairs (IMSERSO) *Online Web accessibility test*
www.tawdis.net

[14] Watchfire *WebXACT*
webxact.watchfire.com

[15] Vicente Luque Centeno, Carlos Delgado Kloos, Luis Sánchez Fernández, Norberto Fernández García *Device independence for Web Wrapper Agents* Workshop on Device Independent Web Engineering (DIWE'04) 26 July 2004, Munich

[16] Peter Blair *A Review of Free, Online Accessibility Tools*
www.webaim.org/techniques/articles/freetools, February 2004

[17] Center for IT Accommodation (CITA) *U.S. Section 508 Guidelines*
www.section508.gov

[18] Fondazione Ugo Bordoni *Torquemada, Web for all*
www.webxtutti.it/testa_en.htm

Automated Web Site Accessibility Evaluation

Shadi Abou-Zahra [1]

Web Accessibility Initiative
World Wide Web Consortium
Sophia-Antipolis, France

Abstract

The Evaluation and Report Language (EARL) is an RDF Schema which allows evaluation tools to express test results in a machine-readable, platform-independent, and vendor-neutral format. Describing test results (such as results from automated Web accessibility checks) using technologies provided by the Semantic Web framework facilitates the aggregation of test results with readily available parsers, libraries, and resources. Furthermore, Semantic Web technologies allow authoring tools (such as editors, content management system, or save-as utilities), Web browsers, search engines, or other types of applications to elegantly process these test results and provide more advanced features to the end users.

Key words: Accessibility, Automated Evaluation, Conformance Testing, Quality Assurance, Validation

1 Introduction

The Web has rapidly evolved to become a key resource for news, information, commerce, and entertainment. It is continuing to displace traditional sources of information and other aspects of society such as leisure, education, at the workspace, civic participation, and government services. However, for some people with visual, hearing, physical, cognitive, or neurological disabilities, there are severe barriers on the Web denying them access to fundamental information and services.

The W3C/WAI Web Content Accessibility Guidelines (WCAG) addresses accessibility requirements of people with disabilities. It is accompanied by a set of *Techniques* documents which explain to Web developers how to implement accessibility features in different Web technologies such as HTML, CSS, SVG, or SMIL. In order to determine conformance of Web sites with WCAG,

[1] Email: shadi@w3.org

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

numerous accessibility checks need to be made on each site. For large or sophisticated Web sites, such conformance testing can be very time cosuming and costly. This is one of the main motivations for developing EARL; to leverage the ability and quality of Web accessibility evaluation tools which significantly reduce the time and effort required to evalute Web sites for accessibility.

This paper will explain how Semantic Web technologies (such as RDF and OWL) can be used by evaluation tools to describe test results and allow a whole range of applications to process this data. While the primary aim of EARL is to enable the exchange of test results between Web accessibility evaluation and repair tools, it is has been designed to be generic and serve several other purposes.

2 Anatomy of the Evaluation and Report Language

EARL is a collection of objects (called *Classes*) and attributes (called *Properties*), as well as the relationships between them, all expressed using the Resource Description Framework (RDF). RDF does not actually define any domain specific vocabulary but rather a basic set of expressions to allow the definition of vocabularies. It is similar to XML which does not define application specific vocabulary but rather a formal grammar for the syntax of these; only that RDF also deals with the representation of semantic knowledge bound to the vocabulary.

2.1 Core Classes

There are four core classes in EARL which define the basic data model of the language, these are expressed below:

- Assertor
  The context information about the test contains information such as who or what carried out the test, when the test was carried out, or information about the platform and configuration in which the test was executed.
- Subject
  The subject which is being evaluated is usually a Web page but could also be anything else such as a piece of software code, a document, or an object which is not available on the Web (e.g. a fax machine).
- Testcase
  The test criteria against which a subject is being evaluated could be any (referenceable) specification, a set of guidelines, a single test from a test suite, or some other test case.
- Result
  The outcomes of a test conducted on a subject can contain information about the success or fail of the test, the confidence in the obtained results, the reason for failing, or other attributes.

2.2 Extensions

RDF allows Classes and Properties to be subclassed into further classes, while still retaining backward compatibility. This happens through simple inference of the class type through inheritance. For example, a fax machine manufacturer may want to subclass the *Subject* to a new class called *SAZ-1* in order to identify a specific model of fax machines which is currently being tested. Outside this company, other systems can still deduce that *SAZ-1* is a subject (because its parent in the hierarchy is a subject) without really knowing much more about what *SAZ-1* actually is. So, while EARL defines a minimum set of entities required to describe test results, it does not confine developers from extending these core classes with domain specific terminology. In fact, EARL ensures the compatibility of the test results despite such extensions.

3 Current State of Web Accessibility Evaluation Tools

Currently, there is a very broad spectrum of Web accessibility evaluation tools available but only very little consistency in their features and performance. Even though there has been substantial development in the quality of these tools, much more work needs to be done:

- higher degree of precise and reliable automated testing capabilities need to be achieved in order to reduce time and effort required for large scale or comprehensive Web site evaluations;
- integration of Web accessibility evaluation tools into existing development environments (such as editors or content management systems needs) to be better facilitated;
- more mature user interfaces which can adapt to the requirements of the visual designers, content authors, programmers, or project managers need to become standard features.

4 Use cases for the Evaluation and Report Language

In the context of evaluation of Web sites for accessibility, the following use cases illustrate some of the ways in which the machine readable Evaluation and Report Language (EARL) can be utilized.

4.1 Combine Reports

Web accessibility evaluation tools vary greatly in their capability to test WCAG Checkpoints. For example, while some evaluation tools have more advanced color contrast analysis algorithms, others perform better in text analysis. EARL provides a standardized data format which allows test results from automated or semi-automated evaluation tools to be collected into a single repository. This enables reviewers to make use of different evaluation tools

during a review process and maximize the number of covered Checkpoints.

4.2 Verify Test Results

EARL allows the test results of different Web accessibility evaluation tools to be compared against each other. For different WCAG Checkpoints, reviewers can prefer to trust the results from certain evaluation tools more than others. Test results claimed by evaluation tools can then be weighted according to these preferences and then verified by comparison against other tools. This minimizes the rate of false positives (not identifying existing errors) and false negatives (incorrectly reporting non-existing errors) in evaluation reports.

4.3 Prioritize Results

Data analysis tools can process EARL reports and prioritize test results according to different policies. For instance, it may sometimes be desirable to sort the test results according to their corresponding severity (for example by matching them to the Priority of the WCAG Checkpoint). In other cases, the relative cost of repair for an accessibility barrier may be the key by which an entity may choose to sort the test results. Data analysis tools can output their reports in EARL format to allow the chaining of EARL enabled tools.

4.4 Provide Data Views

Test results can contain comprehensive information for different end-users. For example line numbers and detailed error messages for Web developers, or less verbose technical detail for project managers and executives. Repair suggestions and educational resources may sometimes be helpful to educate developers new to Web accessibility, but may also be tedious for more experienced ones. The well defined structure of EARL allows customized data views to be made from the same set of test results in order to suite the preferences of the end-users.

4.5 Integrate Authoring Tools

EARL also provides a standardized interface between Web accessibility evaluation tools and authoring tools. Instead of producing reports of the test results directly for end-users, authoring tools could process these machine readable reports and assist Web developers in finding and fixing errors. Evaluation tools could then more easily become vendor neutral plug-ins which can be integrated into any EARL aware editors or content management systems and be utilized like spell- or grammar checking tools.

4.6 Annotate Web Content

While the ultimate aim is to achieve a universal Web which is accessible to everyone, EARL reports can also be used to annotate the current state of Web sites. Search engines or User Agents (such as browsers or media players) can make use of these reports according to user preferences. For example, some users may wish not to receive any links to Web pages with flashing content in their search results or they may prefer the browsers to suppress from displaying such content. Evaluation tools could either serve as third-party annotation services or plug-ins for EARL enabled search engines or User Agents.

5 Summary

On the one hand, the Evaluation and Report Language (EARL) is a mere syntax to structure test results in a standardized way. This enables Web accessibility evaluation tools to exchange data in a vendor-neutral and platform-independent environment amongst themselves and among authoring tools, browsers, or search engines. A common syntax also allows tool results to be collected, combined, and compared in order to achieve the best possible performance.

On the other hand, EARL makes use of the well-founded Semantic Web technologies to add meaning to the formal syntax. The vocabulary defined by EARL to describe test results is part of a rich framework which allows inferences, queries, and mappings to be made to the reports. Ontologies and rules can analyze and prioritize test results as well as infer additional evaluation results indirectly by running queries on the reports.

Which ever aspect we consider, EARL enhances the automated Web site accessibility evaluation by enabling tools to exchange data in an open form. It syndicates results between different types of tools to facilitate their integration into a powerful orchestration. However, EARL is intentionally designed to address the more generic requirement of Quality Assurance in order to be applicable and reusable in many other fields as well.

References

[1] World Wide Web Consortium (W3C)
http://www.w3.org/

[2] Web Accessibility Initiative (WAI)
http://www.w3.org/WAI/

[3] Web Content Accessibility Guidelines (WCAG)
http://www.w3.org/WAI/GL/

[4] Evaluation and Repair Tools Working Group
http://www.w3.org/WAI/ER/

[5] Evaluation and Report Language (EARL)
http://www.w3.org/TR/EARL10/

[6] Evaluation and Repair Tools
http://www.w3.org/WAI/ER/existingtools.html

[7] Resource Description Framework (RDF)
http://www.w3.org/RDF/

[8] Web Ontology Language (OWL)
http://www.w3.org/2004/OWL/

[9] Quality Assurance Activity
http://www.w3.org/QA/

[10] Semantic Web Activity
http://www.w3.org/2001/sw/

Context Sequence Matching for XML

Temur Kutsia [1,2]

Research Institute for Symbolic Computation
Johannes Kepler University
A-4040 Linz, Austria

Abstract

Context and sequence variables allow matching to explore term-trees both in depth and in breadth. It makes context sequence matching a suitable computational mechanism for a rule-based language to query and transform XML, or to specify and verify web sites. Such a language would have advantages of both path-based and pattern-based languages. We develop a context sequence matching algorithm and its extension for regular expression matching, and prove their soundness, termination and completeness properties.

Key words: Context variables, Sequence variables, Matching, Regular expressions, XML querying, Web site verification.

1 Introduction

Context variables allow matching to descend in a term represented as a tree to arbitrary depth. Sequence variables give terms a flexible arity and allow matching to move to arbitrary breadth. The ability to explore these two orthogonal directions makes context sequence matching a useful mechanism for querying data available as a large term, like XML documents [26].

Context variables may be instantiated with a context—a term with a hole. Sequence variables may be instantiated with a finite (maybe empty) sequence of terms. Sequence variables are normally used with flexible arity function symbols. Besides context and sequence variables we have function and individual variables. Function variables may be instantiated with a single function symbol or with another function variable. Individual variables may be bound with a single term. Like context and sequence variables, functional and individual variables can be used to traverse terms in depth and breadth, respectively, but only in one level.

[1] Supported by Austrian Science Foundation (FWF) under the SFB projects 1302 and 1322.

[2] Email: kutsia@risc.uni-linz.ac.at

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

In this paper we develop a matching algorithm for terms built using flexible arity function symbols and involving context, sequence, function, and individual variables. We call it the context sequence matching algorithm underlying the importance of the context and sequence variables. We prove soundness, termination, and completeness of the algorithm. It generates a minimal complete set of solutions for the input matching problem. For solvable problems this set is finite, that indicates that context sequence matching is finitary.

Context matching and unification have been intensively investigated in the recent past years, see e.g, [9,10,19,22,23,24]. Context matching is decidable. Decidability of context unification is still an open question [25]. Schmidt-Schauß and Stuber in [24] gave a context matching algorithm and noted that it can be used similar to XPath [7] matching for XML documents. Sequence matching and unification was addressed, for instance, in [2,12,13,16,17,18]. Both matching and unification with sequence variables are decidable. Sequence unification procedure described in [17,18] was implemented in the constraint logic programming language CLP(Flex) [8] and was used for XML processing. However, to the best of our knowledge, so far there was no attempt to combine these two kinds of variables in a single framework. The main contribution of this paper is exactly to develop such a framework and show its usefulness in XML querying, transformation, and web site verification. Incorporating regular expressions into context sequence matching problems is one of such useful features. We give regular expression matching rules that extend those for context sequence matching and show soundness, termination, and completeness of such an extension. Regular expressions constrain both context and sequence variables, i.e., these expressions can be used both on depth and on breadth in terms, which provides a high degree of flexibility and expressiveness. Also, we can easily express incomplete and unordered queries.

Simulation unification [4] implemented in the Xcerpt language has a *descendant* construct that is similar to context variables in the sense that it allows to descend in terms to arbitrary depth, but it does not allow regular expressions along it. Also, sequence variables are not present there. However, it can process unordered and incomplete queries, and it is a full scale unification, not a matching. Having sequence variables in a full scale unification would make it infinitary (see e.g., [18]).

In our opinion, context sequence matching can serve as a computational mechanism for a declarative, rule-based language to query and transform XML, or to specify and verify web sites. Such a query language can have advantages from both path-based and pattern-based languages that form two important classes of XML query languages. Path-based languages usually allow to access a single set of nodes of the graph or tree representing an XML data. The access is based on relations with other nodes in the graph or tree specified in the path expression. Pattern-based languages allow access to several parts of the graph or tree at once specifying the relations among the accessed nodes by tree or graph patterns. (For a recent survey over query and

transformation languages see [11].) Moreover, with context sequence matching we can achieve improved control on rewriting that can be useful for rewriting-based web site specification and verification techniques [1].

Another application area for context sequence matching is mathematical knowledge management. For instance, it can retrieve algorithms or problems from the schema library [5] of the Theorema system [6].

We made a prototype implementation of the context sequence matching algorithm in the rule-based programming system ρLog [21].

The paper is organized as follows: In Section 2 we introduce preliminary notions and fix the terminology. In Section 3 we design the context sequence matching algorithm and prove its properties. In Section 4 we introduce rules for regular expression matching for context and sequence variables. In Section 5 we discuss usefulness of context sequence matching for languages to query and transform XML and to verify web sites. Section 6 concludes.

2 Preliminaries

We assume fixed pairwise disjoint sets of symbols: individual variables $\mathcal{V}_{\text{Ind}}$, sequence variables $\mathcal{V}_{\text{Seq}}$, function variables $\mathcal{V}_{\text{Fun}}$, context variables $\mathcal{V}_{\text{Con}}$, and function symbols $\mathcal{F}$. The sets $\mathcal{V}_{\text{Ind}}$, $\mathcal{V}_{\text{Seq}}$, $\mathcal{V}_{\text{Fun}}$, and $\mathcal{V}_{\text{Con}}$ are countable. The set $\mathcal{F}$ is finite or countable. All the symbols in $\mathcal{F}$ except a distinguished constant $\circ$ (called a *hole*) have flexible arity. We will use x, y, z for individual variables, $\overline{x}, \overline{y}, \overline{z}$ for sequence variables, F, G, H for function variables, $\overline{C}, \overline{D}, \overline{E}$ for context variables, and a, b, c, f, g, h for function symbols. We may use these meta-variables with indices as well.

Terms are constructed using the following grammar:

$$t ::= x \mid \overline{x} \mid \circ \mid f(t_1, \ldots, t_n) \mid F(t_1, \ldots, t_n) \mid \overline{C}(t)$$

In $\overline{C}(t)$ the term t can not be a sequence variable. We will write a for the term $a()$ where $a \in \mathcal{F}$. The meta-variables s, t, r, maybe with indices, will be used for terms. A *ground* term is a term without variables. A *context* is a term with a single occurrence of the hole constant $\circ$. To emphasize that a term t is a context we will write $t[\circ]$. A context $t[\circ]$ may be applied to a term s that is not a sequence variable, written $t[s]$, and the result is the term consisting of t with $\circ$ replaced by s. We will use C and D, with or without indices, for contexts.

A *substitution* is a mapping from individual variables to those terms which are not sequence variables and contain no holes, from sequence variables to finite, possibly empty sequences of terms without holes, from function variables to function variables and symbols, and from context variables to contexts, such that all but finitely many individual and function variables are mapped to themselves, all but finitely many sequence variables are mapped to themselves considered as singleton sequences, and all but finitely many context

variables are mapped to themselves applied to the hole. For example, the mapping $\{x \mapsto f(a, \overline{y}), \overline{x} \mapsto \ulcorner\urcorner, \overline{y} \mapsto \ulcorner a, \overline{C}(f(b)), x\urcorner, F \mapsto g, \overline{C} \mapsto g(\circ)\}$ is a substitution. Note that we identify a singleton sequence with its sole member. We will use lower case Greek letters $\sigma, \vartheta, \varphi$, and ε for substitutions, where ε will denote the empty substitution. As usual, indices may be used with the meta-variables.

Substitutions are extended to terms as follows:

$$\begin{aligned} x\sigma &= \sigma(x) \\ \overline{x}\sigma &= \sigma(\overline{x}) \\ f(t_1, \ldots, t_n)\sigma &= f(t_1\sigma, \ldots, t_n\sigma) \\ F(t_1, \ldots, t_n)\sigma &= \sigma(F)(t_1\sigma, \ldots, t_n\sigma) \\ \overline{C}(t)\sigma &= \sigma(\overline{C})[t\sigma]. \end{aligned}$$

A substitution σ is *more general* than ϑ, denoted $\sigma \leq \vartheta$, if there exists a φ such that $\sigma\varphi = \vartheta$. A *context sequence matching problem* is a finite multiset of term pairs (*matching equations*), written $\{s_1 \ll t_1, \ldots, s_n \ll t_n\}$, where the s's and the t's contain no holes, the s's are not sequence variables, and the t's are ground. We will also call the s's the *query* and the t's the *data*. Substitutions are extended to matching equations and matching problems in the usual way. A substitution σ is called a *matcher* of the matching problem $\{s_1 \ll t_1, \ldots, s_n \ll t_n\}$ if $s_i\sigma = t_i$ for all $1 \leq i \leq n$. We will use Γ and Δ to denote matching problems. A *complete set of matchers* of a matching problem Γ is a set of substitutions S such that (i) each element of S is a matcher of Γ, and (ii) for each matcher ϑ of Γ there exist a substitution $\sigma \in S$ such that $\sigma \leq \vartheta$. The set S is a *minimal complete set of matchers* of Γ if it is a complete set and two distinct elements of S are incomparable with respect to $\leq$. For solvable problems this set is finite, i.e. context sequence matching is finitary.

Example 2.1 *The minimal complete set of matchers for the context sequence matching problem* $\{\overline{C}(f(\overline{x})) \ll g(f(a, b), h(f(a)))\}$ *consists of two elements:* $\{\overline{C} \mapsto g(\circ, h(f(a))),\ \overline{x} \mapsto \ulcorner a, b\urcorner\}$ *and* $\{\overline{C} \mapsto g(f(a, b), h(\circ)),\ \overline{x} \mapsto a\}$.

3 Matching Algorithm

We now present inference rules for deriving solutions for matching problems. A *system* is either the symbol $\bot$ (representing failure) or a pair $\Gamma; \sigma$, where Γ is a matching problem and σ is a substitution. The inference system $\mathfrak{I}$ consists of the transformation rules on systems listed below. We assume that the indices n and m are non-negative unless otherwise stated.

T: **Trivial**

$\{t \ll t\} \cup \Gamma';\ \sigma \Longrightarrow \Gamma';\ \sigma.$

IVE: **Individual Variable Elimination**

$$\{x \ll t\} \cup \Gamma';\ \sigma \Longrightarrow \Gamma'\vartheta;\ \sigma\vartheta, \qquad \text{where } \vartheta = \{x \mapsto t\}.$$

FVE: **Function Variable Elimination**

$$\{F(s_1, \ldots, s_n) \ll f(t_1, \ldots, t_m)\} \cup \Gamma';\ \sigma \\ \Longrightarrow \{f(s_1\vartheta, \ldots, s_n\vartheta) \ll f(t_1, \ldots, t_m)\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$$

where $\vartheta = \{F \mapsto f\}$.

TD: **Total Decomposition**

$$\{f(s_1, \ldots, s_n) \ll f(t_1, \ldots, t_n)\} \cup \Gamma';\ \sigma \\ \Longrightarrow \{s_1 \ll t_1, \ldots, s_n \ll t_n\} \cup \Gamma';\ \sigma,$$

if $f(s_1, \ldots, s_n) \neq f(t_1, \ldots, t_n)$ and $s_i \notin \mathcal{V}_{\text{Seq}}$ for all $1 \leq i \leq n$.

PD: **Partial Decomposition**

$$\{f(s_1, \ldots, s_n) \ll f(t_1, \ldots, t_m)\} \cup \Gamma';\ \sigma \\ \Longrightarrow \{s_1 \ll t_1, \ldots, s_{k-1} \ll t_{k-1}, f(s_k, \ldots, s_n) \ll f(t_k, \ldots, t_m)\} \cup \Gamma';\ \sigma,$$

if $f(s_1, \ldots, s_n) \neq f(t_1, \ldots, t_m)$, $s_k \in \mathcal{V}_{\text{Seq}}$ for some $1 < k \leq \min(n, m) + 1$, and $s_i \notin \mathcal{V}_{\text{Seq}}$ for all $1 \leq i < k$.

SVD: **Sequence Variable Deletion**

$$\{f(\overline{x}, s_1, \ldots, s_n) \ll t\} \cup \Gamma';\ \sigma \Longrightarrow \{f(s_1\vartheta, \ldots, s_n\vartheta) \ll t\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$$

where $\vartheta = \{\overline{x} \mapsto \ulcorner\urcorner\}$.

W: **Widening**

$$\{f(\overline{x}, s_1, \ldots, s_n) \ll f(t, t_1, \ldots, t_m)\} \cup \Gamma';\ \sigma \\ \Longrightarrow \{f(\overline{x}, s_1\vartheta, \ldots, s_n\vartheta) \ll f(t_1, \ldots, t_m)\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$$

where $\vartheta = \{\overline{x} \mapsto \ulcorner t, \overline{x}\urcorner\}$.

CVD: **Context Variable Deletion**

$$\{\overline{C}(s) \ll t\} \cup \Gamma';\ \sigma \Longrightarrow \{s\vartheta \ll t\} \cup \Gamma'\vartheta;\ \sigma\vartheta, \qquad \text{where } \vartheta = \{\overline{C} \mapsto \circ\}.$$

D: **Deepening**

$$\{\overline{C}(s) \ll f(t_1, \ldots, t_m)\} \cup \Gamma';\ \sigma \Longrightarrow \{\overline{C}(s\vartheta) \ll t_j\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$$

where $\vartheta = \{\overline{C} \mapsto f(t_1, \ldots, t_{j-1}, \overline{C}(\circ), t_{j+1}, \ldots, t_m)\}$ for some $1 \leq j \leq m$, and $m > 0$.

SC: **Symbol Clash**

$$\{f(s_1, \ldots, s_n) \ll g(t_1, \ldots, t_m)\} \cup \Gamma';\ \sigma \Longrightarrow \bot,$$

if $f \notin \mathcal{V}_{\text{Con}} \cup \mathcal{V}_{\text{Fun}}$ and $f \neq g$.

AD: **Arity Disagreement**

$$\{f(s_1, \ldots, s_n) \ll f(t_1, \ldots, t_m)\} \cup \Gamma';\ \sigma \Longrightarrow \bot,$$

if $m \neq n$ and $s_i \notin \mathcal{V}_{\text{Seq}}$ for all $1 \leq i \leq n$.

We may use the rule name abbreviations as subscripts, e.g., $\Gamma_1; \sigma_1 \Longrightarrow_{\mathsf{T}} \Gamma_2; \sigma_2$ for the Trivial rule. SVD, W, CVD, and D are non-deterministic rules.

A *derivation* is a sequence $\Gamma_1; \sigma_1 \Longrightarrow \Gamma_2; \sigma_2 \Longrightarrow \cdots$ of system transformations.

Definition 3.1 A *context sequence matching algorithm* $\mathfrak{M}$ is any program that takes a system $\Gamma; \varepsilon$ as an input and uses the rules in $\mathfrak{I}$ to generate a complete tree of derivations, called the *matching tree for* Γ, in the following way:

(i) The root of the tree is labeled with $\Gamma; \varepsilon$.

(ii) Each branch of the tree is a derivation. The nodes in the tree are systems.

(iii) If several transformation rules, or different instances of the same transformation rule are applicable to a node in the tree, they are applied concurrently.

The leaves of a matching tree are labeled either with the systems of the form $\emptyset; \sigma$ or with $\bot$. The branches that end with $\emptyset; \sigma$ are *successful branches*, and those that end with $\bot$ are *failed branches*. We denote by $\mathcal{S}ol_{\mathfrak{M}}(\Gamma)$ the solution set of Γ generated by $\mathfrak{M}$, i.e., the set of all σ's such that $\emptyset; \sigma$ is a leaf of the matching tree for Γ.

Theorem 3.2 (Soundness of $\mathfrak{M}$) *Let Γ be a matching problem. Then every substitution $\sigma \in \mathcal{S}ol_{\mathfrak{M}}(\Gamma)$ is a matcher of Γ.*

Proof. (Sketch) Inspecting the rules in $\mathfrak{I}$ one can conclude that for a derivation $\Gamma; \varepsilon \Longrightarrow^+ \emptyset; \sigma$ the problems $\Gamma\sigma$ and $\emptyset$ have the same set of matchers. It implies that σ is a matcher of Γ. □

Theorem 3.3 (Termination of $\mathfrak{M}$) *The algorithm $\mathfrak{M}$ terminates on any input.*

Proof. With each matching problem Δ we associate a complexity measure as a triple of non-negative integers $\langle n_1, n_2, n_3 \rangle$, where n_1 is the number of distinct variables in Δ, n_2 is the number of symbols in the ground sides of matching equations in Δ, and n_3 is the number of subterms in Δ of the form $f(s_1, \ldots, s_n)$, where s_1 is not a sequence variable. Measures are compared lexicographically. Every non-failing rule in $\mathfrak{I}$ strictly decreases the measure. Failing rules immediately lead to termination. Hence, $\mathfrak{M}$ terminates on any input. □

Theorem 3.4 (Completeness of $\mathfrak{M}$) *Let Γ be a matching problem and let ϑ be a matcher of Γ. Then there exists a derivation $\Gamma; \varepsilon \Longrightarrow^+ \emptyset; \sigma$ such that $\sigma \preceq \vartheta$.*

Proof. We construct the derivation recursively. For the base case $\Gamma_1; \sigma_1 = \Gamma; \varepsilon$ we have $\varepsilon \preceq \vartheta$. Now assume that the system $\Gamma_n; \sigma_n$, where $n \geq 1$ and $\Gamma_n \neq \emptyset$, belongs to the derivation and find a system $\Gamma_{n+1}; \sigma_{n+1}$ such that

$\Gamma_n;\sigma_n \Longrightarrow \Gamma_{n+1};\sigma_{n+1}$ and $\sigma_{n+1} \preceq \vartheta$. We have $\sigma_n \preceq \vartheta$. Therefore, there exists φ such that $\sigma_n\varphi = \vartheta$ and φ is a matcher of Γ_n. Without loss of generality, we pick an arbitrary matching equation $s \ll t$ from Γ_n and represent Γ_n as $\{s \ll t\} \cup \Gamma'_n$. Depending on the form of $s \ll t$, we have three cases:

Case 1. The terms s and t are the same. We extend the derivation with the step $\Gamma_n;\sigma_n \Longrightarrow_{\mathsf{T}} \Gamma'_n;\sigma_n$. Therefore, $\sigma_{n+1} = \sigma_n \preceq \vartheta$.

Case 2. The term s is an individual variable x. Then $x\varphi = t$. Therefore, for $\psi = \{x \mapsto t\}$ we have $\psi\varphi = \varphi$ and, hence, $\sigma_n\psi\varphi = \vartheta$. We extend the derivation with the step $\Gamma_n;\sigma_n \Longrightarrow_{\mathsf{IVE}} \Gamma'_n;\sigma_{n+1}$, where $\sigma_{n+1} = \sigma_n\psi \preceq \vartheta$.

Case 3. The terms s and t are not the same and s is a compound term. The only non-trivial cases are those when the first argument of s is a sequence variable, or when the head of s is a context variable. If the first argument of s is a sequence variable $\overline{x}$ then φ must contain a binding $\overline{x} \mapsto \ulcorner t_1, \ldots, t_k \urcorner$ for $\overline{x}$, where $k \geq 0$ and t_i's are ground terms. If $k = 0$ then we take $\psi = \{\overline{x} \mapsto \ulcorner\urcorner\}$ and extend the derivation with the step $\Gamma_n;\sigma_n \Longrightarrow_{\mathsf{SVD}} \Gamma'_n;\sigma_{n+1}$, where $\sigma_{n+1} = \sigma_n\psi$. If $k > 0$ then we take $\psi = \{\overline{x} \mapsto \ulcorner t_1, \overline{x}\urcorner\}$ and extend the derivation with the step $\Gamma_n;\sigma_n \Longrightarrow_{\mathsf{W}} \Gamma'_n;\sigma_{n+1}$, where $\sigma_{n+1} = \sigma_n\psi$. In both cases we have $\sigma_{n+1} = \sigma_n\psi \preceq \sigma_n\varphi = \vartheta$. If the head of s is a context variable $\overline{C}$ then φ must contain a binding $\overline{C} \mapsto C$ for $\overline{C}$, where C is a ground context. If $C = \circ$ then we take $\psi = \{\overline{C} \mapsto \circ\}$ and we extend the derivation with the step $\Gamma_n;\sigma_n \Longrightarrow_{\mathsf{CVD}} \Gamma'_n;\sigma_{n+1}$, where $\sigma_{n+1} = \sigma_n\psi$. If $C \neq \circ$ then C should have a form $f(t_1, \ldots, t_{j-1}, D, t_{j+1}, \ldots, t_m)$, where D is a context and $f(t_1, \ldots, t_m) = t$. Then we take $\psi = \{\overline{C} \mapsto f(t_1, \ldots, t_{j-1}, \overline{C}(\circ), t_{j+1}, \ldots, t_m)\}$ and extend the derivation with the step $\Gamma_n;\sigma_n \Longrightarrow_{\mathsf{W}} \Gamma'_n;\sigma_{n+1}$, where $\sigma_{n+1} = \sigma_n\psi$. In both cases $\sigma_{n+1} = \sigma_n\psi \preceq \sigma_n\varphi = \vartheta$. □

Theorem 3.5 (Minimality) *Let Γ be a matching problem. Then $\mathcal{S}ol_{\mathfrak{M}}(\Gamma)$ is a minimal set of matchers of Γ.*

Proof. For any matching problem Δ the set

$$S(\Delta) = \{\varphi \mid \Delta;\varepsilon \Longrightarrow \Phi;\varphi \text{ for some } \Phi\}$$

is minimal. Moreover, every substitution ϑ in $S(\Delta)$ preserves minimality: If $\{\sigma_1, \ldots, \sigma_n\}$ is a minimal set of substitutions then so is the set $\{\vartheta\sigma_1, \ldots, \vartheta\sigma_n\}$. It implies that $\mathcal{S}ol_{\mathfrak{M}}(\Gamma)$ is minimal. □

These results are summarized in the main theorem.

Theorem 3.6 (Main Theorem) *The matching algorithm $\mathfrak{M}$ terminates for any input problem Γ and generates a minimal complete set of matchers of Γ.*

Moreover, note that $\mathfrak{M}$ never computes the same matcher twice.

If we are not interested in bindings for certain variables, we can replace them with the anonymous variables: "_ " for any individual or function variable, and "___ " for any sequence or context variable. It is straightforward to adapt the rules in $\mathfrak{I}$ to anonymous variables: If an anonymous variable occurs

in the rule IVE, FVE, SVD, W, CVD, or D then the substitution ϑ in the same rule is the empty substitution ε. It is interesting to note that a context sequence matching equation $s \ll t$ whose all variables are anonymous variables can be considered as a problem of computing simulations of s in t that can be efficiently solved by the algorithm described in [14].

4 Regular Expressions

Regular expressions provide a powerful mechanism for restricting data values in XML. Many languages have support for them. In [15] regular expression pattern matching is proposed as a core feature of programming languages for manipulating XML. The classical approach uses finite automata for regular expression matching. In this section we show that regular expression matching can be easily incorporated into the rule-based framework of context sequence matching.

Regular expressions on terms are defined by the following grammar:

$$\mathtt{R} ::= t \mid \ulcorner\urcorner \mid \ulcorner \mathtt{R}_1, \mathtt{R}_2 \urcorner \mid \mathtt{R}_1 | \mathtt{R}_2 \mid \mathtt{R}^*,$$

where t is a term without holes, $\ulcorner\urcorner$ is the empty sequence, "," is concatenation, "|" is choice, and $*$ is repetition (Kleene star). The symbols "$\ulcorner$" and "$\urcorner$" are there just for the readability purposes. The operators are right-associative; "*" has the highest precedence, followed by "," and "|".

Substitutions are extended to regular expressions on terms in the usual way: $\ulcorner\urcorner\sigma = \ulcorner\urcorner$, $\ulcorner \mathtt{R}_1, \mathtt{R}_2 \urcorner \sigma = \ulcorner \mathtt{R}_1\sigma, \mathtt{R}_2\sigma \urcorner$, $(\mathtt{R}_1|\mathtt{R}_2)\sigma = \mathtt{R}_1\sigma|\mathtt{R}_2\sigma$, and $\mathtt{R}^*\sigma = (\mathtt{R}\sigma)^*$.

Regular expressions on functions are defined as follows:

$$\mathtt{Q} ::= f \mid F \mid \overline{C} \mid \ulcorner \mathtt{Q}_1, \mathtt{Q}_2 \urcorner \mid \mathtt{Q}_1 | \mathtt{Q}_2 \mid \mathtt{Q}^*.$$

Note that by this definition the hole $\circ$ is a regular expression on functions, because it is a (constant) function symbol.

We introduce a new operation $\diamond$ as a special way of applying substitutions on context variables: For any $\overline{C}$ and σ, $\overline{C} \diamond \sigma = \mathit{path}(\overline{C}\sigma)$, where $\mathit{path}(C)$ is the sequence of symbols from the head of the context C to the hole in C. For instance, $\mathit{path}(f(a, f(g(a), H(b, \overline{D}(h(c), \circ), b), c))) = \ulcorner f, f, H, \overline{D} \urcorner$ and $\mathit{path}(\circ) = \ulcorner\urcorner$. We can extend $\diamond$ to regular expressions on functions: $f \diamond \sigma = f\sigma$, $F \diamond \sigma = F\sigma$, $\ulcorner \mathtt{Q}_1, \mathtt{Q}_2 \urcorner \diamond \sigma = \ulcorner \mathtt{Q}_1 \diamond \sigma, \mathtt{Q}_2 \diamond \sigma \urcorner$, $(\mathtt{Q}_1|\mathtt{Q}_2) \diamond \sigma = \mathtt{Q}_1 \diamond \sigma | \mathtt{Q}_2 \diamond \sigma$, and $\mathtt{Q}^* \diamond \sigma = (\mathtt{Q} \diamond \sigma)^*$.

We write $L(\mathtt{E})$ for a regular language described by the regular expression $\mathtt{E}$. The only element of $L(\ulcorner\urcorner)$ and $L(\circ)$ is the empty sequence $\ulcorner\urcorner$.

Patterns are atoms of the form Ts in $\mathtt{R}$ or Fs in $\mathtt{Q}$, where Ts is a finite, possibly empty, sequence of terms and Fs is a finite, possibly empty, sequence of function symbols, function variables, and context variables. Pattern-pairs are pairs $(p, \mathtt{f})$ where p is a pattern and $\mathtt{f}$ is a flag that is an integer 0 or 1. The intuition behind the pattern-pair $(\overline{x}$ in $\mathtt{R}, \mathtt{f})$ (resp. $(\overline{C}$ in $\mathtt{Q}, \mathtt{f})$) is that if

$\mathtt{f} = 0$ then $\overline{x}$ (resp. $\overline{C}$) is allowed to be replaced with $\ulcorner\urcorner$ (resp. with $\circ$) if $\mathtt{R}$ (resp. $\mathtt{Q}$) permits. If $\mathtt{f} = 1$ then the replacement is impossible, even if the corresponding regular expression permits. It will be needed later to guarantee that the regular pattern matching algorithm terminates. Substitutions are extended to pattern-pairs as follows: $(Ts \ \mathsf{in}\ \mathtt{R}, \mathtt{f})\sigma = (Ts\sigma \ \mathsf{in}\ \mathtt{R}\sigma, \mathtt{f}\sigma)$, and $(Fs \ \mathsf{in}\ \mathtt{Q}, \mathtt{f})\sigma = (Fs \diamond \sigma \ \mathsf{in}\ \mathtt{Q} \diamond \sigma, \mathtt{f} \diamond \sigma)$.

A *context sequence regular pattern matching problem* is a multiset of matching equations and pattern-pairs of the form:

$$\{s_1 \ll t_1, \ldots, s_n \ll t_n,\ (\overline{x}_1 \ \mathsf{in}\ \mathtt{R}_1, \mathtt{f}_1^1), \ldots, (\overline{x}_m \ \mathsf{in}\ \mathtt{R}_m, \mathtt{f}_m^1),\\ (\overline{C}_1 \ \mathsf{in}\ \mathtt{Q}_1, \mathtt{f}_1^2), \ldots, (\overline{C}_k \ \mathsf{in}\ \mathtt{Q}_k, \mathtt{f}_k^2)\},$$

where all $\overline{x}$'s and all $\overline{C}$'s are distinct and do not occur in $\mathtt{R}$'s and $\mathtt{Q}$'s. We will assume that all $\overline{x}$'s and $\overline{C}$'s occur in the matching equations. A substitution σ is called a *regular pattern matcher* for such a problem if $s_i\sigma = t_i$, $\overline{x}_j\sigma \in L(\mathtt{R}_j\sigma)_{\mathtt{f}_j^1}$, and $\overline{C}_l \diamond \sigma \in L(\mathtt{Q}_l \diamond \sigma)_{\mathtt{f}_l^2}$ for all $1 \leq i \leq n$, $1 \leq j \leq m$, and $1 \leq l \leq k$, where $L(\mathtt{P})_0 = L(\mathtt{P})$ and $L(\mathtt{P})_1 = L(\mathtt{P}) \setminus \{\ulcorner\urcorner\}$.

We define the inference system $\mathfrak{I}_R$ to solve context sequence regular pattern matching problems. It operates on systems $\Gamma; \sigma$ where Γ is a regular pattern matching problem and σ is a substitution. The system $\mathfrak{I}_R$ includes all the rules from the system $\mathfrak{I}$, but SVD, W, CVD, and D need an extra condition on applicability: For the variables $\overline{x}$ and $\overline{C}$ in those rules there should be no pattern-pair $(\overline{x} \ \mathsf{in}\ \mathtt{R}, \mathtt{f}^1)$ and $(\overline{C} \ \mathsf{in}\ \mathtt{Q}, \mathtt{f}^2)$ in the matching problem. There are additional rules in $\mathfrak{I}_R$ for the variables constrained by pattern-pairs listed below. The meta-functions $\mathtt{NonEmpty}$ and $\oplus$ used in these rules are defined as follows: $\mathtt{NonEmpty}() = 0$ and $\mathtt{NonEmpty}(r_1, \ldots, r_n) = 1$ if $r_i \notin \mathcal{V}_{\mathrm{Seq}} \cup \mathcal{V}_{\mathrm{Con}}$ for some $1 \leq i \leq n$; $0 \oplus 0 = 1 \oplus 1 = 0$ and $1 \oplus 0 = 0 \oplus 1 = 1$.

ESRET: **Empty Sequence in a Regular Expression for Terms**

$$\{f(\overline{x}, s_1, \ldots, s_n) \ll t,\ (\overline{x} \ \mathsf{in}\ \ulcorner\urcorner, \mathtt{f})\} \cup \Gamma';\ \sigma \\ \Longrightarrow \{f(\overline{x}, s_1, \ldots, s_n)\vartheta \ll t\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$$

where $\vartheta = \{\overline{x} \mapsto \ulcorner\urcorner\}$ if $\mathtt{f} = 0$. If $\mathtt{f} = 1$ the rule fails.

TRET: **Term in a Regular Expression for Terms**

$$\{f(\overline{x}, s_1, \ldots, s_n) \ll t,\ (\overline{x} \ \mathsf{in}\ s, \mathtt{f})\} \cup \Gamma';\ \sigma \\ \Longrightarrow \{f(\overline{x}, s_1, \ldots, s_n)\vartheta \ll t\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$$

where $\vartheta = \{\overline{x} \mapsto s\}$ and $s \notin \mathcal{V}_{\mathrm{Seq}}$.

SVRET: **Sequence Variable in a Regular Expression for Terms**

$$\{f(\overline{x}, s_1, \ldots, s_n) \ll t,\ (\overline{x} \ \mathsf{in}\ \overline{y}, \mathtt{f})\} \cup \Gamma';\ \sigma \\ \Longrightarrow \{f(\overline{x}, s_1, \ldots, s_n)\vartheta \ll t\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$$

where $\vartheta = \{\overline{x} \mapsto \overline{y}\}$ if $\mathtt{f} = 0$. If $\mathtt{f} = 1$ then $\vartheta = \{\overline{x} \mapsto \ulcorner y, \overline{y}\urcorner\}$ where y is a fresh variable.

ChRET: **Choice in a Regular Expression for Terms**

$\{f(\overline{x}, s_1, \ldots, s_n) \ll t,\ (\overline{x}\ \mathsf{in}\ \mathtt{R}_1|\mathtt{R}_2, \mathtt{f})\} \cup \Gamma';\ \sigma$
$\Longrightarrow \{f(\overline{x}, s_1, \ldots, s_n)\vartheta \ll t,\ (\overline{y}_i\ \mathsf{in}\ \mathtt{R}_i, \mathtt{f})\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$

for $i = 1, 2$, where y_i is a fresh variable and $\vartheta = \{\overline{x} \mapsto \overline{y}_i\}$.

CRET: **Concatenation in a Regular Expression for Terms**

$\{f(\overline{x}, s_1, \ldots, s_n) \ll t,\ (\overline{x}\ \mathsf{in}\ \ulcorner \mathtt{R}_1, \mathtt{R}_2 \urcorner, \mathtt{f})\} \cup \Gamma';\ \sigma$
$\Longrightarrow \{f(\overline{x}, s_1, \ldots, s_n)\vartheta \ll t,\ (\overline{y}_1\ \mathsf{in}\ \mathtt{R}_1, \mathtt{f}_1), (\overline{y}_2\ \mathsf{in}\ \mathtt{R}_2, \mathtt{f}_2)\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$

where $\overline{y}_1$ and $\overline{y}_2$ are fresh variables, $\vartheta = \{\overline{x} \mapsto \ulcorner \overline{y}_1, \overline{y}_2 \urcorner\}$, and $\mathtt{f}_1$ and $\mathtt{f}_2$ are computed as follows: If $\mathtt{f} = 0$ then $\mathtt{f}_1 = \mathtt{f}_2 = 0$ else $\mathtt{f}_1 = 0$ and $\mathtt{f}_2 = \mathtt{NonEmpty}(\overline{y}_1) \oplus 1$.

RRET1: **Repetition in a Regular Expression for Terms 1**

$\{f(\overline{x}, s_1, \ldots, s_n) \ll t,\ (\overline{x}\ \mathsf{in}\ \mathtt{R}^*, \mathtt{f})\} \cup \Gamma';\ \sigma$
$\Longrightarrow \{f(\overline{x}, s_1, \ldots, s_n)\vartheta \ll t\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$

where $\vartheta = \{\overline{x} \mapsto \ulcorner \urcorner\}$ and $\mathtt{f} = 0$. If $\mathtt{f} = 1$ the rule fails.

RRET2: **Repetition in a Regular Expression for Terms 2**

$\{f(\overline{x}, s_1, \ldots, s_n) \ll t,\ (\overline{x}\ \mathsf{in}\ \mathtt{R}^*, \mathtt{f})\} \cup \Gamma';\ \sigma$
$\Longrightarrow \{f(\overline{x}, s_1, \ldots, s_n)\vartheta \ll t,\ (\overline{y}\ \mathsf{in}\ \mathtt{R}, 1),\ (\overline{x}\ \mathsf{in}\ \mathtt{R}^*, 0)\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$

where y is a fresh variable and $\vartheta = \{\overline{x} \mapsto \ulcorner \overline{y}, \overline{x} \urcorner\}$.

HREF: **Hole in a Regular Expression for Functions**

$\{\overline{C}(s) \ll t,\ (\overline{C}\ \mathsf{in}\ \circ, \mathtt{f})\} \cup \Gamma';\ \sigma \Longrightarrow \{\overline{C}(s)\vartheta \ll t\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$

where $\vartheta = \{\overline{C} \mapsto \circ\}$ and $\mathtt{f} = 0$. If $\mathtt{f} = 1$ the rule fails.

FREF: **Function in a Regular Expression for Functions**

$\{\overline{C}(s) \ll t,\ (\overline{C}\ \mathsf{in}\ M, \mathtt{f})\} \cup \Gamma';\ \sigma \Longrightarrow \{\overline{C}(s)\vartheta \ll t\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$

where $M \in (\mathcal{F} \setminus \{\circ\}) \cup \mathcal{V}_{\mathrm{Fun}}$, and $\vartheta = \{\overline{C} \mapsto M(\overline{x}, \circ, \overline{y})\}$ with fresh variables $\overline{x}$ and $\overline{y}$.

CVREF: **Context Variable in a Regular Expression for Functions**

$\{\overline{C}(s) \ll t,\ (\overline{C}\ \mathsf{in}\ \overline{D}, \mathtt{f})\} \cup \Gamma';\ \sigma \Longrightarrow \{\overline{C}(s)\vartheta \ll t\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$

where $\vartheta = \{\overline{C} \mapsto \overline{D}(\circ)\}$ if $\mathtt{f} = 0$. If $\mathtt{f} = 1$ then $\vartheta = \{\overline{C} \mapsto F(\overline{x}, \overline{D}(\circ), \overline{y})\}$ where $F, \overline{x}$, and $\overline{y}$ are fresh variables.

ChREF: **Choice in a Regular Expression for Functions**

$\{\overline{C}(s) \ll t,\ (\overline{C}\ \mathsf{in}\ \mathtt{Q}_1|\mathtt{Q}_2, \mathtt{f})\} \cup \Gamma';\ \sigma$
$\Longrightarrow \{\overline{C}(s)\vartheta \ll t,\ (\overline{D}_i\ \mathsf{in}\ \mathtt{Q}_i, \mathtt{f})\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$

for $i = 1, 2$, where $\overline{D}_i$ is a fresh variable and $\vartheta = \{\overline{C} \mapsto \overline{D}_i(\circ)\}$.

CREF: **Concatenation in a Regular Expression for Functions**

$$\{\overline{C}(s) \ll t,\ (\overline{C}\ \mathsf{in}\ \ulcorner \mathtt{Q}_1, \mathtt{Q}_2 \urcorner, \mathtt{f})\} \cup \Gamma';\ \sigma \\ \Longrightarrow \{\overline{C}(s)\vartheta \ll t,\ (\overline{D}_1\ \mathsf{in}\ \mathtt{Q}_1, \mathtt{f}_1), (\overline{D}_2\ \mathsf{in}\ \mathtt{Q}_2, \mathtt{f}_2)\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$$

where $\overline{D}_1$ and $\overline{D}_2$ are fresh variables and $\vartheta = \{\overline{C} \mapsto \overline{D}_1(\overline{D}_2(\circ))\}$, and $\mathtt{f}_1$ and $\mathtt{f}_2$ are computed as follows: If $\mathtt{f} = 0$ then $\mathtt{f}_1 = \mathtt{f}_2 = 0$ else $\mathtt{f}_1 = 0$ and $\mathtt{f}_2 = \mathtt{NonEmpty}(\overline{D}_1) \oplus 1$.

RREF1: **Repetition in a Regular Expression for Functions 1**

$$\{\overline{C}(s) \ll t,\ (\overline{C}\ \mathsf{in}\ \mathtt{Q}^*, \mathtt{f})\} \cup \Gamma';\ \sigma \Longrightarrow \{\overline{C}(s)\vartheta \ll t\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$$

where $\vartheta = \{\overline{C} \mapsto \circ\}$ and $\mathtt{f} = 0$. If $\mathtt{f} = 1$ the rule fails.

RREF2: **Repetition in a Regular Expression for Functions 2**

$$\{\overline{C}(s) \ll t,\ (\overline{C}\ \mathsf{in}\ \mathtt{Q}^*, \mathtt{f})\} \cup \Gamma';\ \sigma \\ \Longrightarrow \{\overline{C}(s)\vartheta \ll t,\ (\overline{D}\ \mathsf{in}\ \mathtt{Q}, 1),\ (\overline{C}\ \mathsf{in}\ \mathtt{Q}^*, 0)\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$$

where $\overline{D}$ is a fresh variable and $\vartheta = \{\overline{C} \mapsto \overline{D}(\overline{C}(\circ))\}$.

A context sequence regular pattern matching algorithm $\mathfrak{M}_R$ is defined in the similar way as the algorithm $\mathfrak{M}$ (Definition 3.1) with the only difference that the rules of $\mathfrak{I}_R$ are used instead of the rules of $\mathfrak{I}$. ¿From the beginning, all the flags in the input problem are set to 0. Note that the rules in $\mathfrak{I}_R$ work either on a selected matching equation, or on a selected pair of a matching equation and a pattern-pair. No rule selects a patter-pair alone. We denote by $\mathcal{S}ol_{\mathfrak{M}_R}(\Gamma)$ the solution set of Γ generated by $\mathfrak{M}_R$.

Theorem 4.1 (Soundness of $\mathfrak{M}_R$) *Let Γ be a regular pattern matching problem. Then every substitution $\sigma \in \mathcal{S}ol_{\mathfrak{M}_R}(\Gamma)$ is a regular pattern matcher of Γ.*

Proof. (Sketch) Inspecting the rules in $\mathfrak{I}_R$ one can conclude that for a derivation $\Gamma; \varepsilon \Longrightarrow^+ \emptyset; \sigma$ every regular pattern matcher of $\emptyset$ is also a regular pattern matcher of $\Gamma\sigma$. It implies that σ is a regular pattern matcher of Γ. □

Theorem 4.2 (Termination of $\mathfrak{M}_R$) *The algorithm $\mathfrak{M}_R$ terminates on any input.*

Proof. The tricky part of the proof is related with patterns containing the star "*". A derivation that contains an application of the RRET2 rule on a system with a selected matching equation and pattern-pair $s_0 \ll t_0, (\overline{x}\ \mathsf{in}\ \mathtt{R}_0^*, \mathtt{f})$ either fails or eventually produces a system that contains a matching equation $s_1 \ll t_1$ and a pattern-pair $(\overline{x}\ \mathsf{in}\ \mathtt{R}_1^*, 0)$ where $\mathtt{R}_1$ is an instance of $\mathtt{R}_0$ and $\overline{x}$ is the first argument of s_1:

$$\{s_0 \ll t_0, (\overline{x}\ \mathsf{in}\ \mathtt{R}_0^*, \mathtt{f})\} \cup \Gamma; \sigma \\ \Longrightarrow_{\mathsf{RRET2}} \{s_0\vartheta \ll t_0, (\overline{y}\ \mathsf{in}\ \mathtt{R}_0, 1), (\overline{x}\ \mathsf{in}\ \mathtt{R}_0^*, \mathtt{f})\} \cup \Gamma\vartheta; \sigma\vartheta \\ \Longrightarrow^+ \{s_1 \ll t_1, (\overline{x}\ \mathsf{in}\ \mathtt{R}_1^*, 0)\} \cup \Delta; \varphi.$$

Hence, the rule RRET2 can apply again on $\{s_1 \ll t_1, (\overline{x} \text{ in } \mathtt{R}_1^*, 0)\} \cup \Delta; \varphi$. The important point is that the total size of the ground sides of the matching equations strictly decreases between these two applications of RRET2: In $\{s_1 \ll t_1\} \cup \Delta$ it is strictly smaller than in $\{s_0 \ll t_0\} \cup \Gamma$. This is guaranteed by the fact that $(\overline{y} \text{ in } \mathtt{R}_0, 1)$ does not allow the variable $\overline{y}$ to be bound with the empty sequence. The same argument applies to derivations that contain an application of the RREF2 rule. Applications of the other rules also lead to a strict decrease of the size of the ground sides after finitely many steps. Since no rule increases the size of the ground sides, the algorithm $\mathfrak{M}_R$ terminates.□

Theorem 4.3 (Completeness of $\mathfrak{M}_R$) *Let Γ be a regular pattern matching problem and let ϑ be a regular pattern matcher of Γ. Then there exists a substitution $\sigma \in \mathcal{S}ol_{\mathfrak{M}_R}$ such that $\sigma \preceq \vartheta$.*

Proof. Similar to the proof of Theorem 3.4. □

Note that we can extend the system $\mathfrak{I}_R$ with some more rules that facilitate an early detection of failure, e.g., $\{f(\overline{x}, s_1, \ldots, s_n) \ll f(), (\overline{x} \text{ in } \mathtt{R}, 1)\} \cup \Gamma'; \sigma \Longrightarrow \bot$ would be one of such rules.

5 Context Sequence Matching and XML

We assume the existence of a declarative, rule-based query and transformation language for XML that uses the context sequence matching to answer queries. Queries are expressed as (conditional) rules *pattern* → *result* `if` *condition*. We do not go into the details, but just mention that regular expression patterns $\overline{x}$ in R and $\overline{C}$ in Q can be used as conditions. In such cases context sequence regular pattern matching can be used to match *pattern* to the data. Arithmetic formulae and matchability tests are other instances of conditions. Note that conditions can also be omitted (assumed to be true). The *pattern* matches the data in the root position. One can choose between getting all the results or only one of them.

To put more syntactic sugar on queries, we borrow some notation from [4]. We write $f\{s_1, \ldots, s_n\}$ if the order of arguments $s_1, \ldots, s_n$ does not matter. The following (rather inefficient) rule relates a matching problem in which the curly bracket construct occurs, to the standard matching problems:

Ord: **Orderless**

$$\{f\{s_1, \ldots, s_n\} \ll t\} \cup \Gamma';\ \sigma \Longrightarrow \{f(s_{\pi(1)}, \ldots, s_{\pi(n)}) \ll t\} \cup \Gamma';\ \sigma,$$

if $f(s_1, \ldots, s_n) \neq t$ and π is a permutation of $1, \ldots, n$.

Moreover, we can use double curly bracket notation $f\{\!\{s_1, \ldots, s_n\}\!\}$ for $f\{\_\_\_, s_1, \_\_\_, \ldots, \_\_\_, s_n, \_\_\_\}$. Similarly, we may use the notation with double brackets and write $f(\!(s_1, \ldots, s_n)\!)$ for $f(\_\_\_, s_1, \_\_\_, \ldots, \_\_\_, s_n, \_\_\_)$. The matching algorithm can be easily modified to work directly (and more efficiently) on such representations.

Now we show how in this language the query operations given in [20] can be expressed. (This benchmark was used to compare five XML query languages in [3].) The case study is that of a car dealer office, with documents from different auto dealers and brokers. The `manufacturer` documents list the manufacturers name, year, and models with their names, front rating, side rating, and rank; the `vehicle` documents list the vendor, make, year, color and price. We consider XML data of the form:

```
<manufacturer>
        <mn-name>Mercury</mn-name>
        <year>1999</year>
        <model>
           <mo-name>Sable LT</mo-name>
           <front-rating>3.84</front-rating>
           <side-rating>2.14</side-rating>
           <rank>9</rank>
        </model>
        <model>...</model>
        ...
</manufacturer>
```

while the dealers and brokers publish information in the form

```
<vehicle>
        <vendor>Scott Thomason</vendor>
        <make>Mercury</make>
        <model>Sable LT</model>
        <year>1999</year>
        <color>metallic blue</color>
        <option opt="sunroof"/>
        <option opt="A/C"/>
        <option opt="lthr seats"/>
        <price>26800</price>
</vehicle>.
```

Translating the data into our syntax is pretty straightforward. For instance, the manufacturer element can be written as:

$$manufacturer(mn\text{-}name(Mercury), year(1999),\\ model(mo\text{-}name(SableLT), front\text{-}rating(3.84), side\text{-}rating(2.14), rank(9))).$$

The query operations and their encoding in our syntax are given below.

Selection and Extraction: We want to select and extract <`manufacturer`> elements where some <`model`> has <`rank`> less or equal to 10:

$$\_(\!(manufacturer(\overline{x}_1, model(\overline{y}_1, rank(x), \overline{y}_2), \overline{x}_2))\!)\\ \rightarrow manufacturer(\overline{x}_1, model(\overline{y}_1, rank(x), \overline{y}_2), \overline{x}_2) \texttt{ if } x \leq 10.$$

Reduction: From the <manufacturer> elements, we want to drop those <model> sub-elements whose <rank> is greater than 10. We also want to elide the <front-rating> and <side-rating> elements from the remaining models.

$$\begin{aligned}&\_\,(\!(manufacturer(\overline{x}_1,\\ &\qquad model(\overline{y}_1, front\text{-}rating(\_\,), side\text{-}rating(\_\,), rank(x), \overline{y}_2), \overline{x}_2)\!)\\ &\to manufacturer(\overline{x}_1, model(\overline{y}_1, rank(x), \overline{y}_2), \overline{x}_2)\ \texttt{if}\ \ x \leq 10.\end{aligned}$$

Joins: We want to generate pairs of <manufacturer> and <vehicle> elements where <mn-name>=<make>, <mo-name>=<model>, and <year>=<year>.

$$\begin{aligned}&\_\,\{\!\{manufacturer(\overline{x}_1, mn\text{-}name(x_1), \overline{x}_2, year(x_2), \overline{x}_3, \overline{C}(mo\text{-}name(y_1)), \overline{x}_4),\\ &\quad vehicle(\overline{z}_1, make(x_1), \overline{z}_2, model(y_1), \overline{z}_3, year(x_2), \overline{z}_4)\}\!\}\\ &\to pair(\\ &\quad manufacturer(\overline{x}_1, mn\text{-}name(x_1), \overline{x}_2, year(x_2), \overline{x}_3, \overline{C}(mo\text{-}name(y_1)), \overline{x}_4),\\ &\quad vehicle(\overline{z}_1, make(x_1), \overline{z}_2, model(x_2), \overline{z}_3, year(y_1), \overline{z}_4)).\end{aligned}$$

Restructuring: We want our query to collect <car> elements listing their make, model, vendor, rank, and price, in this order:

$$\begin{aligned}&\_\,\{\!\{vehicle(\!(vendor(y_1), make(y_2), model(y_3), year(y_4), price(y_5))\!),\\ &\qquad manufacturer(\!(\overline{C}(rank(x_1)))\!)\}\!\}\\ &\to car(make(y_2), model(y_3), vendor(y_1), rank(x_1), price(y_5)).\end{aligned}$$

Hence, all these operations can be easily expressed in our framework.

At the end of this section we give an example how to extract elements from an XML document that do not meet certain requirements (e.g., miss certain information). Such problems arise in web site verification tasks discussed in [1].

We use the data from [1]. Assume that a web site is given in the form of the following term:

website(members(member(name(mario),surname(rossi),status(professor)),
member(name(franca),surname(bianchi),status(technician)),
member(name(anna),surname(gialli),status(professor)),
member(name(giulio),surname(verdi),status(student))),
hpage(name(mario),surname(rossi),phone(3333),status(professor),
hobbies(hobby(reading),hobby(gardening))),
hpage(name(franca),surname(bianchi),status(technician),phone(5555)),
hpage(name(anna),surname(gialli),status(professor),phone(4444),
teaching(course(algebra))),
pubs(pub(name(mario),surname(rossi),title(blahblah1),year(2003)),
pub(name(anna),surname(gialli),title(blahblah1),year(2002)))).

The task is to find those home pages of professors which miss the teaching information. We formulate the question as the following query:

$$\_(\!(\mathit{hpage}(\overline{x}, \mathit{status}(\mathit{professor}), \overline{y})\!) \to \mathit{hpage}(\overline{x}, \mathit{status}(\mathit{professor}), \overline{y})$$
$$\texttt{if } \_\_\_ (\mathit{teaching}(\_\_\_)) \not\ll \mathit{hpage}(\overline{x}, \mathit{status}(\mathit{professor}), \overline{y}).$$

The condition in the query requires the term $\_\_\_ (\mathit{teaching}(\_\_\_))$ not to match $\mathit{hpage}(\overline{x}, \mathit{status}(\mathit{professor}), \overline{y})$. In $\_\_\_ (\mathit{teaching}(\_\_\_))$, the first anonymous variable is the anonymous context variable, and the second one is the anonymous sequence variable. Since context sequence matching is decidable, the condition can be effectively checked. The result of the query is

hpage(name(mario),surname(rossi),phone(3333),status(professor),
hobbies(hobby(reading),hobby(gardening))).

6 Conclusions

Context sequence matching is a matching for flexible arity terms that contain context and sequence variables. These two kinds of variables allow matching to explore terms (represented as trees) in two orthogonal directions: in depth (context) and in breadth (sequence) and, thus, to get more freedom in selecting subterms. Besides context and sequence variables, terms may contain function and individual variables that allow matching to make a single step in depth or in breadth. We developed a rule-based algorithm for context sequence matching and proved its soundness, termination and completeness. Moreover, we showed that regular restrictions can be easily incorporated in the rule-based matching framework extending the algorithm with the rules for matching regular expressions both for context and sequence variables. We showed soundness, termination and completeness of such an extension.

In our opinion, context sequence matching can serve as a computational mechanism for a declarative, rule-based language to query and transform XML, or to specify and verify web sites. The ability of traversing trees both in depth and in breadth would give such a language the advantages from both path-based and pattern-based languages. It would easily support, for instance, a wide range of queries (selection and extraction, reduction, negation, restructuring, combination), parent-child and sibling relations and their closures, access by position, unordered matching, order-preserving result, partial and total queries, optionally, construction, multiple results, and other properties. We expect such a language to have a clean declarative semantics (rule-based paradigm) and to be visualizable.

References

[1] M. Alpuente, D. Ballis, and M. Falaschi. A rewriting-based framework for web sites verification. *Electronic Notes on Theoretical Computer Science*, 2004. To

appear.

[2] H. Boley. *A Tight, Practical Integration of Relations and Functions*, volume 1712 of *LNAI*. Springer, 1999.

[3] A. Bonifati and S. Ceri. Comparative analysis of five XML query languages. *ACM SIGMOD Record*, 29(1):68–79, 2000.

[4] F. Bry and S. Schaffert. Towards a declarative query and transformation language for XML and semistructured data: Simulation unification. In *Proc. of International Conference on Logic Programming (ICLP)*, number 2401 in LNCS, Copenhagen, Denmark, 2002. Springer.

[5] B. Buchberger and A. Crăciun. Algorithm synthesis by lazy thinking: Examples and implementation in Theorema. In *Proc. of the Mathematical Knowledge Management Symposium*, volume 93 of *Electronic Notes on Theoretical Computer Science*, pages 24–59, 2003.

[6] B. Buchberger, C. Dupré, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, and W. Windsteiger. The Theorema project: A progress report. In M. Kerber and M. Kohlhase, editors, *Proc. of Calculemus'2000 Conference*, pages 98–113, 2000.

[7] J. Clark and S. DeRose, editors. *XML Path Language (XPath) Version 1.0*. W3C, 1999. Available from: `http://www.w3.org/TR/xpath/`.

[8] J. Coelho and M. Florido. CLP(Flex): Constraint logic programming applied to XML processing. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE. Proc. of Confederated Int. Conferences*, volume 3291 of *LNCS*, pages 1098–1112. Springer, 2004.

[9] H. Comon. Completion of rewrite systems with membership constraints. Part I: Deduction rules. *J. Symbolic Computation*, 25(4):397–419, 1998.

[10] H. Comon. Completion of rewrite systems with membership constraints. Part II: Constraint solving. *J. Symbolic Computation*, 25(4):421–453, 1998.

[11] T. Furche, F. Bry, S. Schaffert, R. Orsini, I. Horroks, M. Kraus, and O. Bolzer. Survey over existing query and transformation languages. Available from: `http://rewerse.net/deliverables/i4-d1.pdf`, 2004.

[12] M. L. Ginsberg. The MVL theorem proving system. *SIGART Bull.*, 2(3):57–60, 1991.

[13] M. Hamana. Term rewriting with sequences. In: Proc. of the First Int. *Theorema* Workshop. Technical report 97–20, RISC, Johannes Kepler University, Linz, Austria, 1997.

[14] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *Proc. of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 453–462. IEEE Computer Society Press, 1995.

[15] H. Hosoya and B. Pierce. Regular expression pattern matching for XML. *J. Functional Programming*, 13(6):961–1004, 2003.

[16] T. Kutsia. *Solving and Proving in Equational Theories with Sequence Variables and Flexible Arity Symbols.* PhD thesis, Johannes Kepler University, Linz, Austria, 2002.

[17] T. Kutsia. Unification with sequence variables and flexible arity symbols and its extension with pattern-terms. In J. Calmet, B. Benhamou, O. Caprotti, L. Henocque, and V. Sorge, editors, *Artificial Intelligence, Automated Reasoning and Symbolic Computation. Proc. of Joint AISC'2002 – Calculemus'2002 Conference*, volume 2385 of *LNAI*, pages 290–304. Springer, 2002.

[18] T. Kutsia. Solving equations involving sequence variables and sequence functions. In B. Buchberger and J. A. Campbell, editors, *Artificial Intelligence and Symbolic Computation. Proc. of AISC'04 Conference*, volume 3249 of *LNAI*, pages 157–170. Springer, 2004.

[19] J. Levy and M. Villaret. Linear second-order unification and context unification with tree-regular constraints. In L. Bachmair, editor, *Proc. of the 11th Int. Conference on Rewriting Techniques and Applications (RTA'2000)*, volume 1833 of *LNCS*, pages 156–171. Springer, 2000.

[20] D. Maier. Database desiderata for an XML query language. Available from: `http://www.w3.org/TandS/QL/QL98/pp/maier.html`, 1998.

[21] M. Marin. Introducing a rule-based programming system ρLog. Available from: `http://www.score.is.tsukuba.ac.jp/~mmarin/RhoLog/`, 2004.

[22] M. Schmidt-Schauß. A decision algorithm for stratified context unification. *J. Logic and Computation*, 12(6):929–953, 2002.

[23] M. Schmidt-Schauß and K. U. Schulz. Solvability of context equations with two context variables is decidable. *J. Symbolic Computation*, 33(1):77–122, 2002.

[24] M. Schmidt-Schauß and J. Stuber. On the complexity of linear and stratified context matching problems. Research Report 4923, INRIA-Lorraine, France, 2003.

[25] The RTA List of Open Problems. Problem No. 90. Available from: `http://www.lsv.ens-cachan.fr/rtaloop/problems/90.html`.

[26] World Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.0. Second edition. Available from: `http://www.w3.org/`, 1999.

Slicing XML Documents

Josep Silva [1]

DSIC, Technical University of Valencia
Valencia, Spain

Abstract

Program slicing is a well-known technique to extract the program statements that (potentially) affect the values computed at some point of interest. In this work, we introduce a novel slicing method for XML documents. Essentially, given an XML document (which is valid w.r.t. some DTD), we produce a new XML document (a *slice*) that contains the relevant information in the original XML document according to some criterion. Furthermore, we also output a new DTD such that the computed slice is valid w.r.t. this DTD. A prototype implementation of the XML slicer has been undertaken.

Key words: XML, DTD, Program Slicing.

1 Introduction

The increasing complexity of websites demands for tools which are able to aid Web designers in their construction and maintenance. Systematic, formal approaches can bring many benefits to quality website development, giving support for automated website transformations.

One of the most widely used program transformations is program slicing. It consists of a decomposition technique for the extraction of those program statements that (potentially) affect the values computed at some point of interest. Program slicing was originally introduced by Weiser [6] and has now many applications such as debugging, code understanding, program specialization, etc. See [4] for a survey.

In this work, we present a slicing technique which is applicable to XML documents [1]. The result of slicing an XML document is a new XML document (a *slice*) composed by those parts of the original document satisfying some criterion (the *slicing criterion*). We also produce a new DTD such that the computed slice is well-formed and valid with respect to this DTD.

[1] Email: jsilva@dsic.upv.es

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

```
<PersonalInfo>
   <Contact>
      <Status> Professor </Status>
      <Name> Ryan </Name>
      <Surname> Gibson <Surname>
   </Contact>
   <Teaching>
      <Subject>
         <Name> Logic </Name>
         <Sched> Mon/Wed 16-18 </Sched>
         <Course> 4-Mathematics </Course>
      </Subject>
      <Subject>
         <Name> Algebra </Name>
         <Sched> Mon/Tur 11-13 </Sched>
         <Course> 3-Mathematics </Course>
      </Subject>
      ...
   </Teaching>
   <Research>
      <Project
         name = "SysLog"
         year = "2003-2004"
         budget = "16000€" />
      ...
   </Research>
</PersonalInfo>
```

(a) XML document with the contents of a webpage

```
<!ELEMENT PersonalInfo (Contact,
                        Teaching,
                        Research)>
<!ELEMENT Contact (Status,
                   Name,
                   Surname)>
<!ELEMENT Status ANY>
<!ELEMENT Name ANY>
<!ELEMENT Surname ANY>
<!ELEMENT Teaching (Subject+)>
<!ELEMENT Subject (Name,
                   Sched,
                   Course)>
<!ELEMENT Sched ANY>
<!ELEMENT Course ANY>
<!ELEMENT Research (Project*)>
<!ELEMENT Project ANY>
<!ATTLIST Project
   name CDATA #REQUIRED
   year CDATA #REQUIRED
   budget CDATA #IMPLIED
>
```

(b) DTD defining personal info

Fig. 1. XML document (a) and its associated DTD (b)

2 XML

XML [1] was developed by an XML Working Group formed under the auspices of the World Wide Web Consortium (W3C) in 1996. XML documents are made up of basic units called "elements" according to a set of restrictions specified on an independent specification called *Document Type Definition* (DTD). Figure 1 shows an example of XML (a) and DTD (b) documents with some staff information from a university (for the time being, the reader can ignore the differences between black and grey text). An XML document is "well-formed" if it conforms to the standard XML syntax rules described in [1]. A "valid" XML document is a well-formed XML document, which also conforms to the rules of a DTD.

XML documents can easily be transformed to a webpage by means of the *Extensible Stylesheet Language Transformations* (XSLT) [2]. It specifies the presentation of XML documents by describing how their instances are transformed to an XML document that uses a formatting vocabulary, such as (X)HTML or XSL-FO, that can be interpreted by any standard browser. For instance, the XML document in Figure 1 can be automatically translated to a standard webpage showing the information structured with tables and colored

text. Different slices of an XML document could be used to produce (keeping the same XSLT transformation) distinct versions of the generated webpage.

3 The Slicing Technique

Program slicing techniques have been typically based on a data structure called *Program Dependence Graph* (PDG). In the case of XML, we take advantage of the tree-like structure of XML documents and DTDs. Despite the fact that DTDs can be graphs, we only consider trees without lost of generality, since they can easily be converted to trees by duplicating those elements producing cycles. Every vertex in the tree represents a single element of the XML document of type 'ELEMENT' (we refer the reader to [1] for a more detailed explanation about XML elements) containing all its attributes. Edges represent aggregation relations between elements.

In our setting, the slicing criterion consist of a set of elements from one of the trees (the XML or its associated DTD). As a consequence, we distinguish between two types of slicing:

DTD slicing: Given a set of elements from a DTD, we extract from this DTD those elements which are strictly necessary to maintain the tree structure, i.e., all the DTD elements that are in the path from the root to any of the elements in the slicing criterion.
Once the DTD slice is produced, the set of elements in the slice is used as a slicing criterion in order to produce the associated XML slice (see below).

Example 3.1 Consider the XML and DTD documents in Figure 1. Let the set containing the single element "Course" be the slicing criterion. The slices computed for the two documents are highlighted in Figure 1 with black color.

XML slicing: Analogously to DTD slicing, given a set of XML elements, we extract from the XML document those elements which are in the path from the root to any of the elements in the slicing criterion.
Note that this criterion could be more fine-grained than the previous one. While a slicing criterion in a DTD selects a type of elements, a slicing criterion in an XML document can select only some particular instances of this type.

Example 3.2 Consider the XML and DTD documents in Figure 1. Let the element "3-Mathematics" of type "Course" be the slicing criterion. The slice computed for the DTD is exactly the same than the one in Example 3.1. However, the XML slice only contains the information related to the "Algebra" subject, being the information related to the "Logic" subject removed.

The slicer proceeds as follows:

1) Compute the set of relevant elements with respect to the slicing criterion

2) The computed set is modified in order to satisfy all the restrictions of the DTD (for instance, the * and + relations). Attributes are also included

3) The result is used as a slicing criterion for the associated DTD/XML

Let $\mathcal{D}$ be a well-formed DTD, $\mathcal{X}$ an XML document valid with respect to $\mathcal{D}$, $\mathcal{D}'$ and $\mathcal{X}'$ two slices of $\mathcal{D}$ and $\mathcal{X}$ respectively, computed with a DTD-slicing criterion $\mathcal{C}$, and $\mathcal{D}''$ and $\mathcal{X}''$ two slices of $\mathcal{D}$ and $\mathcal{X}$ respectively, computed with a XML-slicing criterion $\mathcal{C}'$, we have proven the following properties:

a) $\mathcal{D}'$ is well-formed and $\mathcal{X}'$ is valid with respect to $\mathcal{D}'$

b) $\mathcal{D}''$ is well-formed and $\mathcal{X}''$ is valid with respect to $\mathcal{D}''$
If all the elements in $\mathcal{C}'$ are of one of the types in $\mathcal{C}$, then

c) $\mathcal{D}' = \mathcal{D}''$

d) $\mathcal{X}''$ is a subtree of $\mathcal{X}'$

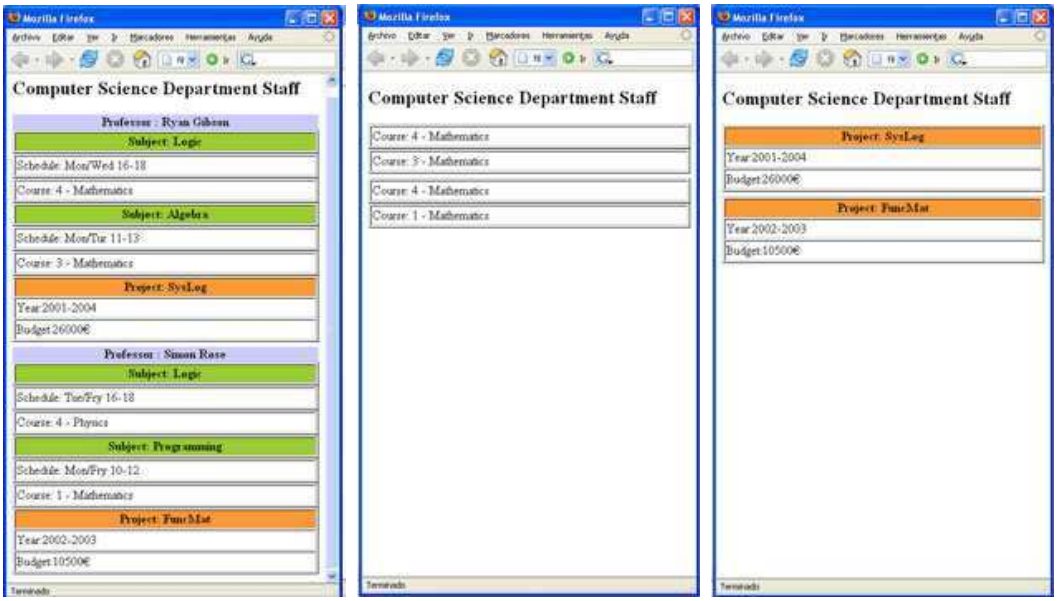
4 Implementation

We have implemented our prototype in Haskell [3]. Haskell provides us a formal basis with many advantages for the manipulation of XML documents such as the *HaXml* library [5]. It allows us to automatically translate XML or HTML documents into a Haskell representation. In particular, we use the following data structures that can represent any XML/HTML document:

```
data Element = Elem Name [Attribute] [Content]
data Attribute = (Name, Value)
data Content = CElem Element
             | CText String
```

One of the main possible applications of XML slicing is webpages slicing, because XML slices have a direct representation in webpages, producing slices of the original webpages. For instance, Figure 2 (a) shows the webpage that is automatically generated with an XSLT file from the XML document of Figure 1. Figure 2 (b) shows the webpage that is automatically generated with the same XSLT file from the XML slice of Figure 1. Figure 2 (c) shows the webpage that is automatically generated with the same XSLT file from the slice of Figure 1 computed with a DTD slicing criterion formed by the "Project" element.

Preliminary results are very encouraging, showing that the slicing technique can bring many benefits when applied in conjunction with XSLT being able to slice complex webpages. The implementation and some other materials are publicly available at: http://www.dsic.upv.es/~jsilva/xml.



(a) WebPage (b) "Courses" Slice (c) "Projects" Slice

Fig. 2. Web Pages automatically generated from the XML in Figure 1

References

[1] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and Franois Yergeau. Extensible markup language (xml) 1.0 (third edition). Available at: http://www.w3.org/TR/REC-xml/, February 2004. W3C Recommendation.

[2] James Clark. Xsl transformations (xslt). Available at: http://www.w3.org/TR/xslt, November 1999. W3C Recommendation.

[3] S. Peyton Jones, editor. *Haskell 98 Language and Libraries: The Revised Report.* Cambridge University Press, 2003.

[4] F. Tip. A Survey of Program Slicing Techniques. *Journal of Programming Languages*, 3:121–189, 1995.

[5] Malcolm Wallace and Colin Runciman. Haskell and XML: Generic combinators or type-based translation? In *Proceedings of the Fourth ACM SIGPLAN International Conference on Functional Programming (ICFP'99)*, volume 34–9, pages 148–159, N.Y., 1999. ACM Press.

[6] M.D. Weiser. Program Slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, 1984.

A Language for Verification and Manipulation of Web Documents

(Extended Abstract)

Luigi Liquori

INRIA, France

Furio Honsell

DIMI, University of Udine, Italy

Rekha Redamalla

Birla Science Center, Adarsh Nagar, Hyderabad, India

Abstract

In this paper we develop the language theory underpinning the logical framework PLF. This language features lambda abstraction with patterns and application via pattern-matching. Reductions are allowed in patterns. The framework is particularly suited as a metalanguage for encoding rewriting logics and logical systems where rules require proof terms to have special syntactic constraints, *e.g.* call-by-value λ-calculus, or term rewriting systems. PLF is a conservative extension of the well-known Edinburgh Logical Framework LF. Because of sophisticated pattern matching facilities PLF is suitable for verification and manipulation of XML documents.

1 Introduction

Since the introduction of Logical Frameworks [4,6]), blending dependent typed λ-calculi with rewriting systems has been a major challenge in the last decades, see [9, 7, 12, 3, 5, 8]). Blending lambda calculi and rewrite rules enhances the usability of the system as a metalanguage for logics. Clearly the expressiveness of the system does not increase, since already the Logical Framework of [6] is a universal language for encoding formal systems. Clearly rewrite systems can provide in many instances much more natural and transparent encodings, thus improving the overall pragmatic usability of the system, and the possibility of automating decision procedures, such as checking and encoding equality.

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

In this paper, we present an uniform framework based on a dependent typed lambda calculus enriched with pattern matching in lambda abstraction, called PLF. In contrast to the simple λ-calculus, the pattern-matching algorithm can either fire a substitution, or stick the computation unless further substitution are provided. The following simple example illustrates the point:

$$M \equiv (\lambda(f\,y).y)\,x$$

is stuck, but

$$(\lambda(f\,x).M)\,(f\,(f\,3)) \mapsto\!\!\!\!\rightarrow_{\beta} 3.$$

Furthermore, for a given pattern inside an abstraction, the user can explicitly discriminate between variables that will be bound in the body, and variables that can be bound in an external context containing the abstraction itself. This freedom is particularly appreciated in patterns that evolve (by reduction or substitution) during execution, *e.g.*

$$(\lambda x{:}A.\lambda P[x].N)\,M \mapsto_{\beta} \lambda P[M].N.$$

PLF extends the subsystem of [1] corresponding to LF in allowing reductions inside patterns. As is well know, since the seminal work of [12], variables in patterns can be bound only if they occur linearly and not actively (*i.e.* not in functional position), hence extra care has to be payed when reductions are allowed in patterns. We presents few simple examples of encoding, which capitalize on patterns, namely Plotkin's call-by-value λ-calculus, and the rewriting calculus (ρ-calculus) [11]. An appendix complete the paper with few classical XML-oriented routines, written in the rewriting calculus, that can be either runned and certified in our PLF. The full metatheory of PLF will appear in a companion paper.

2 The Pattern Logical Framework

We present the syntax of PLF, a logical framework with pattern oriented features. Since patterns occurs as *binders* in abstractions the types of the "matchable" variable in the pattern are decorated in suitable *contexts*, *i.e.* a pattern abstraction has the form $\lambda M{:}\Delta.N$. Therefore, the contexts defining the types of the free variables of these patterns are given explicitly as part of the abstraction. The context Δ can discriminate on variables that are suitable to be matched and variables that are not, the latter are *de facto* considered as constants. This treatment of constants simplifies the presentation of the system since constants are now considered as variables that cannot be bound. This mechanism implies also that some patterns can evolve during reduction via substitution; this operation is rather delicate, but sound, since a pattern, considered with a suitable context recording the types of the free variables, is enforced to preserve suitable good properties, under restriction.

2.1 PLF*'s Terms*

The first definition introduces the pseudo-syntax for kinds, families, objects and contexts.

Definition 2.1 [PLF's Pseudo-syntax]

$$\Gamma, \Delta \in \mathcal{C} \qquad \Gamma \ ::= \emptyset \mid \Gamma, x{:}K \mid x{:}A$$

$$K \in \mathcal{K} \qquad K \ ::= \mathsf{Type} \mid \Pi M{:}\Delta.K \mid \lambda M{:}\Delta.K \mid K\,M$$

$$A, B, C \in \mathcal{F} \qquad A \ ::= x \mid \Pi M{:}\Delta.A \mid \lambda M{:}\Delta.A \mid A\,M$$

$$M, N, P, Q \in \mathcal{O} \qquad M \ ::= x \mid \lambda M{:}\Delta.M \mid M\,M$$

As usual, application associates to the right. Let "T" range over any term in the calculus, and let the symbol "$\checkmark$" range over the set $\{\lambda, \Pi\}$. To ease the notation, we write $\checkmark x{:}T_1.T_2$ for $\checkmark x{:}(x{:}T_1).T_2$ in case of a variable-pattern. Intuitively the context Δ in $\lambda M{:}\Delta.N$ contains the type declarations of some (but not all) of the free variables appearing in the pattern M. These variables are bound in the (pattern and body of the) abstraction and the reduction of an abstraction application strongly depends on them, all the other variables being handled as constants. The free variables of M not declared in Δ are not bound in N but can be bound outside the scope of the abstraction itself. For example, in the abstraction

$$\lambda(x\,y\,z){:}(y{:}w, z{:}w).M$$

the y and z variables (of type w) are bound in M, while the x variable (not declared in the context) is considered as free (*i.e.* it is *de facto* handled as a constant). As in ordinary systems dealing with dependent types, we suppose that in the context $\Gamma, x{:}T$, the variable x does not appear free in Γ, and T. Some remarks to understand this syntax are in order.

Remark 2.2

(i) *In case of variable patterns, we recover the syntax of* LF.

(ii) *The above pseudo-syntax does not enforce any particular shape to patterns in abstractions, but it is well known that allowing in patterns non-linearity, i.e. multiple occurrences of the same variable, e.g.* $\lambda(x\,y\,y){:}(y{:}T_1).T_2$*, and active variables, i.e. variables in functional position, e.g.* $\lambda(x\,y){:}(x{:}T_1).T_2$*, breaks confluence and subject reduction. This anomaly will be repaired by enforcing the syntax with suitable restrictions.*

(iii) *A context* Γ *can be always split in three parts* $\Gamma_{\mathsf{T}}, \Gamma_{\mathsf{S}}, \Gamma_{\mathsf{V}}$*, where* Γ_{T} *contains type definitions, i.e. families variables,* Γ_{S} *contains the type declarations of some constants objects, i.e. variables that cannot be abstracted, and* Γ_{V} *contains the type declarations of the object variables which can be abstracted.*

The following predicates characterize the set of *legal/good* patterns w.r.t. a given context.

Definition 2.3 [Safe Patterns]
A safe pattern is characterized by having no free variables in functional position, and by its linearity. This can be carefully formalized, but for the purpose of this paper we just write

$$\mathsf{SPC}(M;\mathbb{V}) \triangleq \neg\mathsf{APC}(M;\mathbb{V}) \wedge \mathsf{LPC}(M;\mathbb{V}).$$

This definition can be extended poitwise to substitutions, families and kinds.

Given the above restrictions on objects occurring in patterns, we can thus define a *safe/legal* PLF *syntax* as follows:

Definition 2.4 [PLF's Safe Syntax]
A term in the PLF syntax is safe if any subterm $\checkmark M{:}\Delta.T$ occurring in it is such that

$$\mathsf{SPC}(M;\mathsf{Dom}(\Delta)) \wedge \mathsf{SPC}(T;\mathsf{Dom}(\Delta))$$

In the rest of the paper we shall consider only safe terms. The definition of free variables needs to be customized as follows.

Definition 2.5 [Free Variables]
The set Fv of free variables is given by:

$$\mathsf{Fv}(\checkmark T_1{:}\Delta.T_2) \triangleq (\mathsf{Fv}(T_1) \cup \mathsf{Fv}(T_2) \cup \mathsf{Fv}(\Delta)) \setminus \mathsf{Dom}(\Delta)$$

$$\mathsf{Fv}(\Delta, x{:}T) \triangleq \mathsf{Fv}(\Delta) \cup (\mathsf{Fv}(T) \setminus \mathsf{Dom}(\Delta))$$

The set Bv of bound variables of a term is the set of variables in the term which are not free. Since we work modulo α-conversion, we suppose that all bound variables of a term have different names and therefore, the domains of all contexts are distinct. A suitable, intuitive, (re)definition of simultaneous substitution application (denoted by θ) to deal with the new forms of abstraction is assumed.

2.2 Matching and Operational Semantics

PLF features pattern abstractions whose application requires solving matching problems. The next two definitions introduce the notion of matching systems and matching algorithm. Both are an easy modification of the ones presented in [1]. The algorithm is first-order, hence decidable.

Definition 2.6 [Matching System]

(i) A *matching system*

$$\mathsf{T} \triangleq \bigwedge_{i=0\ldots n} M_i <^{\mathbb{V}}_{\mathbb{W}_i} N_i$$

is a conjunction of matching equations. The set $\mathbb{V}$ records the name of the free variables that are matchable, while the sets $\mathbb{W}_i$ record the name of bound variables appearing in abstractions which cannot be matched.

(ii) A matching system T is solved by the substitution θ if for all $i = 0 \dots n$, we have that $M_i\theta \equiv N_i$.

(iii) A matching system T is in normal form when it has the form

$$\mathsf{T} \triangleq \bigwedge_{i=0\dots n} x_i <^{\mathbb{V}}_{\mathbb{W}_i} N_i$$

(iv) A matching system in normal form is solved and produces the substitution $[N_1/x_1 \cdots N_n/x_n]$, if the following conditions are satisfied (otherwise the matching fails)
   (a) for all $h, k = 0 \dots n$, if $x_h \equiv x_k$ then $N_h \equiv N_k$.
   (b) for all $i = 0 \dots n$, if $x_i \in \mathbb{W}_i$, then $N_i \equiv x_i$.
   (c) for all $i = 0 \dots n$, if $\mathsf{Fv}(N_i) \cap \mathbb{W}_i \neq \emptyset$, then $N_i \equiv x_i$.
   (d) for all $i = 0 \dots n$, if $x_i \notin \mathbb{V}$, then $N_i \equiv x_i$.
   (e) for all $x \in \mathbb{V}$, there exists $x <^{\mathbb{V}}_{\mathbb{W}_i} \in \mathsf{T}$.

   Let solve be a function that returns a substitution if a matching system in normal form is solvable, and fails otherwise.

Definition 2.7 [Matching Algorithm $\mathcal{A}lg$]

(i) The reduction $\rightsquigarrow$ is the compatible relation induced by the following two rules:

$$\frac{\mathbb{W} \triangleq \mathbb{U} \cup \mathsf{Dom}(\Delta)}{\checkmark M_1{:}\Delta.N_1 <^{\mathbb{V}}_{\mathbb{U}} \checkmark M_2{:}\Delta.N_2 \rightsquigarrow M_1 <^{\mathbb{V}}_{\mathbb{W}} M_2 \wedge N_1 <^{\mathbb{V}}_{\mathbb{W}} N_2}\ (\mathsf{Lbd/Prod})$$

$$\frac{\mathbb{W} \triangleq \mathbb{U} \cup \mathsf{Dom}(\Delta)}{M_1\, N_1 <^{\mathbb{V}}_{\mathbb{U}} M_2\, N_2 \rightsquigarrow M_1 <^{\mathbb{V}}_{\mathbb{W}} M_2 \wedge N_1 <^{\mathbb{V}}_{\mathbb{W}} N_2}\ (\mathsf{Appl})$$

(ii) The reduction $\rightsquigarrow^*$ is defined as the reflexive and transitive closure of $\rightsquigarrow$. Let $\mathsf{normalize}$ be the function that reduces a matching system in normal form, or fails.

(iii) $\mathcal{A}lg(M \cdot N \cdot \mathbb{V})$ is defined as follows:
   1 $\mathsf{T} := \mathsf{normalize}(M <^{\mathbb{V}}_{\emptyset} N)$;
   2 $\mathsf{if}\ \mathsf{T} \neq \mathsf{fail}\ \mathsf{then}\ \mathsf{return}\ \mathsf{solve}(\mathsf{T})$

The matching algorithm is clearly terminating (since all rules decrease the size of terms), deterministic (no critical pairs), and works modulo α-conversion and Barendregt's hygiene-convention. We write θ for the successful output of $\mathcal{A}lg(M \cdot N \cdot \mathbb{V})$. The next definition introduces the classical notions of one-step, many-steps, and congruence relation of $\rightarrow_\beta$.

Definition 2.8 [One/Many-Steps, Congruence]
Let $\theta = \mathcal{A}lg(P \cdot N \cdot \mathsf{Dom}(\Delta))$.

(i) The top-level rules are

$$(\beta_{\mathsf{p}}\text{–Obj}) \quad (\lambda P{:}\Delta.M)\, N \to_\beta M\theta$$

$$(\beta_{\mathsf{p}}\text{–Fam}) \quad (\lambda P{:}\Delta.A)\, N \to_\beta A\theta$$

$$(\beta_{\mathsf{p}}\text{–Kinds}) \quad (\lambda P{:}\Delta.K)\, N \to_\beta K\theta$$

(ii) Let $\mathsf{C}[-]$ denote a pseudo-context with a "single hole" inside, it is defined as follows

$$\mathsf{C}[-] ::= [-] \mid \mathsf{C}[-]\,T \mid T\,\mathsf{C}[-] \mid \checkmark M{:}\Delta.\mathsf{C}[-] \mid \checkmark M{:}\mathsf{C}[-].T \mid x{:}\mathsf{C}[-] \mid \mathsf{C}[-], x{:}T$$

and let $\mathsf{C}[T]$ be the result of filling the hole with the term T. The one-step evaluation $\mapsto_\beta$ is defined by the following inference rules

$$\frac{T_1 \to_\beta T_2}{\mathsf{C}[T_1] \mapsto_\beta \mathsf{C}[T_2]}\,(\mathsf{Ctx}) \qquad \frac{\begin{array}{ll} M \to_\beta N & \mathsf{SPC}(N; \mathsf{Dom}(\Delta)) \\ \mathsf{Fv}(M) \cap \mathsf{Dom}(\Delta) = \mathsf{Fv}(N) \cap \mathsf{Dom}(\Delta) & \end{array}}{\checkmark M{:}\Delta.T \mapsto_\beta \checkmark N{:}\Delta.T}\,(\mathsf{Ctx}^{\checkmark})$$

The intended meaning of the (Ctx) rule is the usual one. Rule ($\mathsf{Ctx}^{\checkmark}$) forbids $\mathsf{K}\beta$-reductions in patterns enforces the safe pattern condition in both redex and the reduced term.

(iii) The many-step evaluation $\mapsto\!\!\!\!\to_{\beta_{\mathsf{p}}}$ and the congruence relation $=_\beta$ are respectively defined as the reflexive-transitive and reflexive-symmetric-transitive closure of $\mapsto_{\beta_{\mathsf{p}}}$.

2.3 PLF*'s Type System*

PLF involves type judgments of the following shape:

$$\vdash \Gamma \qquad \Gamma \vdash K \qquad \Gamma \vdash A : \mathsf{Type} \quad \Gamma \vdash M : A$$

The type system rules of PLF are presented in Figure 1. Some rules are quite similar to the ones of the classical logical framework, but others deserve a brief explanation:

- The (F·Abs), (O·Abs) rules deal with λ-abstractions in which we bind over (non trivial) patterns; this rule requires that the pattern and the body of the abstraction are typable in the extended context Γ, Δ.
- The (F·Appl), (O·Appl) rules, give a type to an application; this type contrasts with classical type systems which utilize meta-substitutions. Suitable applications of (F·Conv), (O·Conv) rule can reduce this type.
- The (K·Pi), (F·Pi) rules, give a type to a kind and family products. As for λ-abstractions we require that the pattern and the body of the abstraction are typable in the extended context Γ, Δ.

Dependent Syntax

$$\Gamma ::= \emptyset \mid \Gamma, x{:}K \mid x{:}A$$

$$K ::= \mathsf{Type} \mid \Pi M{:}\Delta.K \mid \lambda M{:}\Delta.K \mid K\,M$$

$$A ::= x \mid \Pi M{:}\Delta.A \mid \lambda M{:}\Delta.A \mid A\,M$$

$$M ::= x \mid \lambda M{:}\Delta.M \mid M\,M$$

Safe Syntax

$$\checkmark M{:}\Delta.T \in \mathcal{C}, \mathcal{K}, \mathcal{F}, \mathcal{O} \implies \mathsf{SPC}(M;\Delta) \wedge \mathsf{SPC}(T;\Delta)$$

Contexts rules

$$\frac{}{\vdash \emptyset}\ (\mathsf{C{\cdot}Empty})$$

$$\frac{\vdash \Gamma \quad \Gamma \vdash K \quad x \notin \mathsf{Dom}(\Gamma)}{\vdash \Gamma, x{:}K}\ (\mathsf{C{\cdot}Kind})$$

$$\frac{\vdash \Gamma \quad \Gamma \vdash A : \mathsf{Type} \quad x \notin \mathsf{Dom}(\Gamma)}{\vdash \Gamma, x{:}A}\ (\mathsf{C{\cdot}Type})$$

Kind rules

$$\frac{\vdash \Gamma}{\Gamma \vdash \mathsf{Type}}\ (\mathsf{K{\cdot}Type})$$

$$\frac{\Gamma, \Delta \vdash M : A \quad \Gamma, \Delta \vdash K}{\Gamma \vdash \Pi M{:}\Delta.K}\ (\mathsf{K{\cdot}Pi})$$

$$\frac{\Gamma, \Delta \vdash M : A \quad \Gamma, \Delta \vdash K}{\Gamma \vdash \lambda M{:}\Delta.K}\ (\mathsf{K{\cdot}Abs})$$

$$\frac{\Gamma \vdash \Pi M{:}\Delta.K \quad \Gamma, \Delta \vdash M : A \quad \Gamma \vdash N : A}{\Gamma \vdash (\lambda M{:}\Delta.K)\,N}\ (\mathsf{K{\cdot}Appl})$$

Families rules

$$\frac{\vdash \Gamma \quad x{:}K \in \Gamma}{\Gamma \vdash x : K}\ (\mathsf{F{\cdot}Var})$$

$$\frac{\Gamma, \Delta \vdash M : B \quad \Gamma, \Delta \vdash A : \mathsf{Type}}{\Gamma \vdash \Pi M{:}\Delta.A : \mathsf{Type}}\ (\mathsf{F{\cdot}Pi})$$

$$\frac{\Gamma, \Delta \vdash M : B \quad \Gamma, \Delta \vdash A : K}{\Gamma \vdash \lambda M{:}\Delta.A : \Pi M{:}\Delta.K}\ (\mathsf{F{\cdot}Abs})$$

$$\frac{\Gamma \vdash A : \Pi N{:}\Delta.K \quad \Gamma, \Delta \vdash N : B \quad \Gamma \vdash M : B}{\Gamma \vdash A\,M : (\lambda N{:}\Delta.K)\,M}\ (\mathsf{F{\cdot}Appl})$$

$$\frac{\Gamma \vdash A : K' \quad \Gamma \vdash K \quad K =_\beta K'}{\Gamma \vdash A : K}\ (\mathsf{F{\cdot}Conv})$$

Object rules

$$\frac{\vdash \Gamma \quad x{:}A \in \Gamma}{\Gamma \vdash x : A}\ (\mathsf{O{\cdot}Var})$$

$$\frac{\Gamma, \Delta \vdash M : B \quad \Gamma, \Delta \vdash N : A}{\Gamma \vdash \lambda M{:}\Delta.N : \Pi M{:}\Delta.A}\ (\mathsf{O{\cdot}Abs})$$

$$\frac{\Gamma \vdash M : \Pi P{:}\Delta.A \quad \Gamma, \Delta \vdash P : B \quad \Gamma \vdash N : B}{\Gamma \vdash M\,N : (\lambda P{:}\Delta.A)\,N}\ (\mathsf{O{\cdot}Appl})$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash B : \mathsf{Type} \quad A =_\beta B}{\Gamma \vdash M : B}\ (\mathsf{O{\cdot}Conv})$$

Figure 1. PLF's Type Rules

3 Examples

Readers familiar with the activity of encoding systems in LF will surely appreciate the usability of this metalanguage and will play with it providing interesting examples. For lack of space we will provide only few examples, which illustrate how patterns can safely replace sub-categories and coercions in the encoding of syntax. Moreover, the sophisticated shapes of patterns and the built-ins pattern matching facilities make PLF (and its extensions) suitable for modeling regular languages, theory of functions, and term rewriting systems that are the engine of many XML-manipulations. More interesting examples can be devised in encoding the proof theory, such as, in modal logic. An appendix presents some classical XML-oriented routines, written in the rewriting calculus [11], that can be either runned and certified in our PLF.

Syntactic Categories

o : Type

Operations (we short $\checkmark\mathsf{C}[x]{:}(x{:}o)$ for $\checkmark\mathsf{C}[x^o]$)

$\mathsf{Val} \ : \ o \to o \qquad \mathsf{Lam} \ : \ \Pi f{:}(\Pi(\mathsf{Val}\ x^o).\ o).\ o \qquad \mathsf{App} \ : \ o \to o \to o$

Judgments

$=$: $o \to o \to$ Type

Λ encoding $[\![-]\!] : \Lambda_{\mathsf{v}} \Rightarrow \mathsf{PLF}$

$[\![x]\!] \triangleq \mathsf{Val}\ x \quad [\![\lambda x.M]\!] \triangleq \mathsf{Val}\ (\mathsf{Lam}\ (\lambda(\mathsf{Val}\ x^o).[\![M]\!])) \quad [\![M\,N]\!] \triangleq \mathsf{App}\ [\![M]\!] [\![N]\!]$

Axioms and Rules

$\mathsf{Eq}_{\mathsf{refl}}$: $\Pi x{:}o.\ x = x$

$\mathsf{Eq}_{\mathsf{symm}}$: $\Pi x{:}o.\ \Pi y{:}o.\ (x = y) \to (y = x)$

$\mathsf{Eq}_{\mathsf{trans}}$: $\Pi x{:}o.\ \Pi y{:}o.\ \Pi z{:}o.\ (x = y) \to (y = z) \to (x = z)$

$\mathsf{Eq}_{\mathsf{ctx}}$: $\Pi x{:}o.\ \Pi y{:}o.\ \Pi z{:}o.\ \Pi w{:}o.\ (x = y) \to (z = w) \to (\mathsf{App}\ x\,z = \mathsf{App}\ y\,w)$

Betav : $\Pi(\mathsf{Val}\ (\mathsf{Lam}\ f)){:}(f{:}\Pi(\mathsf{Val}\ x^o).o).\ \Pi(\mathsf{Val}\ a^o).$
$\mathsf{App}\ (\mathsf{Val}\ (\mathsf{Lam}\ f))\ (\mathsf{Val}\ a) = f\ (\mathsf{Val}\ a)$

Etav : $\Pi(\mathsf{Val}\ x^o).\ (\mathsf{Val}\ (\mathsf{Lam}\ (\lambda(\mathsf{Val}\ y^o).\mathsf{App}\ (\mathsf{Val}\ x)\ (\mathsf{Val}\ y)))) = (\mathsf{Val}\ x)$

Xiv : $\Pi(\mathsf{Val}\ (\mathsf{Lam}\ f)){:}(f{:}(\Pi(\mathsf{Val}\ x^o).o).\ \Pi(\mathsf{Val}\ (\mathsf{Lam}\ g)){:}(g{:}(\Pi(\mathsf{Val}\ x^o).o).$
$(\Pi(\mathsf{Val}\ x^o).\ f\ (\mathsf{Val}\ x) = g\ (\mathsf{Val}\ x)) \to (\mathsf{Val}\ (\mathsf{Lam}\ f) = \mathsf{Val}\ (\mathsf{Lam}\ g))$

Figure 2. Plotkin Call-by-value λ-calculus Encoding

3.1 Call-by-value Lambda Calculus

Plotkin's call-by-value λ-calculus (λ_{v}-calculus) [10] differs from the traditional λ-calculus in the formulation of the β-reduction rule, namely

$$(\lambda x.M)\ N \to_{\beta_{\mathsf{v}}} M[N/x]$$

provided that N is a value, that is a variable or an abstraction. The encoding in PLF is shown in Figure 2. We omit adequacy results. We could, *e.g.*, prove that

$$\mathsf{Th} : \mathsf{Etav} \implies \mathsf{Xiv}$$

Syntax and Operational Semantics

$P ::= x \mid A \mid P \twoheadrightarrow M$ $\qquad (P \twoheadrightarrow M)\, N \;\to_\rho\; M\theta$ with $\theta = \mathcal{A}lg(P \cdot N \cdot \mathsf{Fv}(P))$

$A ::= f \mid A\, P$ $\qquad (M_1\,, M_2)\, M_3 \;\to_\delta\; (M_1\, M_3\,, M_2\, M_3)$

$M ::= P \mid M\, M$ $\qquad$ N.B. both $(M_1\,, M_2)$ and $\to_\delta$ are derivable

Syntactic Categories

o : Type

Operations (we short o^n for $\underbrace{o \to \ldots \to o}_{n \text{ times}}$ and $\checkmark\mathsf{C}[\overline{x}]{:}(\overline{x{:}o^n})$ for $\checkmark\mathsf{C}[\overline{x^{o^n}}]$)

$\mathsf{Alg} : o^2$ $\quad \mathsf{Rew} : o^2 \to o$ $\quad \mathsf{App} : o^3$ $\quad \mathsf{Pair} : o^3$

Judgments

$=$: $o \to o \to$ Type

Rewriting encoding $[\![-]\!] : \mathsf{Rho} \Rightarrow \mathsf{PLF}$

$[\![x]\!] \triangleq x$ $\qquad [\![P \twoheadrightarrow M]\!] \triangleq \mathsf{Rew}\ (\lambda [\![P]\!]{:}\Delta.[\![M]\!])\quad \Delta \triangleq \overline{\mathsf{Fv}(P){:}o}$

$[\![f]\!] \triangleq \mathsf{Alg}\ f$ $\qquad [\![M\, N]\!] \triangleq \mathsf{App}\ [\![M]\!]\, [\![N]\!]$

$[\![A\, P]\!] \triangleq \mathsf{App}\ [\![A]\!]\, [\![P]\!]$ $\qquad [\![M\,, N]\!] \triangleq \mathsf{Rew}\ (\lambda x{:}o.\mathsf{Pair}\ (\mathsf{App}\ [\![M]\!]\, x)\, (\mathsf{App}\ [\![N]\!]\, x))$

Axioms and Rules

$\mathsf{Eq}_{\mathsf{refl}}$ $\mathsf{Eq}_{\mathsf{symm}}$ $\mathsf{Eq}_{\mathsf{trans}}$ $\mathsf{Eq}_{\mathsf{ctx}}$ see the lambda calculus

Rho : $\Pi r{:}o^2.\ \Pi a{:}o.\ \mathsf{App}\ (\mathsf{Rew}\ r)\, a = r\, a$

Eta : $\Pi x{:}o.\ \mathsf{Rew}\ (\lambda y{:}o.\mathsf{App}\ x\, y) = x$

Xi : $\Pi r{:}o^2.\ \Pi s{:}o^2.\ (\Pi a{:}o.\ r\, a = s\, a) \to \mathsf{Rew}\ r = \mathsf{Rew}\ s$

Delta : $\Pi\mathsf{Rew}\ (\lambda x{:}o.\mathsf{Pair}\ (\mathsf{App}\ y^o\, x)\, (\mathsf{App}\ z^o\, x)).\ \Pi a{:}o.$
$\mathsf{App}\ (\mathsf{Rew}\ (\lambda x{:}o.\mathsf{Pair}\ (\mathsf{App}\ y\, x)\, (\mathsf{App}\ z\, x)))\, a = (\lambda x{:}o.\mathsf{Pair}\ (\mathsf{App}\ y\, x)\, (\mathsf{App}\ z\, x))\, a$

Figure 3. Classical ρ-calculus Encoding

3.2 Untyped Rewriting Calculus

The rewriting calculus (ρ-calculus) [2, 3], is a simple higher-order calculus where functions can be applied only upon reception of an argument whose "pattern" matches with the one explicitly declared in the function itself. This allows to represents naturally classical lambda calculus and many term rewriting systems. What makes the rewriting calculus appealing for reasoning on the web is precisely its foundational features that allow us to represent the

atomic actions (*i.e.* rules) and the chaining of these actions (*i.e.* what we called above strategies) in order to achieve a global goal like, for example, transforming semi-structured data, extracting informations or inferring new ones. As the matching mechanism of the calculus can be parameterized by a suitable matching theory, this allows us for example to express in a precise way how the semi-structured data should be matched. The encoding in PLF is shown in Figure 3. We omit adequacy results. We could, *e.g.*, prove that

$$\mathsf{Th} : \mathsf{Rho} \Longrightarrow \mathsf{Delta}$$

References

[1] G. Barthe, H. Cirstea, C. Kirchner, and L. Liquori. Pure Pattern Type Systems. In *Proc. of POPL*. The ACM Press, 2003.

[2] H. Cirstea, C. Kirchner, and L. Liquori. Matching Power. In *Proc. of RTA*, volume 2051 of *LNCS*, pages 77–92. Springer-Verlag, 2001.

[3] H. Cirstea, C. Kirchner, and L. Liquori. Rewriting Calculus with(out) Types. In *Proc. of WRLA*, volume 71 of *ENTCS*, 2002.

[4] T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76:95–120, 1988.

[5] D. J. Dougherty. Adding Algebraic Rewriting to the Untyped Lambda Calculus. *Information and Computation*, 101(2):251–267, 1992.

[6] R. Harper, F. Honsell, and G. Plotkin. A Framework for Defining Logics. *Journal of the ACM*, 40(1):143–184, 1992.

[7] J.P. Jouannaud and M. Okada. Executable Higher-Order Algebraic Specification Languages. In *Proc. of LICS*, pages 350–361, 1991.

[8] J.W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993.

[9] M. Okada. Strong Normalizability for the Combined System of the Typed λ Calculus and an Arbitrary Convergent Term Rewrite System. In *Proc. of ISSAC*, pages 357–363. ACM Press, 1989.

[10] G. Plotkin. Call-by-Name, Call-by-Value and the λ-Calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[11] Rho Team. The Rho Home Page, 2005. http://rho.loria.fr/.

[12] V. van Oostrom. Lambda Calculus with Patterns. TR IR-228, Vrije Univ. Amsterdam, 1990.

A Genuine XML-examples in *à la* Rewriting Calculus

```
*******************************************************
* Small XML Galleria written in the Rewriting Calculus *
*     Run it with the interpreter iRho, its free!      *
*  http://www-sop.inria.fr/mirho/Luigi.Liquori/iRho/   *
*******************************************************

*** Some list of numbers  ***
ZERO  = 0;;              ONE  = (succ 0);;       TWO = (succ ONE);;   THREE = (succ TWO);;
FOUR  = (succ THREE);;   FIVE = (succ FOUR);;    SIX = (succ FIVE);;  SEVEN = (succ SIX);;
EIGHT = (succ SEVEN);;   NINE = (succ EIGHT);;   TEN = (succ NINE);;

*** Some list of friends ***
ME   = (person ((first luigi)   ,(last liquori) ,(sex m),(empl inria)   ,(nat it) ,(cat ONE)));;
YOU  = (person ((first jessica) ,(last rabbit)  ,(sex f),(empl disney)  ,(nat usa),(cat TWO)));;
SHE  = (person ((first helene)  ,(last kirchner),(sex f),(empl cnrs)    ,(nat fr) ,(cat ZERO)));;
HIM  = (person ((first claude)  ,(last kirchner),(sex m),(empl inria)   ,(nat fr) ,(cat FOUR)));;
HER  = (person ((first uma)     ,(last thurman) ,(sex f),(empl hollywd) ,(nat usa),(cat FIVE)));;
BIG  = (person ((first bg)      ,(last sidharth),(sex m),(empl birla)   ,(nat in) ,(cat SIX)));;
HEAD = (person ((first moreno)  ,(last falaschi),(sex m),(empl siena)   ,(nat it) ,(cat TWO)));;
BOSS = (person ((first furio)   ,(last honsell) ,(sex m),(empl udine)   ,(nat it) ,(cat ONE)));;
JEFE = (person ((first maria)   ,(last alpuente),(sex f),(empl valencia),(nat es) ,(cat ZERO)));;
GURU = (person ((first salvador),(last lucas)   ,(sex m),(empl papaya)  ,(nat es) ,(cat ONE)));;

*** The DB ***
DB = (group (ME,YOU,SHE,HIM,HER,BIG,HEAD,BOSS,JEFE,GURU,nil));;

*** FINDN: Find in a DB the nth Element in a xml catalogue ***
[FINDN = ((0,nil)                -> fail, ((succ N),(group nil))   -> fail,
         ((succ 0),(group (X,Y))) -> X,    ((succ N),(group (X,Y))) -> (FINDN (N,(group Y))))];
(FINDN (THREE,DB));;

*** KILLIT: Kill in a DB all the items of "it" nationality  ***
[KILLIT = (group X)                          -> (group (DROPIT X)) |
 DROPIT = ((nil)                             -> (nil),
           ((person (X,Y,Z,U,(nat it) ,V)),W) -> (DROPIT W),
           ((person (X,Y,Z,U,(nat fr) ,V)),W) -> ((person (X,Y,Z,U,(nat fr) ,V)),(DROPIT W)),
           ((person (X,Y,Z,U,(nat es) ,V)),W) -> ((person (X,Y,Z,U,(nat es) ,V)),(DROPIT W)),
           ((person (X,Y,Z,U,(nat in) ,V)),W) -> ((person (X,Y,Z,U,(nat in) ,V)),(DROPIT W)),
           ((person (X,Y,Z,U,(nat usa),V)),W) -> ((person (X,Y,Z,U,(nat usa),V)),(DROPIT W)))];
(KILLIT DB);;

*** KILLC: Kill in a DB all the items of a given category ***
[KILLC = (C,(group X))                       -> (group (DROPC (C,X))) |
 DROPC = ((C,(nil))                          -> (nil),
          (C,((person (X,Y,Z,U,V,(cat K))),W)) ->
           (COND ((EQ (C,K)),DUMMY -> (DROPC (C,W)),
                             DUMMY -> ((person (X,Y,Z,U,V,(cat K))),(DROPC (C,W))))))];
(KILLC (ONE,DB));;

*** GETC: Select in a DB all the items of a given category ***
[GETC  = (C,(group X))                       -> (group (SELC (C,X))) |
 SELC  = ((C,(nil))                          -> (nil),
          (C,((person (X,Y,Z,U,V,(cat K))),W)) ->
           (COND ((EQ (C,K)),DUMMY -> ((person (X,Y,Z,U,V,(cat K))),(SELC (C,W))),
                             DUMMY -> (SELC (C,W)))))];
(GETC (ONE,DB));;

*** Auxiliary COND and EQ ***
COND = (BOOL,THEN,ELSE) -> ((true -> (THEN dummy),false -> (ELSE dummy)) BOOL);;
EQ   = ((0,0)->true,(0,(succ X))->false,((succ X),0)->false,((succ X),(succ Y))->(EQ (X,Y)));;
```

Anchoring modularity in HTML

Claude Kirchner

INRIA & LORIA

Hélène Kirchner

CNRS & LORIA

Anderson Santana [1]

INRIA & LORIA

Abstract

Modularity is a key feature at design, programming, proving, testing, and maintenance time, as well as a must for reusability. Most languages and systems provide built-in facilities for encapsulation, importation or parameterization. Nevertheless there exists also languages, like HTML, with poor support for modularization. A natural idea is therefore to provide generic modularization primitives.

To extend an existing language with additional and possibly formal capabilities, the notion of *anchorage* and *Formal Island* has been introduced recently. TOM for example, provides generic matching, rewriting and strategy extensions to JAVA and C.

In this paper, we show on the HTML example, how to add modular features by anchoring modularization primitives in HTML. This allows one to write modular HTML descriptions, therefore facilitating their design, reusability, and maintenance, as well as providing an important step towards HTML validity checking.

Key words: Modularization, parameterization, HTML, TOM, MHTML, formal island, feature anchorage

1 Introduction

Modularity is a key feature of programming environments at all stages of software development, from users requirements analysis to maintenance. It is of course the key feature of reusability policies and therefore a main concept in

[1] Supported by CAPES.

any software library. With the raising time of safe and secure software, modularity appears also as a fundamental feature to make possible the construction of large certified software.

Modularity is thus present in many programming languages and proof environments and we have now a fairly good understanding of the semantics of the main modularity constructs. In particular in functional and algebraic programming, the notions of importation, parameterization and visibility have been given both categorical and operational semantics (e.g. [2,6,12]).

If from the theoretical point of view, the situation is satisfactory, this is not the case from the practical one, in particular because each language has its own modularity features and semantics. Clusters in CLU, packages in Ada, structures in ML, classes in C++ and Java are different constructs facilitating modular programming. Some languages have quite sophisticated modularity features, like CASL, OBJ and Maude, where the notion of *view* precisely formalizes the way parameters are instantiated or modules imported. Others, like ELAN have a more elementary approach. Object-oriented languages like Java take into account classes and inheritance. Functional languages, such as ML, have also evolved towards modularity. Face to this variety of approaches, we are thus in a situation where standard modularity features, mainly independent of the language, would be greatly appreciated.

But modularity is not a universal feature of programming languages and several of them lack of useful capabilities. For example, parameterization does not exist in ASF+SDF nor C. An extreme example in this case is HTML that has no importation nor parameterization capability at all.

So either for standardization or for needed improvement, it is desirable to have the capability of adding modularity features to an existing programming language.

While we understand its usefulness, we have now to address the feasibility of adding modularity primitives in a programming environment. This question has been explored in [12] where an SML-like module system is presented that can accommodate a variety of base languages, provided they satisfy mild assumptions.

Another approach, independently developed, is the formal island paradigm that comes into play in a simple and pragmatic context: indeed, it would be nonsense to throw away the billions of code lines that are in use today in all domains of human activities, nevertheless it is clear that all these software have to be considerably improved in their logic, algorithmic, security and maintenance qualities. As introduced in TOM [2] [13,8] in particular for matching, normalization and strategic rewriting, formal islands allow for the addition to existing programming languages, of formal features that can be compiled later on into the host language itself, therefore inducing no dependency on the formal island mechanism.

[2] `tom.loria.fr`

At the price of a rigorous anchoring, that provides the link between the host language data structure and the formal objects, the formal island approach gives the possibility (*i*) to extend the expressivity of the language with higher-level constructs at design time, (*ii*) to perform formal proof on the formal island constructions, (*iii*) to certify the implementation of the formal island compilation into the host language [11].

In addition to these benefits, what makes formal islands even more attractive is that they are shared between several implementations made in different programming languages. For instance, TOM provides matching, normalization and strategic rewriting in Java (this is jTOM), in C (this is cTOM) or in CAML (mlTOM).

To set-up the general definition of Modular Formal Island is a difficult goal and a first work towards making TOM modular for algebraic specifications in the vein of CASL has been done in [9].

The purpose of this paper is to present a first step for anchoring modularity in an existing language and to illustrate the approach with HTML. This allows writing modular HTML descriptions, therefore facilitating their design, reusability, and maintenance, as well as providing an important step towards HTML validity checking. While we only deal in this paper with the HTML case, one important interest of the proposed approach is to set-up the basis for a generic method.

In order to make precise our objectives, we use in Section 2 a running example of a one block HTML page and show how we would like it to be decomposed in significantly smaller pieces. We then present in Section 3 the modularity features added to HTML and give in Section 4 its operational semantics, thus making clear the compilation process. Related work and further extensions are addressed respectively in Section 5 and Section 6.

2 Motivating example

Let us first consider an illustrative example of how modularity can help for the task of construction and maintenance of web sites.

Commonly, web pages composing a web site share some contents. This is related either to standard information that must appear on each page, or to navigability issues, for example, sets of links repeatedly presented to site visitors leading to other pages on the site.

The current trend among web designers is to drop the use of frames [3], which allows shared content between pages to be repeated in every page of the site. Especially for web sites built without the use of a script language, the webmasters have literally to "copy and paste" the static information from one page to another. Consequently, updates become an annoying task.

[3] http://www.w3c.org/TR/REC-html40/present/frames.html

A typical web site layout is shown in Figure 1. All pages that follow the home (index) page would share the same code for page header, menu, and page footer. The interaction of a visitor with the links contained on the list of pointers on the left, brings a similar page where the "content" box, as indicated in the figure, displays varying information, according to the subject of that page.

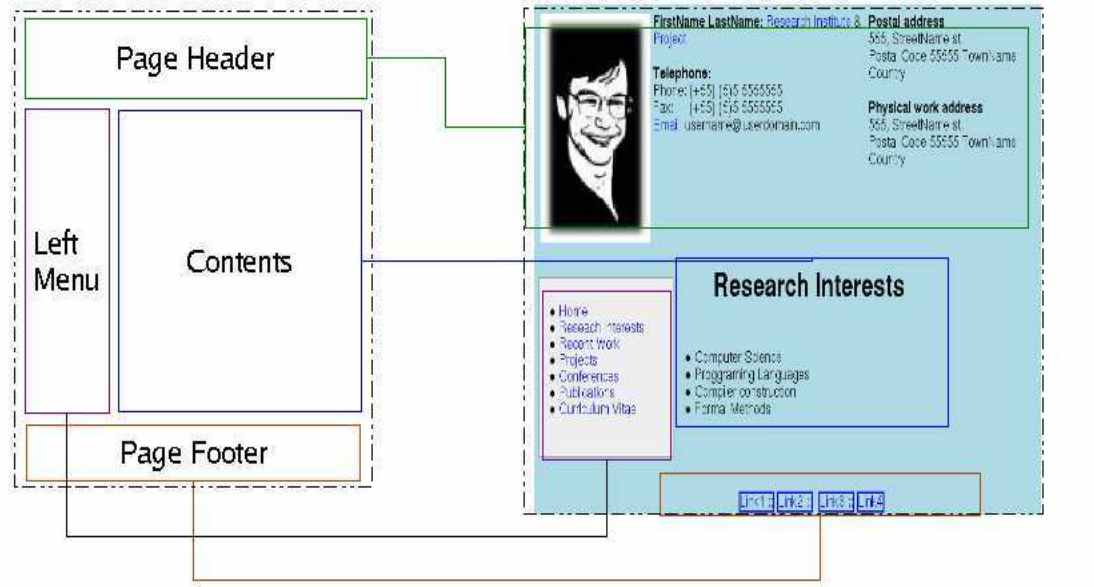


Fig. 1. A typical web site layout

Having the capability to isolate each piece of repeated code, web sites writers could (semi-)automatically compose a new web page from separate sources. In the example above, the page should be broken into reusable named modules for the page heading, navigation menu, and for the page footer.

Moreover, introducing parameters would allow reusing a similar page format for another user, facilitating uniform presentation of web sites.

As a third concern, in this context, each repeated piece of code could be checked only once to be well-formed from an HTML point of view.

Our goal is now to give a practical answer to these concerns, through the proposal of a formal language and its anchoring process in HTML.

3 The Modular HTML language

In this section we present the implementation of Modular HTML (`mHTML`). Basically, we use the algebraic specification formalism to express syntax and semantics of the language. Similar approaches have been used in the definition of Full Maude [5] as well as a module algebra for ASF+SDF [1]. Nevertheless, no sophisticated knowledge about algebraic specifications is required to understand the mechanism on which this execution model is based.

The mHTML language is a combination of the regular HTML markup with structuring primitives that allows the composition of documents through the reuse of existing code fragments. The principles that have guided its design are simplicity and genericity.

Modules

A first concern in modularization is how to define a self-contained piece of reusable code that represents a module. Our approach leaves to the user the task of defining the desired degree of granularity, since we do not restrict the modules to be exactly valid HTML code. Actually, any well-formed set of HTML markup can be considered as a module. For example, by using the `%module` primitive, we can define a reusable "menu" available for all pages in a web site in the following manner:

Example 3.1 The left menu of the web page in Figure 1 is described in the following module.

```
%module menu
 <ul>
      <li><a href="index.html">Home</a></li>
      <li><a href="researchInterests.html">Research Interests</a></li>
      <li><a href="work.html">Recent Work</a></li>
      <li><a href="projects.html">Projects</a></li>
      <li><a href="conferences.html">Conferences</a></li>
      <li><a href="publications.html">Publications</a></li>
      <li><a href="cv.html">Curriculum Vitae</a></li>
 </ul>
```

By convention and in order to facilitate access to modules, we restrict to one module per file, and impose that module and system file containing it have the same names. Thus a module called `foo`, will be found in a system file called `foo.mhtml`.

Imports

Secondly, a common need in the context of reuse is to access symbols declared by other modules. This can be achieved through the `%import` primitive, The effect of the import primitive at compile time is a textual expansion of the document. The imported module is inserted at the position the `%import` mark is found, repeated headers will be ignored, only the main module header is kept. In the case where the imported module has itself importations, this process is recursively performed.

Another important concern is to force the imported modules to be well-formed HTML. Alternatively it would be desirable to import *valid* HTML code, what would mean that the imported code should satisfy all the requirements from the W3C current HTML specification [14]. This is achieved by another feature of the language which provides a theory notion, close to the

notion of type. Setting in mHTML that a module mod is well-formed (resp. valid) is achieved by the declaration mod :: WellFormedHTML (resp. mod :: ValidHTML). At compile time, in the HTML case, this property is checked by calling the appropriate HTML tools.

The example below illustrates these issues considering the web site presented in Section 2:

Example 3.2 The web page of Figure 1 is written as the following module.

```
%module page1
<html>
  <head>
    <link href="styles.css" rel="stylesheet" type="text/css" />
    <title>Home Page</title>
  </head>
  <body>
        %import header :: WellFormedHTML
        <div class="header">
            <h1 class="title">Research Interests</h1>
        </div>
                <div  class="menu">
            %import menu :: WellFormedHTML
        </div>
        <div class="content">
            %import contents :: WellFormedHTML
        </div>
         <div class="footer">
            %import footer :: WellFormedHTML
        </div>
   </body>
</html>
```

Templates

The language also provides a template mechanism, that extends the language with parameterized modules. In this case, actual parameters are expected to be other modules, that in their turn, may be required to conform to a certain theory. In the HTML case, we currently simply consider these as either valid or simply well-formed HTML. Again, the parameters are transformed into a textual expansion when the preprocessor composes instances of such templates. The following example shows the definition of a parameterized mHTML module:

Example 3.3 For reusing the structure of the web page of Figure 1, the following template is provided:

```
%module template1 [
             Title
             Header   :: WellFormedHTML
```

```
              Menu     :: WellFormedHTML
              Contents :: WellFormedHTML
              Footer   :: WellFormedHTML   ]
<html>
  <head>
    <title>%use Title</title>
  </head>
   <body>
           %use Header
         <div class="header">
             <h1 class="title">Research Interests</h1>
         </div>
         <div class="menu">
            %use Menu
         </div>
          <div class="content">
            %use Contents
         </div>
           <div class="footer">
            %use Footer
         </div>
   </body>
</html>
```

This template can generate a new instance with the following instantiation:

Example 3.4 Actual module names are substituted to parameters to obtain an appropriate instance of the previous template:

```
%module publications
%import template1 [myHomePage
                   myPageHeeader
                   myBookmarks
                   myPublications
                   myPageFooter]
```

Figure 2 presents the syntax used in the examples above.

4 Semantics, anchoring and compilation

We are now ready to describe how the modular features provided in MHTML can be compiled into HTML. From an implementation point of view, the compiler accepts as input an MHTML program, consisting of native code combined with the modularity primitives of the language described in Section 3 and generates pure native code as output. For describing this compilation process, we choose to write an algebraic specification with rewrite rules, implemented in ELAN 4 [10]. In this way, we also provide the operational semantics of our MHTML language. Outlines of this specification is given in Figure 3. This is sim-

Module	::=	%*module* ⟨moduleName⟩ ("[" **Param**+ "]")∗ **Body**
Body	::=	([**Import**][**Use**] [**HTML**])∗
Import	::=	%*import* ⟨moduleName⟩ [**Paraminst**+] ["::" **Theory**]
Param	::=	⟨paramName⟩ ["::" **Theory**]
Paraminst	::=	⟨moduleName⟩
Use	::=	%*use* ⟨paramName⟩
Theory	::=	WellFormedHTML | ValidHTML
HTML	::=	Any HTML code with the clear exception of the primitives listed above

Fig. 2. Syntax for the generic structuring constructs

ilar to (but indeed simpler than) the rules given in [5] to describe the (more elaborated) modularity features of the Maude language.

ELAN 4 shares the syntax definition of SDF [4]. HTML code is represented by the sort `HTML`. The next rules can be read as follows:

- The first rule, establishes that modules with no further structuring constructs than `%module` should be transformed into HTML code.

```
[ ] translate ( %module m html ) => html
```

- The couple of rules below state that any importation without parameters nor theories should start a recursive call to the translate function, and as result would concatenate the produced HTML in the position the `%import` mark was found

```
[ ] translate ( %module m %import m1 html ) => translate(load(m1)) html
[ ] translate ( %module m html %import m1 ) => html translate(load(m1))
```

- The following rules have a similar effect as the previous ones, but they ensure that the modules being imported conform to a certain theory, through the `satisfies` function.

```
[ ] translate ( %module m html %import m1 :: th) =>
          html translate(load(m1))
    if satisfies( load(m1), th)
[ ] translate ( %module m  %import m1 :: th html) =>
          translate(load(m1)) html
    if satisfies( load(m1), th)
```

- The next two rules deal with the importation of parameterized modules, in this case a second version of the translation function is activated to treat the actual parameters.

```
[ ] translate ( %module m %import m1 [i] html) =>
       translate(load(m1), i) html
[ ] translate ( %module m %import m1 [i] :: th  html) =>
       translate(load(m1), i) html
    if satisfies( load(m1), th)
```

```
module \mhtmlTranslate

imports basic/ElanLayout
        basic/BuiltinBool
        \mhtmlSyntax

       exports
              context-free syntax

                       translate(Module)                  -> HTML
                       translate ( Module , ParamList ) -> HTML
                       HTML HTML                          -> HTML

                       load( ModuleName )          -> Module
                       satisfies(Module, Theory)    -> BuiltinBool
        hiddens
               variables
                       "m"[0-9]*    -> ModuleName
                       "x"[0-9]*    -> ParamName
                       "p"[0-9]*    -> ParamDecList
                       "i"[0-9]*    -> ParamList
                       "html"[0-9]* -> HTML
                       "th"[0-9]*   -> Theory
rules
   [] ...(c.f. text)
```

Fig. 3. Rewrite rules for the MHTML compiler

- The following rules run through lists of formal parameters to verify them against actual parameters, and perform the corresponding substitution wherever the parameter occurrence was found

```
[ ] translate ( %module m  [x p] html %use x, m2 i ) =>
              translate ( %module m [p] html translate(load(m2)) , i)

[ ] translate ( %module m  [x :: th  p] html %use x, m2 i ) =>
              translate ( %module m [p] html translate(load(m2)) , i)
         if satisfies(load(m2), th)
```

These rewrite rules provide an outline of the compilation of MHTML to HTML. When completed with the full description of the instantiation process, this specification provides a normalization process that compiles MHTML into HTML. Compared to the implementation of *formal islands* provided for matching and rewriting in TOM, this process is simpler from several points of views. With the modularity primitives introduced here, there is no need for an anchoring dealing with the structure of terms. This could change when considering, for example, dependent types. So rather than a "deep" anchoring involving the structure of terms as used in [11], we need here only a "shallow" one dealing with the module content only. Another difference is that we

do not have to certify the compilation process in the same way: in [11], the matching compilation has to be certified, taking into account the properties of the anchoring. In our case, the verification process will concern the validation of the importations and of the parameter instantiations, leading in particular the way to another use of type systems for HTML.

5 Related Work about HTML

Restricting our attention to the specific question of anchoring modularity in HTML, the question arises to identify the improvements provided by our approach with respect to the web site development process. First of all, `mHTML` provides the following main advantages:

- It is independent of the web server hosting the web site.
- It is not necessary to process the language in a graphical environment, like in WYSIWYG HTML editors. This simplifies the maintenance process. It could also be used by the graphical environment to produce modular and hopefully readable code.
- It has a lightweight compilation process.
- It is easy to learn.
- It does not need to be executed every time the site is accessed.
- Finally, we should emphasize that, as for any anchorage, it does not induce any dependence on the language extension, as all the anchored language features are compiled into the target language.

The lack of HTML modularity has of course already drawn attention and we can mention the following web related works.

The jigwig project[3] provides an implementation of a language aimed at designing interactive web services. It is based on a session centered execution model of requests made through the web. As a secondary result, it provides a template language for dynamic web page construction. This language allows the definition of gaps, that may be filled with HTML code. Surely, the approach provides reuse of HTML code, but it is dependent on the whole environment to do simple structuring tasks.

In [7] a complete management environment with a language is developed to attack the management problems that appear in the implementation of data intensive web sites. The system combines a query language to specify the site's structure and content with a template language for its HTML representation. Although reusability is not the main concern of this work, the template language offers flexibility and extensibility to the creation of the site, it presents the same disadvantage as the previous one.

With respect to scripting languages like PHP, ASP, PERL, etc, this approach has the advantage of being simple and straightforward for the user. Another advantage of `mHTML` when compared to scripting languages or server

side includes, available in Apache for example, is that it does not need to be re-executed every time the root module is accessed via the web. Moreover we believe, although this has not yet been considered, that `mHTML` can be combined with other languages in the development of web sites.

Similar forms of template mechanisms are provided by a number of WYSIWYG HTML editors. This restricts the re-use of HTML because the user depends on a graphical environment to generate a new web page from existing templates, whereas the same functionality can be obtained in MTML through a simple text editor. It is also obviously possible for the user to design his page in his favorite WYSIWYG editor, and after, determine what are the parts he would like to reuse from that page in `mHTML`.

6 Conclusion

The approach described in this paper is part of a much larger enterprise towards the non-invasive diffusion of formal methods and algebraic methodology through the concept of Formal Islands. For example, on one hand matching and strategic rewriting may help to model large pieces of code. On the other hand, modularity is of fundamental use in the structuration of large software.

We have developed the idea of *modular anchoring* on the example, simple but useful, of the HTML language. Introducing quite simple primitives for importation and parameterization, we have shown how this can define a modular extension of HTML. The compilation process has been outlined using ELAN 4.

This is of course a preliminary work and we are actively working on deepening several points. First an implementation of `mHTML` in on its way and we naturally chose as implementation framework TOM itself. This will allow us to play with the concept and to validate our initial ideas. Second and quite importantly, verification tools specific to HTML should be used or developed. One can think of course, as we mentioned before, to the (X)HTML code validation as provided by the W3C tools. Specific type systems will also be developed to ensure desirable properties. For example and as for algebraic languages, structured data types could be statically typed using prescriptive type systems. Also, the introduction of views for parameter validation will require for their verification to perform non-trivial proofs and the experience gained again from the algebraic specification community will be of great interest here. We may also think to specific verification tools for HTML, like checking the reachability of linked objects. Of course HTML is a very specific and elementary language. A natural extension will concern XML, in particular for the logic and semantic web, and modularity features will be of great use in projects like `Rewerse` [4] .

Acknowledgments This paper benefits of the many fruiful discussions we had

[4] rewerse.net

with Pierre-Etienne Moreau, Horatiu Cirstea and Antoine Reilles, in particular on formal islands. We also thanks the anonymous referees for their constructive comments on the first version of this paper.

References

[1] Bergstra, J. A., J. Heering and P. Klint, *Module algebra*, Journal of the ACM **37** (1990), pp. 335–372.

[2] Bidoit, M., D. Sannella and A. Tarlecki, *Architectural specifications in* Casl, Formal Aspects of Computing **13** (2002), pp. 252–273.

[3] Christensen, A. S., A. Moller and M. I. Schwartzbach, *Extending Java for high-level Web service construction*, ACM Trans. Program. Lang. Syst. **25** (2003), pp. 814–875.

[4] Deursen, A., J. Heering and P. Klint, "Language Prototyping," World Scientific, 1996, iSBN 981-02-2732-9.

[5] Durán, F., "A Reflective Module Algebra with Applications to the Maude Language," Ph.D. thesis, Universidad de Málaga, Spain (1999), http://maude.csl.sri.com/papers.

[6] Durán, F. and J. Meseguer, *Structured theories and institutions*, in: M. Hofmann, G. Rosolini and D. Pavlović, editors, *Proceedings of 8th Conference on Category Theory and Computer Science, Edinburgh, Scotland, September 1999*, Electronic Notes in Theoretical Computer Science **29** (1999), pp. 71–90, http://www.elsevier.nl/locate/entcs/volume29.html.

[7] Fernandez, M., D. Florescu, A. Levy and D. Suciu, *Declarative specification of web sites with strudel*, The VLDB Journal **9** (2000), pp. 38–55.

[8] Guyon, J., P.-E. Moreau and A. Reilles, *An integrated development environment for pattern matching programming*, in: *2nd eclipse Technology eXchange workshop - eTX'2004, Barcelona, Spain*, Electronic Notes in Theoretical Computer Science, Brian Barry and Oege de Moor, 2004.

[9] Hrvatin, S., *Structuration pour les spécifications à base de règles : Etude et mise en œuvre pour TOM*, Rapport de DEA, Université Henri Poincaré - Nancy 1 (2004).

[10] Kirchner, C. and H. Kirchner, *Rule-based programming and proving : the ELAN experience outcomes*, in: *Ninth Asian Computing Science Conference - ASIAN'04, Chiang Mai, Thailand*, 2004.

[11] Kirchner, C., P.-E. Moreau and A. Reilles, *Formal validation of pattern matching code*, Submitted (2005).

[12] Leroy, X., *A modular module system*, Journal of Functional Programming **10** (2000), pp. 269–303.

[13] Moreau, P.-E., C. Ringeissen and M. Vittek, *A Pattern Matching Compiler for Multiple Target Languages*, in: G. Hedin, editor, *12th Conference on Compiler Construction, Warsaw (Poland)*, LNCS **2622** (2003), pp. 61–76.

[14] Raggett, D., A. L. Hors and I. Jacobs, *Html 4.01 specification* (1999), http://www.w3.org/TR/REC-html40/.

A Rule-based System for Web site Verification [1]

D. Ballis [a] J. García-Vivó [b]

[a] *Dip. Matematica e Informatica, Via delle Scienze 206, 33100 Udine, Italy. Email: demis@dimi.uniud.it.*

[b] *DSIC, Universidad Politécnica de Valencia, Camino de Vera s/n, Apdo. 22012, 46071 Valencia, Spain. Email: jgarciavivo@dsic.upv.es.*

Abstract

In this paper, we describe a system, written in Haskell, for the automated verification of Web sites which can be used to specify (partial) correctness and completeness properties of a given Web site, and then automatically check whether these properties are actually fulfilled. It provides a rule-based, formal specification language which allows us to define syntactic/semantic conditions for the Web site by means of a user-friendly graphical interface as well as a verification facility for recognizing forbidden/incorrect patterns and incomplete/missing Web pages.

Key words: Web site Verification, Rewriting, Formal Methods.

1 Introduction

The management of a complex Web site is a nontrivial task, in which the problematics related to the verification and the correction of the (semistructured) data play a fundamental role. As a matter of fact, it is far simpler to discover inconsistent information on the Web than to find a well-maintained Web site. We believe that formal methods can bring a relevant contribution, giving support to Automated Web site verification. For instance, the system XLINKIT [3] allows one to check the consistency of distributed, heterogeneous documents as well as to fix the (possibly) inconsistent information. Its specification language is a restricted form of first order logic combined with Xpath expressions. [4] presents a framework for modeling Web interactions and a type system, which can be employed to catch errors in interactive Web programs. In our previous work [2], we described the system VERDI which provides a rule-based

[1] This work has been partially supported by MEC under grant TIN2004-07943-C04-02, and by ICT for EU-India Cross Cultural Dissemination Project under grant ALA/95/23/2003/077-054.

language for the specification and the verification of syntactic as well as semantic properties of collections of XML/XHTML documents. Specifically, the system is able to detect missing/incomplete Web pages w.r.t. a given formal specification.

This paper describes an evolution of the Verdi system which improves several aspects of the previous tool. Firstly, the new specification language does offer the expressiveness and the computational power of functions (which are modeled as term rewriting systems [5]) and is enriched by a new class of rules (i.e., *correctness rules*) in order to express properties for the detection of erroneous/forbidden information. Moreover, the verification methodology allows one to investigate both the syntax and the semantics of a Web site, while the typical validation against DTDs and XML Schemata can only check the syntactic layer of the site. Secondly, a graphical interface has been developed which provides a more friendly use of the tool.

The system is based on the theoretical framework we proposed in [1]. We use rewriting-based technology both to specify the required properties and to formalize a verification technique, which is able to check them.

2 Web site denotation

In our framework, a *Web page* is either an XHTML or an XML. document. Since Web pages are provided with a tree-like structure, they can be straightforwardly translated into ordinary terms of a given term algebra. Note that XML/XHTML tag attributes can be considered as common tagged elements, and hence translated in the same way. Therefore, *Web sites* can be represented as finite sets of (ground) terms.

3 Web specification language

A Web specification is a triple (R, I_N, I_M), 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, such as string processing, arithmetic, boolean operators, etc. It is formalized as a term rewriting system, which is handled by standard rewriting [5]. The rewriting mechanism allows one to execute function calls by simply reducing them to their irreducible form (that is, a term that cannot be rewritten any longer).

The set I_N describes constraints for detecting erroneous Web pages (*correctNess rules*). A correctness rule has the following form: `l` $\rightharpoonup$ `error | C`, where `l` is a term, `error` is a reserved constant, and `C` is a (possibly empty) finite sequence containing membership tests w.r.t. a given regular language[2] (e.g. `X` $\in$ `rexp`), and/or equations over terms. For the sake of expressiveness, we also allow to write inequalities of the form `s` $\neq$ `t` in `C`. Such

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

inequalities are just syntactic sugar for $(\mathtt{s} = \mathtt{t}) = \mathtt{false}$. When $\mathtt{C}$ is empty, we simply write $\mathtt{l} \rightharpoonup \mathtt{error}$. Informally, the meaning of a correctness rule $\mathtt{l} \rightharpoonup \mathtt{error} \mid \mathtt{C}$ is the following. Whenever an instance $\mathtt{l}\sigma$ of $\mathtt{l}$ is recognized in some Web page $\mathtt{p}$, and (i) each structured text $X_i\sigma$, $i = 1, \ldots, n$, is contained in the language of the corresponding regular expression $\mathtt{rexp_i}$ in the condition; (ii) each instantiated equation $(\mathtt{s} = \mathtt{t})\sigma$ holds; then, Web page $\mathtt{p}$ is marked as an incorrect page.

The third set of rules I_M specifes some properties for discovering incomplete/missing Web pages (*coMpleteness rules*). A completeness rule is defined as $\mathtt{l} \rightharpoonup \mathtt{r}\ \langle \mathtt{q} \rangle$, where $\mathtt{l}$ and $\mathtt{r}$ 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. We use attributes $\langle \mathtt{A} \rangle$ and $\langle \mathtt{E} \rangle$ to distinguish "universal" from "existential" rules. Right-hand sides of completeness rules can contain functions, which are defined in R. Besides, some symbols in the right-hand sides of the rules may be marked by means of the symbol $\sharp$. Marking information of a given rule r is used to select the subset of the Web site in which we want to check the condition formalized by 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) $\mathtt{l}$ is recognized in W, also (an instance of) the irreducible form of $\mathtt{r}$ must be recognized in *all* (respectively, *some*) of the Web pages which embed (an instance of) the marked part of $\mathtt{r}$.

Example 3.1 Let R be a TRS defining function `Nat(X)`, which converts a string `X` to a natural number, `append(X,Y)` which concatenates two strings, and `add(X,Y)` which sums two natural numbers. Let (R, I_N, I_M) be a Web specification where I_N and I_M are defined as follows:

$\mathtt{member(name(X), surname(Y))} \rightharpoonup \sharp\mathtt{hpage(fullname(append(X,Y)),status)}\ \langle \mathtt{E} \rangle$
$\mathtt{hpage(status(professor))} \rightharpoonup \sharp\mathtt{hpage(\sharp status(\sharp professor), teaching))}\ \langle \mathtt{A} \rangle$
$\mathtt{hpage(X)} \rightharpoonup \mathtt{error} \mid \mathtt{X\ in\ [:TextTag:]*\ sex\ [:TextTag:]*}$
$\mathtt{blink(X)} \rightharpoonup \mathtt{error}$
$\mathtt{project(grant1(X), grant2(Y), total(Z))} \rightharpoonup \mathtt{error} \mid \mathtt{add(Nat(X), Nat(Y))} \neq \mathtt{Nat(Z)}$

The given Web specification models some required properties for a Web site of a research group. The first two rules are completeness rules, while the remaining ones are correctness rules. 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 applying the function *append* to the name and the surname of the member. 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 mark given on the tag "sta-

tus". The third rule forbids sexual contents from being published in the home pages of the group members. Precisely, we check 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 the set of all the tags and raw texts, containing that word. The fourth 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 fifth rule states that, for each research project, the total project budget must be equal to the sum of the funds, which has been granted for the first and the second research periods.

Diagnoses are carried out by running Web specifications on Web sites. The operational mechanism is based on a novel rewriting-based technique called *partial rewriting*, which is able to extract partial structure from a term, and then rewrite it. Roughly speaking, partial rewriting is a rewriting relation in which pattern matching is replaced by a *simulation* algorithm (cf. [1]).

4 The verification system

The verification system has been implemented in Haskell (GHC v6.2.2). The implementation consists of approximately 1100 lines of source code. It includes a parser for semistructured expressions (i.e. XML/XHTML documents) and Web specifications, and several modules implementing the partial rewriting mechanism, the verification technique, and the graphical user interface. The system allows the user to load a Web site together with a Web specification. Additionally, he/she can inspect the loaded data and finally check the Web pages w.r.t. the Web site specification. We have tested the system on several Web sites. As we plan to show in our system demonstration, we are able to detect both missing/incomplete and incorrect Web pages efficiently.

References

[1] M. Alpuente, D. Ballis, and M. Falaschi. Automated Verification of Web Sites Using Partial Rewriting. In *Proc. of ISoLA'04*, pp. 81—88, 2004.

[2] M. Alpuente, D. Ballis, and M. Falaschi. VERDI: An Automated Tool for Web Sites Verification. In *Proc. of JELIA'04*, pp. 726–729. Springer LNCS 3229, 2004.

[3] 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.

[4] P. T. Graunke, R. B. Findler, S. Krishnamurthi, and M. Felleisen. Modeling Web Interactions. In *Proc. of ESOP'03*, pp. 238–252. Springer LNCS 2618, 2003.

[5] J.W. Klop. Term Rewriting Systems. In *Handbook of Logic in Computer Science*, vol. I, pages 1–112. Oxford University Press, 1992.

Rewriting-based navigation of Web sites [1]

Salvador Lucas [a]

[a] *DSIC, UPV, Camino de Vera s/n, 46022 Valencia, Spain.* slucas@dsic.upv.es

Abstract

In this paper we sketch the use of term rewriting techniques for modeling the dynamic behavior of Web sites.

Key words: Hypertext browsing, Semantic modeling of Web sites, Term rewriting.

1 Introduction

The World Wide Web (WWW) provides easy and flexible access to information and resources distributed all around the world. Although Web sites are usually connected via Internet, many hypertext-based systems like on-line help in compilers, programming language reference manuals, electronic books, or software systems are now organized in a very similar way, also using the same description language (HTML) of Web sites. *Browsing* such systems is an essential aspect of their design and use. Having appropriate dynamic models of Web sites is essential for guaranteeing the expected behavioral properties.

Rewriting techniques [BN98,Ohl02,Ter03] have been recently used to reason about the *static* contents of Web sites [ABF05]. In this paper we show that term rewriting techniques are also well-suited for modeling the dynamic behavior of Web sites. We use Maude [CDEL+02] as a suitable specification language for the rewriting models which also permits to explore interesting properties like the reachability of Web pages within the site.

2 From ARSs to TRSs

We use a (finite) set of symbols (an alphabet) $\mathcal{P}$ to give name to the Web *pages* of a Web site. Regarding its dynamic modeling, the most relevant information contained in a Web page is, of course, that of the links which

[1] Work partially supported by Spanish MEC grant SELF TIN 2004-07943-C04-02, Acción Integrada HU 2003-0003, and EU-India Cross-Cultural Dissemination project ALA/95/23/2003/077-054.

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

can originate that a new Web page is downloaded and then used to further browsing the site. The obvious way to express the different transitions between Web pages is to give the (finite) set of transitions among them, i.e., for each Web page p, we can define $\rightarrow_p = \{(p, p_1), \ldots, (p, p_{n_p})\} \subseteq \mathcal{P} \times \mathcal{P}$ which is the abstract relation between the page p and its immediate successors (i.e., the pages $p_1, \ldots, p_{n_p} \in \mathcal{P}$ which are reachable from p in a single step). The pair $(\mathcal{P}, \rightarrow_{\mathcal{P}})$, where $\rightarrow_{\mathcal{P}} = \bigcup_{p \in \mathcal{P}} \rightarrow_p$ is an *Abstract Reduction System* (ARS [BN98, Chapter 2]) and we can use the associated computational relations $\rightarrow_{\mathcal{P}}$, $\rightarrow^+_{\mathcal{P}}$, etc., to describe the dynamic behavior of our Web site. For instance, reachability of a Web page p' from another page p can be rephrased as $p \rightarrow^*_{\mathcal{P}} p'$.

This abstract model is intuitively clear and can, then, be used as a reference for building more elaborated ones. For many applications, however, this ARS-based framework becomes too restrictive. For instance, modeling *safe* (user-sensitive) access to a Web page requires to represent information about the users and modeling some kind of *validation* before granting any access.

Term Rewriting Systems (TRSs [BN98,Ter03]) provide a more expressive setting by allowing the use of *signatures*, i.e., sets of symbols which can be used to build structured objects (terms) by joining terms below a symbol of the signature. For instance, a safe Web page p can take now an argument representing the *user* who is trying to get access to this page. Web pages p containing no link are just constant symbols p (without any transition). Web pages p without safety requirements are represented by rewrite rules $p(U) \rightarrow p_i(U)$ for $1 \leq i \leq n_p$. The definition of a safe page p is as follows:

$$\begin{array}{lll} p(U) \rightarrow vp(U) & vp(u_1) \rightarrow bp(u_1) & bp(U) \rightarrow p_1(U) \\ & \vdots & \vdots \\ & vp(u_{m_p}) \rightarrow bp(u_{m_p}) & bp(U) \rightarrow p_{n_p}(U) \end{array}$$

where vp and bp stand for *validate* and *browse* page p, respectively, and u_i for $1 \leq i \leq m_p$ are terms (e.g., constant symbols) representing the users who are allowed to gain access to the Web page p. The resulting TRS is *shallow* and *linear*[2]; thus, reachability is decidable [Com00]. Then, reachability of a Web page from another one is decidable too.

Now, after representing the Web site as a Maude rewriting module, it is possible to ask Maude about reachability issues. For instance, the following Maude module provides a partial representation of the WWV'05 site (see http://www.dsic.upv.es/workshops/wwv05):

```
mod WebWWV05 is
   sort S .
   ops wwv05 submission speakers org valencia accomodation travelling
      : S -> S .
   ops sbmlink entcs entcswwv05 : S -> S .
```

[2] A TRS is shallow if variables occur (at most) at depth 1 both in the left- and right-hand sides of the rules [Com00, Section 4]. A TRS is linear if variables occur at most once both in left- and right-hand sides of the rules [BN98, Definition 6.3.1].

```
   ops login vlogin blogin : S -> S .
   ops forgotten register submit : S -> S .
   ops krishnamurthi finkelstein : S -> S .
   ops alpuente ballis escobar : S -> S .
   op cfp : -> S .
   ops slucas smith : -> S .
   vars P PS X U : S .
   rl wwv05(U) => submission(U) .    rl wwv05(U) => speakers(U) .
   rl wwv05(U) => org(U) .           rl wwv05(U) => cfp .
   rl wwv05(U) => valencia(U) .      rl wwv05(U) => accomodation(U) .
   rl wwv05(U) => travelling(U) .    rl submission(U) => sbmlink(U) .
   rl submission(U) =>  entcs(U) .   rl submission(U) => entcswwv05(U) .
   rl sbmlink(U) => login(U) .       rl sbmlink(U) => forgotten(U) .
   rl sbmlink(U) => register(U) .    rl speakers(U) => finkelstein(U) .
   rl speakers(U) => krishnamurthi(U) .     rl org(U) => alpuente(U) .
   rl org(U) => ballis(U) .          rl org(U) => escobar(U) .
   rl login(U) => vlogin(U) .        rl vlogin(slucas) => blogin(slucas) .
   rl blogin(U) => submit(U) .
endm
```

The only safe page is `login`, which grants access to the submission system. For the sake of simplicity, we have omitted many links. In fact, the only 'terminal' page is `cfp`, containing the textual version of the WWV'05 call for papers. We can check whether `slucas` (who has been previously registered) can get access to the submission system (page `submit`).

```
Maude> search wwv05(slucas) =>+ submit(slucas) .
search in WebWWV05safe : wwv05(slucas) =>+ submit(slucas) .

Solution 1 (state 21)
states: 22  rewrites: 21 in 0ms cpu (0ms real) (~ rewrites/second)
empty substitution

No more solutions.
states: 22  rewrites: 21 in 0ms cpu (1ms real) (~ rewrites/second)
```

Maude tells us that there is only one way for `slucas` to reach the submission page. The command `show path 21` provides the concrete path:

```
wwv05(slucas) → submission(slucas) → sbmlink(slucas)
  → login(slucas) → vlogin(slucas) → blogin(slucas)
  → submit(slucas)
```

The non-registered user `smith` cannot reach this protected part of the site:

```
Maude> search wwv05(smith) =>+ submit(smith) .
search in WebWWV05safe : wwv05(smith) =>+ submit(smith) .

No solution.
states: 20  rewrites: 19 in 0ms cpu (0ms real) (~ rewrites/second)
```

3 Further improvements and applications

The basic model in Section 2 can be improved in a number of different ways to obtain more expressive models and/or analyze other behavioral issues:

(i) *Structured* names of users and Web pages allowing for more intuitive and hierarchical naming systems.

(ii) Efficiency of browsing paths; e.g., shortest path (if any) leading from a Web page to another one.

(iii) Finiteness of the search space. Of course, the number of pages in a Web site is always finite, but this could eventually be missed in more expressive models. The use of type information and/or syntactic replacement restrictions [Luc02] could be helpful to avoid this problem.

(iv) Frequency of use of the different links (by applying the recently introduced *probabilistic* approaches to term rewriting).

Finally, the rewriting theory could also benefit from the new research directions pointed by the analysis of the Web. Some challenging aspects are:

(i) Structured definition of Web sites: a given site can often be considered as composed by many smaller sites. This kind of issues correspond to the analysis of *modular* properties in Term Rewriting [Ohl02], but the current developments are probably too weak for modeling Web site structures.

(ii) Evolving Web sites: adding new pages to a Web site is quite usual. This corresponds to dinamically adding new rules to the model of the site.

References

[ABF05] M. Alpuente, D. Ballis, and M. Falaschi. A Rewriting-based Framework for Web Sites Verification. Electronic Notes in Theoretical Computer Science, 124:41-61, 2005.

[BN98] F. Baader and T. Nipkow. Term Rewriting and All That. Cambridge University Press, 1998.

[CDEL+02] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science* 285(2):187-243, 2002.

[Com00] H. Comon. Sequentiality, Monadic Second-Order Logic and Tree Automata. *Information and Computation*, 157(1-2): 25-51, 2000.

[Luc02] S. Lucas. Context-sensitive rewriting strategies. *Information and Computation*, 178(1):293-343, 2002.

[Ohl02] E. Ohlebusch. Advanced Topics in Term Rewriting. Springer-Verlag, Berlin, 2002.

[Ter03] TeReSe, editor, *Term Rewriting Systems*, Cambridge Univ. Press, 2003.

Modeling Web Applications by the Multiple Levels of Integrity Policy

G. Amato [1]

Dipartimento di Scienze
Università degli Studi "G. d'Annunzio", Italy

M. Coppola, S. Gnesi [2]

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
CNR Pisa, Italy

F. Scozzari, L. Semini [3]

Dipartimento di Informatica
Università di Pisa, Italy

Abstract

We propose a formal method to validate the reliability of a web application, by modeling interactions among its constituent objects. Modeling exploits the recent "Multiple Levels of Integrity" mechanism which allows objects with dynamically changing reliability to cooperate within the application. The novelty of the method is the ability to describe systems where objects can modify their own integrity level, and react to such changes in other objects. The model is formalized with a process algebra, properties are expressed using the ACTL temporal logic, and can be verified by means of a model checker. Any instance of the above model inherits both the established properties and the proof techniques. To substantiate our proposal we consider several case-studies of web applications, showing how to express specific useful properties, and their validation schemata. Examples range from on-line travel agencies, inverted Turing test to detect malicious web-bots, to content cross-validation in peer to peer systems.

Key words: Formal Methods, Model Checking, Process Algebra, Temporal Logic.

[1] Email: amato@sci.unich.it
[2] Email: {coppola,gnesi}@isti.cnr.it
[3] Email: {scozzari,semini}@di.unipi.it

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

1 Introduction

Formal methods are increasingly being used to validate the design of distributed systems and have already proved successful in specifying commercial and safety-critical software and in verifying protocol standards and hardware design [4,8]. It is increasingly accepted that the adoption of formal methods in the life cycle development of systems would guarantee higher levels of dependability and greatly increase the understanding of a system by revealing, right from the earliest phases of the software development, inconsistencies, ambiguities and incompletenesses, which could cause subsequent faults. In particular model checking techniques [6,7] have emerged as successful formal verification techniques. They have been defined to automatically check the truth of system properties, expressed as temporal logic formulae, on the finite state model representing the behavior of a system. Model checkers can easily be used by non–expert users too. For this reason model checking has often been preferred in industries to other verification tools, and many efficient verification environments are currently available, based on model checking algorithms [5,11,15].

In the last few years distributed applications over the WEB have gained wider popularity. Several systems have led to an increasing demand of evolutionary paradigms to design and control the development of applications over the WEB. The main advantages of exploiting the WEB as underlying platform can be summarized as follows. The WEB provides uniform mechanisms to handle computing problems which involve a large number of heterogeneous components that are physically distributed and (inter)operate autonomously. Conceptually, WEB services are stand-alone components that reside over the nodes of the network. Each WEB service has an interface which is network accessible through standard network protocols and describes the interaction capabilities of the service. Applications over the WEB are developed by combining and integrating together WEB services. Web applications show the same verification problems of classical distributed systems. We may hence extend techniques and tool used for their verification also in the case of Web applications.

The formalization framework that we propose in this paper is based on some results presented in [14], where the formal validation of an interaction policy between communicating objects was carried out. The policy is the Multiple Levels of Integrity policy, defined in the context of the design of fault tolerant systems to enhance systems dependability. The original definition of the policy simply consists of a set of declarative rules: it can be operationally realized defining a communication protocol. The protocol which carries out the integrity policy is formally specified as a collection of interacting processes in a process algebra. We consider specific interaction patterns, which subsume the most complex interaction schemata, and check on them temporal logic formulae expressing the non-violation of integrity rules.

2 The Multiple Levels of Integrity policy

Integrity policies are defined in the field of fault tolerant systems. The design of fault tolerant systems usually includes the modeling of faults and failures or the definition of fault tolerant schemata. At the software architecture level, a fault tolerant schema usually describes a set of components and their interactions. A component that is part of a fault tolerant system is said to be *critical* if its failure can seriously affect the reliability of the overall system. Fault tolerant schemata, and in particular integrity policies, are defined to prevent failure propagation from non critical to critical components. An integrity policy assigns a *level of integrity*, ranging over a finite set of natural values, to each system component, and states the communication patterns. Components that may be critical are assigned a high integrity level.

The *Multiple Levels of Integrity* policy has been defined within an object–oriented framework, to provide flexible fault tolerant schemata. Instead of forbidding data flow from low level to high level objects, this policy permits some objects to receive low level data, by decreasing their integrity level. The policy is based on the following concepts:

Integrity levels (il) range from 0, the lowest, to 3, the highest. Data are assigned the integrity level of the object which produced them.

Single Level Objects (SLO) are objects whose integrity level does not change during computations. Consequently, an SLO of level n is only allowed to receive data from objects of level $\geq n$.

Multiple Level Objects (MLO) are the core of the policy: their integrity level can be dynamically modified, since they are allowed to receive low level data. To this purpose, an MLO is assigned three values:

maxil which represents the maximum integrity level that the MLO can have. It is also called the *intrinsic level* of the MLO, since it is assigned during the design of the application. It is a static value.

minil which represents the minimum value the integrity level of the MLO can reach while interacting with other objects. It is set at invocation time, on the bases of the invocation level. No memory of it is kept after the answer to the invocation is returned: *minil* is local to an invocation.

il which is the current integrity level. It is set at invocation time to a value ranging between *maxil* and *minil* and decreases if lower level data are received during the computation to serve the invocation. Also *il* is local to each invocation.

The policy requires a new MLO instance to be created every time the MLO is invoked. As a consequence, an MLO cannot be used to implement a component which has to store some data. This means that an MLO, from a functional point of view, is a stateless object: only SLOs can store data. In Fig. 1, we provide an example of the evolution of an MLO in response to an invocation: when an MLO with $maxil = 3$ receives a read request of level 1, it

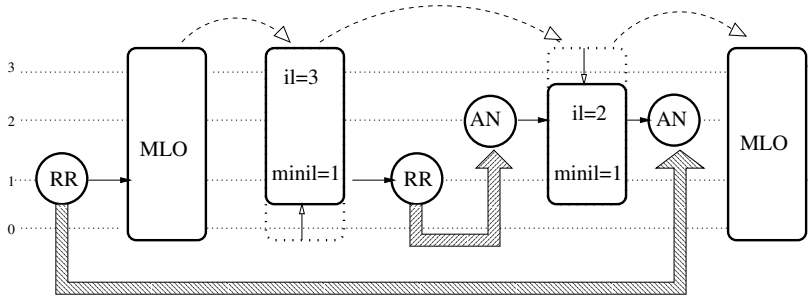


Fig. 1. Behaviour of an MLO: dotted arrows follow the MLO's evolution, thick arrows bind requests to the corresponding answers.

sets its *minil*: no answer with integrity level smaller than 1 can be returned. The value of *il* equals *maxil*: a read request does not corrupt the integrity level of the MLO. Suppose the MLO needs to delegate part of the answer construction, sending another read request to a third object. The level assigned to the request equals *minil*: an answer to this request is accepted if greater or equal to *minil*. Since the integrity level of the answer is 2, the MLO can accept it but *il* is decreased to level 2. Finally, an answer to the first request is provided, whose level equals the current *il*, and the MLO restores its initial state.

Validation Objects (VO) are used to extract reliable data from low level objects and to provide information at a fixed level of integrity. In real systems, it is sometimes necessary to get data from unreliable sources, such as sensors, and use them in critical tasks. However, this use could either lower the level of the entire system or violate the integrity policy. Validation Objects represent a safe way to upgrade the integrity level of these data. An example of Validation Object is the one that uses a redundant number of data sources, and filters them with appropriate algorithms. For instance, a voting policy can be used. These policies are well known in the literature, in particular in the distributed fault tolerant community. Among them, we recall the solutions to the Byzantine Generals problem [16], where an agreement among multiple nodes is sought in the presence of faults. To validate a voting algorithm we can apply the results presented in [2].

A set of **rules** is given, describing all the possible communication patterns among pairs of objects, depending on the respective integrity levels. We list them in Table 1: we call A and B the invoking and the invoked objects, respectively. The first part of the table considers invocation conditions. The invocation is refused if the specified condition is not satisfied. If it is accepted, the invoked object (if an MLO) might have to change its integrity level, as shown in the second part of the table, where invocation effects are considered. In the case of read or read–write invocation, an answer is returned at the end of the method execution. If the invoking object was an MLO, then the returned data may decrease its integrity level as follows: $il(A) := min(il(A), il(B))$.

The **communication model** is based on the notion of method invocation.

Conditions	$A\&B$ SLOs	A SLO, B MLO	A MLO, B SLO	$A\&B$ MLOs
A reads B	$il(A) \leq il(B)$	$il(A) \leq maxil(B)$	$minil(A) \leq il(B)$	$minil(A) \leq maxil(B)$
A writes B	$il(B) \leq il(A)$	*always*	$il(B) \leq il(A)$	*always*
A r-w B	$il(A) = il(B)$	$il(A) \leq maxil(B)$	$minil(A) \leq il(B) \leq il(A)$	$minil(A) \leq maxil(B)$

Effect	A SLO, B MLO	$A\&B$ MLOs
A reads B	$minil(B) := il(A)$; $il(B) := maxil(B)$	$minil(B) := minil(A)$; $il(B) := maxil(B)$
A writes B	$il(B) := min(il(A),\ maxil(B))$	$il(B) := min(il(A),\ maxil(B))$
A r-w B	$minil(B), il(B) := il(A)$	$minil(B) := minil(A)$; $il(B) := min(il(A),\ maxil(B))$

Table 1
Conditions to be satisfied for a method invocation to be accepted, and the effect on the level of objects after acceptance.

Method invocations are assigned an integrity level too. In particular, *read*, *write* and *read–write requests* are considered as abstractions of any method, with respect to the effect on the state of objects. The level of a write request corresponds to the level of the data which are written, the level of a read request corresponds to the minimum acceptable level of the data to be read. Read–write requests are assigned two integrity levels, one for read and one for write.

3 Formal Validation Methodology

The Multiple Levels of Integrity policy has been validated according to the following steps. We follow the same methodology to validate the case studies.

- Formal specification of the mechanism using the CCS process algebra [17]. Process algebras are based on a simple syntax and are provided with a rigorous semantics defined in terms of Labeled Transition Systems (LTSs). Process algebras are well suited to describing interaction policies, since a policy definition abstracts from the functionalities of the objects, and the relevant events to be specified are the object invocations (the actions) which may change the object integrity level (the state). In Table 2 we present the subset of the CCS operators used in the following.
- Use of the ACTL temporal logic [10] to describe the desired properties. ACTL is a branching-time temporal logic whose interpretation domains are LTSs. It is the action based version of CTL [13] and is well suited to expressing the properties of a system in terms of the actions it performs. We use a fragment of ACTL, given by the following grammar, where ϕ denotes a state property:

$a : P$	Action prefix	Action a is performed, and then process P is executed. Action a is in Act_τ
$P + Q$	Nondeterministic choice	Alternative choice between the behaviour of process P and that of process Q
$P \parallel Q$	Parallel composition	Interleaved executions of processes P and Q. The two processes synchronize on complementary input and output actions (i.e. actions with the same name but a different suffix)
$P \setminus a$	Action restriction	The action a can only be performed within a synchronization
$P = P'$	Process definition	It includes recursion

Table 2
A fragment of CCS syntax

$$\phi ::= true \mid \phi \,\&\, \phi' \mid [\mu]\phi \mid AG\,\phi \mid A[\phi\{\mu\}U\{\mu'\}\phi']$$

In the above rules μ is an action formula defined by:

$$\mu ::= true \mid a \mid \mu \vee \mu \mid \sim\mu \qquad \text{for } a \in Act$$

We provide here an informal description of the semantics of ACTL operators. The formal semantics is given in [10]. Any state satisfies *true*. A state satisfies ϕ & ϕ' if and only if it satisfies both ϕ and ϕ'. A state satisfies $[a]\phi$ if for all next states reachable with a, ϕ is true. The meaning of $AG\,\phi$ is that ϕ is true now and *always* in the future.

A state P satisfies $A[\phi\{\mu\}U\{\mu'\}\phi']$ if and only if in each path exiting from P, μ' will eventually be executed. It is also required that ϕ' holds after μ', and all the intermediate states satisfy ϕ; finally, before μ' only μ or τ actions can be executed. A useful formula is $A[\phi\{true\}U\{\mu'\}\phi']$ where the first action formula is *true*: this means that any action can be executed before μ'.

- Generation of the (finite state) model. To this end, we use the tools of the JACK (*Just Another Concurrency Kit*) verification environment [3], which is based on the use of process algebras, LTSs, and temporal logic formalisms, and supports many phases of the systems development process.
- Model checking of the ACTL formulae against the model, using the model checker for ACTL available in JACK, FMC.

3.1 Validation of the Multiple Levels of Integrity policy

The Multiple Levels of Integrity policy has to guarantee that the interaction among different components does not affect the overall confidence of the application, i.e., that a non–critical component does not corrupt a critical one. In particular, data of a low integrity level cannot flow to a higher integrity level

(unless through a Validation Object). This condition should hold for isolated objects and in any schema of interaction among objects. In [14], the following properties have been validated:

(i) An object with intrinsic level i cannot provide answers of level $j > i$.

(ii) An object with intrinsic level i does not accept read requests of level $j > i$.

(iii) If an MLO with intrinsic level i receives a read request of level $j \leq i$, and, to serve the request, it invokes with a read request a third object of intrinsic level *maxil* smaller than j, then it cannot answer the initial request. Indeed, its level is decreased to the *maxil* value of the third object because of the new data received.

(iv) If an MLO with intrinsic level i receives a read request of level $j \leq i$, and then a write request of level $k < j$, then it can still answer the read request. In other words, its level is not decreased by the concurrent invocation.

4 A concept of interface

The model proposed in [14] assumes that all the components of a system and their relationships are known. This assumption cannot be satisfied in the case of web site specification, since in most cases we only analyze a piece of the system, while of the remaining components we only know the interface toward the components of interest. We therefore need to define a formal concept of interface for a component of a system expressed in the CCS process algebra. This is accomplished using the restriction operator together with a dummy process which simulates the rest of the world. To be more precise, let P be a process over the set of actions Act_P. We could image to have a process W describing the rest of the world, thus we would like to verify the overall system $P \parallel W$. Of course, this is not possible, since we cannot specify all the possible components. Actually, since we are not interested in other communications than those among our process P and the rest of the world, we can restrict ourselves to study the process $(P \parallel W) \setminus (Act_W \smallsetminus Act_P)$, where Act_W is the set of actions of W. But this is equivalent to considering the process $P \parallel (W \setminus (Act_W \smallsetminus Act_P))$. Our idea is to consider, instead of the process $W \setminus (Act_W \smallsetminus Act_P)$ its interface toward P. To this aim, we need to introduce the notion of *dummy process*, that we use to separate the proper interface of W we are interested in. Let $D_{W,P}$ be the dummy process

$$\mathtt{D_{W,P}} = \mathtt{a_1} : \mathtt{D_{W,P}} + \mathtt{a_2} : \mathtt{D_{W,P}} + \ldots + \mathtt{a_n} : \mathtt{D_{W,P}} \tag{1}$$

where $\{a_1, \ldots, a_n\} = Act_W \setminus Act_P$. We define the *interface of W w.r.t. P* the process $W_P = (W||D_{W,P}) \setminus (Act_W \smallsetminus Act_P)$. Actually, for our purpose, any process trace-equivalent to W_P would suffice, that is any process which exhibits the same behaviour w.r.t. P when we observe only the traces of the

system. In the following, we call *interface* any of such processes. Thus, given any interface I of W w.r.t. P, we simply consider the system $P||I$.

For example, given $\mathtt{P} = ?\mathtt{request} :!\mathtt{ask\_google} :?\mathtt{read\_google} :!\mathtt{reply}$ we do not want to observe the interaction of the Google web site with the rest of the world, then we may choose $?\mathtt{ask\_google}$ and $!\mathtt{read\_google}$ as the only actions we are interested in, and which should be described in the interface of Google.

Our aim is to verify ACTL formulas on processes defined by CCS agents. Since we adopt the above concept of interface, we are particularly interested in those formulas such that, once proved for $P \parallel I$, where I is any interface of W w.r.t. P, they also hold for $P||W$. It is well–known that every formula in ACTL which does not contain any existential path quantifier E and negation $\sim$, enjoys the above property, since we can observe only the traces of the system. This motivates our choice of the ACTL fragment, as presented in Section 3.

5 Case Study: the Travel Agency

Our first case study concerns the modeling and analyzing of the architecture of the subset of the Web, which is of interest for a user willing to organize a travel by booking flights and hotels. The user interacts with an on–line travel agency. The travel agency, in turn, accesses the web sites of air companies, tour operators, single hotels as well as other travel agencies specialized in hotel booking, and so on. Here, we consider a simplified scenario, with two reliable sites, namely those of the travel agency and the air company Alitalia[4], and a fairly reliable hotel booking site.

We model Alitalia and the travel agency as MLOs with *maxil* 3, called $\mathtt{ALITALIA_3}$ and $\mathtt{TA_3}$, respectively, and the hotel booking site as HOTELSEEK, an MLO with *maxil* 2. All these sites are supposed to interact and receive data from many distinguished other components. We want them to perform their job even if the received data have a lower integrity level. At the same time, we recall that MLOs cannot store data: we can imagine that these components interact with some private SLOs, where to store the information of any finalized reservation. To exemplify this, we specify the SLOs of the travel agency, and call them disks. Since the travel agency is an MLO of level 3, it may be at any level when accessing its disks with a write request. We hence introduce 4 disks, one for each integrity level. They are specified parametrically by the process $\mathtt{DISK_x}$. We also need a disk manager, specified as an MLO of level 3, in order to choose the right $\mathtt{DISK_x}$ according to the integrity level.

[4] Disclaimer: The company names and the integrity level we use, are freely introduced for the purposes of the example, and have no correspondence with the reliability of the actual sites, when they exists.

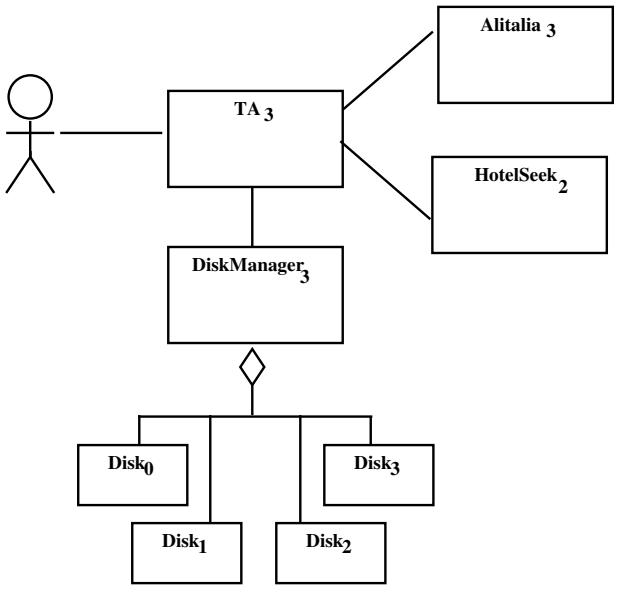


Fig. 2. The travel agency architecture.

The architecture of the resulting system is described in Figure 2. The full specification is given below, by instantiating the process defined in [14]. A disk can receive a read request when the travel agency needs to access previous reservations. $\texttt{read\_request}_{\texttt{x}}$ is a read request action of level `x`. In general, this means that the invocation was issued either by an SLO with `x` as *il* or by an MLO with `x` as *minil*. A disk can receive a write request too, when the travel agency needs to store a new reservation. Only requests at level `x` are served. A write request at a different level will be served by another disk.

```
DISK_MANAGER(3) =
     ?read_data(y). !read_disk(y).!answer_data(y).DISK_MANAGER(3) +
     ?write_data(y).!write_disk(y).DISK_MANAGER(3)

DISK_0 = ?read_disk(0).DISK_0 +
         ?write_disk(0).DISK_0
DISK_1 = ?read_disk(1).DISK_1 +
         ?write_disk(1).DISK_1
DISK_2 = ?read_disk(2).DISK_2 +
         ?write_disk(2).DISK_2
DISK_3 = ?read_disk(3).DISK_3 +
         ?write_disk(3).DISK_3
```

The agent `HOTELSEEK` accepts read–write hotel reservation requests, and write–only confirmation requests. $\texttt{r\_w\_hreserve}_{\texttt{y,z}}$ denotes a request issued either by an MLO with `y` as *minil* and `z` as *il* or by an SLO with $il = \texttt{y} = \texttt{z}$. Variable `y` denotes the read level of the request, variable `z` denotes the write level. $\texttt{w\_confirm}_{\texttt{y}}$ denotes a write request of level `y`, issued by an object with `y` as *il*. Hotel reservation requests are served as specified by process `HOTEL_RES`.

```
HOTELSEEK(2) = ?r_w_hreserve(y,z).!hotel.(
```

```
                    ( [y <= z] [z <= 2] HOTEL_RES(y,z,2) ) +
                    (   [y <= 2] [2 <= z] HOTEL_RES(y,2,2) ) +
                    (   [y > 2] !answer_hres(-1). HOTELSEEK(2) ) ) +
                ?w_confirm(y).  HOTELSEEK(2)

HOTEL_RES(min,il,max) =
      ( [min <= 0] [0 <= il]  !answer_hres(0). HOTELSEEK(2) )  +
      ( [min <= 1] [1 <= il]  !answer_hres(1). HOTELSEEK(2) )  +
      ( [min <= 2] [2 <= il]  !answer_hres(2). HOTELSEEK(2) )  +
      ( [min <= 3] [3 <= il]  !answer_hres(3). HOTELSEEK(2) )  +
      !answer_hres(-1). HOTELSEEK(2)
```

The Alitalia specification is very simple. A web site such as the Alitalia one can be implemented using a groupware protocol. These protocols address, among others, the concurrency control problems that arise in systems with multiple users (namely, groupware systems [1,12]) whose actions may be conflicting. A typical example is to reserve the same seat to two or more users that are concurrently booking a flight. The high integrity level of the Alitalia site can be guaranteed by formally specifying the protocol and by proving the non interference properties of interest. Validation can be done by model checking using, for instance, the results given in [18] where some properties of a public subscribe groupware protocol have been proved.

```
ALITALIA(3) = ?r_w_freserve(y,z). !flight.
                      [y <= z] !answer_fres(z). ALITALIA(3) +
                ?w_confirm(y). ALITALIA(3)
```

Finally, the travel agency.

```
TA(3) = ?r_w_booktravel(y,z). [y <= z] TA_BOOK(y,z,3) +
        ?r_infotravel(y). TA_INFO(y,3,3)

TA_BOOK(min,il,3) = F_BOOK(min,il,3) +
                    H_BOOK(min,il,3) +
                    F_BOOK(min,il,3).H_BOOK(min,il,3)

F_BOOK(min,il,3) = !r_w_freserve(min,il). ?answer_fres(x).
                      ( [x < min] !answer_booktravel(-1). TA(3) +
                        [min <= x] [x <= il] TA_FINALIZE(min,x,3) +
                        [il <= x]  TA_FINALIZE(min,il,3) )

H_BOOK(min,il,3) = !r_w_hreserve(min,il). ?answer_hres(x).
                      ( [x < min] !answer_booktravel(-1). TA(3) +
                        [min <= x] [x <= il] TA_FINALIZE(min,x,3) +
                        [il <= x]  TA_FINALIZE(min,il,3) )

TA_FINALIZE(min,il,3) = !write_data(il). !w_confirm(il).
                        !answer_booktravel(il). TA(3)

TA_INFO(min,3,3) = !read_data(min). ?answer_data(x).
```

```
                    ( [x < min] !answer_info(-1). TA(3) +
                      [x >= min] !answer_info(x). TA(3) )
```

We also need to specify a generic user of the system, which can ask for information or book a travel.

```
User(x) = !info. ( ( !r_infotravel(x). ?answer_info(y).
                   (  ( [y < 0  ] !failure. User(x) ) +
                      ( [y >= 0 ] !success. User(x) ) ) ) +
          !book. ( !r_w_booktravel(0,x). ?answer_booktravel(y).
                      (  ( [y < 0  ] !failure. User(x) ) +
                      ( [y >= 0 ] !success. User(x) ) ) )
```

In our test, we use a generic process consisting of the travel agency, the air company, the hotel booking site, a generic user of level 2 and the disks.

```
( HOTELSEEK(2) || ALITALIA(3) || TA(3) || User(2) || DISK_MANAGER(3)
  || DISK_0 || DISK_1 || DISK_2 || DISK_3 ) \read_data \answer_data
 \write_data \answer_hres \r_w_hreserve \w_confirm \r_w_freserve
 \answer_fres \r_w_booktravel \r_infotravel \answer_booktravel
 \answer_info \read_disk \write_disk
```

The only non restricted actions are `info`, `book`, `hotel` and `flight`. Therefore we will use them when specifying the ACTL formula. As a first example, we want to prove that, if a client requires a booking service (action !`book`), the travel agency will either book an hotel (action !`hotel`) or a flight (action !`flight`) before any positive answer (action !`success`). Formally, we require to verify the following ACTL formula:

```
AG [ !book ] A [ true { ~ !success } U { !hotel | !flight } true ]
```

The result of the model checker is that the formula is true and that 153 states has been observed. The following formula:

```
AG [ !info ] A [ true { ~ !success } U { !hotel | !flight } true ]
```

states that at any request of information will follow the booking of an hotel or a flight. Of course, in this case the result of the model checker is that the formula is false.

6 Case Study: Peer to Peer Validation Service

We here describe a peer to peer architecture that a user can query to download a video. This is a simplified instance of the concrete problem of identifying remote file content before downloading in peer to peer systems, where some or all of the peers are untrusted, or content-based access control has to be enforced. In the example we assume that two peers exist, at level 1 and 2 respectively. Moreover, the system includes two refutation lists which collect information of help to know whether the expected content of a file corresponds to the file name. The download is filtered by a Validation object that first looks for the video with a `read_video` request, and then validates the answer by querying the refutation lists. The system is described in Figure 3.

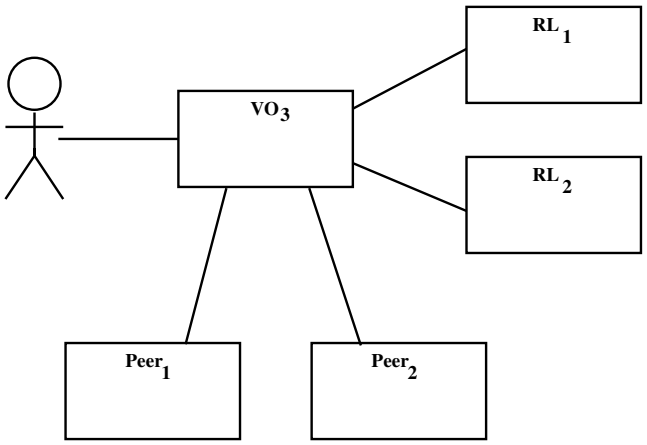


Fig. 3. The Peer to Peer Validation Service architecture.

A peer's answer to a `read_video` request carries two values: the peer integrity level, and an integer holding -1 if the peer does not have the video, a different value otherwise. If the video is not found from both peers `P`, the validator `VO` sends a negative answer to the user, otherwise it validates the video content with the help of the clauses of the agents `VAL` and `VAL2`. This involves querying one or more of the refutation lists processes `RL`.

In the example, we abstract from actual validation algorithms in `VAL` and `VAL2`, and show a completely non-deterministic behaviour that can be refined in any concrete solution. Our validation approach is compositional: to prove the correctness of the final system, we only need to validate the refinement step. Indeed, the abstract behaviour of the `VO` specified here corresponds to the interface of any actual validation object, with a specific validation algorithm demanded to the `VAL` agent.

To complete the example description, we assume that peers perform a visible action `video` when the video is available, and the user performs the visible actions `start` at search beginning, then `success`, or `failure`. The last two actions discriminate the cases where a valid video was found from the cases where either no video was found, or the video content was not correct.

```
P(x) = ?read_video(y). ( ( [y <= x] (!video. !answer_video(x,x). P(x) +
                                    !answer_video(x,-1). P(x) )       ) +
                         ( [y > x] !answer_video(-1,-1). P(x)         ) )

RL(x) = ?query_video(y). ( ( [y <= x]  !query_answer(x). RL(x)  ) +
                           ( [y > x]   !query_answer(-1). RL(x) )  )

VO(x) = ?user_req(y). ( ( [y <= x]  !read_video(0). ?answer_video(z,w).
                          ( ( [z = -1]  !user_answer(-1). VO(x) ) +
                            ( [z >=0]    VAL(x,w)              )  ) +
                      ( [y > x]   !user_answer(-1).VO(x)           )  )  )

VAL(x,w) = [w = -1] !user_answer(-1). VO(x) +
           [w >= 0] ( !query_video(0). ?query_answer(y).
```

```
                                    ( !user_answer(x). VO(x)  +
                                      !user_answer(-1). VO(x) +
                                       VAL2(x)                 )  )

VAL2(x) = !query_video(0). ?query_answer(y). ( !user_answer(x). VO(x) +
                                               !user_answer(-1). VO(x)  )

User(x) = !start. !user_req(x). ?user_answer(y).
              ( ( [y < 0  ] !failure. User(x) ) +
                ( [y >= 0 ] !success. User(x) )  )

net Net = ( VO(3) || P(1) || P(2) || RL(1) || RL(2) || User(1) )
        \read_video \query_video \user_req \answer_video \query_answer
        \user_answer
```

The validation process has lead to the generation of a model with 524 states, against which the following properties have been checked, returning the expected results.

```
AG [ !start ] A [ true { ~ !start } U { !failure | !video } true ]
--        The formula is TRUE              --

AG [ !start ] A [ true { true } U { !video } true ]
--        The formula is FALSE             --
```

7 The inverted Turing Test

The Inverted Turing test, proposed by Watt[19] as an alternative to the conventional Turing Test for artificial intelligence, requires:

- to put a system in the role of the observer;
- the observer to discriminate between humans and machines.

The machine that wants to mimic humans should show naive psychology, that faculty which predisposes us to anthropomorphism and enables us to ascribe intelligence to others. An example test is the one that asks many questions like "how close is a building to a house, how close is a hotel to a house, how close is a lodge to a house, how close is a cavern to a house", with answers in a finite range, say 1–100. The observer compares the answers to a table obtained by making the same questions to a sufficiently large population.

A variant of the Inverted Turing Test is the Editing Test [9], often used to discriminate humans from machines when assigning a new e-mail address. It is based on the so-called interpretative asymmetry, that is the asymmetry of the skillful way in which humans "repair" deficiencies in speech, written texts, handwriting, etc., and the failure of computers to achieve the same interpretative competence. For instance, an optical sequence of characters like the one in Figure 4 is printed on the screen, and the observed entity is asked to type the characters with the keyboard.

Fig. 4. The editing test: only humans are supposed to read the sequence of characters.

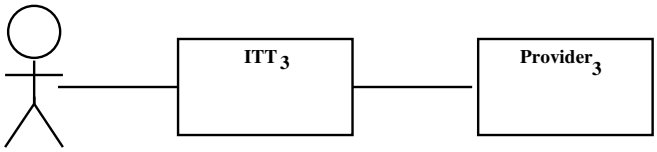


Fig. 5. The architecture of the subset of the WEB including an Inverted Turing test.

The component implementing the Inverted Turing test can be modeled in our framework as a validation object. The architecture of the subset of the WEB of interest can be modeled as described in Figure 5: the validation object intercepts the interactions between the entity (human or machine) asking for an e-mail address.

8 Conclusion

We have proposed a formal method to describe web applications by means of a process algebra which can be automatically verified by a model checker. By considering a fragment of the ACTL logic which does not contain negation and existential path quantification, we can introduce a formal notion of interface which allows us to prove properties expressed by temporal formulae in a modular way. We exploit the notion of validation object proposed in fault tolerant system verification and show examples of web applications where validation objects play a fundamental role. We describe in details two case studies validating some formulae with the help of the FMC model checker. Moreover, we briefly sketch another example where a commonly used web application can be easily modeled as a validation object. As a future work, we intend to investigate the possibility to separately verify different parts of the system and to compose the results.

References

[1] Baecker, R., editor, "Readings in Groupware and Computer Supported Cooperation Work–Assisting Human-Human Collaboration," 1992.

[2] Bernardeschi, C., A. Fantechi and S. Gnesi, *Formal validation of fault–tolerance*

mechanisms inside Guards, Reliability, Engineering and System Safety (RE&SS) **71** (2001), pp. 261–270, elsevier.

[3] Bouali, A., S. Gnesi and S. Larosa, *JACK: Just another concurrency kit*, Bulletin of the European Association for Theoretical Computer Science **54** (1994), pp. 207–224.

[4] Bowen, J. and M. Hinchey, *Seven more myths of formal methods*, IEEE Software **12** (1995), pp. 34–41.

[5] Burch, J., E.M.Clarke, K. McMillan, D. Dill and J. Hwang., *Symbolic Model Checking* 10^{20} *states and beyond*, in: *Proceedings of Symposium on Logics in Computer Science*, 1990.

[6] Clarke, E., E. Emerson and A. Sistla, *Automatic Verification of Finite–State Concurrent Systems Using Temporal Logic Specification*, ACM Transaction on Programming Languages and Systems **8** (1986), pp. 244–263.

[7] Clarke, E., O. Grumberg and D.Peled, "Model Checking," MIT Press, 1999.

[8] Clarke, E. and J. Wing, *Formal methods: state of the Art and Future Directions*, ACM Computing Surveys **28** (1996), pp. 627–643.

[9] Collins, H., *The editing test for the deep problem of AI*, Psycoloquy **8** (1997).

[10] De Nicola, R. and F. Vaandrager, *Action versus State based Logics for Transition Systems*, in: *Proceedings Ecole de Printemps on Semantics of Concurrency*, Lecture Notes in Computer Science **469** (1990), pp. 407–419.

[11] Dill, D., A. Drexler, A. Hu and C. H. Yang, *Protocol Verification as a Hardware Design Aid*, in: *IEEE International Conference on Computer Design: VLSI in Computers and Processors* (1992), pp. 522–525.

[12] Ellis, C. and S. Gibbs, *Concurrency control in groupware systems*, in: *In Proc. SIGMOD'89* (1989), pp. 399–407.

[13] Emerson, E. and J. Halpern, *Sometimes and Not Never Revisited: on Branching Time versus Linear Time Temporal Logic*, Journal of ACM **33** (1986), pp. 151–178.

[14] Fantechi, A., S. Gnesi and L. Semini, *Formal Description and Validation for an Integrity Policy Supporting Multiple Levels of Criticality*, in: C. Weinstock and J. Rushby, editors, *Proc. DCCA–7, Seventh IFIP International Conference on Dependable Computing for Critical Applications* (1999), pp. 129–146.

[15] Holzmann, G., *The Model Checker SPIN*, IEEE Transaction on Software Engineering **5** (1997), pp. 279–295.

[16] Lamport, L., R. Shostack and M. Pease, *The Byzantine Generals problem*, ACM Transactions on Programming Languages and Systems **4** (1982), pp. 282–401.

[17] Milner, R., "A Calculus of Communicating Systems," Lecture Notes in Computer Science **92**, Springer-Verlag, Berlin, 1980.

[18] ter Beek, M., M. Massink, D. Latella and S. Gnesi, *Model checking groupware protocols.*, in: F. Darses, R. Dieng, C. Simone and M. Zacklad, editors, *Cooperative Systems Design - Scenario-Based Design of Collaborative Systems*, Frontiers in Artificial Intelligence and Applications **107** (2004), pp. 179–194.

[19] Watt, S., *Naive-psychology and the inverted turing test*, Psycoloquy **7** (1996).

Verification of Web Services with Timed Automata

Gregorio Diaz [2] Juan-José Pardo [3] María-Emilia Cambronero [4]
Valentín Valero [5] Fernando Cuartero [6]

Departamento de Informática
Universidad de Castilla-La Mancha
Escuela Politécnica Superior de Albacete. 02071 - SPAIN

Abstract

In this paper we show how we can use formal methods for describing and analyzing the behavior of Web Services, and more specifically those including time restrictions. Then, our starting point are Web Services descriptions written in WSCI - WSCDL (XML-based description languages). These descriptions are then translated into timed automata, and then, we use a well known tool that supports this formalism (UPPAAL) to simulate and analyze the system behavior. As illustration we take a particular case study, a travel reservation system.

1 Introduction

Nowadays the society model is changing. Our society is based on the information exchange due to the growth of Internet and Telecommunications. For example in the European Union the annual expenditure on ICT (Information and Communication Technology) amounted to an estimated of more than 500 billion EUR which was approximately 6% of total Gross Domestic Product. And the Internet access has increased for household and enterprises. In 2003, the access level of household to the Internet was 45%. The access of enterprisers was higher, reaching in some countries over 90% of all enterprises (source: EUROSTAT [8]).

[1] This work has been supported by the CICYT project "Description and Evaluation of Distributed Systems and Application to Multimedia Systems",TIC2003-07848-C02-02.
[2] Email:gregorio@info-ab.uclm.es
[3] Email:jpardo@info-ab.uclm.es
[4] Email:emicp@info-ab.uclm.es
[5] Email:valentin@info-ab.uclm.es
[6] Email:fernando@info-ab.uclm.es

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

Due to this change in the society model it becomes necessary to increase the research in the development of systems based in Internet, whose objective is to develop solutions for automating their peer-to-peer collaborations, in an effort to improve productivity and reduce operating costs.

Thus, in the last years some new techniques and languages for developing this kind of distributed systems have appeared, such as the Extensible Markup Language, XML [14], and some new Web Services frameworks [5,9,15] for describing interoperable data and platform neutral business interfaces, enabling more open business transactions to be developed.

Web Services are a key component of the emerging, loosely coupled, Web-based computing architecture. A Web Service is an autonomous, standards-based component whose public interfaces are defined and described using XML [11]. Other systems may interact with a Web Service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.

The Web Services specifications offer a communication bridge between the heterogeneous computational environments used to develop and host applications. The future of E-Business applications requires the ability to perform long-lived, peer-to-peer collaborations between the participating services, within or across the trusted domains of an organization.

The Web Service architecture stack targeted for integrating interacting applications consists of the following components [11]:

- **SOAP[9]:** It defines the basic formatting of a message and the basic delivery options independent of programming language, operating system, or platform. A SOAP compliant Web Service knows how to send and receive SOAP-based messages.
- **WSDL[15]:** It describes the static interface of a Web Service. Then, at this point the message set and the message characteristics of end points are here defined. Data types are defined by XML Schema specifications, which support rich type definitions and allow expressing any kind of XML type requirement for the application data.
- **Registry[5]:** It makes visible an available Web Service and allows the service requesters to discover it by means of relatively sophisticated searching mechanims. It also describes the concrete capabilities of a Web Service.
- **Security layer:** Its goal is to ensure that exchanged informations are not modified or forged in a verifiable manner and that parties can be authenticated.
- **Reliable Messaging layer:** It provides a reliable layer for the exchange of information between parties, guaranteeing the delivery of information with an exactly-once semantics.
- **Context, Coordination and Transaction layer:** It defines interoperable mechanisms for propagating context of long-lived business transactions and enables parties to meet correctness requirements by following a global

agreement protocol.

- **Business Process Languages layer[2,3]:** It describes the execution logic of Web Services based applications by defining their control flows (such as conditional, sequential, parallel and exceptional execution) and prescribing the rules for consistently managing their non-observable data.
- **Choreography layer[11]:** It describes collaborations of parties by defining from a global viewpoint their common and complementary observable behavior, where information exchanges occur, when the jointly agreed ordering rules are satisfied.

The Web Services Choreography specification is aimed at the composition of interoperable collaborations between any type of party regardless of the supporting platform or programming model used by the implementation of the hosting environment.

Web Services cover a wide range of systems, which in many cases have strong time constraints (for instance, peer-to-peer collaborations may have time limits to be completed). Then, in many Web Services descriptions these time aspects can become very important. Actually, they are currently covered by the top level layers in Web Services architectures with elements such as time-outs and alignments. Time-outs allow to each party to fix the available time for an action to occur, while alignments are synchronizations between two peer-to-peer parties.

Thus, it becomes important for Web Services frameworks to ensure the correctness of systems with time constraints. For instance, we can think in a failure of a bank to receive a large electronic funds transfer on time, which may result in huge financial losses. Then, there is growing consensus that the use of formal methods, development methods based on some formalism, could have significant benefits in developing E-business systems due to the enhanced rigor these methods bring [10]. Furthermore, these formalisms allow us to reason with the constructed models, analysing and verifying some properties of interest of the described systems. One of these formalisms are timed automata [1], which are very used in practice and there are some well-known tools supporting them, like UPPAAL [6,7,12] and KHRONOS [4].

Then, our goal with this paper is to describe how we can verify Web Services with time constraints using model checking techniques. This verification process starts from the top level layers of Web Services architectures (Business Process Language Layer and Choreography layer). The particular Business Process Language layer that we use here is the Web Service Choreography Interface (WS-CI) [2], and the concrete Choreography Layer that we use is the Web Service Choreography Description Language (WS-CDL) [11]. Therefore, the starting point are specification documents written in WS-CDL and WS-CI. However, these description languages are not very useful for the verification process. Thus, these descriptions are translated into timed automata, and the UPPAAL tool is used to simulate and verify the correctness

of the system.

As illustration of this methodology of verification we use a particular case study, which is an airline ticket reservation system, whose description contains some time constraints.

The paper is structured as follows. In Section 2 we present the case study that will be used to illustrate the methodology we propose for the verification of Web Services with time restrictions. In Section 3 we describe WSCI - WSCDL and how they are used to describe the case study. In Section 4 we show how we can model the case study and we use the UPPAAL tool to simulate and verify the system behavior. Finally, the conclusions and the future work are presented in Section 5.

2 Case Study: Travel Reservation System

In this section we present the case study that we consider in order to illustrate our methodology of verification. The scenario consists of three participants: a Traveler, a Travel Agent and an Airline Reservation System, whose behavior is as follows:

A Traveler is planning on taking a trip. Once he has decided the concrete trip he wants to make he submits it to a Travel Agent by means of his local Web Service software (*Order Trip*). The Travel Agent selects the best itinerary according to the criteria established by the Traveler. For each leg of this itinerary, the Travel Agent asks the Airline Reservation System to verify the availability of seats (*Verify Seats Availability*). Thus, the Traveler has the choice of accepting or rejecting the proposed itinerary, and he can also decide not to take the trip at all.

- In case he rejects the proposed itinerary, he may submit the modifications (*Change Itinerary*), and wait for a new proposal from the Travel Agent.
- In case he decides not to take the trip, he informs the Travel Agent (*Cancel Itinerary*) and the process ends.
- In case he decides to accept the proposed itinerary (*Reserve Tickets*), he will provide the Travel Agent with his Credit Card information in order to properly book the itinerary.

Once the Traveler has accepted the proposed itinerary, the Travel Agent connects with the Airline Reservation System in order to reserve the seats (*Reserve Seats*). However, it may occur that at that moment no seat is available for a particular leg of the trip, because some time has elapsed from the moment in which the availability check was made. In that case the Travel Agent is informed by the Airline Reservation System of that situation (*No seats*), and the Travel Agent informs the Traveler that the itinerary is not possible (*Notify of Cancellation*). Once made the reservation the Travel Agent informs the Traveler (*Seats Reserved*). However, this reservation is only valid for a

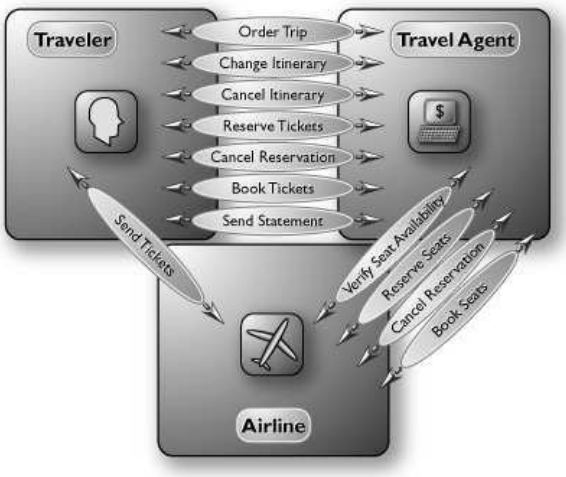


Fig. 1. Flow of the messages exchanged.

period of just one day, which means that if a final confirmation has not been received in that period, the seats are unreserved and the Travel Agent is informed. Thus, the Traveler can now either finalize the reservation or cancel it. If he confirms the reservation (*Book Tickets*), the Travel Agent asks the Airline Reservation System to finally book the seats (*Book Seats*).

According to the previous description, the high level flow of the messages exchanged within the global process (which is called *PlanAndBookTrip*) is that shown in Fig. 1, and a more complete description, including the actions performed by each participant is shown in Fig. 2.

3 The WSCI - WSCDL Description

The Web Services Choreography specification is aimed at being able to precisely describe collaborations between any type of party, regardless of the supporting platform or programming model used by the implementation of the hosting environment. Using the Web Services Choreography specification, a contract containing a "global" definition of the common ordering conditions and constraints under which messages are exchanged is produced that describes, from a global viewpoint, the common and complementary observable behavior of all the parties involved. Each party can then use the global definition to build and test solutions that conform to it. The global specification is in turn realized by combination of the resulting local systems, on the basis of appropriate infrastructure support.

In real-world scenarios, corporate entities are often unwilling to delegate control of their business processes to their integration partners. Choreography offers a means by which the rules of participation within a collaboration can be clearly defined and agreed to, jointly. Each entity may then implement its portion of the Choreography as determined by the common or global view.

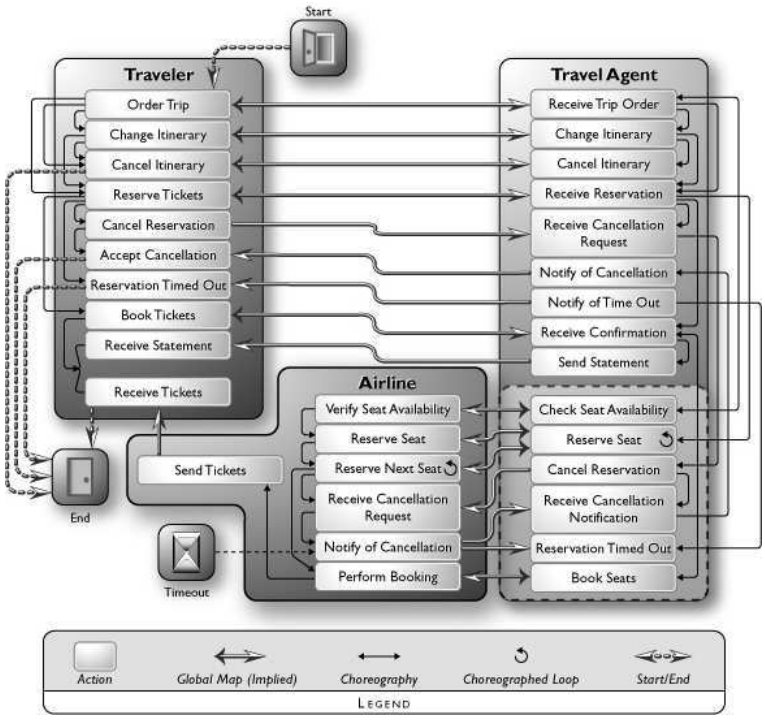


Fig. 2. Overall picture of *PlanAndBookTrip*.

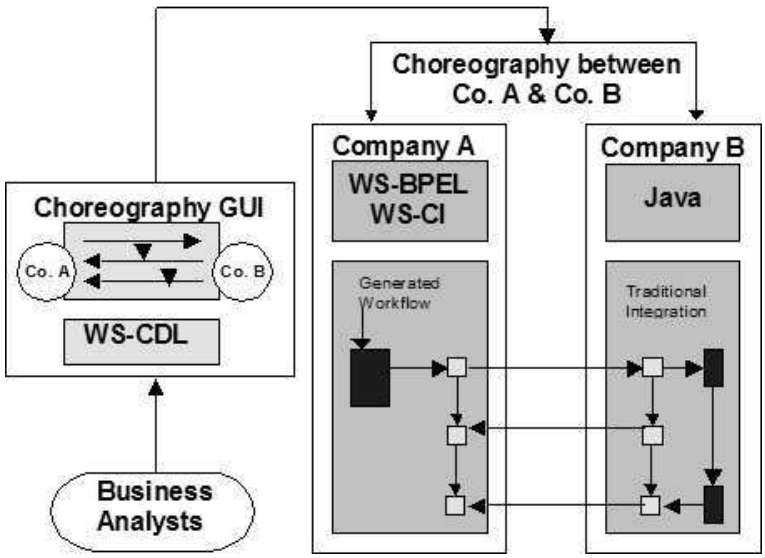


Fig. 3. WS-CDL and WS-CI usage.

It is the intent of WS-CDL that the conformance of each implementation to the common view expressed in WS-CDL is easy to determine. Figure 3 shows a possible usage of the Choreography Description Language. In the particular example we are using we take WS-CI as the Business Process Execution Layer (BPEL for short). However, before that we must provide the WS-CDL description.

WS-CDL describes interoperable collaborations between parties. In order to facilitate these collaborations, services commit to mutual responsibilities by establishing relationships. Their collaboration takes place in a jointly agreed set of ordering and constraint rules, whereby information is exchanged between the parties. The WS-CDL model consists of the following entities:

- **Participant Types, Role Types and Relationship Types**: within a Choreography the information is always exchanged between parties within or across trust boundaries. A Role Type enumerates the observable behavior a party exhibits in order to collaborate with other parties. A Relationship Type identifies the mutual commitments that must be made between two parties for them to collaborate successfully. A Participant Type is grouping together those parts of the observable behavior that must be implemented by the same logical entity or organization.
- **Information Types, Variables and Tokens**: Variables contain information about commonly observable objects in a collaboration, such as the information exchanged or the observable information of the Roles involved. Tokens are aliases that can be used to reference parts of a Variable. Both Variables and Tokens have Types that define the structure of what the Variable contains or the Token references.
- **Choreographies**: They define collaborations between interacting parties:
  - **Choreography Life-line**, which expresses the progression of a collaboration. Initially, the collaboration is established between parties, then work is performed within it and finally it completes either normally or abnormally.
  - **Choreography Exception Block**, which specifies the additional interactions should occur when a Choreography behaves in an abnormal way.
  - **Choreography Finalizer Block**, which describes how to specify additional interactions that should occur to modify the effect of an earlier successfully completed Choreography (for example to confirm or undo the effect).
- **Channels**: They establish a point of collaboration between parties by specifying where and how information is exchanged.
- **Work Units**: They prescribe the constraints that must be fulfilled for making progress and thus performing actual work within a Choreography.
- **Activities and Ordering Structures**: Activities are the lowest level components of the Choreography that perform the actual work. Ordering Structures combine activities with other Ordering Structures in a nested structure to express the ordering conditions in which information within the Choreography is exchanged.
- **Interaction Activity**: It is the basic building block of a Choreography, which results in an exchange of information between parties and possible synchronization of their observable information changes and the actual val-

```
  <interaction name="reservation&booking"
               channelVariable="travelAgentAirlineChannel"
               operation="reservation&booking"
               align="true"
               initiate="true" >
    <participate  relationshipType="TravelAgentAirline"
               fromRole="TravelAgent" toRole="Airline" />
    <exchange  name="reservation"
               informationType="reservation" action="request" >
      <send    variable="tns:reservationOrderID" causeException="true" />
      <receive variable="tns:reservationAckID" causeException="true" />
    </exchange>
    <exchange  name="booking" informationType="booking" action="respond" >
      <send    variable="tns:bookingRequestID" causeException="true" />
      <receive variable="bookingAckID" causeException="true" />
    </exchange>
    <timeout time-to-complete="24:00" />
    <record  name="bookingTimeout" when="timeout" causeException="true" />
      <source
        variable="AL:getVariable('tns:reservationOrderCancel', '', '')" />
      <target
        variable="TA:getVariable('tns:reservationOrderCancel', '', '')" />
    </record>
</interaction>
```

Fig. 4. Part of the WS-CDL specification

ues of the exchanged information.

- **Semantics**: It allows the creation of descriptions that can record the semantic definitions of every component in the model.

Figure 4 shows a part of the WS-CDL document that describes our case study. This part shows the relationship between the Airline and the Travel Agent. We can see that this interaction description determines that the maximum time a reservation is available is just of one day.

3.1 WSCI

WSCI is an interface description language. It describes the observable behavior of a service and the rules for interacting with the service from outside. It is not an executable language but it is precise and unambiguous enough.

The observable behavior of each party in a message exchange is described independently of the others.

The basic construct of WSCI is the Action, which is bound to some WS-CDL operation.

The main concepts in WSCI language are the following:

Interface: WSCI maps the description of a web service to the notion of interface.

Activities and choreography of activities: WSCI describes the behavior

of a Web Service in terms of choreographed activities. A choreography describes temporal and logical dependencies among activities, where atomic activities represent the basic unit of behavior of a Web Service.

Processes and units of reuse: A process is a portion of behavior labeled with a name. It can be reused by referencing its name.

Properties: It allows us to reference a value within the interface definition. They are the equivalent of variables on other languages.

Context: It describes the environment in which a set of activities is executed. Each activity is defined in exactly one context definition.

Message correlation: It describes how conversations are structured and which properties must be exchanged to do the service correctly.

Exceptions: The definiton of exception is part of the context definition. There are three kinds of exceptions and when an exception occurs the current context must terminate after the activities associated with the exception have been performed.

Transactions and compensation activities .- A transaction asserts that a set of activities is executed in an all-or-nothing way. A transaction may declare a set of compensation activities that will be executed if the transaction has completed successfully, but needs to be undone.

Global model: The global model is described by a collection of interfaces of the participating services and a collection of links between the operations of communicating services.

3.2 Example. Travel Reservation System

We now present the modeling details for the case study under consideration.

3.2.1 Travel Agent Interface

The model for the travel agent has the following elements:

- The main activities of the travel agent are represented via nested processes.
- The iterative processes are described by means of *while* activities.
- We use exceptions to capture the withdrawal of the trip request or the reservation request.
- The interface uses two different correlations, which identify the same conversation involving the travel agent with both the traveler and the airline reservation system.

Figure 5 shows a part of travel agent specification, in which an exception to handle the reservation timeout is defined.

```
...
<context>
 <process name ="BookSeats" instantiation="other">
  <action name="bookSeats"
          role="tns:travelAgent"
          operation="tns:TAtoAirline/bookSeats">
  </action>
 </process>
 <exception>
  <onMessage>
   <action name="ReservationTimeOut"
           role="tns:TravelAgent"
           operation="tns:TAtoAirline/AcceptCancellation">
      <correlate
          correlation="defs:reservationCorrelation"/>
   </action>
   <action name="NotifyOfTimeOut"
           role="tns:TravelAgent"
           operation="tns:TAtotraveler/NotifyofCancellation"/>
                     <fault code="tns:reservationTimedOut"/>
  </onMessage>
  ...
  </exception>
  ...
  </context>
```

Fig. 5. Part of the Travel Agent Specification

3.2.2 Traveler Interface

The main top-level process describing the Traveler is declared with *instantiation=other* attribute to describe the fact that the traveler is actually the entity starting the message exchange. Notice that the model captures the possibility of canceling the reservation or the ticket booking, by means of a new context with a new exception.

We use a correlation to ensure that both the travel agent and the airline reservation system know how to fulfill the correlation requirements exhibited by the traveler interface.

3.2.3 Airline Reservation System

The airline reservation system interface is modeled by an interface with two top-level processes, both with the *instantiation=message* attribute.

The seat reservation for each leg is defined as a transaction which defines a compensation activity which probably will withdraw the reservations for all seats.

Figure 6 shows a part of the specification (the timeout control).

```
 ...
<sequence>
 <context>
  <exception>
   <onTimeout property ="tnsd:expireTime"
              type="duration"
              reference="tns:ReserveSeats@end">
      <compensate name="CompensateReservation"
      transaction="seatReservation"/>
   </onTimeout>
  </exception>
 </context>
   ...
</sequence>
```

Fig. 6. Part of the Travel Agent Specification

4 Modeling, Simulation and Verification

The previous descriptions can be translated into timed automata, thus obtaining three automata, which correspond to the traveler, the travel agent and the airline company. These automata are shown in Figures 7, 8 and 9.

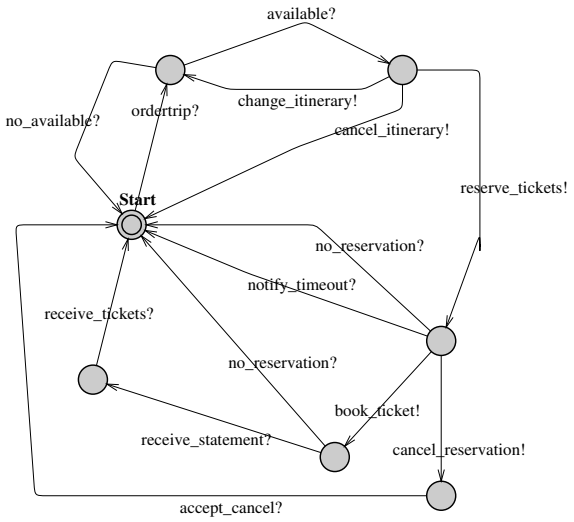


Fig. 7. Timed automaton for Traveler.

Notice the use of clock x in Fig. 8, to control when the reservation expires. This clock is initialized once *reserved_seat* is done.

By means of simulations we can check whether or not the system model holds the expected behavior. These simulations are made by choosing different transitions and delays along the system evolution. At any moment during the simulation, you can see the variable values and the enabled transitions that you can select. Thus, you can choose the transition you want to execute. Nevertheless, you can also select a random execution of transitions, and thus, the

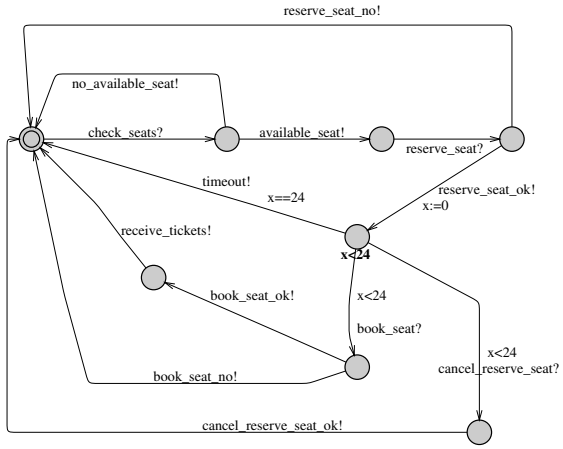


Fig. 8. Timed automaton for Airline Reservation System.

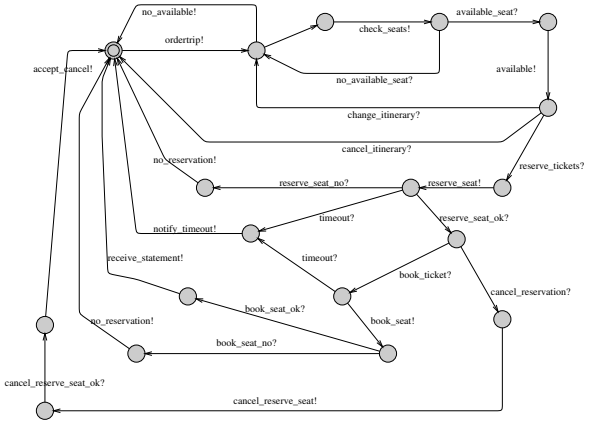


Fig. 9. Timed automaton for Travel Agent.

system evolves by executing transitions and delays in a random way. We have some other options in the Simulator. For instance, you can save simulations traces that can be later used to recover a specific execution trace. Actually, the simulation is quite flexible at this point, and you can back or forward in the sequence.

Then, our main goal in the validation phase of our case study is to check the correctness of the message flow and time-outs, taking into account the protocol definition. We have made a number of simulations, and we have concluded that the system design satisfies the expected behavior in terms of the message flow between the parties.

Before starting the automatic verification, we must establish which are the properties that the model must fulfill. We have divided these properties into three classes: Safety, Liveness and Deadlocks. These properties are specified by means of a *Temporal Logic*, and all of them have been checked by using the UPPAAL tool. The temporal Logic used by UPPAAL is described in [13].

Safety Properties: They allow us to check if our model satisfies some security restrictions. For example, if we have two trains that have to cross the same bridge, a security property is that both trains cannot cross at the same time the bridge:

$$\forall\Box\neg(Train1.crossing \wedge Train2.crossing) \text{ or}$$
$$\neg\exists\Diamond(Train1.crossing \wedge Train2.crossing)$$

The main Safety properties for our case study are the following:

- The TravelAgent always sends the itinerary on traveler's demand:

(1) $$\forall\Box Traveler.Itinerary \Rightarrow TravelAgent.sendItinerary$$

- The TravelAgent always changes the itinerary on traveler's demand:

(2) $$\forall\Box Traveler.ChangeItinerary \Rightarrow TravelAgent.PerformChange$$

- The TravelAgent always cancels the reservation on traveler's demand:

(3) $$\forall\Box Traveler.CancelReservation \rightarrow (TravelAgent.CancelReservtRcv \wedge Airline.PerformCancel \wedge Airline.Clockx < 24)$$

- A reservation is only available 24 hours before performing the booking:

(4) $$\forall\Box(TravelAgent.Booking \wedge Airline.ReceiveBoking \wedge Airline.ClockX <= 24)$$

- A Traveler always receives his tickets and the statement after the payment:

(5) $$\forall\Box Traveler.PaymentPerform \rightarrow (Traveler.Finish \wedge Airline.SnddTckt \wedge TravelAgent.SenddSttment)$$

Liveness Properties: They intend to check that our model can evolve in the right order. Returning to the train example, if a train approaches the bridge, some time later the train will be able to cross it:

$$Train.approach \rightarrow Train.crossed$$

Liveness Properties for our model are simple, for instance, if a Traveler sends a trip demand, some time later the TravelAgent will send the itineraries. Translating it into Temporal Logic we have:

(6) $$Traveler.PlanOrder \longrightarrow TravelAgent.SendItinerary$$

Another liveness property of interest is the following: if a Traveler orders a book within the next 24 hours after the reservation, the Airline performs the booking. Translating it into Temporal Logic we have:

(7) $$(Traveler.BookOdr \wedge Airline.ClockX < 24) \longrightarrow Airline.PerformBook$$

Deadlocks: These are clear restrictions. We could check if our model is deadlock free with the following formula:

(8) $$\forall\Box\neg Deadlock$$

5 Conclusions and Future Work

In this paper we have shown how we can apply formal methods to ensure the correctness of Web Services with time restrictions. We have shown that we can translate the descriptions written in WSCI-WSCDL into timed automata, and thus, we can use the UPPAAL tool to simulate and verify the system behavior.

In the particular case study we have used to illustrate how this methodology works (the airline ticket reservation system) this translation has been made manually, but our intention is to study if this translation can be made automatically, and in that case to implement a tool supporting this translation.

References

[1] R. Alur and D. Dill. *Automata for modeling real–time systems.* In Proceedings of the 17th International Colloquium on Automata, Languages and Programming, volume 443, Editors. Springer–Verlag, 1990.

[2] Assaf Arkin et al. *Web Service Choreography Interface (WSCI) 1.0.* In http:/www.w3.org/TR/wsci/.

[3] Assaf Arkin, Sid Askary, Ben Bloch, et. al., *Web Services Business Process Execution Language Version 2.0*, Editors. OASIS Open, December 2004. In http://www.oasis-open.org/committees/download.php/10347/wsbpel-specification-draft-120204.htm.

[4] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis and S. Yovine. *Kronos: A model-checking tool for real-time systems.* In Proc. 1998 Computer-Aided Verification, CAV'98, Vancouver, Canada, June 1998. Lecture Notes in Computer Science 1427, Springer-Verlag.

[5] Luc Clement, Andrew Hately, Claus von Riegen and Tony Rogers. *UDDI Version 3.0.2*, Editors. OASIS Open, 19 October 2004. In http://uddi.org/pubs/uddi_v3.htm.

[6] G. Diaz, F. Cuartero, V. Valero and F. Pelayo. *Automatic Verification of the TLS Handshake Protocol.* In proceedings of the 2004 ACM Symposium on Applied Computing.

[7] G. Diaz, K.G. Larsen, J. Pardo, F. Cuartero and V. Valero. *An approach to handle Real Time and Probabilistic behaviors in e-commerce: Validating the SET Protocol.* In proceedings of the 2005 ACM Symposium on Applied Computing.

[8] Eurostat yearbook 2004. *The statistical guide to Europe. Data 1992-2002.* European Commission: EUROSTAT, Office for Official Publications of the European Communities, 2004

[9] Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, et. al. *SOAP Version 1.2 Part 1: Messaging Framework*, Editors. World Wide Web Consortium, 24 June 2003. In http://www.w3.org/TR/soap12-part1.

[10] Constance Heitmeyer and Dino Mandrioli. *Formal Methods for Real-Time Computing*. John Wiley & Sons. 1996.

[11] Nickolas Kavantzas et al. *Web Service Choreography Description Language (WSCDL) 1.0*. In http://www.w3.org/TR/ws-cdl-10/.

[12] K. Larsen and P. Pettersson and Wang Yi. Uppaal *in a Nutshell*. Int. Journal on Software Tools for Technology Transfer, Editors. Springer–Verlag vol.1, 1997.

[13] Kim G. Larsen, Paul Petterson, Wang Yi. *Diagnostic Model-Checking for Real-Time Systems*. Proc. of Workshop on Verification and Control of Hybrid Systems III, Lecture Notes in Computer Science, vol. 1066, pp. 575-586, 1995.

[14] Jean Paoli, Eve Maler, Tim Bray, et. al.*Extensible Markup Language (XML) 1.0 (Third Edition)*, Editors. World Wide Web Consortium, 04 February 2004. In http://www.w3.org/TR/2004/REC-xml-20040204.

[15] Sanjiva Weerawarana, Roberto Chinnici, Martin Gudgin, et. al. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, Editors. World Wide Web Consortium, 03 August 2004. In http://www.w3.org/TR/2004/WD-wsdl20.

[16] Simon Woodman, et al. *Specification and Verification of Composite Web Services*. In proocedings of The 8th Enterprise Distributed Object Computing Conference 2004.

Improving the Quality of Web-based Enterprise Applications with Extended Static Checking: A Case Study

Frédéric Rioux [1], Patrice Chalin [1]

Dependable Software Research Group (DSRG)
Department of Computer Science and Software Engineering
Concordia University
Montréal, Canada

Abstract

ESC/Java2 is a tool that statically detects errors in Java programs and that uses the Java Modeling Language (JML) as its annotation language. ESC/Java2 can modularly reason about the code of a Java Web-based Enterprise Application (WEA) and uncover potential errors. In this paper, we assessed the effectiveness of ESC/Java2 at helping developers increase WEA quality by detecting design and implementation issues.

Key words: Web-based Enterprise Application, Extended Static Checking, Design by Contract, Java Modeling Language

1 Introduction

The evolution of programming languages has allowed software engineers to develop increasingly larger software systems while maintaining, or improving, product quality. This has been achieved in part by increasing the level of abstraction of the languages, exploiting new paradigms (such as object-orientedness), and enabling compilers to perform static checks and/or embed run-time checking code when the former is neither possible nor practical. In light of the above, one may ask: what will be the next programming language and tool advances that are likely to go mainstream? We believe that for most domains of application, these advances are likely to include Design by Contract (DBC) and Extended Static Checking (ESC). DBC can gradually be integrated into projects as a lightweight formal method allowing developers to become accustomed to the method. Among others, lint-like [7] ESC tools

[1] Email: {f_rioux, chalin}@cse.concordia.ca

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

— familiar to many programmers — can then make use of this extra formal documentation to perform rapid and effective checks.

Enterprise Applications (EAs) are generally characterized by the large volume of data they manipulate, the business rules they embed, and their need to integrate with other (often legacy) applications [6]. An important class of EAs are web-based enterprise applications (WEAs) that businesses and consumers are increasingly coming to make use of — in the third quarter of 2004 alone, retail e-commerce sales in the US have been estimated at $17.6 billion US, an increase of 4.7% from the previous quarter. This figure has been on the rise since 1999 [16].

This paper presents the preliminary results of a case study whose main goal is to assess the effectiveness of ESC/Java2 [1][2][4] at helping developers increase WEA quality by detecting design and implementation issues. In doing so, we wrote lightweight JML specifications for some *javax* and *java.sql* classes that can be reused not only by ESC/Java2 users, but by the whole JML community. The subject of our study is a collection of WEAs mainly based on a small framework (named *SoenEA*) that has been in use at Concordia for almost two years now in courses where WEA architecture and design is being taught.

This paper is organized as follows. DBC, ESC and ESC/Java2 are covered in Section 2. Section 3 reiterates our goals and presents the case study including an explanation for our choice of application domain. The remaining sections offer a discussion of the case study results, future work and conclusion.

2 Background

2.1 Design by Contract (DBC)

DBC is an approach to design which views the relationship between two classes — a supplier and a client — as a formal agreement, or a contract [12]. Such an agreement expresses each party's rights and obligations. Contracts usually take the form of preconditions, post-conditions, and invariants.

Contracts are a form of module specifications. As such, it is possible to raise our level of trust of large scale and complex systems if their contracts are unambiguous, correct, and verifiable. DBC is currently supported by the Eiffel programming language and some specification languages, including Behavioral Interface Specification Languages (BISLs).

A BISL is a language that can be used to accurately describe a module's interface, hence, by definition, implementing DBC. The main characteristic of a BISL is that it is tailored to a particular programming language. BISLs are an interesting kind of specification language, since they can be used to bridge the gap between the specification and design activities of the software development lifecycle. They also set the basis for powerful and automatic verification. Moreover, they are flexible, allowing them to be introduced incrementally, as

```
//@ requires y >= 0;
public static int isqrt(int y){
  ...
}
```

Fig. 1. Sample lightweight specification

needed, to new developments as well as legacy code.

The Java Modeling Language (JML) is a Java BISL that is actively being developed by an international team of researchers who collaborate on the language definition and tool implementation. Details on the latest tools and applications can be found in [1]. Use of JML for DBC is described in [8].

JML specifications can be embedded in Java source file as specially formatted comments or in external specification files. A JML expression look a lot like a Java expression since JML keeps most of Java's syntax and semantics [10]. Such a tight coupling between Java and JML lowers the burden required of developers to learn and use JML.

JML supports both lightweight and heavyweight specifications. Lightweight specifications are less detailed and complete than heavyweight specifications and are often composed of individual clauses, describing only one aspect of the intended behavior. JML was designed to support lightweight specifications and has semantics that allow most clauses to be omitted.

In JML, the specifications are usually written just before the methods. Preconditions are represented by the *requires* clause and the post-condition by the *ensures* clause. Every clause is a Boolean expression. Method calls can be used in the specification, but they must be calls to methods declared as *pure*, i.e. that they have no side-effects [9].

Figure 1 and Figure 2 show partial and more detailed specifications, respectively, of an integer square root function. In the first case, the only constraint is that the number must be a positive integer; however, in the later case, the function is also guaranteed to return a number that is in the range [-*y*, *y*], whose square is smaller or equal to y and whose square of the absolute value increased of 1 is greater than *y*.

2.2 Extended Static Checking (ESC)

While run-time checking code can certainly be useful for detecting and reporting errors, it leaves developers with the problem of handling them at run-time. Whenever possible, it would be more desirable if those errors could be prevented from occurring in the first place. Also, the earlier an error is found, the less expensive it is to correct. Static program checkers allow us to detect some of these run-time errors by static program analysis.

There exists a wide variety of static checking tools ranging from simple type checkers to formal program verifier. ESC is a special class of checker that, according to [11], generates verification conditions and logical formulas

```
/*@ public normal_behavior
  @  requires y >= 0;
  @  ensures -y <= \result
  @   && \result <= y
  @   && \result * \result <= y
  @   && y < (Math.abs(\result) + 1)
  @        * (Math.abs(\result) + 1);
  @*/
public static int isqrt(int y){
  ...
}
```

Fig. 2. Sample heavyweight specification

```
/*@ behavior
  @   requires        true;
  @   assignable      \everything;
  @   ensures         true;
  @   signals         (Exception) true;
  @   ...
  @*/
```

Fig. 3. Default (i.e. implicit) method specification

from a given program and passes them to an automatic theorem prover. Ideal characteristics of ESC tools are completeness and soundness i.e., catching all the errors, and triggering no false alarms (by reporting an error where there is none). On engineering grounds, such characteristics are not needed to benefit from ESC, especially since meeting such characteristics would imply over specification and reduced performances of the checkers [11][5].

ESC tools warn the user when there is an error or when the code does not implement its specification. By default, trivial specifications are assumed. A trivial specification implies that there are no special preconditions, that the execution may have any side effect, and that no specific post-condition should be expected. A default specification, if written in JML, would look like Figure 3 [9].

A false alarm may indicate that the specification is too weak and needs to be strengthened. To do such a strengthening, the programmers need to record the design decisions using the tool's annotation language that takes the form of a comment in the code. According to Flanagan et al. the annotation burden coupled with the fact that un-annotated code generates an excessive quantity of warnings may lower the benefit/effort ratio below the acceptable boundary of the mainstream programmer [5].

2.3 ESC/Java2

ESC/Java is an ESC tool for Java that was developed subsequently to the ESC/Modula-3 tool. These two ESC tools cumulate more than 12 years of experience and have been successfully applied to tens of thousands of lines of code [11]. Even if ESC/Java uses a complex program verification technology, it looks like a simple type checker for the programmer. Moreover, since it performs modular checking (as opposed to whole-program checking) the level of complexity can be kept at minimal level [2]. The second version of ESC/Java features JML as its annotation language. Both heavy and lightweight specifications are understood. Like its predecessor, ESC/Java2 is neither sound nor complete.

Coupling ESC with JML by using the phrases from a specification language as the ESC annotation language implies that the design decisions are recorded using a rich language and that such specifications can be reused, expanded, and passed to other JML tools-such as the runtime assertion compiler of Iowa University, JML RAC, and the formal program verification tool of Nijmegen University, LOOP [1]. Moreover, such an annotation language can take advantages of all existing JML specifications in order to better reason about the code it is checking.

An important feature of JML is its adoption of a behavioral subtyping semantics [9] in which a method overridden in a subclass is required to preserve the contracts of its corresponding super class method [3]. In the next section we mention how we believe behavioral subtyping can be used to advantage.

2.4 Java Web-based Enterprise Application (WEA)

Most Enterprise Applications (EAs) involve a significant quantity of persistent data. Most often, databases are used to store and access that data. Another characteristic of EAs is the high number of user interaction screens. For WEAs, Fowler recommends using the common three layered scheme, which isolates the domain logic (also referred to as business logic) from the presentation and data sources [6]. An interesting aspect of Java WEAs, is that most of them use the same small set of external libraries, super classes and interfaces for user interaction (e.g. *servlets.jar*) and database access (e.g. *java.sql.**). Due to the behavioral subtyping semantics of JML, we believe that in such a situation, the investment of specifying commonly used super classes and interfaces should reduce the effort required in specifying subclasses and lower the number of false alarms thus allowing developers to identify potential faults more easily.

3 Case study

3.1 Goals and approach

The main goal of our case study has been to assess the effectiveness of ESC/Java2 at helping developers increase software quality by detecting design and implementation faults. One of our early decisions was concerning the choice of application area. We opted for WEAs because it is one of the most active areas in Java development. We were also influenced by the availability of *SoenEA*, a small WEA framework that has been developed over the past two years for use in software architecture and design courses at Concordia University. *SoenEA* wraps code, such as the database connection and the servlet interface, so that students can use them easily without thorough knowledge of these technologies. The framework also comes with application samples.

Our study had two phases. During the first phase we applied ESC/Java2 to the *SoenEA* core and to sample applications that made use of the *SoenEA*. We will refer to this package as *A1*. The purpose of the first phase was to:

- gain experience in using ESC/Java2, while at the same time
- incrementally developing specifications for:
  - *A1* application
  - *SoenEA* core
  - *javax* and *java.sql* package modules used by *A1*.

ESC/Java2 performs modular checking [5][2] thus allowing us to work on one *A1* file at a time. This is very useful as ESC/Java2 initially reported hundreds of faults for *A1*. *A1* contained 1.6K source lines of code (SLOC), or 2.1K LOC.

The purpose of phase two was to measure the reduction in false alarms due to the use of the annotated versions of *SoenEA* core, *javax*, *java.sql* modules developed in phase 1. In addition to the *A1* package we analyzed two more applications totaling 5K SLOC (or 7.6K LOC). We will refer to the latter as the *A3* package.

3.2 General results

These two phases required approximately 4 person-weeks of effort (full-time). At the conclusion of phase I we had created:

- An annotated version of the *A1* package (i.e. *SoenEA* core and the *A1* application). Overall this represented an increase in size by approximately 4% due to the annotations (i.e. about 100 LOC).
- Lightweight specifications for the *javax* and *java.sql* package modules-90 SLOC (379 LOC). These lightweight specifications consisted mostly of annotations about the class attributes and method arguments restricting them to being non-null.

In phase II, use of the created specifications reduced false alarms by 9% on average for the *A1* and *A3* packages as compared to use of ESC/Java2 without the specifications.

In the subsections that follow we relate some of our experiences in specifying the external libraries (*javax* and *java.sql*) as well as the SoenEA core. The material presents issues of increasing complexity (to emulate, to some extent, the order in which we usually had to deal with them). The final subsection covers the most interesting faults reported by ESC/Java2 for the *A1* application.

Before discussing the number of faults uncovered by ESC/Java2 it becomes essential at this point to provide our definition of "fault". By "fault", we refer to something that is wrong with respect to the intended use of a method (or in general, a class). The intended use of a method is what is recorded in its specification. When there is no explicit documented specification (be it formal or informal), a default implicit specification is assumed. A "fault" occurs when the code does not satisfy the specification. According to DBC this can occur either because a client calls a method when the method's pre-condition is not satisfied or when a method returns and its post-condition is not satisfied. A false alarm due to a missing explicit specification will be recognized as a fault and named a specification fault. False alarms over external libraries denote missing specifications in ESC/Java2, whereas false alarms over user modules denote true specification faults.

3.3 Specifying javax, java.sql and the SoenEA core

ESC/Java2 performs static checking and reasoning using the JML annotations present in the code or specification files. In the absence of a specification (as would be the case for a method of the *javax* class at the start of phase I) ESC/Java2 assumes the default specification (Section 2.2) which is very permissive and seldom satisfactory. This implies that in the absence of specification, ESC/Java2 may trigger warnings, many of which would be false alarms or specification faults. (Thankfully ESC/Java2 comes with JML specifications for the most common Java classes, which, e.g. prevents it from warning about possible null dereferences in the case of a simple *System.out*.)

An example of a specification fault that ESC/Java2 reported early in phase I is given in Figure 4. The corresponding code is given in Figure 5. ESC/Java2 reports that the variable *dbStatement* could be null at the indicated point in line 35. This is not the case since this variable is set to the return value of *db.preparedStatement()* which always returns a reference to an object [15]. This is a false alarm due to a lack of explicit specification of *preparedStatement()*.

Fixing this false alarm is simple: we create an explicit specification for the method stating that it does not return a null result-see Figure 6. Most of the specifications created for *javax* and *java.sql* modules where of this nature:

```
soenEA.applications.assignment.a3.ts.TaskTDG: findAll() ...

------------------------------------------------------------
.\soenEA\applications\assignment\a3\ts\TaskTDG.java:35:
  Warning: Possible null dereference (Null)
                return dbStatement.executeQuery();
                                   ^
------------------------------------------------------------
    [0.19 s 10054064 bytes]  failed
```

Fig. 4. Sample false alarm / specification fault

```
public class TaskTDG {
  ...
  public static ResultSet findAll()
  throws Exception, SQLException {
    Connection db = DbRegistry.getDbConnection();
    PreparedStatement dbStatement =
      db.prepareStatement("SELECT * from " + TABLE_NAME);
    return dbStatement.executeQuery();      // line 35
  }
  ...
}
```

Fig. 5. TaskTDG code excerpt

```
package java.sql;

public interface Connection {

  //@ ensures \result != null;
  public PreparedStatement prepareStatement(String sql)
  throws SQLException;

}
```

Fig. 6. Sample lightweight spec created for Connection.preparedStatement()

i.e. specifying that arguments and or returned results would not be null. ESC/Java2 is capable of static checking of far more properties but statically detecting all possible null dereferences was deemed sufficient for this iteration of the case study.

Like the specifications of the external libraries, the majority of the annotations added to the SoenEA core were of the same lightweight nature. This allowed us to uncover some very interesting faults nonetheless, which we report in the next section.

4 Specific design and implementation faults

In this section we report on the most interesting faults uncovered by ESC/Java2 in the *A1* package (i.e. *SoenEA* core and the *A1* application) during the second phase of our study. ESC/Java2 reported 102 faults for the *A1* package. Of the 102 faults that were identified, 90% were specification faults, i.e. they were due to a lack of design documentation/specifications. Such faults are eliminated by writing design documentation in the form of method API specifications. The faults presented next are taken from the remaining 10% of the faults.

4.1 Incompletely propagated design changes

At least two of the faults were manifestations of incompletely propagated design changes. For example, one of the domain logic classes representing a *Task* in the *A1* application underwent the following design change: initially the class fields were initialized by means of setters; subsequently it was decided that all fields were to be initialized by means of the (non-default) class constructors. A consequence of this design change is that class constructor arguments of reference types could no longer be null. Such a fact had been properly documented in the *Task* class but not all calls of the constructors respected these constraints either because:

- not all client classes were appropriately updated, or, as often happens,
- in subsequent updates to the application, developers were still relying on the old API semantics.

These two scenarios demonstrate the advantage of formally documented design decisions and the use of extended static checking.

Another similar example occurred in the *SoenEA* database registry. An initial version of the class tried to enforce that a connection field would never be null. However, it was realized that this was infeasible. It is interesting to note that had ESC/Java2 been used earlier, this design error would have been uncovered from the start. The class was redesigned, but ESC/Java2 helped uncover some situations in which designers had not anticipated that the connection field could still be null.

4.2 Possible violation of behavioral subtyping

In the *User* class a method named *equals* was defined (Figure 7), thus overriding *Object*'s *equals* method. ESC/Java2 reported that the superclass specification (i.e. the specification of *Object.equals()*) might not hold for *User.equals*. It was initially thought that the cause of this fault was that *User.equals* only compared four of its five attributes. However, after discussion with designers, it was realized that this was deliberate. Hence, in order to avoid confusion, it was decided that the proper corrective action was to rename *User.equals*

```
public class User { ...
  private long id;
  private String loginId;
  private String name;
  private List originatedTasks;
  private String password;

  public boolean equals(Object obj) {
    if(obj == null || !(obj instanceof User))
    return false;
    User other = (User) obj;
    return this.id == other.id &&
           this.loginId.equals(other.loginId) &&
           this.name.equals(other.name) &&
           this.password.equals(other.password);
  }
}
```

Fig. 7. User class excerpt

```
soenEA.general.app.Servlet: forwardAbsolute(java.lang.String,
 javax.servlet.http.HttpServletRequest,
 javax.servlet.http.HttpServletResponse) ...
-------------------------------------------------------------
soenEA/general/app/Servlet.java:109:
 Warning: Possible null dereference (Null)
        dispatcher.forward(request, response);
                   ^
```

Fig. 8. Null dispatcher error

to *User.similar*. This example illustrates the power of behavioral subtyping: carefully written super class specifications allow ESC/Java2 to uncover semantic errors.

4.3 Unchecked dispatcher

ESC/Java2 detected a fault in the *SoenEA* core that, we have come to know, is apparently common for novice WEA developers (Figure 8). The framework assumed that the servlet dispatcher was always initialized. This is not the case [14]. ESC/Java2 provided a warning to this effect (based on default implicit specifications of *getRequestDispatcher*, which in this case were correct). The faulty code is given in Figure 9.

```
public void forwardAbsolute(String target,
HttpServletRequest request, HttpServletResponse response)
throws ServletException, java.io.IOException
{
    RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher(target);
    dispatcher.forward(request, response);
}
```

Fig. 9. Servlet class excerpt

4.4 Missed exceptional condition

Database connection information (e.g. login id, password) is recorded in a property file that the *SoenEA* framework reads. While the case where the file is not present was properly handled, the case where the file did not contain the required properties was not. This was not caught during code review nor during testing.

4.5 Other faults

A few of the remaining faults were simple possible null pointer dereferences. An example of this occurred in a file that was recently added to the framework, which had not been tested carefully. ESC/Java2 detected three possible null pointer dereferences in it. Only an extensive use of the framework would have been able to catch these, since the class was an exception adapter that would only be used under exceptional circumstances.

5 Conclusion

We believe writing JML specifications and verifying the match between them and the code with ESC/Java2 was a very helpful exercise. The time spent during this case study has allowed us not only to identify and correct design issues, but it has also forced us to think about and document the design decisions that had previously been made. Moreover, this documentation can be automatically verified by the tool. We have raised our level of trust in the framework and, through the enforcement of behavioral subtyping, feel the framework is a better candidate for reuse and expansion.

In this case study, ESC/Java2 proved itself to be useful for WEAs. However, it is a general purpose tool that is not limited to a specific domain of application. The use of ESC/Java2 is likely to spread to other domains of application. To our knowledge, this case study was the first one involving ESC/Java2 and the WEA domain. Perhaps a drawback of JML in that particular domain is that is does not currently support reasoning about concurrency. However, a proposal has just been published to address this issue [13]. Nonetheless the lightweight JML specifications we had to write for *javax*

and *java.sql* can be reused or supplemented by the JML community and can contribute to making JML and its supporting tools, like ESC/Java2, more convenient to use for Java developers.

Our case study involved a relatively small framework and application, but since it performs modular checking, we believe that ESC/Java2 will scale up to larger application. For this case study, we used the latest development release of ESC/Java2 (December 2004). It is considered to be in a late alpha cycle. As such we did encounter errors with the tool but none that could not be worked around. In fact the lead developers were very responsive to our problem reports and requests for assistance when the tool failed.

Of course there is a cost associated with the creation of specifications. As this was our first case study involving ESC/Java2, an important part of our effort was dedicated to learning about the tool and about the particular form of JML specifications that best suite it. Several false alarms were also due to missing specifications of *javax* and *java.sql* classes. Moreover, as of now, ESC/Java2 is not integrated into any IDE. We are confident that as research progresses and ESC/Java2 becomes more mature and easy to use, the cost/benefit ratio will become more and more appealing.

6 Future work

Now that some of the *javax* and *java.sql* classes have been specified, in the future, it would be an interesting exercise to perform this case study again with other WEAs and see if the annotation burden has been lowered in a significant manner. Since every year the *SoenEA* framework is used by more than a hundred students, we plan to provide them with the annotated version of the framework and verify whether they think using ESC/Java2 can improve the quality of their WEAs and help them identify and correct faults. Yet another interesting avenue would be to reuse the annotated *SoenEA* framework, applying other JML compatible tools to it, and then checking how much the written specification can be reused and what benefits WEA developers can get out of it.

7 Acknowledgement

The authors would like to thank Daniel Sinnig and Stuart Thiel for their valuable feedback as well David Cok and Joseph Kiniry for helping us with ESC/Java2 and answering all our questions.

References

[1] Burdy, L., Y. Cheon, D. Cok, M. Ernst, J. Kiniry, G. T. Leavens, K. R. M. Leino and E. Poll, *An overview of JML tools and applications*, International

Journal on Software Tools for Technology Transfer (STTT) (2004), to appear.
URL ftp://ftp.cs.iastate.edu/pub/leavens/JML/sttt04.pdf

[2] Cok, D. R. and J. R. Kiniry, *Esc/java2: Uniting esc/java and jml*, in: *Construction and Analysis of Safe, Secure, and Interoperable Smart Devices* (2004), pp. 108–128.

[3] Dhara, K. K. and G. T. Leavens, *Forcing behavioral subtyping through specification inheritance*, Technical Report 95-20c, Department of Computer Science, Iowa State University, Ames, Iowa, 50011 (1995).
URL ftp://ftp.cs.iastate.edu/pub/techreports/TR95-20/TR.ps.gz

[4] *Esc/java2*.
URL www.sos.cs.ru.nl/research/escjava/

[5] Flanagan, C., K. R. M. Leino, L. Lillibridge, G. Nelson, J. B. Saxe and R. Stata, *Extended static checking for Java*, ACM SIGPLAN Notices **37** (2002), pp. 234–245.

[6] Fowler, M., "Patterns of Enterprise Application Architecture," Addison Wesley, 2002.

[7] Johnson, S., *Lint, a c program checker* (1978).
URL citeseer.ist.psu.edu/johnson78lint.html

[8] Leavens, G. and Y. Cheon, *Design by contract with jml* (2003).
URL citeseer.lcs.mit.edu/leavens03design.html

[9] Leavens, G. T., A. L. Baker and C. Ruby, *Preliminary design of JML: A behavioral interface specification language for Java*, Technical Report 98-06y, Iowa State University, Department of Computer Science (2004), see www.jmlspecs.org.
URL ftp://ftp.cs.iastate.edu/pub/techreports/TR98-06/TR.ps.gz

[10] Leavens, G. T., K. R. M. Leino, E. Poll, C. Ruby and B. Jacobs, *JML: notations and tools supporting detailed design in Java*, in: *OOPSLA 2000 Companion, Minneapolis, Minnesota*, ACM, 2000, pp. 105–106.
URL ftp://ftp.cs.iastate.edu/pub/techreports/TR00-15/TR.ps.gz

[11] Leino, K. R. M., *Extended static checking: A ten-year perspective*, Lecture Notes in Computer Science **2000** (2001), pp. 157–??
URL http://link.springer-ny.com/link/service/series/0558/bibs/2000/20000157.htm

[12] Meyer, B., "Object-Oriented Software Construction," Prentice-Hall, Englewood Cliffs, 1997, second edition.
URL http://www.prenhall.com/allbooks/ptr_0136291554.html

[13] Rodríguez, E., M. B. Dwyer, C. Flanagan, J. Hatcliff, G. T. Leavens and Robby, *Extending sequential specification techniques for modular specification and verification of multi-threaded programs*, Technical Report SAnToS-TR2004-10, Kansas State University, Department of Computing and Information Sciences

(2004), to appear in *ECOOP 2005*.
URL http://spex.projects.cis.ksu.edu/papers/SAnToS-TR2004-10.pdf

[14] *Sun java servlet api documentation*.
URL
java.sun.com/j2ee/1.4/docs/api/javax/servlet/package-summary.html

[15] *Sun java sql api documentation*.
URL
java.sun.com/j2se/1.4.2/docs/api/java/sql/package-summary.html

[16] *Quarterly retail e-commerce sales 3rd quarter 2004* (2004).
URL www.census.gov/mrts/www/data/html/3q2004.html

Author Index