Université de Montréal

A Formal Framework for Run-Time Verification of Web Applications

An approach supported by Scope-Extended Linear Temporal Logic

par

May Haydar

Département d'informatique et de recherches opérationnelles

Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures

en vue de l'obtention du grade de Doctorat

en Informatique

Décembre, 2007

Université de Montréal

Faculté des études supérieures

Cette thèse intitulée :

A Formal Framework for Run-Time Verification of Web Applications: An approach supported by Scope-Extended Linear Temporal Logic

présentée par :

May Haydar

a été évaluée par un jury composé des personnes suivantes :

Pierre McKenzie, président-rapporteur

Houari Sahraoui, directeur de recherche

Alexandre Petrenko, co-directeur

Julie Vachon, membre du jury

Ferhat Khendek, examinateur externe

Jean-Pierre Marquis, représentant du doyen de la FES

Thèse acceptée le 12 décembre 2007

Résumé

Les travaux de recherches présentés dans cette thèse touchent trois domaines principaux dans un effort de développer une approche formelle et un cadre pour l'analyse et la vérification des applications web. Ce travail de recherche a pour but d'assurer et maintenir une haute qualité des applications web d'une manière automatique et facile à utiliser.

La contribution principale de recherche est dédiée au développement de méthodes pour la modélisation formelle d'une application web donnée en utilisant un modèle d'automates finis communicants, basé sur des propriétés définies par l'usager pour leur validation. Nous élaborons une méthode pour la génération automatique de ce modèle à partir des traces d'exécution produites par l'application web durant son exploration par un operateur humain ou un *crawler*. Le modèle obtenu pourrait être utilisé pour vérifier des propriétés avec un *model checker*, ainsi que pour des tests de régression et de la documentation. Certaines propriétés reliées au web concernent tous les états du modèle, tandis que d'autres ont trait à seulement un sous-ensemble. Pour cela, nous raffinons notre modèle pour designer le sous-ensemble des états globaux auquel on est intéressé.

La seconde contribution de recherche consiste à résoudre le problème de spécification de propriétés en logique linéaire temporelle sur un sous-ensemble d'états d'un système sous test, tout en ignorant la validité des propriétés pour le reste des états. Nous introduisons pour cela des opérateurs spécialisés qui facilitent la spécification des propriétés sur des *scopes* propositionnels, où chaque scope constitue un sous-ensemble d'états satisfaisant une formule de logique propositionnelle. En utilisant les opérateurs proposés, l'usager peut spécifier les propriétés du web d'une façon plus concise et plus intuitive. En dépit du fait que la motivation derrière ce problème vient du besoin de différencier les états stables des états transitoires du modèle des applications web, la solution proposée est générique et applicable à n'importe quel domaine de problème.

Spécifier des propriétés en utilisant la logique temporelle est souvent complexe même pour les experts. C'est également une tâche difficile et sujette aux erreurs pour les

usagers non-experts. Pour assister les développeurs et les testeurs du web à spécifier formellement des propriétés reliées au web, nous présentons une librairie de patrons de spécifications du web exprimés en LTL. Cette librairie est le résultat d'une inspection de diverses sources dans le domaine de l'assurance de qualité des applications web. Les patrons sont classés en deux catégories principales : fonctionnels et non-fonctionnels.

Finalement, nous présentons une implémentation de notre framework utilisant le *model checker* Spin, ainsi qu'un prototype qui inspecte et analyse les exécutions d'une application web donnée et produit un modèle à automates communicants. Ce modèle peut être représenté en Promela (le langage de Spin) ou en XML-Promela.

Mots-clés : Analyse dynamique, model checking, vérification, logique temporelle linéaire, model checker Spin, application web, session de navigation, structure de Kripke, patrons de propriétés, automates communicants.

Abstract

The research work presented in this thesis encompasses three main subject areas in an effort to develop a formal approach and framework for the analysis and verification of Web Applications. This research aims at ensuring and maintaining high quality Web Applications in an efficient, automatic, and easy to use manner.

The main research venue is dedicated to developing methods for formal modeling of a given web application using communicating finite automata model, based on the user-defined properties to be validated. We elaborate a method for automatic generation of such a model from execution traces produced by a web application while it is explored by a human operator or a crawler. The obtained model could then be used to verify properties with a model checker, as well as for regression testing and documentation. Some of the web related properties concern all states of the model, while others – only a proper subset of them. Therefore, we refine our model to designate the subset of the global states of interest.

The second research venue involves solving the problem of property specification in Linear Temporal Logic (LTL) over a subset of states of a system under test while ignoring the valuation of the properties in the rest of them. We introduce specialized operators that facilitate specifying properties over propositional scopes, where each scope constitutes a subset of states that satisfy a propositional logic formula. Using the proposed operators, the user can specify web properties more concisely and intuitively. Although the motivation behind this problem stems from the context of distinguishing between stable and transient states of the proposed model for Web Applications, the anticipated solution is generic and applicable to any problem domain.

Specifying properties using temporal logic is often complex even to experts, while it is a daunting task and error prone for non-expert users. To assist web developer and testers in formally specifying web related properties, we present a library of web specification patterns mapped into LTL. This library is a result of a survey of various resources in the field of quality assurance of Web Applications, which characterize successful web

application using a set of standardized attributes. The patterns are categorized into two main classes: functional and non-functional.

We finally present our implementation of the proposed framework using Spin model checker, where we develop a prototype tool that monitors and analyzes executions of a given web application, and produces a communicating automata model which could be represented either in Promela (Spin' input language) or XML-Promela.

Keywords : Dynamic Analysis, Model Checking, Verification, Linear Temporal Logic, Spin Model Checker, Web Application, Browsing Session, Kripke Structure, Property Patterns, Communicating Automata.

To my father, my first teacher

Acknowledgments

First and foremost, I would like to thank my supervisors, Professor Houari Sahraoui and Dr. Alexandre Petrenko who provided continuous support, both technical and financial. The difference of their research directions has enriched this work by benefiting from both expertises. Without them, this work would have never been accomplished.

Dr. Petrenko has offered me the opportunity to work in an intellectually stimulating environment in his ASYD group. His proficiency, sharpness, critical thinking, high ethical standards, constantly aiming towards perfection, and above all his shrewdness have been fundamental to the success of this work. He has invested a considerable amount of time and effort in directing and reviewing my research. On various occasions, he gave me the opportunity to present my work to researchers from various companies like SAP Labs, and Siemens AG, and to the Conseil Scientifique du CRIM, where I received constructive feedback that contributed to the advancement of my research. In addition to research coordination, Dr. Petrenko provides a framework of commitment, and dedication, coupled with utmost kindness, which allowed me to reconcile my life as a mother with the one as a student. For guiding me through this phase of my life, I am forever indebted to him.

Prof. Sahraoui has helped me shape my research work into applicable contributions to the web domain. His expertise in quality assurance and software maintenance has given this work the prospect of having a significant impact in the domain of quality assurance of Web Applications. He offered me a stimulating environment within the context of his GEODES lab, and the seminar sessions where insightful discussions helped me shape this work. His understanding and guidance throughout this journey is exceptionally appreciated and valued.

I am also grateful to Sergiy Boroday, researcher in ASYD group, who has contributed to the success of this thesis. His ideas, and constantly challenging discussions, greatly participated in shaping the research work. He also gave a lot of his time reading many of my writings and meticulously assessing them.

I am thankful to Ghazwa Malak for the resources she provided, and the discussions we had, which contributed to parts of this thesis. Ghazwa is a Ph.D. student in the group of Prof. Sahraoui working on quality evaluation of Web Applications using a probabilistic approach.

I also greatly appreciate the collaboration of masters' students and trainees who helped in the development and implementation of the tools related to this thesis, as well as interesting comments from various members of ASYD and GEODES groups.

I also wish to acknowledge the financial support provided by CRIM throughout my Ph.D. studies, which I greatly appreciate, as well as the bursaries offered by the FES, and DIRO. CRIM also provided me with a pleasant work place in R&D. Within its environment, I had the opportunity to perceive how theoretical research can be transferred to practical solutions for the benefit of technology advancement.

I am also thankful to Daphne Belizaire, the responsible for the documentation center at CRIM. She is efficient and dedicated in providing almost any resource I needed, and which could not be found in university libraries in Montreal.

I appreciate also insightful discussions with several researchers, namely Andreas Ulrich from Siemens AG, and Gerard Holzmann who corresponded with me by email addressing all my concerns regarding Linear Temporal Logic and Spin model checker. I also acknowledge the feedback of the anonymous reviewers of my publications as well as the audience of my presentations.

On a personal level, I am grateful to my mother, sisters, and family members and friends for their unfailing encouragement even from far apart. My thanks go my husband Hesham who has always been at my side, extending his support and help; I could not have completed this work without his patience and understanding. To my precious Riham and Mohamad Ali, I cherish every moment I spend with them, and all the hope they have given me throughout this journey. Last but not least, to my father, whose fondness of knowledge, wisdom, patience, and perseverance, and mostly his love and compassion, have contributed to my success. This thesis is the fruit of your teachings.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The accelerating growth of Web Applications and the numerous constantly evolving technologies used in the development of such applications have led to an increased complexity of maintaining high quality for Web Applications. In particular, developers do not have sufficient tool support to create high quality applications. Thus development tools should be complemented with analysis and validation tools and methods. In this chapter, we introduce the motivations and problems addressed in this thesis as well as the main contributions.

1.1 Web Applications

The Internet has reshaped the way people deal with information. Few years ago, simple web sites existed where the components were text documents interconnected through hyper links. The aim of those web sites was to supply information across the web in a simplistic and intuitive manner. Therefore, quality assurance, nonetheless overlooked, was a relatively simple task.

Nowadays, the Internet and the web affect the daily life in many ways. They are used to run large-scale software applications involving almost all aspects of life including information management/gathering, information distribution, e-commerce (business-to-customer, business to business), software development, learning, education, collaborative work, and lately, an integral part of Service Oriented Applications (SOA). According to [84], diversity is a key description of Web Applications (WAs) in many aspects that led to the notion of "web engineering". WAs are developed with cutting edge technologies, and interact with users, databases, and other applications. They also use software components that could be geographically distributed, and communicate through different media. WAs are constructed of many heterogeneous components including plain HTML files, mixtures

of HTML, XML, and programs, scripting languages (CGI, ASP, JSP, PHP, servlets, etc.), databases, graphical images, and complex user interfaces. These diversities led to the need for large teams who do not share the same talents, skills, and knowledge. These include programmers, usability engineers, data communications and network experts, database administrators, information layout specialists, and graphic designers [84]. Figure 1 is a typical representation of WAs architecture [61].

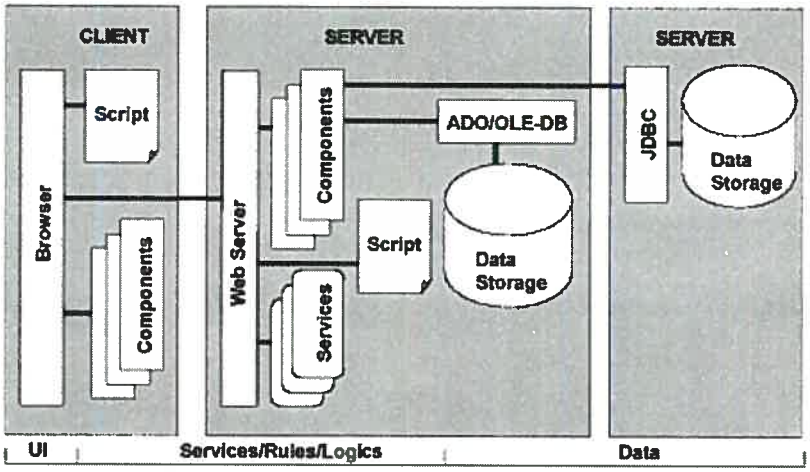


Figure 1. Web Applications Architecture

With such pervasive and radical growth of WAs, correctness is a primary concern. Unlike traditional software, WAs have an extremely short development and evolution life cycle and have to meet stringent time to market requirements. WAs often have a large number of untrained users who could experiment with the WA unpredictably. The success of WAs solely depends on their users and their satisfaction. In fact, [68] reports that human errors together with software failure account for about 80% of WAs failures. Hence, a low quality of these applications can be very costly; as an example, the four days outage of Microsoft Money in 2004 was caused by a glitch that prevented users from accessing their online personal finance files [68]. Therefore, thorough analysis and verification of WAs is indispensable to assure the release of high quality applications.

1.2 Web Analysis Tools

Various types of web analysis tools exist in the market. Tools, such as [94] and [95], generally verify the syntax of HTML documents, confirm the hyperlink integrity of a set of HTML documents, test the GUI components embedded in the browsers, and measure the performance of the WA. In [90], 340 web analysis tools (both commercial and open source) are listed and classified into 12 categories. There exists various types of web analysis tools [61]; we describe some of them:

Rule-based analyzers such as HTML validators and scripting languages validators, examine the source code, compare it with the language-specific rules, and report inconsistencies and potential errors.

Web load and performance testing tools generate test scripts to simulate thousands of users accessing a given WA by requesting data, and submitting transactions. Performance metrics such as response time and data throughput are then tracked, reported, and plotted in several tabular and graphical formats for further analysis.

GUI capture and playback tools are used for functional and regression testing. They recognize GUI controls such as form buttons, links, and Java applets in web pages, and capture input events (user activities) applied on those controls. The input events are converted into test scripts used by a playback engine that replays the prerecorded activities repeatedly. These tools usually report error logs, which indicate discrepancies and inconsistencies discovered during playback.

Run-time error detectors and Log analyzers are also called dynamic analyzers. Run-time error detectors analyze the execution of programs rather than the source code. Some of these tools report erroneous operations during the execution of the program. Log analyzers employ the same concept of analyzing the execution of a given WA by logging the server's requests and responses. Those analyzers report on the number of hits of pages, the peak times of server access, number of visitors who accessed the WA, HTTP errors, user activity statistics, accessed files, paths taken through the WA, and other statistics related to the usage of the WA.

Most of these tools test only one or some aspects of WAs and are insufficient to ensure a high quality of WAs. Therefore, there is an increasing need for tools that perform thorough analysis and verification of WAs.

1.3 Motivation and Problem Statement

In recent years, the software community started to adopt formal methods as a practical and reliable solution to analyze applications in various domains. However, applying formal methods to web development, one has to keep in mind two main issues:

1. the costly learning curve associated to understanding the underlying mathematical foundations and
2. the scalability of the involved modeling and analysis techniques.

Still many research groups have identified ways to work around these obstacles and the literature shows how formal methods are being used in the white-box and black-box analysis of distributed systems [46,29], real-time systems[55,14], source code analysis [18,19], etc. In almost all the proposed techniques, an application under test is analyzed by:

- Inferring a formal model from its design documents, source code, or even execution logs, and
- Verifying the inferred model against existing specifications or user-defined properties that are of relevance to the analyzed application.

To obtain a formal model of a WA in case when the code of an application is available, one may apply abstraction techniques developed in software reverse engineering following a *static*, white box approach [5,18,19]. However, the source code is not always available, or access to the code could breach copyrights or trade secrets (especially when verification is performed by a third party). Moreover, a WA can be written using different languages and even different paradigms, which makes the static analysis difficult to perform. When the code is not used for modeling, one can build a formal model following a *dynamic*, black-box based approach, by executing the application and using only the observations of an external behavior of the application [60,96,78].

Also, formal verification by model checking techniques has increasingly been used and in many cases preferred over testing and simulation, since model checking can perform an exhaustive exploration of all possible behaviors of a given system. Indeed, testing and simulation methods are not exhaustive and deal with a part of the system leaving the unexplored behaviors of the system unchecked. Model checking is fully automatic, and in case the design does not satisfy a given property, the model checker produces a counter example that points out to the behavior that violates the property.

In this thesis, we address the problem of formal verification of WAs without having access to source code. To solve it, we develop a formal framework, which allows the analysis of existing WAs by providing proper tools to automate the analysis process.

The work in this thesis addresses the following main problems:

1. Inferring automata based models from a web application under test (WAUT) to verify user-defined properties,
   a. independently from the browser, since WAs can be adapted to run on several different platforms and devices with different navigation programs,
   b. and without any access to the underlying code,
2. Defining property patterns related to good practices in developing WAs, to alleviate the burden on the user/tester to learn the foundations of temporal logic for property specification.
3. Extending Linear Temporal Logic (LTL) with scope operators to allow the specification of user defined properties over an arbitrary subset of states of a given model more succinctly.

The properties addressed in this work relate the requirements that ensure a high quality of WAs. By quality requirements here, we mean the ones that are related to the correctness and functionality of WAs, as well as those that are related to the ergonomics and design of web pages. Also, those properties can be generic to all types of WAs or specific to certain WAs such as online banking or governmental WAs. They also encompass specifications related to e-commerce WAs which differ in many aspects from

those of other types of applications. For example, a generic specification related to the functionality of WAs is "*Home page is always reachable from all pages*". Another property related to the ergonomics of web pages is "*Number of links in each page should not exceed a certain threshold*".

One could argue that thorough analysis could be performed on the actual code of the WAUT. However, such approach reduces the application domain eliminating the possibility of analysis of a wide range of web applications where the code is not available. We rely on a non-intrusive dynamic approach by inferring automata models from the run-time behavior of the WAUT.

On the other hand, one could build specific tools [7,21] for verifying a restricted range of properties, usually predefined and embedded in the tools. Such tools have two major inefficiencies. First, they do not give the freedom to verify a wide range of properties on a WAUT model, thus reducing the problem domain. Second, they do not scale to accept relatively large models.

Hence, using formal methods, we could benefit from the availability of various reliable tools, used for several years in industry and academia. Such tools allow the specification of general properties using temporal logic, thus, solving a wider range of problems related to WAUT. In addition, these tools have undergone years of development, enhancement, and upgrades to solve many of the scalability problems related to the state explosion problem [16]. In the next section, we give an overview on one of these tools, Spin model checker, which is used in the work of this thesis.

1.4 Contributions of the Thesis

The prevalence of Web-based applications leads to the increasing demand for validation frameworks and tools that could verify that the WA in question possesses a number of useful properties ensuring its successful deployment, while certain unwanted properties are absent in it. Such analysis could be done by model checking of WAs.

The main objective of the research is to develop an integrated framework and a prototype tool environment to automate verification of properties of WAs. The idea is to translate a WA into an automata model and reuse an existing model checker to verify user-defined properties. We adopt a methodology based on the finite state machine model, in particular, the theory of model checking. Developing a prototype verification environment for WAs, we reuse an off-the-shelf model checker, Spin [44]. The research work of this thesis contributes to three different areas: formal modeling and verification of WAs, temporal logic extensions, and property patterns. The publications [35,36,37,38,39,40,41,42] reflect the contributions of the thesis.

1.4.1 Methods for Run-Time Verification of Web Applications

In this thesis, we develop a formal approach to build a finite automata model tuned to features of WA that have to be validated, while delegating the task of property verification to an existing model checker. We follow a black-box (dynamic) approach by executing the WAUT and analyzing only its external behavior without any access to server programs or databases. The model built is a system of communicating automata representing all windows and frames of the WAUT. The existence of frames and windows reflects concurrent behavior of the WAUT, where these objects affect each other behaviors via links and forms with specified targets. Therefore, communicating automata is a suitable and natural modeling technique, where the burden of building a global state graph of the model is left to a model checker. As opposed to the existing approaches discussed in Chapter 2, we model not only static pages, but also dynamic pages with form filling (with Get and Post methods), frames behavior, multiple windows, and their concurrent behavior. Generally speaking, one could build a special web-oriented model checker, as in [21], to verify specific properties, building thus all the necessary algorithms from scratch. We opt for the use of an existing model checker, Spin, used in several industrial applications [44], such that we only had to describe our model in the model checker's input language. The behavior of the WA is represented by the composition of all the component automata. A

global state is *stable* if it represents a page that is fully loaded and displayed to the user; otherwise, the global state is called *transient*. Distinction between stable and transient states becomes evident in applications that use frames, where the default pages of frames of a given web page are not loaded simultaneously. We argue that some properties are relevant to all the global states (stable and transient), while others should be verified only on either stable or transient global states. Thus, the scopes of states of interest constitute an important element in web model checking. For this reason, we develop a high-level algorithm to refine our model and extend it with local variables and updates on them to be able to designate global stable and transient states.

1.4.2 Extension of Linear Temporal Logic with Scopes

We further address in this thesis the problem of property specification in LTL over a subset of the states of a given system. We also realize that it is cumbersome to the expert, and virtually impossible [6] for the novice, to specify meaningful (often complex) properties using the usual temporal logic formalisms of model checking. The problem of property specification becomes even more difficult when complex properties specified on a part of a given system behavior while ignoring the rest of it, are considered. Therefore, we tackle the problem of property specification in LTL assuming that the user is interested in checking the properties over a subset of the states of a given system while ignoring the rest of the states. To filter out from a system's model the "uninteresting" states one could apply abstraction techniques and specify properties on the resulting model. However, such a solution cannot be generic since various languages, in which they are written, usually require specific abstractions. At the same time, different abstractions might be needed for each of the properties to be verified on a single system, which is error prone and unfeasible. We propose a generic and practical solution to ease the problem of property specification in LTL over subsets of states. This solution also does not require any changes in the system model by defining specialized operators in LTL using scopes. The new operators do not affect the expressiveness of LTL, but rather help specifying the properties more intuitively and succinctly. Though our solution is generic and can be applied to any type of systems,

this LTL scoping problem stems from the context of our formal framework described in this thesis, where in case of WAs with frames, some properties should be verified over stable global states only.

1.4.3 System of Web Property Patterns

Model checking requires learning the mathematical foundations behind temporal logic so that properties can be specified. Specifying properties can be difficult to specialist, let alone web users and testers. For this reason, we provide a library of property patterns for WAs specifications. The properties are gathered from various resources of the web community [58,79,92]. They constitute mainly web quality attributes that are required to ensure high quality WAs. We categorized these properties into functional and nonfunctional and translated them into linear temporal logic formula. The resulting LTL properties constitute a library of web specification patterns.

1.5 Contributions of Authors

The Web Specification Pattern System presented in Chapter 3 is based on the full system published in [40]. I was the main writer of the report and major contributor of the results. The second author participated as my supervisor in providing resources as well as directing the collaboration with Ghazwa Malak, a Ph.D. student working on quality assurance of WAs to narrow down the spectrum of quality requirements that could be useful in our specification system; he also participated in conceiving and revising the text of the report. My contributions to the report are:

- surveying various resources on quality of WAs, collecting the set of quality requirements, and classifying them, and
- mapping the quality requirements into LTL and building the system of web specification patterns.

Most of Chapter 4, and some empirical results from Chapter 7 include the derivation of methods and algorithms for run-time verification of WAs, as well as the developed

framework, and implementation of the toolset. These are published in [41,42]. I was the main writer of the papers and major contributor of the results. The second author of [41] participated as my supervisor in conceiving the formal framework and approach, writing, revising, and correcting the papers. The third author participated as my supervisor in revising the papers. My contributions to the papers are:

- Formalizing the framework, including developing the formal definitions, algorithms to infer automata models, as well as formulating properties in LTL; and
- Doing the case studies and model checking properties.

Chapter 6 is based on the results on extending LTL with propositional scopes and its application to the web domain, published in [35,36,37]. I was the main writer of the papers and major contributor of the results. The third author, my supervisor, was the one to raise the problem of scopes in our web modeling. The second author, a researcher in my supervisor's group (ASYD), as well as the third author assessed and corrected the system of definitions and formal proofs, and participated in conceiving, writing, and revising the paper. The fourth author participated as my supervisor in revising and commenting on the papers. My contributions to the papers are:

- solving the problem of property formulation in LTL over arbitrary sets of states, by introducing new LTL operators;
- conceiving and proving the lemmas and theorems of the papers; and
- demonstrating the usefulness of the proposed solution to web property specification.

Chapter 4, Chapter 5, and the major parts of Chapters 6 and 7 are based on the results published in [39]. They include the formal framework as in [41], as well as a discussion on web related properties which could be verified, and proofs that an inferred session automaton preserves these properties. The results also include methods and algorithms to extend our automata model with event tracking variables, and evaluation of the prototype tool using case studies. I was the main writer of the paper and major contributor of the results. The second author participated as my supervisor in conceiving, writing, and revising the paper. He also ensured that the system of definitions, proofs, and

algorithms are correct and consistent throughout the paper. The third author participated as a researcher in my supervisor's group (ASYD) in conceiving and correcting the definitions, theorems, and proofs of Section 4.3, as well as in writing and revising the paper. The fourth author participated as my supervisor assessing the evaluation of the toolset, its performance, the case studies, and empirical results, as well as in writing and revising the paper. My contributions to the paper are:

- formalizing the framework, including developing the formal definitions, theorems, proofs, and algorithms, as well as formulating properties in LTL;
- assessing the development of the toolset of our framework;
- doing the case studies and model checking properties; and
- evaluating the toolset, its performance, and scalability.

1.6 Thesis Plan

The rest of the thesis is organized as follows:

Chapter 2. It provides a literature review which starts by a synopsis of web related terminology and major components. It also includes a definition of a variant of LTL for both finite and infinite sequences, and a brief overview of the system of specification patterns. Finally, the chapter discusses the related work on modeling, analysis, testing and verification of WA, as well as work related to temporal logic extensions.

Chapter 3. It describes the web related criteria that ensure high quality WAs. Those criteria are categorized into functional and non-functional and translated into LTL formulae.

Chapter 4. In this chapter, the modeling approach and methodology are elaborated. In addition, we suggest an algorithm to model a browsing session, aka execution trace, of a single window WA (explored by a human operator or a crawler) by an automaton, discuss properties of WAs that could be verified using the inferred model, and prove that this model preserves the discussed properties. We also describe a method to partition a single

execution trace of a multi frame/window WA into local traces and to convert the local traces into communicating automata.

Chapter 5. We present a model refinement by extending the automata model with variables and updates to designate stable/transient global states.

Chapter 6. We explain the problem related to known difficulties in the specification of properties in LTL formalism and describe our solution, namely the syntax and semantics of the new operators. We also show the effectiveness of our approach for the analysis and verification of WAs and how our results can be used to specify web related properties over the states of interest.

Chapter 7. We present the implementation of the approach using Spin and provide an evaluation of the prototype tool developed as well as case studies. We conclude the chapter with a discussion evaluating the scalability of the prototype tool to generate relatively large models.

Chapter 8. We summarize the results of the thesis and discuss potential future work.

Apendix 1. We present the XML model of a WA with three frames, generated by the prototype tool described in Chapter 7.

Apendix 2. We present the Promela model of a WA with three frames, generated by the prototype tool.

Chapter 2

Literature Survey

The unabated growth and increasing significance of the world wide web has resulted in a flurry of research activity to improve the web capacity for serving information more effectively; for this reason, we dedicate this chapter mainly to discuss the state of the art on the related work developed in two main venues: modeling, verification, and testing of WA, and temporal logic extensions. First, however, we present the terminology of the World Wide Web used in the literature and elaborate on the usage of the HTTP protocol, HTML forms, and frames. We also present an overview of Linear Temporal Logic (LTL) and the Specification Pattern System.

2.1 Web Literature and Terminology

Based on several online resources [85,86,89], we present the major terms encountered in studying WAs. We also present an overview of basic elements that constitute WAs, namely Hypertext Transfer Protocol, HTML frames, and HTML forms.

2.1.1 Preliminaries

A WA is defined in [89] as "a software application that is accessible using a web browser or HTTP user agent. It typically consists of a thin-client tier (the web browser), a presentation tier (web servers), an application tier (application servers) and a database tier. An application may be spread over multiple presentation tiers and indeed use multiple application tiers while handling multiple database sources". A WA is also defined in [17] as "a web system (web server, network, HTTP, browser) in which user input (navigation and data input) affects the state of the business". We see a *web application* as an application providing interactive services by rendering web resources in the form of web pages (containing text and images, forms, etc.).

A WA handles web resources. A *web resource* is any entity residing on the web and identified by a single URI (Uniform Resource Identifier). The first part of the URI indicates what protocol to use, and the second part specifies the domain name or the IP address of the server, where the resource is located. A URI can be further classified as a locator (URL), a name (URN), or both. Familiar examples of web resources include an electronic document, an image, a service, etc. [89]. Web resources are rendered in web pages, also called URL pages. A *web page* is an HTML document on the World Wide Web. Every page is identified by a unique URI. A page can be static, residing on the server, or dynamic, resulting from the execution of a script at the server or the client side. A page can contain text, media or hyperlinks that can be transferred from an HTTP server to a browser that renders pages to the user in windows. A window may be split into regions each called a *frame*. A *browser* is a software application used to locate and display web pages in browser's main or independent windows. Windows are addressed, in links and forms, by *targets*.

A *target* is a string that is the name of a specific window or frame. It can be associated with a *link* or a *form*. When the target is defined, the browser loads the page in the window/frame bearing the target name. Otherwise, when the target is absent, the corresponding page is loaded within the same window where the link was clicked or the form was submitted. If the window/frame does not exist, a new window is opened and the page is loaded into it.

A *link*, synonym of hyperlink or anchor, is a highlighted text in a page that corresponds to a URI either locating a page on a server or referring to a section name on the same page. When referring to a section of the page, the URI of the link starts by the character "#" followed by the designated section name while it starts by a domain name followed by the file name and its extension when referring to another page. Thus, when a link is clicked, it either causes the loading of a new page or simply causes the scrolling to another place in the same page.

A *form* is a formatted document containing blank fields that the user can fill in with data. The user does this by selecting options with a pointing device or typing in text from the computer keyboard. The data is then sent directly to a form processing application such as a CGI program or JSP servlets. Forms are common on the World Wide Web because the HTML language has built-in codes for displaying form elements such as text fields and check boxes.

The content of each frame is usually a web page. Web pages displayed in frames can contain links targeting other frames in the window or other windows in the applications. Framing is a feature supported by modern web browsers, namely by Netscape 3+ and Internet Explorer 3+, though not evenly, which explains why many web authors avoid frames in spite a high flexibility framing offers to them in designing web pages.

2.1.2 Hyper Text Transfer Protocol

The HTTP Protocol specifies the syntax and semantics with which web clients and servers communicate (Figure 2). An HTTP message is a structured collection of octets having a specific syntax [50]. A *request* is an HTTP message sent from a client to a server and a *response* is an HTTP message sent back to the client by the server. A request consists of the following elements: the HTTP method, a URI, a protocol version identifier, optional header fields, and an optional message body that represents the data sent with the request. An HTTP method can be one of the following: Get, Head, Post, Put, or Delete; only the first three are widely used and implemented in most servers. The URI and the message body, if any, identify the requested resource. A request is generated when the user types a URI in the browser window, clicks a link, or submits a filled form. The browser itself can initiate requests, without the user's intervention, for an embedded object or for a page whose URI is specified in an HTML HTTP-EQUIV tag. In the case of link clicking, form submission, or browser initiated requests, the browser uses an optional header in the request, the "referer" [50], to include the URI of the page from which the request URI was obtained. The "referer" field is included in the request if the requested resource and the referred page are fetched from the same server; otherwise, the "referer" field is not included

in the request. In the case when the user types the URI in the browser's location field, the "referer" field is not included in the corresponding request.

A response consists of a numerical status code to determine the status of the response, a human-readable response status line, optional response header fields, and an optional body that includes the requested resource. HTTP status codes are divided into five categories and are in the range 100-599. The first digit determines the overall meaning of the status code and thus its category; the remaining two digits specify the condition in more details [57]:

- the 1xx codes constitute the *informational* category,
- the 2xx codes constitute the *successful* category,
- the 3xx codes are the *redirection* category,
- the 4xx codes are the *client error* category, and
- the 5xx codes are the *server error* category.

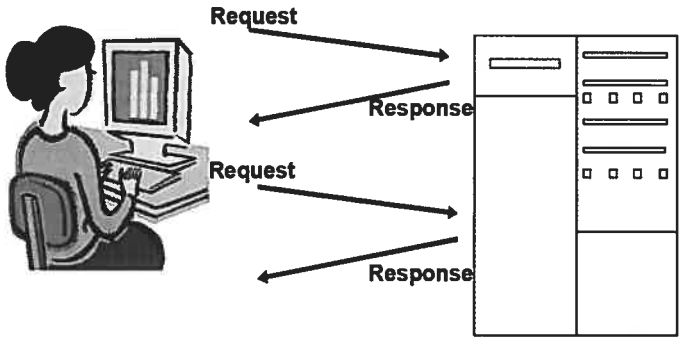


Figure 2. HTTP Interactions between Web Client and Server.

2.1.3 HTML Forms

An HTML form is a section of a web page that includes textual content, special elements called *controls* (checkboxes, radio buttons, menus, etc.), and optional labels for those controls. Users usually complete a form by modifying its controls such as entering text or selecting menu items, before submitting the form to the server for processing [1]. When a form is submitted for processing, some controls have their name paired with their current (default or user specified) value and these control pairs are submitted with the form.

Those controls for which name/value pairs are submitted are called *successful controls*. The unsuccessful controls are thus ignored by the submitted form request.

The following control types are defined in HTML: buttons, checkboxes, radio buttons, menus, text input, file select, hidden controls, and object controls [86]. Buttons can be of three types, submit buttons to submit a form, reset buttons to reset all controls to their initial (default) values, and push buttons which have no default behavior, instead they may be associated with client-side scripts. Checkboxes are on/off switches that may be toggled by the user. When a form is submitted, only "on" checkbox controls are successful. Radio buttons are similar to checkboxes except that when several share the same control name, they are mutually exclusive: when one is switched "on", all others with the same name are switched "off". Menus offer users options to choose from. Text input allows the user to enter textual data and it can be of two types, a single-line or multi-line input control, and a password input control. File select allows the user to select files so that their contents may be submitted with a form. Hidden controls are controls not rendered but whose values are submitted with a form; the authors generally use this control type to store information exchanged between client and server. Object controls (frequently called image controls) are generic objects inserted in forms such that when clicked the form is immediately submitted with their associated values; in general, they are decorative submit buttons. Figure 3 is an example of a form for flight reservation.

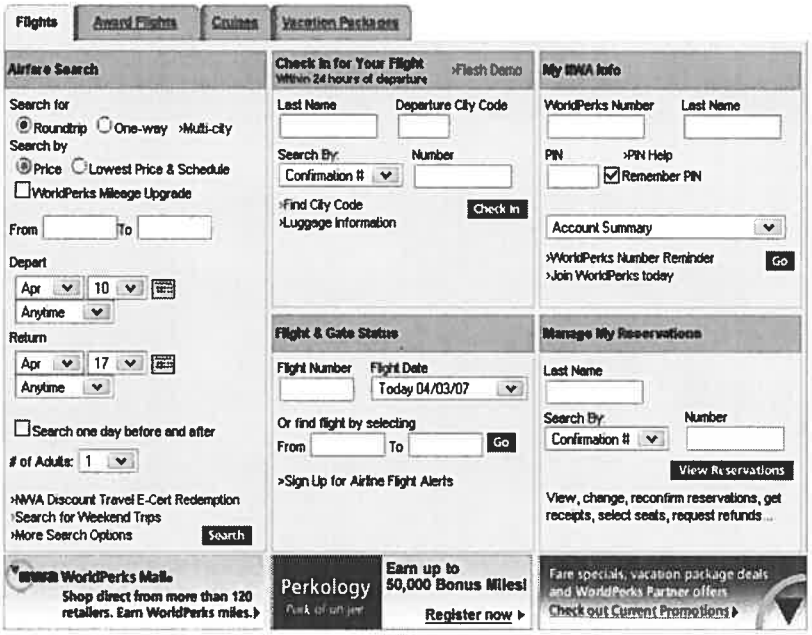


Figure 3. Form for Flight Reservation

Besides the form content, two attributes are defined in a form: Action and Method. The Action is the URI of a program on the server executed when the form is submitted. The Method specifies the HTTP method triggered to submit the form. The method can be either a GET method or a POST method (advantages and disadvantages of these methods are discussed in [33]). With the GET method, the form data set, which is the string of concatenated successful controls, is appended to the URI specified by the action attribute (with a question mark as separator) and this newly formed URI is sent to the server. With the POST method, the form data set is included in the request as a data stream and sent to the server.

When the server receives the submitted data, it hands it over to a program for processing. Such a program can be a CGI program or Java servlet classes that process requests for Java server pages (JSP). The program generates the corresponding output and hands it to the server. The server is responsible for dynamically generating an HTML page based on the processing program output and sends the page as a response to the user. There are cases of format errors in the user data such as entering a string in a date field or in a number field, or entering a birth date that is greater than the current date. In these cases,

either the data is processed by the processing program and an error message is returned to the user, or a client side script is executed to check the validity of the user input before sending the data to the server [33].

2.1.4 HTML Frames

HTML frames allow web authors to divide the browser's display area into multiple regions, called frames. For example, within the same window, one frame might have a static banner, a second a navigation menu, and a third a document that can be scrolled through or replaced by clicking a link in the second frame [89].

A frame element is an HTML tag that defines a frame. The frame element has a source *src* attribute that specifies the URI of the source (initial) page loaded in the frame. In addition, it may have an optional *name* attribute that assigns a name to the frame, so the frame can be *targeted* by links from other pages. A *frameset* element is an HTML tag that groups frame elements and possibly other frameset elements. The HTML document that describes the layout of frames is called the frameset document. The frameset document includes a frameset element that may be nested to any level. Frames and framesets at a lower level are the children of the frameset at one level higher. A Frameset document can be viewed as a *frame tree* whose leaves are frame elements.

When the target attribute is associated to a link or a form action, the corresponding page (which can be another frameset document) is loaded into the frame that has the same name as the target attribute. If none of the frames has such a name, the browser opens a new window, assigns to it the value of the target attribute as a name, and loads the corresponding page in it. The frame or the window in which the link was clicked does not change its displayed page. There are four predefined target names with which the browser takes specific actions: with the "_blank" target, the browser loads the designated page in a new, but unnamed window. With "_self", the browser loads the page in the same frame as the element referring to this target. With "_parent", the browser loads the page into the immediate frameset parent of the current frame (thus canceling the current frame and any other frame/frameset defined at the same level as the current frame); this target is

equivalent to "_self" if the current frame has no parent. With "_top", the browser loads the page into the full, original window (thus canceling all other frames); this target is equivalent to "_self" if the current frame has no parent [89].

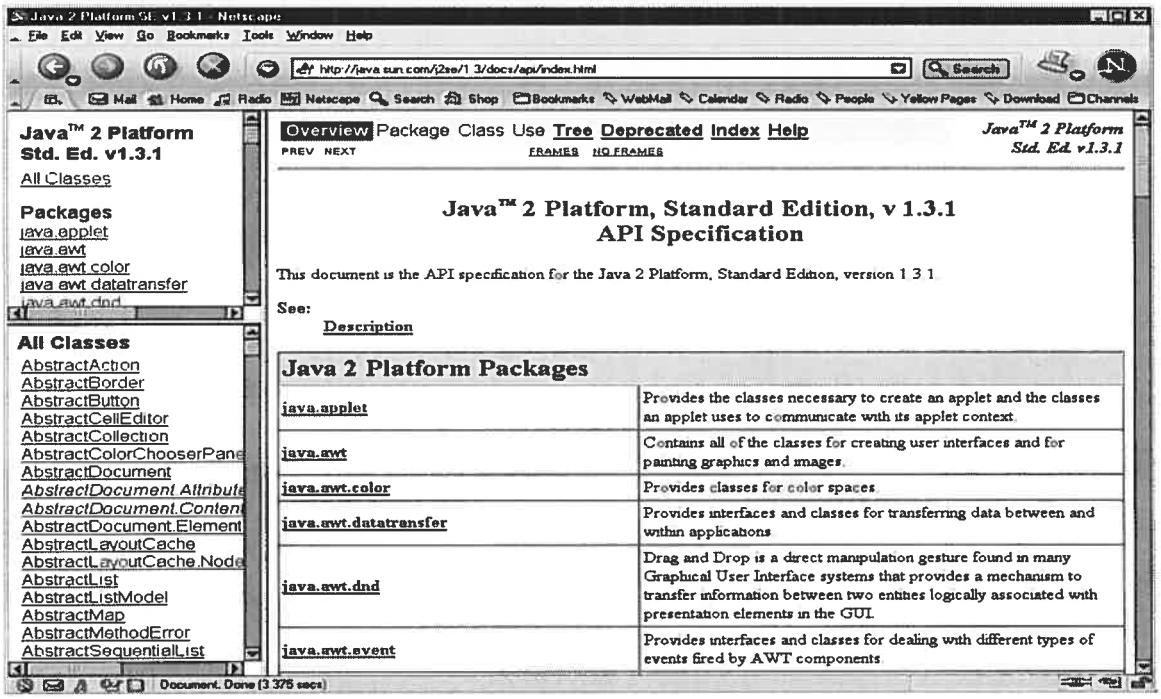


Figure 4. Example of a Page with Three Frames

Figure 4 shows an example of a web page with frames. A portion of the html code in the frameset document is as follows:

```
<FRAMESET cols="20%,80%">
        <FRAMESET rows="30%,70%">
                <FRAME src="overview-frame.html" NAME="packageListFrame">
                <FRAME src="allclasses-frame.html" NAME="packageFrame">
        </FRAMESET>
        <FRAME src="overview-summary.html" NAME="classFrame">
</FRAMESET>
```

In the example, the web page is decomposed into two main regions: left and right. The left region represents a frameset which contains two frames named *packageListFrame* and *packageFrame*. The right region, on the other hand, represents one frame named *classFrame*. Thus, the frame tree in this example has one root node and three leaves representing the frames *packageListFrame*, *packageFrame*, and *classFrame*.

The links in the *packageListFrame* frame have as target the *packageFrame* frame, and thus they are loaded into the lower left frame. Meanwhile, the links of *packageFrame* frame have as target *classFrame* and are loaded into the frame on the right.

2.2 LTL for both infinite and finite sequences

In this section, we present the syntax and semantics of a variant of LTL that represents both finite and infinite behaviors. LTL (sometimes called PTL or PLTL) extends traditional propositional logic with temporal operators. Thus, LTL allows assertions about the temporal behavior of a system [44,45,71], in our case a WA. An LTL formula φ has the following syntax:

$$\varphi ::= p \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \,\mathtt{U}\, \varphi) \mid (\mathtt{G}\, \varphi) \mid (\mathtt{F}\, \varphi) \mid (\mathtt{X}\, \varphi)$$

where p is an atomic proposition, U is the *until* operator, G (or $\Box$) is the *always* operator, F (or $\Diamond$) is the *eventually* operator, and X (or o) is the *next* operator.

LTL semantics is defined by Pnueli [71] over infinite sequences of states that correspond to infinite or non-terminating sequences of computations. LTL deals only with infinite behavior. Finite traces could be tackled with certain workarounds, such as adding a looping transition to the last state of a finite sequence of computations. Nevertheless, here we consider a variant of LTL that naturally applies to both, finite and infinite, cases [8]. Such logics are increasingly popular [4] in the context of LTL monitoring. Our choice is partially motivated by efficiency of new operators used on both finite and infinite scopes in WA analysis [41,30]. While this variant of LTL differs from the classical one, in case of infinite computation sequences, it coincides with the classical LTL.

In spirit of [8], we define a Kripke structure as follows. A *Kripke structure* is a tuple $M = (S, T, S_0, P, \mathcal{L})$, where S is a set of states, $T \subseteq S \times S$ is a transition relation, $S_0 \subseteq S$ is a set of initial states, P is a set of atomic propositions, and $\mathcal{L}$ is a labeling function from S to the power set of P. KS could have both infinite and finite paths (aka executions). An infinite state sequence $\pi = \langle s_0, s_1, \ldots \rangle$ is called a *path* (execution) of M if $s_0 \in S_0$, $(s_i, s_{i+1}) \in$

T for all i, $i \geq 0$. A finite state sequence of the form $\langle s_0, s_1, \ldots, s_k \rangle$, such that $s_0 \in S_0$, $(s_i, s_{i+1}) \in T$, and for all $s \in S$ $(s_k, s) \notin T$ is called *finite* path. $\pi^i = \langle s_i, s_{i+1}, \ldots \rangle$ denotes the suffix of a sequence $\pi = \langle s_0, s_1, \ldots \rangle$ starting at s_i. We denote by $|\pi|$ the length of a given state sequence π; if π is an infinite sequence of states, then $|\pi| = \infty$, assuming that ∞ is greater than any integer. An empty sequence of states is denoted ε; $|\varepsilon| = 0$. $\pi_j = \langle s_0, s_1, \ldots, s_j \rangle$ denotes the prefix of a sequence $\pi = \langle s_0, s_1, \ldots, s_j, \ldots \rangle$ starting at s_0 and ending at s_j; if $j = \infty$, then $\pi_j = \pi$. We assume that $\pi^i = \varepsilon$ for $|\pi| \leq i$. Also, note that $\pi^0 = \pi$. In the rest of the thesis, we will use path and execution interchangeably.

The semantics of LTL formulae is defined as follows:

1. $\pi \models p \Leftrightarrow |\pi| > 0$, and $p \in \mathcal{L}(s_0)$,
2. $\pi \models \neg\varphi \Leftrightarrow \pi \not\models \varphi$,
3. $\pi \models \varphi \wedge \psi \Leftrightarrow \pi \models \varphi$ and $\pi \models \psi$,
4. $\pi \models \mathtt{X}\,\varphi \Leftrightarrow |\pi| > 1$ and $\pi^1 \models \varphi$,
5. $\pi \models \mathtt{G}\,\varphi \Leftrightarrow$ for all i, $0 \leq i < |\pi|$, $\pi^i \models \varphi$,
6. $\pi \models \mathtt{F}\,\varphi \Leftrightarrow$ for some i, $0 \leq i < |\pi|$, $\pi^i \models \varphi$,
7. $\pi \models \varphi\ \mathtt{U}\ \psi \Leftrightarrow$ there exists an i, $0 \leq i < |\pi|$, such that $\pi^i \models \psi$ and for all j, $0 \leq j < i$, $\pi^j \models \varphi$.

Unlike the classical definition of LTL [71], our definition, inspired by [4], takes into account both infinite and finite cases. Note that next operator X is defined to be strong (existential). Thus, that for the case of $\pi = \varepsilon$, $\pi \models \mathtt{X}\,\varphi$, as well as $\pi \models \mathtt{F}\,\varphi$, never hold. A KS satisfies a given formula φ, denoted $M \models \varphi$, if and only if for every path π of M, $\pi \models \varphi$.

2.3 Verification using Spin

Spin [44] is a model checker used for verification of asynchronous software systems, mainly communicating protocols. It is built at Bell Laboratories by Gerard Holzmann and Doron Peled. It employs a number of algorithms and techniques for efficient LTL model checking, notably: explicit state enumeration, on the fly model checking, and partial order reduction [16].

Spin supports the specification of correctness requirements expressible in LTL. It can also be used as an efficient on-the-fly verifier for more basic safety and liveness properties, which can be expressed and verified without the use of LTL. Correctness properties can be specified as invariants (using assertions), as LTL properties, or as omega-regular automata in the syntax of never claims [44]. The tool also supports random, interactive, and guided simulation, and both exhaustive and partial proof techniques, based on either depth-first or breadth-first search.

To verify a LTL property, Spin translates it into an omega-regular automaton specifying the negation of the property. By negating the property, a language that formalizes all the error sequences for that property is produced. The Promela model to be verified defines the language of all possible executions of the modeled system. Spin computes the intersection of these two languages. If the intersection is empty, Spin reports that the original property is satisfied. If the intersection is not empty, then it contains the execution sequences that violate the original property. In the latter case, Spin reports these sequences as counter examples to the original property.

2.4 Specification Pattern System

In [88,28], patterns are classified into two categories: Order and Occurrence, and they include:

- *Absence* - A state/event does not occur within a scope;
- *Existence* - A state/event must occur within a scope;

- *Universality* - A state/event occurs throughout a scope;
- *Response* - A state/event P must be always followed by a state/event Q within a scope;
- *Precedence* - A state/event P must be always preceded by a state/event Q within a scope.

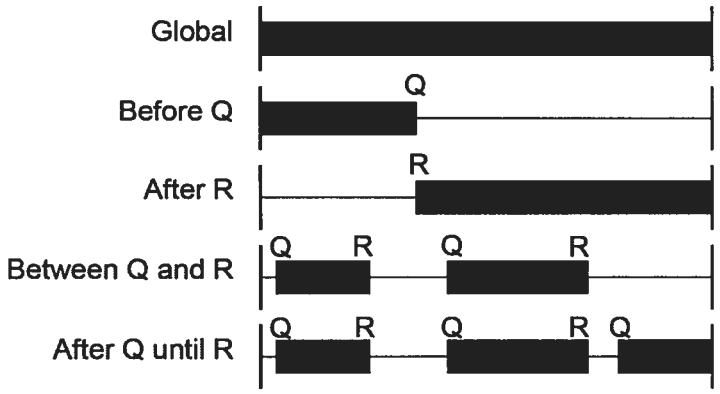


Figure 5. Pattern Scopes

In addition, there are five scopes (Figure 5) that can be associated to patterns:

- *Global* - The entire execution path;
- *Before* - The execution path up to a given state/event;
- *After* - The execution path after a given state/event;
- *Between* - Any part of the execution path from one given state/event to another given state/event;
- *After-until* - Similar to scope Between, except that the designated part of the execution path continues even if the second state/event does not occur.

2.5 Related Work

Formal modeling of WAs is a relatively new research direction. Previous work on the topic includes modeling approaches that target the verification of such applications [21,22,76,77,67,24,25], modeling for testing [7,17,72,73,80,84,2], design and implementation [17,56,64,65,23], and reverse engineering [9,63,81]. We also discuss the related work in temporal logic extensions with scopes.

2.5.1 Verification of Web Applications

In [21,22] an approach is presented where a web site is modeled as a directed graph. A node in the graph represents a web page and the edges represent links clicked. If the page contains frames, the graph node is then a tree whose tree nodes are pages loaded in frames and tree edges are labeled by frame names. The so-called constructive μ-calculus is introduced, based on the standard equational μ-calculus [16] to represent web properties. Based on this formalism, the emphasis is on the verification of properties that are global to a web site including connectivity properties such as the reachability of web pages, frame-dependent properties that are related to the detection of errors in the structure of frames, and cost-of-traversal properties such as the computation of the longest and shortest paths between sets of web pages.

However, the author treats only static pages, concurrent behavior of multiple windows is not modeled, and all the links whose targets could create new independent windows are treated as broken links. Álso, in the author's model, a page loaded in an unnamed window (as a result of the predefined target "_blank" associated with a link) is represented as a graph node that replaces the existing node as if the page is loaded on top of page where the link was clicked; this is due to the inadequacy of the approach to model concurrent behavior of multiple windows.

In [76,77] the authors present a model based on Petri nets to model check static hyperdocuments [76] and framed pages [77]. In [76], Petri nets are used to model Trellis and Hyperties [65] documents to build their corresponding links-automaton. To formally specify browsing properties, HTL and HTL* (Hypertext Temporal Logic), a branching temporal logic notation based on CTL and CTL* [16] is defined. The authors do not extend the basic power of the latter temporal logic languages, but they define specific temporal operators for more naturally expressing browsing properties of hyperdocuments.

Also, the properties defined are not general enough to make them applicable to a wide range of WAs. They relate mainly to navigational aspects of static hyperdocuments such as checking if a certain link is always present/absent, or the menu is always visible.

WAs may include different complex structures and components such as multiple windows, frames, forms, dynamic pages, and thus may need different properties du to the specifics of these components. Correspondingly, the presented model has certain limitations that hinder the representation of those complex aspects of WAs. In [77], an approach based on colored Petri-nets to verify web sites with frames is introduced. The approach is implemented using the SMV model checker. While Petri-nets can express parallel and concurrent behavior, the authors build the overall state space as input of the model checker, which limits the possible use of efficient on-the-fly model checking algorithms. [21,22,76,77] do not tackle the modeling and verification of form-based pages that are dynamically generated by a server program, neither concurrent behavior of applications with multiple windows.

In [67], the authors use symbolic model checking approach to verify properties of e-commerce applications. However, their model relates only to checking functionality of e-commerce applications and cannot be applied to other types of applications such as multi frame/window applications. Similarly, in [24,25], the authors propose using the symbolic model checker NuSMV in a static approach to verifying WAs. However, states in their graph model [25] designate windows, pages, links and actions, which raise questions about potential state explosion.

Each of the above related work concentrates on some aspects of WAs. We conclude that they do not offer a comprehensive solution to ensure correctness and quality of WAs, and which could address a wide range of related properties.

2.5.2 Testing of Web Applications

As described in Section 1.2, there exist numerous web testing tools in the market as well as open source tools which are not sufficient for thorough analysis of WAs. Also, in the literature there exist work on developing approaches and tools for testing WAs. In [7], the authors introduce VeriWeb, a tool that automatically explores web-site execution paths which can be followed by a user in a WA. The main contribution of this work lies in the capability of the exhaustive automatic navigation through dynamic components of web sites. When forms are encountered, they are automatically populated using SmartProfiles,

user specified sets of attribute-value pairs, during web site exploration. Although the underlying approach contributes to the techniques of functional and regression testing, it does not address concurrent behavior of multi frame/window applications. Also, it limits the validation of correctness of dynamic components of WAs to standard features, mainly safety properties, and the graph is traversed only up to a certain predefined depth. This is due to the limited features of VeriSoft [93], the state exploration tool used to detect errors that allows detection of deadlocks and user specified assertions.

The work in [72,73] focuses on inferring a UML model for WAs. This model, merely a class diagram, is mainly used for the static analysis of WAs: HTML code inspection and scanning, data flow analysis, and semi automatic test case generation. In [80], the authors extend the above mentioned modeling technique based on WA execution to extract models for dynamic web pages using server's access logs. These logs present limited information on the requests since they only include the request headers. In case dynamic pages are generated based on POST method requests, the form data submitted is usually stored in the message body of the request; thus, making those pages requests undistinguishable and introduce unnecessary non-determinism into the resulting model. Besides, the modeling approach is inadequate to model concurrent behavior of frames and multiple windows.

In [84], the authors present a modeling technique for WAs based on regular expressions. The focus is on presenting a model of the behavior of WAs merely dynamically generated pages for the purpose of functional testing.

In [2], the authors propose an approach for test case generation of WA. They use hierarchical finite state machines to model WAs.

2.5.3 Design and Implementation of Web Applications

Other approaches for modeling WAs are oriented towards the design rather than analysis of WA. These include object oriented based models [17] and statechart based models [56,64,65,23], that are tailored to forward engineering, logical and hierarchical

representation of WAs. Such models are not suitable for model checking behavioral aspects of WAs. Also, they are not available for analyzing existing WAs developed without formal models.

In [23], the authors introduce HMBS (Hypermedia Model Based on Statecharts), a formal statechart-based approach to model WAs. The authors use the structure and execution semantics of statecharts to specify the structural organization as well as the browsing semantics of hypermedia applications. The approach proposes a mapping of statechart objects into hypertext objects specifically states, transitions, events into pages, links and anchors. HMBS is more concerned with the hierarchical structuring of hypermedia than the navigational aspects included in WAs. Links are described as events that represent transitions of states treating all kinds of hyperlinks in the same manner.

Paulo et al. [64,65] introduce Hypercharts, a formal notation that extends the statechart formalism. In their approach, the authors maintained the statechart characteristics of HMBS such as information structuring, separation of concerns between structure and content, and navigation. In addition, they provide a model to describe timing and synchronization requirements. Hypercharts extends statecharts with timed history, timed transitions, and a set of synchronization mechanisms. Timed history is a mechanism that allows the execution of activities to resume from the point where they stopped. Timed transitions specify the temporal behavior of a multimedia presentation activity. Synchronization mechanisms are used to specify hypermedia synchronization requirements such as multiple data streams. The novelty of the Hypercharts notation lies in its suitability to specify hyperdocuments with embedded dynamic multimedia applications with timing constraints and synchronization. However, as in [23], this approach does not allow the modeling of general hyperdocuments with multiple windows and frames, neither dynamic pages based on form filling and submission.

In [17], the author presents a UML (Unified Modelling Language) model of WAs that represents the business logic of WAs. Different complex aspects of WAs are analysed where dynamic client and server pages, forms, and frames structures are represented as

UML classes. Contents are modelled as stereotyped attributes of classes and links between pages are perceived as class associations.

Leung's approach [56], on the other hand, employs statecharts as the underlying formalism of WAs. His focus is on modeling web navigation rather than presentation details. The presented model depicts navigation of web pages, hyperlinks, navigation within the same page, and frame-based navigation, multiple windows navigation, and dynamic server and client side contents.

2.5.4 Reverse Engineering of Web Applications

In reverse engineering area, there is work aiming at migrating the presentation of WAs to different interfaces with different platforms [81], others aim at maintenance of web sites [9] by checking for style consistency, outdated links, or duplicated content in web pages. In [63], an approach is presented to automatically create task models from web site code where a task model is a tree like structure of tasks describing activities of the web site. Each of the existing related work concentrate on some aspects of WAs leaving out other aspects that remain unaddressed or unfeasible to model using the corresponding suggested approach.

2.5.5 Temporal Logic Extensions and Property Patterns

In logic theory scope imposing is known as relativization, and is used for producing inner models of set theory. relativization was also addressed in dynamic epistemic logic, which is equivalent in expressive power to modal logic. These results are used for public communications in the context of multi-agent systems [70].

Manna and Wolper [59] propose the so-called relativization rules in the context of synthesis of communicating processes. Relativization takes a PTL specification of a single process and transforms it into a specification of the global system. There are certain similarities between these rules and our work. However, there are two main differences between the approaches. First, the relativization procedure they propose applies only on infinite scopes, while our definitions equally apply to both finite and infinite scopes and

models. The second difference is that the relativization transformations [29] apply when the atomic propositions of a given formula hold only in the scope, while we do not make any assumptions on dependencies between the property and the scope.

There exist also several works on introducing and/or extending specification logics based on existing ones, such as LTL and CTL, to ease the expression of properties of systems or allow a wider range of specifications to be expressed using these logics. Examples of such logics are QCTL [52], an extension of CTL with quantification over propositional formulae, XCTL [13], an extension of CTL to De Morgan Algebras, and DLTL [43], a logic that extends LTL by indexing the until operator with regular programs.

The most recent work we know about is the one of Chaki et al [11]. The authors introduce a framework for modeling and verification of concurrent software systems, where both states and events are incorporated. For this purpose, they introduce SE-LTL, a specification logic based on LTL that allows both state and event requirements to be easily expressed.

Beer et al [6] propose the so-called Temporal Logic Sugar. They extend CTL with regular expressions and introduce new operators to formulate properties in CTL. These operators do not add expressive power to CTL, but make it easier for the non-expert user of CTL to specify properties of interest. The operator "next" defined in Sugar is close to the "next in scope" operator we provide, but our definition is more general. The Sugar next is defined as the next state in which a Boolean expression is valid. Our definition states that the property has to be true in the next state, in which a propositional logic expression is valid, relatively to the first state in which the propositional logic expression is valid and not to the first state of the execution path.

Dwyer et al [27,28] present and analyze over 500 temporal properties, classified in the specification pattern system [88]. The patterns introduced constitute abstractions of specifications formulated for different formalisms in which such abstractions are not supported. The patterns are defined on five scopes that represent intervals/regions in which properties should be validated. These scopes have a start state and an end state. However,

the authors did not address the problem of specifying these patterns in a scope of arbitrary set of states. Although the authors mention a class of properties that could be defined based on Boolean variables true in a state, but they state that the specification of these properties is not trivial and offer no solution to this problem. Moreover, there is no methodology to impose those scopes on a given pattern/formula. Since the specification pattern system does not use automatic combinations of scopes and basic patterns, some attempts are made to translate the system to lower-level automata specification languages. Such translations make pattern visualization, fiddling, and elucidation [34,75] possible.

Chechik et al [12,13] extend the pattern system of [27,28] and introduce edge based LTL property formulations in the same scopes introduced by Dwyer. The notion of an edge/event is represented by the "next" operator. Dillon et al [26] introduce in the Graphical Interval logic (GIL) which is similar to the pattern system. GIL provides a mechanism to identify the region/interval of the system execution over which the property should be validated. The operators that identify the intervals introduce a wider and more general range of scopes than the five scopes introduced by Dwyer. However, the operators are used to define segments of the execution path that must have a start and an end. There is no means offered to identify an arbitrary set of states in which properties should be verified.

Though the above cited related work tackled the problem of property specification in certain scope, their solutions cannot be generalized to arbitrary scopes of states. Manna and Wolper [59] proposed a relatively generic solution, but yet with constraining conditions as discussed earlier.

2.6 Summary

In this chapter, we surveyed the major web terminology and components, as well as LTL and the specification pattern system introduced by Dwyer. We also presented a literature review of research work in the areas of verification, testing, analysis and design of WAs. We concluded that each of the existing related work concentrate on some aspects of

WAs leaving out other aspects that remain unaddressed or unfeasible to model using the corresponding suggested approach. In particular, some approaches focus on static WAs with frames, but do not model dynamic pages. Others focus solely on dynamic pages (as well as static) without considering multiple displays (frames and windows). We consider that our proposed approach is more generic, since our models represent static and dynamic pages, single window and multi window/frame applications. Furthermore, unlike some approaches, we use a generic model checker, Spin, such that a wider range of properties could be verified on the inferred models. In addition, we surveyed the literature related to temporal logic extensions, where we found out that solving the problem of property specification in arbitrary scopes of states has not been widely addressed. One research addressed this problem but with constraining conditions as discussed previously.

In the next chapter, we present a specification pattern system tailored to the verification of web applications.

Chapter 3

Web Specification Patterns System

At the heart of the research efforts dedicated to the web development and analysis, lie implicit postulations about "quality" and "usefulness" of web resources and services. In order to maintain a high quality of WAs by means of the application of our formal approach to verify properties of WAs, we study the attributes that designate the good quality of such applications. Web quality assurance specialists have long studied the requirement, design, and implementation of various types of WAs and were able to set norms which could serve as vital guidelines to the quality assurance of WAs.

3.1 Quality Assurance of Web Applications

In this chapter, we present a discussion over the main attributes that determine the quality of WAs. We also present a non-exhaustive set of web quality requirements that are indispensable to meet those attributes and that could serve as web specifications necessary for quality assurance of WAs. In the context of our formal framework, these attributes designate formal properties to be verified against WA models. However, to alleviate the hurdle of learning the mathematical foundations underlying the temporal logic and model checking theory for web users, we propose a formalization of those quality requirements into LTL. The formalized properties could serve as library of web specification patterns that can be verified in a given WAUT. Finally, we discern existing quality requirements that could not be formulated in LTL and thus cannot be verified using our formal approach.

In [47,62], the authors discuss the main quality attributes of WAs, which are explicitly dubbed as success criteria [47] for WAs. The most important quality attributes of WAs are found to be *reliability and functionality*, *usability*, and *security and privacy*. These attributes embody the appeal and trust of users, whom if dissatisfied simply move away to

another better quality WA. Therefore, these attributes are crucial for the success of WAs. Other quality attributes of WAs are *availability*, *scalability*, *maintainability*, and *performance*. In this research, we study the first three attributes, namely reliability and functionality, usability, and security and privacy. The main reason is that since we adopt an approach based on external observation of executions of WAs without any access to source codes or web servers, checking the latter four attributes (availability, scalability, maintainability, and performance) in WAs necessitate mostly static analysis approach and access to server's files and programs. Therefore, the four latter attributes fall outside the scope of this thesis. On the other hand, we can identify another attribute which we believe is indispensable to the success of WAs, which can be called the *custom* attribute.

3.1.1 Reliability and Functionality

Reliability has long been perceived as an attribute of safety critical software applications, such as telecommunications, aerospace, and medical devices. Therefore, most of the industry's software applications are non-safety critical and thus do not need to be highly reliable. However, WAs are critical to the commercial success of many businesses and if their WAs do not work reliably, the businesses will not succeed. In [47], the author characterizes web users as customers, unlike software applications users. Therefore, web users are able to make choices based on how reliable the WA is, while software application users tend to tolerate many flaws of the software as long as it can serve their expected purposes. For example, e-commerce applications deal with crucial items, which are money and personal information, such as credit cards, addresses, and buying habits. These applications also offer unrecoverable transactions when making online purchases, and have deferred results since shipping purchases takes days or weeks. These factors mean that if WAs are unreliable, businesses will lose customers, and may lose large amounts of money. Therefore, WAs reliability is vital to their business's success and companies are willing to spend resources to ensure high reliability.

Functionality of WAs is a quality attribute that is intertwined with reliability since a dysfunctional WA cannot be reliable. WA pages and components should reflect its

requirements. Therefore, each component and web page should have specific functions that it must perform in order to fulfill one or more of the WA requirements such as catalogue information presentation or users' payment data gathering.

3.1.2 Usability

Web users expect WAs to be as simple to learn as shopping at a store. Over the years, many web specialists have studied software usability and web usability. However, many WAs still do not meet the usability requirements anticipated by web users. On the other hand, web users exhibit little loyalty to a given WA since they expect to use it without any training; i.e., WAs that are not usable will not be used [62]. In summary, WAs must be behave according to the user's expectations, offer only needed information, and provide navigation controls that are clear and obvious.

Specifically, WAs features need to be logical, accessible, and intuitive. Therefore, web developers should provide web users with a pleasant and efficient experience. Usability engineering represents the effort to make WAs understandable and usable by each web user.

3.1.3 Security and Privacy

At times when the web consisted only of static documents accessed via hyperlinks, the consequences of security breaches were relatively small. With the evolution and growth of WAs nowadays, business corporations face significant losses in revenue, large repair costs, and legal consequences due to security breaches of their WAs, let alone the loss of their credibility with their customers. Thus, it is essential that WAs safeguard user's data and other electronic information. At the same time, web users should be confident that no unauthorized users can access transactions or personal information stored and used by the WA.

3.1.4 Custom Attribute

While the above stated attributes are generic to WAs and relate to the satisfaction and appeal of the user, the custom attribute is related to the satisfaction of the various stakeholders that are involved and have interest in the WAs. For example, while business owners are mostly interested in features which could result in directly achieving financial gains from the WA, advertisers are more interested to ensure that their commercial advertisements properly appear in key web pages of WAs and strategically published. Also, some WAs require particular specifications that have to be covered by the existing quality requirements related to the above stated attributes. For instance, in WAs related to governments or political parties, the interest might be in carefully using some statements, phrases or words, which has to appear a certain number of times, in key web pages. Also, in banking WAs, where security is the most regarded feature even above many other features, clients are allowed to mistakenly try to login to their accounts no more than three times, where the system is permanently locked afterwards. We list few examples of these requirements that we were able to formulate:

- *The number of a certain string/object should not appear more than a specific threshold*
- *If yahoo search is available, then google search should also be available*
- *A page X should be reachable without going through page Y*
- *The User visits Authentication page then Secure page then returns to Authentication and does this exactly twice*
- *A combination of specific strings/objects is absent throughout the WA*
- *Incorrect login info is allowed only three times, and then login is forbidden*
- *Number of links is balanced among multiple displays*

Therefore, the requirements or specifications to satisfy the custom attribute are pretty much subjective to the stakeholders themselves. For this reason, it is not possible to set a fix number of requirements which can satisfy the custom attribute. We collected a number of these requirements from the surveyed research work in Chapter 2, as well as few of them we were able to deduce from browsing through particular WAs.

3.2 Pattern based Approach to Property Verification of WAs

Formal methods have been recently recognized as a valuable means to provide dependable and practical solutions for the analysis and verification of applications in various domains. In particular, model checking techniques have been widely supported by diverse tools and methods for automatic verification of systems, overcoming several drawbacks related to the scalability of the involved modeling and analysis techniques, and the state space explosion problem. However, researchers have long acknowledged an important impediment to the transfer of those techniques from research to practice. Despite the automation of verification techniques, users of model checkers still must be able to specify system requirements in the specification language of the model checker. Namely, there exists a costly learning curve associated to the understanding of the underlying mathematical foundations, in particular, mastering the temporal logic theory for property specifications for model checking of systems.

As an example, we present the following requirement for some types of WAs:

Incorrect login info is allowed only three times, and then login is forbidden.

To verify this requirement using LTL model checker, the web developer or tester has to translate it into the following LTL formula:

G (*login* → X (!(*forbidden*) or (*login* and X (!(*forbidden*) or (*login* and X (!(*forbidden*) or X (*forbidden* and G (!*login*)))))))

Clearly, writing such formula is a daunting task for web developers and testers. Not only the formula is difficult to read and understand, but it also difficult to correctly formulate without having the expertise in LTL.

To alleviate the above stated problem, we propose a pattern based approach to present the web specifications in an intuitive and easy to use manner. Originally, design patterns [32] are developed to capture descriptions of recurring solutions to design problems, requirements of these solutions, the means to satisfy those requirements, and instances of the solutions. Practitioners, when identifying similar requirements in their

systems, use the appropriate design patterns addressing those requirements, and instantiate solutions that represent those patterns. Recently, Dwyer and al. have developed the Specification Pattern System (SPS) [88,27,28], where a property specification pattern describes the essential structure of some aspect of a system's behavior and provides expressions of this behavior in a range of common formalisms [88]. However, we are not aware of any attempt to build a library of specification patterns that catalogue the requirements necessary to ensure high quality WAs, and represent them in an easy to use manner mainly to the web developer and tester.

One could argue that the SPS could be used to represent web related specifications. Nonetheless, the SPS is a result of a conducted survey of property specification examples collected from various resources that deal with finite state verification of distributed systems, reactive systems, and timed systems; none of the surveyed resources included verification of WAs. For this reason, when we formulate the collected web quality requirements, although we employ patterns from the SPS, many of the web related formulae fell outside of the range of the SPS's patterns. In other cases, we should use several SPS patterns to specify a single web pattern.

On the other hand, the SPS defines a set of scopes in which a property can be verified. Each of the scopes is an interval of states/events with start and end delimiters. Meanwhile, in our work we identify scopes that constitute arbitrary sets of states due to two main reasons:

1. Many of the quality requirements surveyed concerned only a subset of the web pages of the WAs.
2. Due the nature of our modeling approach that relies on communicating automata, in the case of multi-display WAs, we identify two subsets of states, namely stable and transient, as explained in Chapter 5. Some of the properties need to be checked in the stable states only, otherwise the verification of the properties results in erroneous outcomes.

Our work on extending LTL with scope operators to specify properties in arbitrary sets of states is discussed in details in Chapter 6. Therefore, a number of our patterns employ the proposed scope operators to specify the related properties. On the other hand, we believe that the scope operators can be easily used in most of the patterns to specify them for multi frame WAs.

In the following, we present our results in developing a library of web property specifications; we call it the Web Specification Pattern System (WeSPaS). This work is based on surveying several resources that deal with web quality assurance [88,62,58,92,79], and web usability namely, IBM usability group [92], and various research work in the area of analysis and verification of WAs, we also relied on results of a Ph.D. research work in progress [58]. In addition, we deduce some requirements by observing particularities related to some types of applications. In these resources, various requirements are developed for quality assurance of WAs. Most of surveyed requirements are listed in the above mentioned resources as rules that satisfy the quality attributes described in Section 3.1, namely reliability and functionality, usability, and security and privacy. We have identified 120 common requirements that can be formally specified and used in model checking of WAs.

3.2.1 Categorization

A standardized categorization of property specifications have long been that of the known *safety* and *liveness*. Safety designates the class of those properties that ensure that nothing bad will ever happen, and liveness the class of those properties that ensure that something good will eventually happen. Other proposed categorization is that of the SPS which classifies properties into order and occurrence as described in Chapter 2. However, we believe that those mentioned categorizations are too abstract and difficult for a web developer or tester to employ during the analysis of WAs.

On the other hand, we do not keep the classical categorization of quality attributes. The reason is that some specifications identified as custom, or stakeholders specific, might overlap with the other quality attributes, while others might not fit in any of them. On the

other hand, when analyzing the various quality requirements, we are able to distinguish between requirements related to the ergonomics and design of web pages, and requirements related to the functionality of WAs which could cover a range of web pages of interest. For this reason, we propose a classification of those requirements (thus patterns) following two main categories:

Non-Functional. These requirements apply mainly to the design and ergonomics of web pages. They deal with standardization requirements (as in page 57) related to links in pages, content management of pages, and navigation between pages. Therefore, we identify sub-categories:

1. Links
2. Content
3. Navigation

Functional. These requirements concern the functionality of WAs. We notice that many of these requirements concern a wide range of WAs. On the other hand, we realize that e-commerce WAs have specific requirements when it comes to the functionality, which do not necessarily apply to other types of WAs. Therefore, we identify two sub-categories, one concerns WAs in general, and the other concerns e-commerce applications. Within each sub-category, we identified the following groups:

1. General
    1.1. Reachability
    1.2. Security and Authentication
    1.3. Other
2. E-commerce
    2.1. Customer Service
    2.2. Trust
    2.3. Product Info and Navigation
    2.4. Purchase Transaction

3.2.2 Template

In order to archive the web property patterns in our library, we have to define a standard template that provides useful information about a given property pattern. We propose the following items to be included in the template:

- ID: a unique identifier for each pattern.
- Pattern description: an English description of the quality requirement.
- Category: the category and subcategory to which the pattern belongs.
- Attributes: the involved page attributes in the LTL formulation.
- LTL Mapping: the mapping of the quality requirement into LTL formula.
- Comments: additional information concerning the pattern and its formulation.
- Source: the source where the quality requirement of the pattern has been found.

These fields of the template help the web developer and tester identify the properties of interest, and provide the LTL formulation of the properties to verify. Also, the template's fields help in extending the library of patterns with new patterns. Note that each pattern is assigned a unique ID which encodes the first letters of the main category and subcategories followed by the number of the requirement. For convenience, we identify the sources of the quality requirements as follows:

- *Opquast* for [79],
- *IBM usability group* for [92],
- *Research in progress* for [58],
- *Literature* for [76,22], and
- *Newly introduced* for patterns identified during the research work of this thesis.

3.3 Library of Web Property Patterns

In this section, we present the web property patterns we have archived in our library, categorized in their corresponding groups. Quality requirements are formalized as LTL properties that could be checked by a model checker given that the atomic

propositions that constitute those properties are attribute valuations that exist in individual pages. We only present examples of full patterns for each category and sub-category; the complete library can be found in [40]. Finally, we discuss examples of quality requirements that are not checkable using a model checker.

3.3.1 Non-Functional Patterns

In this section we present the non-functional patterns. They are classified into Links, Content, and Navigation. They merely concern standard requirements for good design and ergonomics of WAs.

3.3.1.1 Links

1. Broken links are absent.
2. Deadlocks are absent.
3. URLs have "/" at the end.
4. Only links are underlined.
5. Number of links in each display (page) should not be more than a given number (depends on size of application).
6. Internal URLs have no spaces.
7. URL of current page is not a link in page itself.

ID	NFL5
Pattern description	Number of links in each display (page) should not be more than *n* (depends on size of application)
Category	Non-functional – Links
Attributes	*num_links*: number of links in individual pages *n:* threshold of number of links
LTL Mapping	UniversalityGlobally ($num\_links \leq n$)
Comments	SPS pattern "Universality" is used within the scope "Globally"
Source	Research in progress

Table 1. Link Pattern Number 5.

We present in Table 1 an example template of one the Link pattern number 5.

3.3.1.2 Contents

1. Number of fonts used is less than 4.
2. Size of text is greater than 10.
3. Always name of the site or application is visible.
4. Always logo of the site or application is visible.
5. Site map exists.
6. Index exists.
7. Glossary exists.
8. Always there is a link to home page.
9. Always there is a link to important information.
10. Titles are short (less than 25 characters).
11. Date of last changes exists.
12. Number of frames less than 4.
13. If images exist, their format is gif or jpeg.
14. Address is present (email, phone, fax, postal).
15. Visitors counter is present.
16. Mandatory and optional form fields are distinct.
17. Every text field has a label in forms.
18. Frames are absent in application.
19. In all addresses country is mentioned.
20. Area code is present with telephone number.
21. Menus are present on all pages.
22. 404 page is personalized and main menu appears on it.
23. If plug in is suggested then size should be indicated.
24. Pop-ups have a close button.
25. The number of a certain string/object should not appear more than a specific threshold.
26. If yahoo search is available then google search should also be available.
27. A combination of specific strings/objects is absent.

Table 2 is an illustration of Content pattern number 22.

ID	NFC22
Pattern description	On 404 page, main menu is always present
Category	Non-functional – Content
Attributes	*status*: HTTP status code *num_links*: counts the number of links *menu_exists:* Boolean indicating the presence of menu
LTL Mapping	UniversalityGlobally (*status* = 404 → *menu_exists* and *num_links* > 0)
Comments	SPS pattern "Universality" is used within the scope "Globally"
Source	Opquast

Table 2. Content Pattern Number 22.

3.3.1.3 Navigation

1. Link to a printable version exists.
2. Link to FAQ exists.
3. Link to "what's new" page exist.
4. Link to copyright exists.
5. Link to Author name, qualifications, editor, webmaster, department, etc. exists.
6. Link to date of creation, publication, expiration, last modified exists.
7. There exist 1 or 2 links to Author or webmaster.
8. There exist 1 or 2 links to moderator of public forums.
9. If forum exists, then conditions of moderation exist.
10. If plug in is suggested then a link for it is present.
11. If plug in is suggested then a link for it is present in all pages.
12. After an order is submitted, the time for delivery is indicated.
13. If newsletter exists, frequency of sending newsletters is present in application.
14. Schedule and charge of services are present.

As an illustration, we present in Table 3, the Navigation pattern number 6.

ID	NFN7
Pattern description	There exist 1 or 2 links to Author or webmaster
Category	Non-functional – Navigation
Attributes	*webmaster*: integer identification of webmaster page *webmaster_link*: Boolean indicating the presence for link to webmaster page
LTL Mapping	ExistenceGlobally (*webmaster*) → BoundedExistenceGlobally (*webmaster_link*)
Comments	SPS pattern "Existence" is used within the scope "Globally"
Source	Opquast

Table 3. Navigation Pattern Number 7.

3.3.2 Functional Patterns

Here we present the functional patterns. They are classified into two main sub-categories: General, and E-commerce. These patterns are related to the functionality and expected behavior of WAs.

3.3.2.1 General WAs

The patterns of this class concern WAs in general. They include specifications about the reachability of certain pages. They also include specifications concerning the security and authentication in WAs, as well as other general specifications.

Reachability

1. Home page is reachable from every page.
2. Always link to a text version of page exists.
3. Always help page is accessible.
4. Always reference page is accessible.
5. Site map is reachable from every page.
6. Copyright information is reachable from every page.
7. If the WA is for a specific audience, it should be mentioned at least in home page.

8. If different language of application is available, then it is reachable from every page.
9. A page X is reachable without going through a page Y.

As an illustration, we present in Table 4, the Reachability pattern number 1.

ID	FGR1
Pattern description	Home page is reachable from every page
Category	Functional – General – Reachability
Attributes	*home*: integer identifying home page
LTL Mapping	Negation: AbcenseGlobally (*home*)
Comments	The SPS pattern "Absence" is used within the scope "Globally". To check this pattern, it is negated. The property formulated is "on all paths home page is absent". If the result of verification gives a counter example, it means the model checker found at least a path containing the home page. Then the original property is valid.
Source	Litterature

Table 4. Reachability Pattern Number 1.

Security/Authentication

1. Always the logo for security license is present.
2. Always the link for security information is present.
3. Incorrect login is allowed only 3 times, and then login is forbidden.
4. Secure pages are opened in a different window.
5. Secure pages are not reachable without going through authorization pages.
6. Banking information is entered no more than once before submitting form.
7. User visits Authentication page then Secure page then returns to Authentication and does this exactly twice.

As an illustration, we present in Table 5, the Security pattern number 6.

ID	FGS6
Pattern description	Banking information is entered no more than once before submitting form
Category	Functional – General – Security and Authentication
Attributes	*Banking_info*: Boolean identifying the presence of fields for banking information *Submit*: identification of page where form submit action exists
LTL Mapping	PrecedenceGlobally ((!(*banking_info*) W (*banking_info*))), *submit*)
Comments	SPS pattern "Precedence" is used within the scope "Globally"
Source	Newly introduced

Table 5. Security Pattern Number 6.

Others

1. Search function is always present on all pages.
2. If WA is larger than 15 pages, then advanced search is always available.
3. If page is a questionnaire or comments, then feedback page is next.
4. In forms, a link to go back and correct fields exists.
5. In forms, a validation / feedback page exists before submit action.
6. In forms, a confirmation of submission exists after submit action.
7. In forms, a link to info on how to fill out forms exists.
8. In forms, a link to security and privacy info is always present.
9. In forms, a preview page allowing changing info exists before submit action.
10. User can proceed navigation with links after form submission.
11. Pop-ups appear only once.
12. Pop-ups are absent from internal pages and home page.
13. Certain contexts (adult, violence, etc.) are absent.
14. Subscribe to / unsubscribe from newsletters exists.

In Table 6, we present the pattern number 13 of this category.

ID	FGO13
Pattern description	Certain contexts (adult, violence, etc.) are absent
Category	Functional – General – Other
Attributes	*adult_str*: string count of a specific prohibited adult related phrase/word *violence_str*: string count of a specific prohibited violence related phrase/word
LTL Mapping	AbsenceGlobally ((*adult_str* > 0) and (*violence_str* > 0))
Comments	SPS pattern "Absence" is used within the scope "Globally"
Source	Newly introduced

Table 6. Others Pattern Number 13.

3.3.2.2 E-Commerce WAs

The class of E-commerce patterns comprises specification patterns that concern mainly e-commerce WAs. We archive those patterns into the following sub-categories: Customer support, Trust, Product Info and Navigation, and Purchase Transaction. Note that this classification is manly adopted from the Usability Group at IBM [92].

Customer Support

1. Contact info exists and is accessible from every page.
2. Assistance for passwords (change, remember, …) exist.
3. There are three modes of product search: search function, list of products, product suggestion.
4. FAQ / glossary are accessible within 3 clicks.
5. Link to FAQ exists on all shopping pages and "How to order" pages.
6. Shipping and delivery date info is accessible within 3 clicks from any page.
7. After submitting an order, there is a link for tracking order or the shipper's site.
8. It is possible to change or cancel an order after submission.
9. After order submission, info on warranties and service agreements is accessible.
10. Info on terms and conditions is accessible before order submission or check out.

11. Link to info on reimbursement / return conditions and procedure is present on order pages.
12. Info on security / level / mode is always present.
13. Info on conditions of buying / usage is reachable from all pages.

In Table 7, we present the pattern number 4 of this category.

ID	FEC4
Pattern description	FAQ / glossary is accessible within 3 clicks
Category	Functional – E-commerce – Customer Support
Attributes	*faq*: integer identification of the FAQ page
LTL Mapping	Negation: AbsenceGlobally(!*faq* → X (*faq* or X (*faq* or X (*faq*))))
Comments	The SPS pattern "Absence" is used within the scope "Globally". To check this pattern, it is negated. The property formulated is "on all paths FAQ/glossary cannot be found within 3 clicks of every page ". If the result of verification gives a counter example, it means the model checker found at least a path containing the home page. Then the original property is valid.
Source	IBM usability group

Table 7. Customer Support Pattern Number 4.

Trust

1. Privacy policy is accessed from every page.
2. Security policy is accessed from store front, order list, shopping cart, and order form.
3. There exists third party validation seal for trust.
4. Mission statement, history, etc. exist.

In Table 8, we present the pattern number 2 of this category.

ID	FET2
Pattern description	Security policy is accessed from store front, order list, shopping cart, and order form
Category	Functional – E-commerce – Trust
Attributes	*front*, *order_list*, *shopping_cart*, *security_policy*: integers identifying the corresponding web pages
LTL Mapping	G (*front* or *order_list* or *shopping_cart* → X (*security_policy*))
Comments	
Source	IBM usability group

Table 8. Trust Pattern Number 2.

Product Info and Navigation

1. Product info is reachable within 3 clicks.
2. From home page the following is provided: search products, browse categories, and recommend products.
3. Shopping pages are accessed from every other page.
4. "Best selling" pages are accessed directly either from home page or product category page.
5. Link to a product description and adding product is available from shopping cart page.
6. Within the same product category, product description pages are linked with forward and backward buttons/links.
7. A link to catalogue exists on every page.
8. Promotions exist only on home page and category navigation pages.
9. 1 or 2 promotions exist simultaneously with a link to more promotions.
10. Promotions of certain products are only present either on the Home page or on Shopping pages and their number does not exceed 2.
11. Promotions are absent on product comparison pages and ordering pages.
12. Cross-selling and up-selling pages are reachable only from product detail pages.
13. Types of accepted payments are indicated and present on all product pages.

In Table 9, we present the pattern number 4 of this category.

ID	FEPN4
Pattern description	"Best selling" pages are accessed directly either from home page or product category page
Category	Functional – E-commerce – Product Info and Navigation
Attributes	*home*, *category*, *best_selling*: integers identifying the corresponding web pages
LTL Mapping	UniversalityGlobally (*home* or *category* → X (*best_selling*))
Comments	SPS pattern "Universality" is used within the scope "Globally"
Source	IBM usability group

Table 9. Product Info and Navigation Pattern Number 4.

Purchase Transaction

1. It is possible to view a product description, total cost, change a quantity, add/remove a product, save an order, and proceed to order form.
2. After an order is submitted, a confirmation page is shown to the user.
3. From confirmation page, it is possible to reach registration page (acquire ID, and password).
4. If the user registers, then he can use shortcut order (by entering only ID, password, and credit card number).
5. There is a link from the order form to the order list.
6. Credit card info is submitted no more than once before submitting an order.
7. After ordering, conditions of guarantee are available for the user.
8. After ordering, address of return is indicated.
9. After ordering, return service (charge, hours of work, contact, etc.) are indicated.
10. Availability of products is indicated before validation of order.

In Table 10, we present the pattern number 6 of this category.

ID	FEPT6
Pattern description	Credit card info is entered no more than once before submitting an order
Category	Functional – E-commerce – Purchase Transaction
Attributes	*Credit_card*: Boolean indicating the presence of fields requesting credit card info in pages *submit*: Boolean indicating the submit order page
LTL Mapping	PrecedenceGlobally ((!(*credit_card*) W (*credit_card*)), *submit*)
Comments	SPS pattern "Precedence" is used within the scope "Globally"
Source	Newly introduced

Table 10. Purchase Transaction Pattern Number 6.

3.4 Discussion

In this work, the library of web specification patterns covers a number of important quality requirements of WAs. The list of those patterns is non exhaustive and can be further extended with new patterns. As discussed before, the web properties addressed here are mostly generic and apply to a wide range of WAs as well as to e-commerce WAs. Therefore, we introduced properties that can be seen custom to specific WAs or stakeholders such as banking, or government related applications as well as special e-commerce applications such as e-Bay. The list of those properties cannot be limited and is open to further contributions. Also, this library does include all types of quality requirements. The reason is that our approach is based on formally specified patterns. Therefore, it is important to identify the elements of the quality requirements as attributes that can be evaluated within the web pages of the WA. Those attributes when evaluated, constitute the atomic propositions of the LTL formulae of the corresponding patterns.

For this reason, there exists a range of quality requirements which we have identified but are not able to specify as LTL properties. Those requirements are either:

1. too generic to the extent that it is impossible to identify corresponding attributes in web pages of the WAUT, or
2. they describe necessary requirements for the quality of WAs which do not directly involve the WAUT itself, but rather involve the business or company that owns the WA.

We list a few examples of such requirements:

- 404 error page is personalized.
- At least two modes of payments can be performed by the user.
- An answer to a request for information is sent within five working days of reception.
- An acknowledgment of reception of a request is sent by email within 60 minutes.
- The nature and quantifiable characteristics of products and services (dimensions, duration, capacity, etc.) are stated.
- Subscription to newsletters is submitted to a verification procedure.

Also, it is known that temporal logics, especially LTL and CTL, have certain limitations to express all types of properties. Therefore, we acknowledge that some properties which could be expressed in CTL cannot be expressed in LTL.

On the other hand, many of the patterns of our library employ specification patterns from the SPS. Also, the patterns of the SPS can be mapped not only to LTL, but also to a range of specification languages such as CTL, and QRE. Therefore, we believe that this adds to the flexibility and richness of our library of patterns. In particular, the web patterns which are composed of SPS patterns could be easily formulated using the available SPS's mappings to the various specification languages. Such flexibility makes our system of web specification patterns more generic, and the verification of WAs possible using various model checkers and analysis tools.

In this chapter, we discussed key quality attributes of WAs and related requirements to assure a high quality of those applications. These requirements were collected from a variety of sources dealing with quality assurance of WAs. We introduced the WeSPaS, a web specification pattern system, where each pattern designate a quality requirement,

identified by a unique ID, its involved page attributes, mapping to LTL, and its source. The WesPaS is classified into functional and non-functional patterns. The complete system of web specification patterns can be found in [40].

Having defined the major types of properties of WAs to be verified, we elaborate in the next chapter our formal approach and methodology to run-time verification of WAs.

Chapter 4

Run-Time Verification of Web Applications

In this chapter, we describe our methodology and approach for analysis and verification of WAs, and develop a method that infers an automaton from a behavior a single window application observed during browsing. We also discuss properties which could be verified using the inferred automaton and prove that this automaton preserves these properties. We also present methods to infer communicating automata models for multi window/frame WA.

4.1 Dynamic Approach and Methodology

The purpose of building a formal model for a WAUT is to verify whether the application exhibits certain predefined properties using model checking techniques. We assume that the properties to be specified in a temporal logic of a chosen model checker are composed of atomic propositions, and for each page the value of each proposition is uniquely determined by the content of the page, be it dynamic or static. These propositions refer to the page attributes that are user defined and have to be checked (and thus reflected in a model). As described in Chapter 1, when the code is not used for modeling, one can build a formal model following a dynamic, black-box based approach, by executing the application and using only the observations of an external behavior of the application. Verification of such models is often called run-time verification [3,10]. In case of WAs that rely on the HTTP protocol considered in this work, an "external" behavior consists of requests and responses. We follow the dynamic approach and assume that the request/response traffic between a client side and a server in the WAUT is observable. One

possible way of achieving this is to use a proxy server [57]. A proxy server monitors the traffic between the client and the server and records it in proxy logs. The proxy logs contain the requests for the pages and the responses to these requests.

With this approach, a behavior of a WAUT, called a *browsing session*, is interpreted as a possible sequence of web pages that have the same domain name intermittent with the corresponding requests. Usually, those requests are triggered by the user's actions (clicking links, submitting forms). Note that a behavior of a WAUT is independent of the navigation aids provided by the browser (back button, forward button, etc.). In other words, we assume that the user navigates the WAUT using only the provided links and forms, by clicking links and submitting filled forms; thus, we build a model that is independent of a browser. We assume that a next request is not submitted before the browser delivers a response to a previous request. The user may next choose to click on a certain link or submit a filled form, for which another web page is loaded (either in the full browser window or in a frame/frameset within the same window, or in a newly opened window different from the browser window), and so on. When the user clicks links that cause scrolling within the same page, the WA is not affected, as scrolling is performed by the browser and not by the server. Such links do not trigger any request to the server and hence should not be represented in a model that describes an external behavior of the WA. If the user clicks a link which leads to a page with k frames, then k+1 request/response pairs are observed. The first request/response pair corresponds to the link clicked and, thus, to the frameset document; and k requests, initiated by the browser, along with their responses, correspond to the URIs defined in the frameset document. In other scenarios of WA navigation, the browser automatically initiates a request for a page when an HTML HTTP-EQUIV tag is specified in the already displayed page. An example of an HTTP-EQUIV tag is: <META HTTP-EQUIV = "Refresh" CONTENT = "2; URL = http://nextdoc.html">; when HTTP-EQUIV = "Refresh", the browser performs the reload action. The CONTENT field specifies a delay, in seconds, before the browser automatically loads either the current page or, when the URL field has a different value than the URI of the current page, the page that corresponds to the URL field. Such scenarios are used to notify the user of a redirection to a

different page, or to notify the user to wait until the resulting page is loaded. A browsing session terminates when the user explores the pages he intends to visit or decides simply to stop browsing. Exhaustive exploration could hardly be achieved for non-trivial WAs with a database tier. This is why, following the dynamic approach, one can model only a part of the WAUT, which is exposed in a browsing session. Note that in the web pages, there could be links that use protocols different from HTTP, such as FTP for data transfer, SMTP or IMAP for e-mail transfer. We assume that these links take the user out of the WAUT and require, therefore, that the user do not click on those links and only navigates through HTTP based links that keep him within the application under test. There are cases where the user clicks on links that are broken or do not result in displaying the requested page. Instead, the observed response would be a web page containing an error message that explains the status of the requested page or a pop up window with an ok button displaying the error. Therefore, to distinguish successful responses from erroneous ones, it is important to extract the status code of each observed response. Our interest is in the 4xx and 5xx codes that indicate errors in the request due to the client or errors on the server side. The observation of these codes is essential to distinguish error pages responses from normal successful ones. This enables checking certain properties such as the validity of the page, the accessibility of the page, or its reachability only through other pages that perform authorization, etc. In case of responses with the redirection status code 3xx, the server fetches the new location and sends the new URI back with the response header field called "Location".

To generate sequences of requests one may consider using a crawler instead of the user to explore links in the WAUT [91], though in case of pages with forms to fill, the user actions would still be required. Moreover, web masters, wary of threats related to crawler use, overload on the web server or copying of copyrighted material, prohibit automatic exploration of their websites [49]. In the next section, we present an approach for building finite automata that model a browsing session. Although usually model checking techniques rely on KS based modeling, many model checkers, such as Spin [44], use

automata based languages to describe KSs. Therefore, the finite automata modeling is justified for our approach to web model checking.

4.2 Modeling Single Window Applications

We first present our modeling approach for WAs whose web pages do not have frames and assume that the WA is browsed in a single browser window, in other words, that all the links have their target attributes either undefined or equal to "_self". Later we provide extensions to more complex applications.

As we mentioned before, the purpose of building a formal model for a WA is to validate whether the application exhibits certain predefined properties. We assume that the properties to be specified in a temporal logic of a chosen model checker are composed of atomic propositions, and for each visited page the value of each proposition is uniquely determined by the content of the page, be it dynamic or static. These propositions refer to the page attributes that have to be checked (and reflected in a model). Page attributes can be of various types, for instance: a numerical type to count the occurrence of a certain entity, a string type to denote the domain name of a page, or features of a page link, such as a hypertext associated with the link. However, there are cases when an attribute representing a certain feature of the visited page cannot be defined for another page. For instance, a Boolean attribute that indicates whether the menu is framed in a page that does not contain menus, or an attribute representing the percentage of the number of occurrences of a certain string with respect to the number of all the strings in a page that contains no text. In such cases, we assign to these attributes the value "not available". Unfortunately, only few experimental model checkers support multi-valued logic. One way of solving the problem is to introduce an auxiliary flag variable that indicates appropriateness of the attribute. However, in many cases, as a workaround, it is sufficient to assume that the atomic propositions corresponding to those attributes whose value is "not available", are false in the corresponding pages.

In the following, we describe how to use automata to model an observed behavior of a WA based on the information available in the corresponding browsing session. The session includes requests initiated by the user, namely link clicks and filled form submissions, as well as requests initiated by the browser, namely requests for URIs present in an HTTP-EQUIV tag; for simplicity, we call those URIs *implicit* links.

4.2.1 Definitions

Each request in the browsing session is represented by a string l. In case the request method is Get or Head, l is the URI sent in the request. If the request is for a filled form then we represent it in the form $a?d$, where a is the form action and d is the form data set which represents the data fields filled in the form; in case of the Get method, data set is a part of the URI sent in the request, while in case of Post method, data set is included in the message body as a data stream.

Each response corresponding to a visited page is abstracted by a tuple $<u, c, I, L, V>$, where u denotes the request l, identifying the page (we use the terms response, page, and response page interchangeably); $c \in C$ represents the status code of the page, C is the set of valid status codes defined as integers ranging between 100 and 599 [57]; I is the set of URIs specified by the action attribute of each form in the page; L is the set of URIs associated with links, including the implicit links if any, in the page (L does not include links that cause the scrolling to sections in the same page); and V is a vector $<v_1, \ldots, v_k>$, where v_i is the valuation of the page attribute i and k is the number of all the page attributes over which the atomic propositions are defined. Pages with status code 2xx are the ones rendered successfully and have their URL u equal the request l. Pages with status code 3xx have their URL u different from the request l that triggered the response due to a redirection to another location of the pages. Pages with status code 4xx or 5xx may or may not have links leading back to the application.

A *browsing session* is a Request/Response Sequence $RRS = <u_0, c_0, I_0, L_0, V_0> l_1 <u_1, c_1, I_1, L_1, V_1> \ldots l_n <u_n, c_n, I_n, L_n, V_n>$, where $<u_0, c_0, I_0, L_0, V_0>$ represents the default page displayed in the browser window from which the first request was triggered; this page

is not observed in the browsing session, therefore, u_0 and c_0 are null, and I_0, L_0, and V_0 are empty sets; l_i is a request that is followed by the response page $<u_i, c_i, I_i, L_i, V_i>$; for all $i > 1$, $l_i \in L_{i-1}$ if l_i is a request corresponding to a clicked or implicit link, or if l_i is of the form $a_i?d_i$, then $a_i \in I_{i-1}$; and for all $i > 0$ $l_i = u_i$ if $c_i \neq 3xx$; (otherwise, $l_i \neq u_i$); and n is the total number of requests in the browsing session, starting from the first request l_1 for the home page of the application. Page attributes, along with u and c, are considered as state attributes and used for model checking in a way similar to KS [16]. We denote U the set of all user defined attributes.

We say that a link of the WAUT is *explored* in a browsing session if its URI is one of the requests in the browsing session; otherwise, we say that the link is *unexplored*. Similarly, we say that a form is explored if its action a appears in one of the requests $a?d$ in the browsing session; otherwise we say the form is unexplored. Two pages $<u_i, c_i, I_i, L_i, V_i>$ and $<u_j, c_j, I_j, L_j, V_j>$ have a *repeated* (common) link if $L_i \cap L_j \neq \emptyset$; similarly, a repeated form exists, if $I_i \cap I_j \neq \emptyset$.

4.2.2 Converting a Browsing Session into an Automaton

In this section, we provide a high-level description of the algorithm that converts *RRS* into an automaton, called a *session automaton*. Each transition of the session automaton represents a link or a submitted form, and each state represents all pages (states) that have the same valuations of attributes, and the same sets of links and form actions. The automaton is built as follows.

Procedure 1. Given a browsing session $RRS = <u_0, c_0, I_0, L_0, V_0> l_1 <u_1, c_1, I_1, L_1, V_1> \ldots l_n <u_n, c_n, I_n, L_n, V_n>$, where n is the total number of observed requests, the corresponding automaton $A_{RRS} = < S \cup \{trap\}, s_0, \Sigma, T>$, is built as follows:

1. The tuple $<u_0, c_0, I_0, L_0, V_0>$ is mapped into a designated state called *inactive*, denoted s_0, where u_0 and c_0 are null, and I_0, L_0, and V_0 are empty sets.

2. The set of states S is defined as follows. For all $i > 0$, a tuple $<u_i, c_i, I_i, L_i, V_i>$ corresponds to a state of the session automaton. Tuples $<u_i, c_i, I_i, L_i, V_i>$ and $<u_j, c_j, I_j, L_j, V_j>$, where $j > i$, are mapped into the same state, if $c_i = c_j$, $I_i = I_j$, $L_i = L_j$, and $V_i = V_j$.
3. The alphabet (i.e., set of events) of the automaton is defined by the union of the sets Γ, Δ, *Req*. $\Gamma = \bigcup_{i=1}^{n} L_i$ is the set of all the URIs associated with links in the observed responses, $\Delta \subseteq \bigcup_{i=1}^{n} I_i$ is the set of all actions that correspond to the unexplored forms in the observed responses, *Req* is the set of all the observed requests. Thus, $\Gamma \cup \Delta \cup Req$ is the alphabet of the automaton, denoted Σ.
4. Each triple $(<u_i, c_i, I_i, L_i, V_i>, l_{i+1}, <u_{i+1}, c_{i+1}, I_{i+1}, L_{i+1}, V_{i+1}>)$ defines a transition (s_i, l_{i+1}, s_{i+1}), where s_i, s_{i+1} correspond to the response pages $<u_i, c_i, I_i, L_i, V_i>$, $<u_{i+1}, c_{i+1}, I_{i+1}, L_{i+1}, V_{i+1}>$ respectively, and $l_{i+1} \in L_i$ if l_{i+1} is a request corresponding to a clicked or implicit link, or if l_{i+1} is of the form $a_{i+1}?d_{i+1}$, then $a_{i+1} \in I_i$; and $l_{i+1} = u_{i+1}$, if $c_{i+1} \neq 3xx$; (otherwise, $l_{i+1} \neq u_{i+1}$);
5. Each request corresponding to an explored repeated form or link defines a transition from the state, where it occurs, to the state that corresponds to the response of the submitted filled form or clicked link.
6. Each event in Σ corresponding to an unexplored link $l \in L_i$ or unexplored form $a \in I_i$ defines a transition from the state representing the page $<u_i, c_i, I_i, L_i, V_i>$ to a designated state, called a *trap* state that represents the unexplored part of the WAUT and whose attributes are not available. Let T denote the set of thus defined transitions in steps 4, 5, and 6.

The obtained automaton models a complete behavior of a WAUT or its fragment, depending on the size of the session. In case of a static strongly-connected WA (where each page is reachable from every other page), the automaton that models all its behavior could be built from an exhaustive browsing session obtained by exploring each link and filling in every possible way and submitting each form on every page of the application, which is, as

mentioned before, often unfeasible. Therefore, using Procedure 1, we are able to infer from a single RRS (trace) an automata model which includes several possible behaviors (traces).

The following example is a fragment of a browsing session representing five web pages:

```
GET http://www.crim.ca HTTP/1.0
Host: www.crim.ca
Accept: application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint, image/gif, image/x-xbitmap,
image/jpeg, image/pjpeg, */*
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 4.0)
Accept-Language: en-us
----------END OF HTTP REQUEST----------
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 18316
Server: Apache/1.3.9 (Unix) mod_perl/1.21 mod_ssl/2.4.9 OpenSSL/0.9.4
Date: Wed, 10 Apr 2003 19:40:02 GMT
<HTML>
<HEAD>
<LINK rel="stylesheet" href="/styles.css">
<TITLE> CRIM</TITLE></HEAD> ...
...<a href="/rd/"> recherche-d&eacute;veloppement </a>
...
</HTML>
----------END OF HTTP RESPONSE----------
```

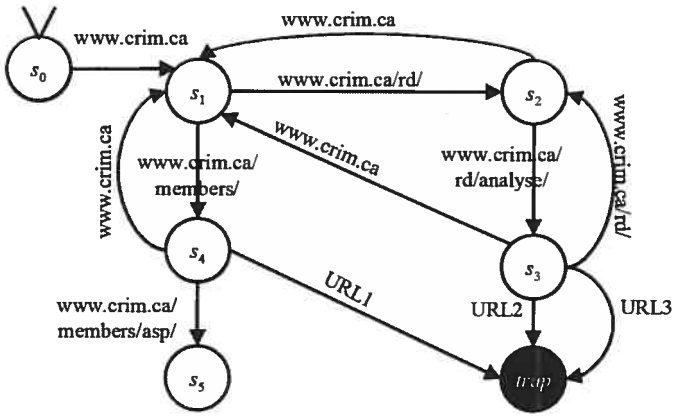


Figure 6. Example of a Session Automaton

Figure 6 shows the automaton that represents the browsing session, where state s_5 is a deadlock state representing an error page whose status code is 404. Transitions from s_2 to

s_1, and s_3 to s_2, represent deduced links which were not clicked by the user. URL1, URL2, and URL3 (named as such for simplicity) represent unexplored links that label transitions to the trap state.

4.3 Verification of Web Applications Using Session Automaton

In this section, we discuss properties of a WAUT which could be verified by analyzing the session automaton, and prove that an inferred session automaton preserves these properties. First, we present a few definitions.

We assume a given WAUT, where each page is a tuple that represents all the content of the page, i.e., a page is of the form $<u, c, I, L, V>$ where u, c, I and L are the same as described in Section 3.2 and V is the complete set of all possible attributes that define the content of an HTML page. Let $\mathsf{P} = \bigcup_{i=1}^{n} < u_{i_{WA}}, c_{i_{WA}}, I_{i_{WA}}, L_{i_{WA}}, V_{i_{WA}} >$ be the set of all the reachable pages of the WAUT, n is the total number of pages, and $\mathsf{V} = \bigcup_{i=1}^{n} V_{i_{WA}}$ be the set of all attribute valuations that identify the contents of the pages in P. Note that $\mathsf{V} \supseteq \mathsf{U}$, i.e., the set of attributes of the WAUT includes the set of user defined attributes in the browsing session. Moreover, we assume that the set of attributes in each page of the browsing session is $V_{i_{RBS}} = V_{i_{WA}} \cap \mathsf{U}$, $i = 1, \ldots, n$. This means that during monitoring, some attributes of web pages could be lost or disregarded (abstracted); however, if an attribute is abstracted from one observed page, it is consistently abstracted from all the others. Without loss of generality, we assume the following about the WAUT. First, all the pages of the WA and consequently browsing session are static pages (do not involve scripts or forms), and the WAUT itself is a static one (that is the set of HTML pages, which represent the WAUT, does not change during observation and verification). Moreover, every unique URI corresponds to a unique page; in other words, there exists a one-to-one correspondence between links and pages. Second, the WAUT starts from an initial page $<u_0, c_0, I_0, L_0, V_0>$ (other than the home page), where u_0, c_0, I_0, L_0, and V_0 are undefined. Third, the status code

$c = 200$ in each page, meaning that all the pages always render successfully, and the URI of the response coincide with the URI of request. Then, a WA automaton model is defined as follows.

Definition 1. *Given a web application with the set of all the reachable pages* $\mathsf{P} = \bigcup_{i=1}^{n} < u_{i_{WA}}, c_{i_{WA}}, I_{i_{WA}}, L_{i_{WA}}, V_{i_{WA}} >$ *, the (finite) web application automaton, denoted* A_{WA}*, is a tuple* $< S, s_0, \Sigma, T >$*, where*

1. *S is the set of all the pages* P*;*
2. $s_0 \in S$ *is the initial page* $<u_0, c_0, I_0, L_0, V_0>$*;*
3. $\Sigma = \bigcup_{i=1}^{n} L_i$, $n = |S|$. *It is the set of all the links in all the pages* P*; and*
4. $T \subseteq S \times \Sigma \times S$ *is a set of triples* $(<u_i, c_i, I_i, L_i, V_i>, l_{i+1}, <u_{i+1}, c_{i+1}, I_{i+1}, L_{i+1}, V_{i+1}>)$*, such that* $l_{i+1} = u_{i+1}, l_{i+1} \in L_i$.

The states of the session automaton do not necessarily represent all the pages of a given WAUT, while the state attributes of the session automaton are user defined and $\mathsf{U} \subseteq \mathsf{V}$. From Definition 1 the following proposition follows.

Proposition 1. *Given* $A_{WA} = < S, s_0, \Sigma, T >$*, the web application automaton, for every session automaton* $A_{RRS} = < S' \cup \{trap\}, s_0', \Sigma', T' >$*, the following holds*

1. $|S'| \leq |S|$,
2. $\Sigma' \subseteq \Sigma$*, and*
3. *For every transition* $(s_i', l_{i+1}', s_{i+1}') \in T'$, $s_i', s_{i+1}' \in S'$, $1 \leq i \leq |S'|$*, there exists* $(s_i, l_{i+1}, s_{i+1}) \in T$*, such that* $l_{i+1}' = l_{i+1}$, $c_i' = c_i$, $I_i' = I_i$, $L_i' = L_i$, $V_i' \subseteq V_i$*, and* $c_{i+1}' = c_{i+1}$, $I_{i+1}' = I_{i+1}$, $L_{i+1}' = L_{i+1}$, $V_{i+1}' \subseteq V_{i+1}$.

We call a *trace* of A_{WA} in state s the sequence of actions $l_1 l_2 \ldots l_k \in \Sigma^*$ such that there exist states $s_1, s_2, \ldots, s_k$, $s_1 = s$, such that $(s_i, l_i, s_{i+1}) \in T$ for all $i = 1, \ldots, k\text{-}1$. Let *traces*($s$) denote the set of all the traces of A_{WA} in a state s. We denote *traces*(A_{WA}) = *traces*(s_0). We Similarly define *traces*(A_{RRS}).

To demonstrate the correctness of our approach, we prove the following theorem (see Annex A) that states that given the web application automaton and the session automaton generated using Procedure 1 from a given browsing session of this application, each trace of the session automaton is a trace of the web application automaton.

Theorem 1. *Given* $A_{WA} = < S, s_0, \Sigma, T >$ *the web application automaton of a WA, for every session automaton of WA* $A_{RRS} = < S' \cup \{trap\}, s_0', \Sigma', T' >$ *it holds that*

$$traces(A_{WA}) \supseteq traces(A_{RRS}).$$

Proof. Let $l_1 l_2 \ldots \in traces(A_{RRS})$ be a trace of a session automaton A_{RRS}. By the definition of the trace, there exist states s_0', s_1', …, where for all $i \geq 0$, $(s_i', l_{i+1}, s_{i+1}') \in T'$. According to the definition of a browsing session, for each $s_i' = <u_i, c_i, I_i, L_i, V_i>$, such that $i > 0$, $l_{i+1} \in L_i$ (though $l_1 \notin L_0$). From Proposition 1, it follows that $l_{i+1} \in \Sigma$, for all $i \geq 0$, and there exist s_i, $s_{i+1} \in S$, such that $l_{i+1} \in L_i$ of s_i. By Definition 1, l_{i+1} is the URI of s_{i+1} in A_{WA}. Therefore, for each l_{i+1}, $i \geq 0$, there exists a triple $(s_i, l_{i+1}, s_{i+1}) \in T$. By Proposition 1, we conclude that $l_1 l_2 \ldots \in traces(A_{WA})$. QED

Thus, we have shown that a session automaton models a fragment of the behavior of the WAUT. In the rest of the section, we determine properties of WAs (or their violation) that are preserved by a session automaton. Since properties are usually defined using a KS model, we define a KS of WA and session automata. The mapping is more or less direct, preserving states, transitions, and attributes (in the form of atomic propositions). The only exception is *trap* state, whose attributes are undefined. There are two options to deal with this state, either to replace it with a "chaos" set of states with all possible attribute valuations that produce any possible paths, or just remove it. The first option would result in a conservative abstraction, so CEGAR (counterexample guided abstraction refinement) methods [51] become applicable. However, this would complicate the model. Thus, here we opt for removing the *trap* state. Now the model, obtained from a web session, could be considered as a restriction of the WAUT.

As described in Section 3.2, safety properties [48] designate an important category of specifications of a given system. They are often described as properties of the system, which have finite counterexamples (for any incorrect system) or, more informally, properties requiring that "*nothing bad happens*" [54]. Invariants constitute a subclass of safety properties, which require, for instance, that the value of a variable remains positive.

Negation of safety properties are either simple reachability properties, which express reachability of some state(s), such as "user could reach a page with e-shop return policy", or conditional reachability properties, such as the "user can always reach home page even without using site map". The reachability properties are not universal since they may require more than just one path to make a counterexample. In other words, reachability properties cannot be formulated in LTL since LTL checks the property validity on all the paths of the system, which is not the case of CTL where properties can be checked existentially, and not universally only.

The web session automaton, derived from observations, could be seen as a kind of abstraction of WA automaton by restriction. Abstraction by restriction [8] is known to preserve reachability properties, in the sense that whenever reachability property holds for the restricted system, it also holds for the original. Thus, if verification of web session automaton for a reachability property succeeds (or verification of the corresponding safety property fails), this reachability property is also proven for the original WAUT. If verification of reachability property fails, more observations are required to validate the WAUT. Here we do not discuss which exactly observations are needed, which is out of scope of this thesis. For safety properties, property preservation is "inverse": whenever a safety property is violated in the session KS (defined below) the safety property is disproved for the WAUT as well. Also, each infinite counterexample of the session KS is also a counterexample of the original one (similar to Property 5.1 in [44]).

While the above facts are intuitive, we prefer to prove them formally.

Definition 2. *Let* $M = (S, T, S_0, P, \mathcal{L})$ *and* $M' = (S', T', S_0', P', \mathcal{L}')$ *be two KSs. We say that* M' *is* an abstraction *of* M*, iff*

1. $P' \subseteq P$, *and*
2. *for every path* $\pi' = \langle s_0', s_1', \ldots \rangle$ *of* M', *there exists a path* $\pi = \langle s_0, s_1, \ldots \rangle$ *of* M, *such that for each* $i \geq 0$, $\mathcal{L}'(s_i') = \mathcal{L}(s_i) \cap P'$.

This abstraction is close to the well-known notion of "universal abstraction" for KSs or "under-approximation" [15].

Definition 3. *Let* $M = (S, T, S_0, P, \mathcal{L})$ *and* $M' = (S', T', S_0', P', \mathcal{L}')$ *be two KSs. We say that* M' *is* a reduction *of* M, *denoted by* $M' \preccurlyeq M$, *iff*

1. $S' \subseteq S$,
2. $P' = P$, and
3. for every path $\pi' = \langle s_0', s_1', \ldots \rangle$ of M', there exists a path $\pi = \langle s_0, s_1, \ldots \rangle$ of M such that π' is a prefix of π.

This definition states that M' is a reduction of M, if and only if every execution of M' is a prefix of an execution of M. It is similar to the "abstraction by restriction" defined in [8], where some behaviors are removed from the original model.

From Definition 2 and Definition 3, the following definition is derived.

Definition 4. *Let* M *and* M' *be two KSs. We say that* M' *is a* reduced abstraction *of* M, *iff there exists a KS* M'', *such that* M'' *is an abstraction of* M *and* $M' \preccurlyeq M''$.

A KS can directly be derived from a web application automaton, we call it a *web application structure* and define it as follows.

Definition 5. *Let* $A_{WA} = <S, s_0, \Sigma, T>$ *be a given web application automaton and* V *be the set of all attribute valuations present in* A_{WA} , *the web application structure* $M_{WA} = (S_{WA}, T_{WA}, S_{0_{WA}}, P_{WA}, \mathcal{L}_{WA})$ *is a KS, where* $S_{WA} = S$, $S_{0_{WA}} = \{s_0\}$, $T_{WA} \subseteq S_{WA} \times S_{WA}$, *such that for every transition* $(s_i, l_{i+1}, s_{i+1}) \in T$, $(s_i, s_{i+1}) \in T_{WA}$, $P_{WA} = \mathsf{V} \cup \Sigma$, *and* $\mathcal{L}_{WA} : S_{WA} \rightarrow 2^{P_{WA}}$ *labels the states with their corresponding attribute valuations.*

Note that each transition label in the web application automaton is a state label of the destination state in the corresponding web application structure.

Similarly, we derive the *browsing session structure* M_{RRS} of the automaton A_{RRS} of a given browsing session of WA. Note that the set of states of M_{RRS} does not include the *trap* state, since the properties are specified over the set of states that only represent browsed pages.

Definition 6. *Let* $A_{RRS} = \langle S \cup \{trap\}, s_0, \Sigma, T \rangle$ *be a given browsing session automaton and* U *the set of all attribute valuations present in all the states of* A_{RRS}*, the browsing session structure* $M_{RRS} = (S_{RRS}, T_{RRS}, S_{0_{RRS}}, P_{RRS}, \mathcal{L}_{RRS})$ *is a KS, where* $S_{RRS} = S$, $S_{0_{RRS}} = \{s_0\}$, $T_{RRS} \subseteq S_{RRS} \times S_{RRS}$ *is such that for every transition* $(s_i, l_{i+1}, s_{i+1}) \in T$, $(s_i, s_{i+1}) \in T_{RRS}$, $P_{RRS} = U \cup \Sigma$, *and* $\mathcal{L}_{RRS} : S_{RRS} \rightarrow 2^{P_{RRS}}$ *labels the states with their corresponding attribute valuations.*

From the properties of the automata models of a given WAUT and a corresponding browsing session defined in Proposition 1 and Theorem 1, the following proposition follows.

Proposition 2. *Given an application structure* $M_{WA} = (S_{WA}, T_{WA}, S_{0_{WA}}, P_{WA}, \mathcal{L}_{WA})$*, for every browsing session structure of WA* $M_{RRS} = (S_{RRS}, T_{RRS}, S_{0_{RRS}}, P_{RRS}, \mathcal{L}_{RRS})$*,* M_{RRS} *is a reduced abstraction of* M_{WA}.

Proof. Let A_{WA} and A_{RRS} be the web application automaton and a session automaton of WA from which M_{WA} and M_{RRS} are derived according to Definition 5 and Definition 6. From Definition 4, let a KS $M_{int} = (S_{int}, T_{int}, S_{0_{int}}, P_{int}, \mathcal{L}_{int})$ be an abstraction of M, such that $P_{int} = P_{RRS}$. Based on Definition 5, there exists an automaton A_{int} that corresponds to M_{int}, such the set of attributes is restricted to U instead of V. In other words, A_{int} preserves the behavior of A_{WA}, meaning that $traces(A_{int}) = traces(A_{WA})$, but restricts the state attributes. Therefore, Proposition 1 applies to A_{RRS} and A_{int} with the exception that state attributes of A_{RRS} are equal to state attributes of A_{int}. Since $traces(A_{int}) = traces(A_{WA})$, then from Theorem 1, $traces(A_{int}) \supseteq traces(A_{RRS})$. Since *trap* state is not included in M_{RRS}, only the traces that do not lead to *trap* are considered. From Proposition 1 and Theorem 1, it follows that for every

trace $l_1 l_2 \ldots \in traces(A_{RRS})$, there exists a state sequence $\langle s_0, s_1, \ldots \rangle$ in M_{RRS}, which is a prefix of a state sequence $\langle s_0', s_1', \ldots \rangle$ in M_{int}. So, by Definition 3, $M_{RRS} \preccurlyeq M_{int}$. Therefore, M_{RRS} is a reduced abstraction of M_{WA}. QED

We claim two theorems stating that when a property is violated in the model of the browsing session of a given WAUT, it is violated in the model of the WAUT if the counterexample in the model of the browsing session is an infinite execution. If the counterexample is a finite execution in the model of the browsing session, the claim holds only for safety properties as discussed earlier in Section 4.3. For this reason we first define a safety property. Here we give a topological definition of a safety similar to the one in [8].

Definition 7. *A safety property is any property* φ *such that whenever* φ *holds on a KS, it also holds on all the reductions of this KS.*

The following proposition follows from the definition of abstraction, and path semantics of LTL.

Proposition 3. *Let* $M = (S, T, S_0, P, \mathcal{L})$ *and* $M' = (S', T', S_0', P', \mathcal{L}')$ *be two KSs, such that* M' *is an abstraction of* M. *Given an LTL formula* φ *defined on atomic propositions* P' *of* M', *if* $M' \nvDash \varphi$, *then* $M \nvDash \varphi$.

Here we claim a theorem which states that when a property violation in the model of the browsing session of a given WAUT, is also a violation in the model of the WAUT if the counterexample in the model of the browsing session is an infinite execution.

Theorem 2. *Given a web application structure* $M_{WA} = (S_{WA}, T_{WA}, S_{0_{WA}}, P_{WA}, \mathcal{L}_{WA})$, *for every browsing session structure of WA* $M_{RRS} = (S_{RRS}, T_{RRS}, S_{0_{RRS}}, P_{RRS}, \mathcal{L}_{RRS})$, *if there is a property* φ, *such that* $M_{RRS} \nvDash \varphi$ *and a path in* M_{RRS} *that violates* φ *is infinite, then* $M_{WA} \nvDash \varphi$.

Proof. From Proposition 2, there exists a KS $M_{int} = (S_{int}, T_{int}, S_{0_{int}}, P_{int}, \mathcal{L}_{int})$, such that $M_{RRS} \preccurlyeq M_{int}$ and M_{int} is an abstraction of M_{WA}. From the definition of $\preccurlyeq$ (Definition 3), every path

of M_{RRS} is a prefix of a path of M_{int}. A path in M_{RRS} that violates φ is infinite, hence, there exists an infinite path π' in M_{int} that also violates φ. Thus, $M_{int} \not\models \varphi$. On the other hand, M_{int} is an abstraction of M_{WA}. From the definition of abstraction (Definition 2), for every path $\langle s_0', s_1', \ldots \rangle$ of M_{int}, there exists a path $\langle s_0, s_1, \ldots \rangle$ of M_{WA}, such that $\mathcal{L}_{int}(s_i') = \mathcal{L}_{WA}(s_i) \cap P_{int}$, $i \geq 0$. By Proposition 3, $M_{WA} \not\models \varphi$. QED

Now we claim a theorem similar to Theorem 2, but which concerns preservation of violation of safety properties.

Theorem 3. *Given a web application structure* $M_{WA} = (S_{WA}, T_{WA}, S0_{WA}, P_{WA}, \mathcal{L}_{WA})$, *for every browsing session structure of WA* $M_{RRS} = (S_{RRS}, T_{RRS}, S0_{RRS}, P_{RRS}, \mathcal{L}_{RRS})$, *if there is a property* φ *such that* $M_{RRS} \not\models \varphi$, *a path in* M_{RRS} *that violates* φ *is finite, and* φ *is a safety property, then* $M_{WA} \not\models \varphi$.

Proof. From Proposition 2, there exists a KS $M_{int} = (S_{int}, T_{int}, S0_{int}, P_{int}, \mathcal{L}_{int})$, such that $M_{RRS} \preccurlyeq M_{int}$ and M_{int} is an abstraction of M_{WA}. By the definition of a safety property (Definition 7), $M_{int} \not\models \varphi$. On the other hand, M_{int} is an abstraction of M_{WA}. Therefore, from Proposition 3, $M_{WA} \not\models \varphi$. QED

Theorems 2 and 3 demonstrate the consistency of our proposed approach. In conclusion, we claim that violations of safety properties are preserved, as well as those properties (safety or liveness) whose counterexamples are infinite executions. In the next section, we present a methodology to model WAs with multiple frames and windows using communicating automata.

4.4 Modeling Web Applications with Frames and Multiple Windows

In the previous section, we presented an automata model for single window WAs. However, WAs often use frames and multiple windows. These options allow rendering several pages at the same time, thus introducing concurrency in the behaviors of such WAs. Therefore, a single automaton is cumbersome to adequately model a concurrent behavior of WAs with several frames/windows. In this section, we further elaborate our approach using communicating automata to model such WAs, which we call *multi-display WA* for simplicity. Before we introduce our extended approach, we define elements of a browsing session of a multi-display WA.

4.4.1 Definitions

A response in a multi-display WA is defined as a tuple $<u, c, I, F, L, V>$, where u denotes the request that identifies the page; c represents the status code of the page; I is the set of URIs specified by the action attribute of each form in the page; L is the set of URIs associated with links, including the implicit links if any, in the page; and V is a vector $<v_1, \ldots, v_k>$, where v_i is the valuation of the page attribute i and k is the number of all the page attributes over which the atomic propositions are defined. I and L are extended to include the target for each action and link. Therefore, an element of L is a tuple $<l, t>$, where l is a URI associated with a link and t is the corresponding target; if no target is defined, then $t = \varepsilon$, which denotes the empty target. Similarly, an element of I becomes a tuple $<a, t>$, where a denotes a form action and t its corresponding target. F is a frame tree defined in the page and whose leaves are frames. A frame is a tuple of the form $<f, b>$, where f is the URI defined by the value of the *src* attribute of the HTML frame element and b is the frame name. We denote by *leaves*(F) a function that returns the set of leaf nodes (frames) of the tree F. In other words, *leaves*(F) denotes the URIs of the default pages loaded in the frames along with those frame names.

We define a browsing session of a multi-display WA as a sequence of requests (along with their corresponding targets) and responses. For simplicity, we keep using the term Request/Response Sequence (RRS) to represent a browsing session.

A RRS = $<u_0, c_0, I_0, F_0, L_0, V_0> <r_1, l_1, t_1> <u_1, c_1, I_1, F_1, L_1, V_1> \ldots <r_n, l_n, t_n> <u_n, c_n, I_n, F_n, L_n, V_n>$, where n is the total number of requests in the browsing session starting from $<r_1, l_1, t_1>$. $<r_i, l_i, t_i>$ represents a request, such that r_i is a string denoting the request header field, "referer", which is the URI of the page, where the request was triggered; and $<l_i, t_i>$ is defined as follows:

- if the request is for a filled form, then l_i is of the form $a_i?d_i$, where a_i forms with the target t_i a tuple $<a_i, t_i> \in I_j$ of the page $<u_j, c_j, I_j, F_j, L_j, V_j>$, where $u_j = r_i$,
- if the request is for a frame source page, then $<l_i, t_i> \in leaves(F_j)$ of the page $<u_j, c_j, I_j, F_j, L_j, V_j>$, where $u_j = r_i$,
- otherwise (if the request is for a link, clicked or implicit), then $<l_i, t_i> \in L_j$ of the page $<u_j, c_j, I_j, F_j, L_j, V_j>$, where $u_j = r_i$,

Notice that, similar to the case of a single window WA, $<u_0, c_0, I_0, F_0, L_0, V_0>$ corresponds to the initial default page displayed in the browser window, such that u_0, c_0 are null, and I_0, F_0, L_0, V_0 are empty sets; $<r_1, l_1, t_1>$ includes the URI l_1 of the starting page, and r_1 and t_1 are the empty string ε. In addition, $l_i = u_i$, if $c_i \neq 3xx$; otherwise, $l_i \neq u_i$ and $<u_i, c_i, I_i, F_i, L_i, V_i>$ immediately follows r_i in the RRS.

4.4.2 Basic Assumptions

Before we elaborate the model of a multi-display WA, we state basic assumptions about the observed browsing session of the WA. Such assumptions are essential due to inability to directly determine from a request the window/frame from which it was triggered. An observed request/response pair does not include the name of the window/frame targeted by the corresponding URI. To determine the window/frame, we track the "referer" header field in the request, which is the URI of the page, where the

request is triggered. Thus, the following assumptions must hold in the observed browsing session:

1. At each moment, different pages are displayed in frames/windows. If two pages have links to the same page, then only one request corresponding to one of the links is present in the session.
2. If a link is repeated in the same page with different targets and a request for that link is in the session, then the request corresponds to the first instance of that link appearing on the page.

These assumptions are not difficult to satisfy when the browsing session is created by the tester.

4.4.3 Communicating Automata Model of Multi-Display Web Applications

Here we describe how an observed browsing session can be modeled by a system of communicating automata. Given the browsing session, we first determine local browsing sessions that correspond to the behaviors of the entities in the browsed part of the WAUT, such as windows, frames, and framesets, each of which is modeled by an automaton. Then we explain how to convert the local browsing sessions into communicating automata and present the corresponding algorithm, which is an extension of Procedure 1 presented in Section 3.4.

Automata communicate synchronously by rendezvous, executing common (rendezvous) actions. Such communication is formalized by the parallel composition operator on automata. Formally, two communicating automata $A_1 = < S_1, s_{01}, \Sigma_1, T_1 >$ and $A_2 = < S_2, s_{02}, \Sigma_2, T_2 >$ are composed using the $\|$ operator. The resulting automaton, denoted $A_1 \| A_2$, is a tuple $< S, s_0, \Sigma, T >$, where $s_0 = (s_{01}, s_{02})$ and $s_0 \in S$; $\Sigma = \Sigma_1 \cup \Sigma_2$; and $S \subseteq S_1 \times S_2$ and T are the smallest sets obtained by applying the following rules:

- If $(s_1, e, s'_1) \in T_1$, $e \notin \Sigma_2$, and $(s_1, s_2) \in S$, then $(s'_1, s_2) \in S$, and $((s_1, s_2), e, (s'_1, s_2)) \in T$.
- If $(s_2, e, s'_2) \in T_2$, $e \notin \Sigma_1$, and $(s_1, s_2) \in S$, then $(s_1, s'_2) \in S$, and $((s_1, s_2), e, (s_1, s'_2)) \in T$.

- If $(s_1, e, s'_1) \in T_1$, $(s_2, e, s'_2) \in T_2$, and $(s_1, s_2) \in S$, then $(s'_1, s'_2) \in S$, and $((s_1, s_2), e, (s'_1, s'_2)) \in T$.

The composition is associative and can be applied to finitely many automata.

4.4.3.1 Local Browsing Sessions

A browsing session represents the behavior of communicating entities, namely, browser's main and independent windows, and frames, denoted $o_1, o_2, \ldots, o_k$, where o_1 corresponds to the browser's main window and k is the number of communicating entities. The entities corresponding to independent windows are determined by analyzing the targets present in the requests; if the target in a request is not an existing frame name, it corresponds to an independent window; for each request, whose target is "_blank", a new entity is defined corresponding to a new unnamed independent window. The entities that correspond to frames are determined by the frame names indicated in the frame trees of the response pages; where each frame entity is uniquely identified by $<f, b>$ and the URI u of the frameset document, where the corresponding frame tree is defined. The number of communicating entities k is then defined as follows. Given a browsing session, $<u_0, c_0, I_0, F_0, L_0, V_0> <r_1, l_1, t_1> <u_1, c_1, I_1, F_1, L_1, V_1> \ldots <r_n, l_n, t_n> <u_n, c_n, I_n, F_n, L_n, V_n>$, let $\{t_1, \ldots, t_q\}$, such that $q \leq n$, be the set of all the distinct targets observed in the requests, including window names, frame names, and predefined targets ("_parent", "_top", "_self", "_blank"). Let $\{b_1', \ldots, b_p'\}$ be the set of all the unique frame names defined in all the responses, such that b_i' represent a frame identified with $<f_i, b_i>$ and the URI u of the corresponding frameset document. Then, $k = 1 + |\{t_1, \ldots, t_q\} \cup \{b_1', \ldots, b_p'\} - \{t_i \mid t_i = \text{"\_top"} \text{ or } t_i = \text{"\_parent"} \text{ or } t_i = \text{"\_self"} \text{ or } t_i = \text{"\_blank"} \text{ or } t_i = \varepsilon\}| + |\{<r_j, l_j, t_j> \mid t_i = \text{"\_blank"}\}|$. We further analyze the hierarchical relationship among the different entities of the application. We consider each window entity as a *window tree*, whose root node represents the window itself. The first frame tree occurring in (frameset document loaded into) the window is appended to the root of the window tree. If a request's target is a frame name, such that the response is another frameset document (having a frame tree), in the window tree, the response's frame tree is appended to the node of the targeted frame. Similarly, if the target

is a frame name or the window itself, any subsequent children are removed from the node of the targeted entity and replaced by the response's frame tree, if any.

The local browsing sessions (RRS_1, ..., RRS_k) corresponding to the observed behavior of k entities of the WA are determined as follows. A request/response pair $<r_j, l_j, t_j> <u_j, c_j, I_j, F_j, L_j, V_j>$ belongs to a RRS_i, if the target t_j refers to the entity o_i. Also, the *RRS* of each frame that could be a child of o_i, contains the same request $<r_j, l_j, t_j>$, whose response is the inactive page. At the same time, the *RRS* of the (targeting) entity, from which $<r_j, l_j, t_j>$ is triggered, must contain $<r_j, l_j, t_j>$ itself with its response being the page, where the request is initiated. This is explained by the fact that the targeting entity does not change its displayed page. However, if the target t_j is "_parent", "_top", or a parent entity name, then the response in the *RRS* of the targeting entity is the inactive page. Similarly, the *RRS* of each frame that is a child of the targeted entity contains the same request $<r_j, l_j, t_j>$ whose response is the inactive page. This means that those frames are deactivated and erased from the window. If the target attribute is absent or "_self" then $<r_j, l_j, t_j> <u_j, c_j, I_j, F_j, L_j, V_j>$ belongs to a RRS_i, provided that the request is triggered from the last page displayed in the corresponding entity o_i. Following is a high-level description of steps determining the local sessions.

Procedure 2. Sessions RRS_i, $i = 1, \ldots, k$, are formed using the following procedure:

1. $RRS_1 := <u_0, c_0, I_0, F_0, L_0, V_0>$ corresponds to the inactive page of the *RRS* of the main window similar to the inactive page defined in Section 4.2.1. For $i > 1$, $RRS_i := <u_\Theta, c_\Theta, I_\Theta, F_\Theta, L_\Theta, V_\Theta>$, is defined similarly to $<u_0, c_0, I_0, F_0, L_0, V_0>$, which corresponds to the inactive page, where the local session starts.
2. The first request response pair $<r_1, l_1, t_1> <u_1, c_1, I_1, F_1, L_1, V_1>$ is appended to the session of the browser's main window, i.e., $RRS_1 := RRS_1 <r_1, l_1, t_1> <u_1, c_1, I_1, F_1, L_1, V_1>$.
3. For each request/response pair $<r_j, l_j, t_j> <u_j, c_j, I_j, F_j, L_j, V_j>, j > 1$,
   3.1. if the target t_j refers to entity o_i, $<r_j, l_j, t_j> <u_j, c_j, I_j, F_j, L_j, V_j>$ is appended to RRS_i, i.e., $RRS_i := RRS_i <r_j, l_j, t_j> <u_j, c_j, I_j, F_j, L_j, V_j>$. At the same time, $<r_j, l_j, t_j> <u_\Theta,$

c_Θ, I_Θ, F_Θ, L_Θ, V_Θ> is appended to the sessions of all the frames and framesets (if any) that are children of o_i.

3.2. If the "referer" r_j is equal to the URI of the last response in RRS_i then

3.2.1. If the target t_j corresponds to a parent entity, the response corresponding to <r_j, l_j, t_j> in RRS_i is the inactive page <u_Θ, c_Θ, I_Θ, F_Θ, L_Θ, V_Θ>. Thus, RRS_i := RRS_i <r_j, l_j, t_j> <u_Θ, c_Θ, I_Θ, F_Θ, L_Θ, V_Θ>. At the same time, <r_j, l_j, t_j> <u_Θ, c_Θ, I_Θ, F_Θ, L_Θ, V_Θ> is also appended to the sessions of all the frames that are children of the targeted parent; otherwise,

3.2.2. the response to <r_j, l_j, t_j> is a tuple <u, c, I, F, L, V>, such that $r_j = u$. Thus, RRS_i := RRS_i <r_j, l_j, t_j> <u, c, I, F, L, V>.

3.3. If the target t_j = "_self" or t_j = ε and r_j is the URI of the last page displayed in RRS_i, then RRS_i := RRS_i <r_j, l_j, t_j> <u_j, c_j, I_j, F_j, L_j, V_j>.

The following is a fragment of a browsing session. It starts with a request followed by a response containing a frameset document that specifies the source of two frames named *toc* and *main*. Then two request/response pairs are observed for the source pages of the two frames *toc* and *main*. The source page for *toc* contains several links and a form whose target attributes are the frame *main*. The source page for *main* contains links whose target attributes are "_top". The last request is for a link from the *main* source page and whose response page is displayed in the full window, thus deleting all the frames.

```
GET http://sec.eecs.berkeley.edu:80/body.htm HTTP/1.0
Referer: http://sec.eecs.berkeley.edu/

----------END OF HTTP REQUEST----------
HTTP/1.1 200 OK

<html>
<frameset cols="151,*" border=0 frameborder="no">
   <frame name="toc" src="toc.htm" frameborder=0 marginwidth=4>
   <frame name="main" src="main.htm" frameborder=0  marginwidth=4>
</frameset>
</html>
----------END OF HTTP RESPONSE----------

GET http://sec.eecs.berkeley.edu:80/toc.htm HTTP/1.0
Referer: http://sec.eecs.berkeley.edu/body.htm
Accept-Language: en-us

----------END OF HTTP REQUEST----------
HTTP/1.1 200 OK
```

```
<html>
<br><br><a href="people/main.htm" target="main">People</A>
   <br><a href="http://robotics.eecs.berkeley.edu/~sastry/darpa.sec/prop/prop225.html" target="main">Proposal</A>

<form method="post" action="/cgi-bin/htsearch" target="main">
<input type="text" size="12" name="words" value="">
<input type="submit" value="Search">
<input type=hidden name=config value=htdigsec>
<input type=hidden name=restrict value="">
<input type=hidden name=exclude value="">
</form>
...
</html>
----------END OF HTTP RESPONSE----------
GET http://sec.eecs.berkeley.edu:80/main.htm HTTP/1.0
Referer: http://sec.eecs.berkeley.edu/body.htm
----------END OF HTTP REQUEST----------
HTTP/1.1 200 OK

<html>
        <a href="http://www.eecs.berkeley.edu/~tah/" target="_top"> Shankar Sastry</a>
        <a href="http://www.eecs.berkeley.edu/~sastry/" target="_top"> Tom Henzinger</a>
        <a href="http://ptolemy.eecs.berkeley.edu/~eal/" target="_top"> Edward A. Lee</a>
...
</html>
----------END OF HTTP RESPONSE----------
GET http://www.eecs.berkeley.edu:80/~sastry/ HTTP/1.0
Referer: http://sec.eecs.berkeley.edu/main.htm

----------END OF HTTP REQUEST----------
HTTP/1.1 200 OK

<HTML>
----------END OF HTTP RESPONSE----------
```

The number of entities determined from the fragment of RRS is $|\{toc, main, \_top, \varepsilon\} \cup \{toc, main\} - \{\_top, \varepsilon\}| + |\{r_i \mid t_i = \_blank\}| + 1 = 3$. These entities correspond to the main browser window, and frames *toc* and *main*. By applying Procedure 2, we obtain the three local browsing sessions (RRS_1, RRS_2, RRS_3) illustrated in Figure 7, Figure 8, and Figure 9.

```
<<<<<<<<<<<<<<<INACTIVE >>>>>>>>>>>>>>>>>>>
-------------------------------------------------------------------
GET http://sec.eecs.berkeley.edu:80/body.htm HTTP/1.0
Referer: http://sec.eecs.berkeley.edu/
----------------------END OF HTTP REQUEST----------------
HTTP/1.1 200 OK

<html>
<frameset cols="151,*" border=0 frameborder="no">
   <frame name="toc" src="toc.htm" frameborder=0
marginwidth=4>
   <frame name="main" src="main.htm" frameborder=0
marginwidth=4>
</frameset>
</html>
----------------------END OF HTTP RESPONSE----------------
GET http://sec.eecs.berkeley.edu:80/toc.htm HTTP/1.0
Referer: http://sec.eecs.berkeley.edu/body.htm
Accept-Language: en-us
----------------------END OF HTTP REQUEST----------------
HTTP/1.1 200 OK

<html>
<frameset cols="151,*" border=0 frameborder="no">
   <frame name="toc" src="toc.htm" frameborder=0
marginwidth=4>
   <frame name="main" src="main.htm" frameborder=0
marginwidth=4>
</frameset>
</html>
----------------------END OF HTTP RESPONSE----------------
GET http://sec.eecs.berkeley.edu:80/main.htm HTTP/1.0
Referer: http://sec.eecs.berkeley.edu/body.htm
----------------------END OF HTTP REQUEST----------------
HTTP/1.1 200 OK

<html>
<frameset cols="151,*" border=0 frameborder="no">
   <frame name="toc" src="toc.htm" frameborder=0
marginwidth=4>
   <frame name="main" src="main.htm" frameborder=0
marginwidth=4>
</frameset>
</html>
----------------------END OF HTTP RESPONSE----------------
GET http://www.eecs.berkeley.edu:80/~sastry/ HTTP/1.0
Referer: http://sec.eecs.berkeley.edu/main.htm
----------------------END OF HTTP REQUEST----------------
HTTP/1.1 200 OK

<HTML> ...
----------------------END OF HTTP RESPONSE----------------
```

Figure 7. RRS_1 Corresponding to the Main Browser Window

```
<<<<<<<<<<<<<<<INACTIVE >>>>>>>>>>>>>>>>>>>
-------------------------------------------------------------------
GET http://sec.eecs.berkeley.edu:80/toc.htm HTTP/1.0
Referer: http://sec.eecs.berkeley.edu/body.htm
Accept-Language: en-us

----------------------END OF HTTP REQUEST----------------
HTTP/1.1 200 OK

<html>
...
   <br><br><a href="people/main.htm" target="main">People</A>
   <br><a href="http://robotics.eecs.berkeley.edu/~sastry/darpa.sec/prop/prop225.html"
target="main">Proposal</A>

<form method="post" action="/cgi-bin/htsearch" target="main">
<input type="text" size="12" name="words" value="">
<input type="submit" value="Search">
<input type=hidden name=config value=htdigsec>
<input type=hidden name=restrict value="">
<input type=hidden name=exclude value="">
</form>
...
</html>
----------------------END OF HTTP RESPONSE----------------
GET http://www.eecs.berkeley.edu:80/~sastry/ HTTP/1.0
Referer: http://sec.eecs.berkeley.edu/main.htm

----------------------END OF HTTP REQUEST----------------
<<<<<<<<<<<<<<<INACTIVE >>>>>>>>>>>>>>>>>>>
```

Figure 8. RRS_2 Corresponding to the Frame *toc*

```
<<<<<<<<<<<<<<<<INACTIVE >>>>>>>>>>>>>>>>>>>
-------------------------------------------------------------------
GET http://sec.eecs.berkeley.edu:80/main.htm HTTP/1.0
Referer: http://sec.eecs.berkeley.edu/body.htm

----------------------END OF HTTP REQUEST----------------------
HTTP/1.1 200 OK

<html>
<a href="http://www.eecs.berkeley.edu/~tah/" target="_top"> Tom Henzinger </a>
<a href="http://www.eecs.berkeley.edu/~sastry/" target="_top"> Shankar Sastry </a>
<a href="http://ptolemy.eecs.berkeley.edu/~eal/" target="_top"> Edward A. Lee</a>
...
</html>
----------------------END OF HTTP RESPONSE----------------------
GET http://www.eecs.berkeley.edu:80/~sastry/ HTTP/1.0
Referer: http://sec.eecs.berkeley.edu/main.htm

----------------------END OF HTTP REQUEST----------------------
<<<<<<<<<<<<<<<<INACTIVE >>>>>>>>>>>>>>>>>>>
```

Figure 9. RRS_3 Corresponding to the Frame *main*

4.4.3.2 Converting Local Browsing Sessions into Communicating Automata

To build an automata model of a browsing session of a multi-display WA, we convert each local browsing session $RRS_i = \langle u_{i\Theta}, c_{i\Theta}, I_{i\Theta}, F_{i\Theta}, L_{i\Theta}, V_{i\Theta}\rangle \langle r_{i1}, l_{i1}, t_{i1}\rangle \langle u_{i1}, c_{i1}, I_{i1}, F_{i1}, L_{i1}, V_{i1}\rangle \ldots \langle r_{im}, l_{im}, t_{im}\rangle \langle u_{im}, c_{im}, I_{im}, F_{im}, L_{im}, V_{im}\rangle$ into an automaton A_i, called the *local session automaton*, by extending Procedure 1 of Section 4.2.2.

The set of events Σ_i of the automaton A_i is defined by the union of the following four sets Γ_i , Δ_i, Req_i, and Φ_i. Similar to what is previously defined, $\Gamma_i = \{\langle l_i, t_i\rangle \mid \langle l_i, t_i\rangle \in L_{iw}, 1 \le w \le m\}$ is the set of all the URIs associated with links in the observed responses, $\Delta_i \subseteq \{\langle a_i, t_i\rangle \mid \langle a_i, t_i\rangle \in I_{iw}, 1 \le w \le m\}$ is the set of all form actions that correspond to the unexplored forms in the observed responses, and Req_i is the set of all the observed requests. $\Phi_i = \{\langle f_i, b_i\rangle \mid \langle f_i, b_i\rangle \in leaves(F_{iw}), 1 \le w \le m\}$ is the set of URIs corresponding to the source pages loaded in the frames. The following procedure extends Procedure 1.

Procedure 3. Given the entities o_i and their local browsing sessions RRS_i, $i = 1, \ldots, k$, each RRS_i is converted into a local session automaton A_i as follows.

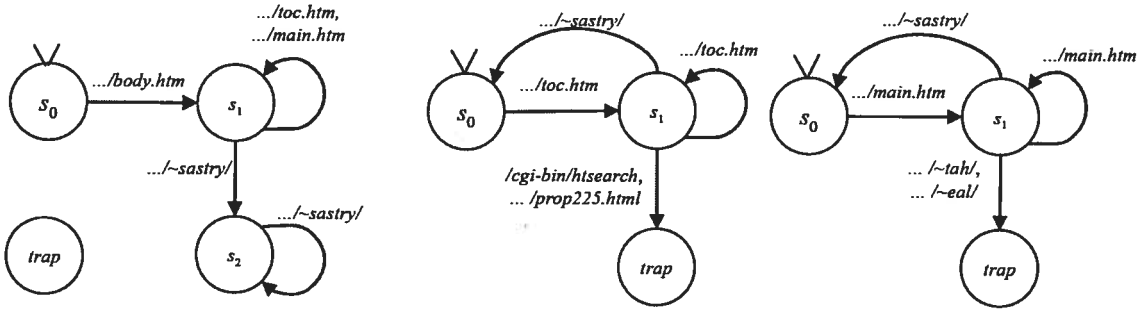
1. Procedure 1 is used to convert RRS_i into A_i.

2. The set of events Σ_i is extended to include the set Φ_i of URIs corresponding to the source pages loaded in the frames; thus, $\Sigma_i := \Sigma_i \cup \Phi_i$.
3. Each triple ($<u_{ij}, c_{ij}, I_{ij}, F_{ij}, L_{ij}, V_{ij}> <r_{ij}, l_{ij}, t_{ij}> <u_{i\Theta}, c_{i\Theta}, I_{i\Theta}, F_{i\Theta}, L_{i\Theta}, V_{i\Theta}>$) defines a transition ($s_{ij}$, $<r_{ij}, l_{ij}, t_{ij}>$, s_{0i}), where s_{ij}, s_{0i} correspond to the pages $<u_{ij}, c_{ij}, I_{ij}, F_{ij}, L_{ij}, V_{ij}>$, $<u_{i\Theta}, c_{i\Theta}, I_{i\Theta}, F_{i\Theta}, L_{i\Theta}, V_{i\Theta}>$, respectively;
4. Each triple ($<u_{ij}, c_{ij}, I_{ij}, F_{ij}, L_{ij}, V_{ij}> <r_{ij}, l_{ij}, t_{ij}> <u_{ij+1}, c_{ij+1}, I_{ij+1}, F_{ij+1}, L_{ij+1}, V_{ij+1}>$), such that $<u_{ij}, c_{ij}, I_{ij}, F_{ij}, L_{ij}, V_{ij}> = <u_{ij+1}, c_{ij+1}, I_{ij+1}, F_{ij+1}, L_{ij+1}, V_{ij+1}>$, defines a transition ($s_{ij}$, $<r_{ij}, l_{ij}, t_{ij}>$, s_{ij}), where s_{ij} corresponds to $<u_{ij}, c_{ij}, I_{ij}, F_{ij}, L_{ij}, V_{ij}>$;
5. Every event, not in Φ_i, corresponding to a request targeting o_i itself labels a transition from every state of the automaton to the state of the corresponding response page.
6. Each unexplored link $<r_{ij}, l_{ij}, t_{ij}>$ that targets an entity o_z, $1 \leq z \leq k$, and $z \neq i$, and whose corresponding pages exist in RRS_z, is represented as follows:
   6.1. The event corresponding to $<r_{ij}, l_{ij}, t_{ij}>$ labels a looping transition from the state s_{ij} to itself.
   6.2. The same event is added to the set of events of o_z, $\Sigma_z := \Sigma_z \cup \{<r_{ij}, l_{ij}, t_{ij}>\}$, and labels a transition from every state of the automaton to the state of the corresponding response page.

Steps 3 to 6 define the transitions labeled by the events shared by different automata. Step 3 defines transitions labeled by a request initiated by o_i or one of its siblings/children and whose target is a parent entity. Then, o_i is deactivated, and A_i is in the inactive state s_{i0}. Step 4 defines transitions labeled by a request initiated by o_i targeting another entity which is not a parent of o_i. In this case, o_i does not change its displayed page and A_i remains in the current state. Step 5 states that a shared event, which does not correspond to a frame source page URI, targeting o_i is not under the control of A_i and, thus, should label transitions from every state of A_i to the corresponding state. The reason behind the exception of frame source page URIs is that the requests corresponding to those URIs are triggered by the browser, and not the user, thus are executed only once. The last step of the procedure defines transitions that represent deduced links (with defined targets)

between different entities, which were not clicked by the user in the browsing session. For convenience, the state that corresponds to the source page of a given frame is called the first state. Note that in case of an ill-designed application or unreasonable user behavior, where multiple instances of a same window created using the predefined target "_blank", are all treated as a single entity, to avoid state explosion.

Let $A_1, \ldots, A_k$ be the automata that model k windows and frames. The composition automata is $A = A_1 \parallel \ldots \parallel A_k = \langle S \cup \{trap\}, s_0, \Sigma, T \rangle$, where $s_0 = (s_{01}, \ldots, s_{0k})$; the set of events Σ of A is the union of all Σ_i; the set of states S and the transition relation T of A are defined according to the semantics of the composition operator $\parallel$. The trap state of A is $trap = (trap_1, \ldots, trap_k)$. Fig. 2 illustrates an example of a fragment of a model for three entities, *browser window*, *Frame*1, and *Frame*2. These entities are modeled by three automata, A_1, A_2, and A_3, respectively, which communicate by executing common events.



(a) (b) (c)

Figure 10. (a) A_1 for the Browser Window, (b) A_2 for Frame *toc*, (c) A_3 for Frame *main*

(a) (b) (c)

Figure 10 illustrates the communicating automata A_1, A_2, and A_3 obtained by applying Procedure 3, representing the local sessions (RRS_1, RRS_2, RRS_3) of the previous section. Note that the transitions having the same source and destination state, and the transitions to each *trap* state are shown as a single edge with several labels separated by commas.

4.5 Summary

We presented in this chapter our formal approach to run-time verification of WAs. We elaborated algorithms that infer automata-based models from observed browsing sessions of WAs. In case of multi-display applications, we apply an algorithm that translates a single browsing session into local browsing sessions where each corresponds to a web entity (window or frame). The results discussed in this chapter are published in [41,42,39].

In the next chapter, we present an algorithm that extends the automata model presented in this chapter with variables and updates. Namely, we extend each automaton with a variable that tracks browser triggered events. This extension is necessary to distinguish between stable and transient global states. Distinction between stable and transient states is necessary in WAs that use frames, where the default pages of frames of a given web page are not loaded simultaneously. Therefore, some properties are relevant to all the global states (stable and transient), while others should be verified only on either stable or transient global states.

Chapter 5

Scopes of States in Web Modeling

In this chapter, we present an algorithm to extend the communicating automata model of WAs with variables and updates. This extension stems from the fact that our modeling approach takes into account possible behaviors of WAs with multiple frames. Specifically, our modeling approach represent the cases where frames could not be loaded completely, and thus the user is able to initiate an action in one of the frames while other frames' pages are not loaded yet. Therefore, the global state graph of such WAUT comprises what we call stable and transient states, where stable states represent the display of fully loaded frames' pages while transient states represent those displays where some of the frames' pages are not loaded yet.

5.1 Stable and Transient States in Web Modeling

In case of a WA with frames, a page is loaded in each frame. When all the pages in the frames (in the browser window) are completely loaded, the display of the WA becomes stable; otherwise, the display is unstable and is in a transient mode; accordingly global states of automata models are either transient or stable. The reason is that when, in a given window, a request for a page with frames is sent to the server, the response is a frameset document containing URIs of the frames' source pages. Then, the browser, without the intervention of the user, initiates the requests for the source pages of the frames. These requests cannot be initiated simultaneously, and their response pages are not loaded at the same time. At this point, a user action can interleave with these requests and responses. This includes, for example, clicking a link in one frame, while the source page in the second frame is not yet fully loaded. Such scenario is considered in our web analysis

framework [41,42], where the frameset document is a state (representing the transient state of the display) in the automaton of the window and the browser initiated requests are rendezvous events labeling looping transitions in the automata of the corresponding frames. The communicating automata model is used to represent WAs, defined by composition of the automata of the different entities. Then, user defined properties are verified over the whole global state space. However, there exist properties of WAs that should be verified by considering only a subset of the states. As an instance of interesting properties of WAs, consider the requirement that the number of certain objects should (not) exceed a certain limit, for example:

An important flashing advertisement should be present in every display shown to the user of a WA with (at least) two frames.

Assume that, initially, the advertisement appears in the first frame. Then, in response to a user action, the advertisement appears in the second frame. Though the initial and final displays of the WA respect the requirement (one instance of the advertisement is always shown to the user), this cannot be said about all the transient global states, where the display is not yet stable. Due to the concurrent behavior of frames or a slow server response the user will be shown a display without any instances of the advertisement, which violates the requirement. Such violation may jeopardize the business value of the application.

The following property, important for some WAs, constitutes another example:

The number of links shown to the user should always be greater than a predefined threshold.

The above properties have to be checked in each and every state of the model and do not require distinction between stable and transient global states.

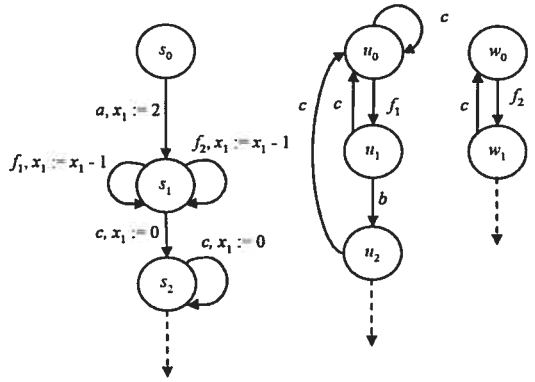


Figure 11. (a) A_1 for Browser Window, (b) A_2 for *Frame*1, (c) A_3 for *Frame*2

To define a way of distinguishing global states, we consider the example of a WA with two frames. In Figure 11, we show a fragment of the model with three entities, the *browser window*, *Frame*1, and *Frame*2. We invite the reader to consider only events labeling transitions in Figure 11; the variable updates on these transitions are later explained in Section 5.4. These entities are modeled by three automata, A_1, A_2, and A_3, respectively, which communicate by executing common events. Initially, the three automata are in their inactive states s_0, u_0, and w_0, respectively. The event a is a link, clicked by the user, which makes A_1 move to state s_1 that is the frameset document containing URIs of *Frame*1 and *Frame*2. The events f_1 and f_2 are from the browser window received by the two frames, respectively, and represent the browser triggered requests for frames source pages. A_2 and A_3 are then active, while A_1 remains in s_1. In *Frame*1, the user can click the link b so that A_1 moves to state u_2 by executing the transition labeled by action b. In *Frame*2, the user can click the link c whose target is "_top", such that the corresponding page is loaded in the full window, thus canceling the two frames. In this case, c is a multi-rendezvous of A_3, A_1, and A_2; as a result, A_2 and A_3 move to their inactive states u_0 and w_0, and A_1 moves to state s_2. Note that different possible behaviors of the WA that can occur in reality are all represented in the model. For example, if the server is slow, *Frame*2 can be activated before *Frame*1 and the user can click on c before the browser triggers the request for the source page of *Frame*1. Similarly, the link b can be clicked

before *Frame*2 becomes active. These possible behaviors can be seen in the composition of the automata shown in Figure 12.

In the example, the global states that correspond to the "completed" stabilized WA displays can be distinguished from the "uncompleted" transient ones, and eventually treated differently in web model checking. In the WA, once the request for the page with two frames is sent to the server, the display is considered unstable, until both frames are active or are canceled and replaced by another page in the browser window. Therefore, some of the stable global states of the model are (s_0,u_0,w_0), (s_1,u_1,w_1), (s_1,u_2,w_1), and (s_2,u_0,w_0), while the remaining states are transient (unstable). Note that the designated transient states (Figure 12) have f_1 and/or f_2, as actions labeling outgoing transitions. These actions correspond to the requests triggered by the browser. Thus, we conclude that any global state with at least one enabled browser triggered action is transient. However, the distinction between transient and stable global states is neglected by model checkers, as they usually do not distinguish types of actions enabled in the states. This hinders their applicability to WA. Therefore, in the next section, we introduce a refined model of WA that allows the identification of stable and transient global states for model checking.

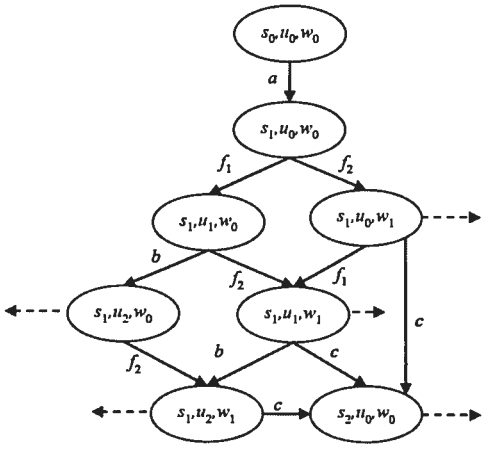


Figure 12. Composition of A_1, A_2, and A_3

5.2 Refining the Model of Web Applications

Given a system of communicating automata, we consider the characterization of state stability based on the type of enabled events. The global behavior of such systems is determined by composition of the component automata, such that if designated events are enabled at a given global state, the state is transient, otherwise the state is stable. It is assumed that in the global state graph, the designated events which determine stability of global states do not label cycles; in other words, we assume that the system does not stay in a transient mode forever. This implies that the execution of the designated events always leads to a stable global state. To facilitate different treatment of various types of states by model checkers, a suitable indication of state stability recognizable by a model checker is needed, namely, identifying stable/transient global states by adding variables to component automata. We, thus, extend our automata model with variables and operations on them, obtaining an Extended Automaton (EA) model, similar to, e.g., [69].

5.2.1 Rationale

In Chapter 4, we presented a procedure that converts local browsing sessions corresponding to entities (windows and frames) of a WAUT into a communicating automata model. Each local browsing session is converted into a local session automaton. In a WAUT, browser triggered requests (corresponding to frames source page URIs) are initiated by the browser not more than once each time a corresponding frameset document is loaded. On the other hand, in case of user triggered requests, the user can click on the same link several times and, thus, trigger the same request several times (potentially an unbounded number of times).

In the automata model of the WAUT, browser triggered requests are the designated events that determine stability of global states. These requests label looping transitions in local automata. Every event, corresponding to a request targeting an entity, labels a transition from every state of the automaton of the targeted entity to the state of the corresponding response page, except for the browser triggered events. The looping

transitions labeled by those events are rendezvous in the automata that model the targeted frames, such that each of the browser triggered events labels a single transition from the inactive state to the first state (source page). Therefore, each of these looping transitions is executed only once on each execution path. In order to distinguish stable from transient global states of the composition automaton, we extend each component automaton with a variable, we call it the *event tracking* variable, which tracks the number of browser triggered events enabled at each state and their execution. Consequently, a model checker can distinguish stable from transient global states of the composition automaton based on the current value of event tracking variables, which are in component automata.

The event tracking variable assigned to a session automaton is set as follows. Initially, the variable is zero. For each state of the automaton, the variable takes the value equal to the number of browser triggered events defined in that state, thus, if it is greater than zero the current global state with this component state is transient. When a transition labeled by a browser triggered event is executed, the variable is decremented by one, indicating that there are fewer such events enabled at the global state. When all transitions labeled by browser triggered events are executed, the event tracking variable is zero indicating that no browser triggered events of the corresponding automaton are enabled. If this is the case for all the component automata, the global state is then stable. For the states of the automaton whose enabled actions do not include browser triggered events, the event tracking variable is set to zero.

Note that transient global states exist only in models of WAs that have multiple frames. In the case of WAs that are single or multi window, and with no frames, the corresponding automata models do not have transient global states, and the event tracking variables are always zero, thus all the global states of those models are stable.

Next, we present an extended automata model and an algorithm that extends an automata model into an EA model with an auxiliary event tracking variable.

5.2.2 Preliminaries

An Extended Automaton (EA) is an automaton extended with a set of context variables V, operations, and predicates defined over the context variables. We denote D_V the set of context variable valuations v. A context variable valuation $v \in D_V$ is called a *context*.

Definition 8. *An Extended Automaton (EA) is a tuple* $Q = \langle S, s_0, \Sigma, V, v_0, T \rangle$, *where* S *is a finite set of states,* $s_0 \subseteq S$ *is the initial state,* Σ *is a finite set of events,* V *is a finite set of context variables,* v_0 *is the initial context, and* T *is a finite set of transitions, such that each transition* $t \in T$ *is a tuple* (s, P, a, up, s'), *where*

1. $s, s' \in S$, *are the initial and final states of the transition, respectively;*
2. $a \in \Sigma$ *is the event of the transition;*
3. P *and* up *are functions defined over context variables* V, *such that*
   - $P: D_V \rightarrow \{True, False\}$ *is the predicate (guard) of the transition;*
   - $up: D_V \rightarrow D_V$ *is the context update function of the transition.*

A *configuration* of Q is a tuple (s, v) of state s and context v. If the set of context variables is empty, then configuration and state are indistinguishable. An EA with finite domains of variables assumed in this thesis can be unfolded into an automaton, where configurations of the EA constitute the states of the automaton. A transition (s, P, a, up, s') is said to be *enabled*, if P = True. An EA is *deterministic* if any two transitions outgoing from the same state with the same event have disjoint predicates.

Given a state s of Q, we denote the set of events defined at s by $init(s) = \{a \in \Sigma \mid \exists\ s' \in S \text{ s.t. } (s, P, a, up, s') \in T\}$.

An EA can be unfolded into an automaton as follows:

Definition 9. *Let* $Q = \langle S, s_0, \Sigma, V, v_0, T \rangle$ *be an EA. The unfolding of* Q *is the automaton* $Q' = \langle S', \Sigma, T', s_0' \rangle$ *where* $S' \subseteq S \times D_V$ *is the set of states;* $s_0' \in S'$ *is the initial state,* $s_0' = (s_0,$

v_0); *the transition relation* $T' \subseteq S' \times \Sigma \times S'$ *is such that for every transition* $(s, P, a, up, s') \in T$, P = *True, if* $(s, v) \in S'$ *then* $((s, v), a, (s', v')) \in T'$, *where* $v' = up(v)$.

We define the composition of Communicating EA (CEA) by applying the parallel composition || on their unfoldings. We assume that the sets of states as well as sets of variables of each composing EA are disjoint.

Definition 10. *Let* $Q_1 = \langle S_1 \times D_{V1}, \Sigma_1, T_1, (s_{01}, v_{01})\rangle$ *and* $Q_2 = \langle S_2 \times D_{V2}, \Sigma_2, T_2, (s_{02}, v_{02})\rangle$ *be the unfolding of two EA, the composition automaton* $Q = Q_1 \parallel Q_2$ *is* $Q = \langle S, \Sigma, T, s_0\rangle$, *where* $s_0 = (s_{01}, s_{02}, v_{01}, v_{02})$ *is the initial state,* $\Sigma = \Sigma_1 \cup \Sigma_2$ *is the set of events, and* $S \subseteq S_1 \times S_2 \times D_{V1} \times D_{V2}$ *and* T *are the smallest sets obtained by applying the following rules:*

1. *If* $a \in \Sigma_1 \cap \Sigma_2$, $((s_1, v_1), a, (s_1', v_1')) \in T_1$, $((s_2, v_2), a, (s_2', v_2')) \in T_2$, *and* $((s_1, s_2), v_1, v_2) \in S$, *then* $((s_1', s_2'), v_1', v_2') \in S$ *and* $(((s_1, s_2), v_1, v_2), a, ((s_1', s_2'), v_1', v_2')) \in T$;
2. *If* $a \in \Sigma_1 \backslash \Sigma_2$, $((s_1, v_1), a, (s_1', v_1')) \in T_1$, *and* $((s_1, s_2), v_1, v_2) \in S$, *then* $((s'_1, s_2), v'_1, v_2) \in S$, *and* $(((s_1, s_2), v_1, v_2), a, ((s_1', s_2), v_1', v_2)) \in T$;
3. *If* $a \in \Sigma_2 \backslash \Sigma_1$, $((s_2, v_2), a, (s_2', v_2')) \in T_2$, *and* $((s_1, s_2), v_1, v_2) \in S$, *then* $((s_1, s_2'), v_1, v'_2) \in S$, *and* $(((s_1, s_2), v_1, v_2), a, ((s_1, s_2'), v_1, v_2')) \in T$.

The composition of Communicating Extended Automata is usually built by a model checker when such a model is used for property verification.

5.2.3 Extending a System of Communicating Automata

Given a system of communicating automata, we present a procedure that extends them into Communicating Extended Automata. Each automaton is extended with a variable to track the number and execution of a designated type of events. The variables of all the automata thus designate stable and transient global states based on the type of the designated events enabled. Let Σ_d be a designated subset of events of the automaton, which decide stability of global states. As discussed in Section 5.2.1, these events labeling looping transitions cannot be executed more than once if the automaton does not reenter the state again due to other event occurrence. This assumption stems from the specifics of our

automata model (Chapter 4), where browser triggered events label only looping transitions, and are rendezvous labeling in the targeted entities a single transition from the inactive state to the first state. For each EA, the presence of this variable does not change its behavior (its traces or language), so all the predicates of the transitions are set to True; this is because the introduced event tracking variable only decorates states to provide an additional characterization. With these assumptions, communicating automata are extended as follows.

Procedure 4. Given a system of communicating automata, $A_1, \ldots, A_k$, for each $A_i = <S_i, s_{0_i}, \Sigma_i, T_i>$, $i = 1, \ldots, k$, let $\Sigma_{d_i} \subseteq \Sigma_i$ be a designated set of events, x_i the event tracking variable. A_i is extended into an EA Q_i as follows.

1. S_i, Σ_i, and s_{0_i} are the set of states, set of events, and the initial state of Q_i respectively.
2. Q_i has a single context variable x_i, and $x_i := 0$, denoted $\boldsymbol{x_{0_i}}$, defines the initial context of Q_i;
3. For each transition $(s, a, s') \in T_i$, $s, s' \in S_i$, $a \in \Sigma_i$,
   3.1. if $s = s'$ and $a \in \Sigma_{d_i}$, then $(s, a, x_i := x_i - 1, s)$ is a transition in Q_i, where $x_i := x_i - 1$ is the update of the transition; otherwise,
   3.2. $(s, a, x_i := |init(s') \cap \Sigma_{d_i}|, s')$ is a transition in Q_i, where $x_i := |init(s') \cap \Sigma_{d_i}|$ is the update of the transition;
4. Let T_{Q_i} denote the set of thus defined transitions;
5. The EA is $Q_i = <S_i, s_{0_i}, \Sigma_i, \{x_i\}, \boldsymbol{x_{0_i}}, T_{Q_i}>$.

5.3 Communicating Extended Automata Model of Web Applications

In this section, we discuss how the automata model of a WAUT defined in Section 4.3 is extended into a CEA model in order to identify global stable and transient states. Applying Procedure 3, k local browsing sessions of a WAUT are mapped into a communicating automata model, where each local session automaton A_i, $i = 1, \ldots, k$,

models a local browsing session RRS_i. $A_i = <S_i \cup \{trap_i\}, s_{0i}, \Sigma_i, T_i>$ where S_i is the set of response pages, $trap_i$ is the trap state, s_{0i} is the inactive state, and Σ_i and T_i are defined by applying steps of Procedure 3. We recall that Σ_i includes the four sets Γ_i, the set of all the URIs associated with links in the observed responses, Δ_i, the set of all form actions that correspond to the unexplored forms in the observed responses, Req_i, the set of all the observed requests, and Φ_i, the set of URIs corresponding to the source pages loaded in the frames.

Applying Procedure 4, each A_i is refined into an EA Q_i. The set $\Phi_i \subseteq \Sigma_i$ is the designated subset of events of the automaton, which together indicate stability of global states; and let x_i, the event tracking variable, be the context variable of Q_i (initially zero), $\boldsymbol{x_i}$ its valuation, such that for each state $s \in S_i$, $\boldsymbol{x_i}$ is the number of browser triggered events enabled at s, and determined as follows. For each incoming transition of s, if the labeling event is not in Φ_i, the update is $x_i := |init_i(s) \cap \Phi_i|$; otherwise, if a looping transition labeled by an event from Φ_i is executed, x_i is decremented by one; hence, the update function is $x_i := x_i - 1$. If all the transitions labeled by events from Φ_i are executed, then x_i becomes zero.

Let $Q_1, \ldots, Q_k$ be the EA that model k windows and frames, and $x_1, \ldots, x_k$ be their corresponding stability variables. The unfolding of each of $Q_1, \ldots, Q_k$ result in the automata $Q_1', \ldots, Q_k'$. The composition automata Q' is $Q_1' \parallel \ldots \parallel Q_k'$, such that $Q' = <S' \cup \{trap'\}, s_0', \Sigma', T'>$. The initial state of Q' is $s_0' = (s_{01}, \ldots, s_{0k}, \boldsymbol{x_{01}}, \ldots, \boldsymbol{x_{0k}})$; the set of events Σ' of Q' is the union of all Σ_i'; the set of states S' and the transition relation T' of Q' are defined according to the semantics of the composition operator $\parallel$ given in Definition 10. The trap state of Q' is $trap' = (trap_1, \ldots, trap_k, \boldsymbol{x_1}, \ldots, \boldsymbol{x_k})$. A state $s' = (s_1, \ldots, s_k, \boldsymbol{x_1}, \ldots, \boldsymbol{x_k})$ of Q' is stable if no browser triggered events are enabled, i.e., $\boldsymbol{x_1}, \ldots, \boldsymbol{x_k}$ are all zero; s' is transient if at least one browser triggered event is enabled, i.e., if at least one event tracking variable, say $\boldsymbol{x_i}$, $i = 1, \ldots, k$, is strictly positive.

As an illustration, consider the example described in Chapter 4, whose EA depicted in Fig. 2. In Q_1 (A_1), the initial state is s_0, and the set of states include s_1 and s_2; $\{f_1, f_2\} \subseteq$

Φ_1, $\{a,c\} \subseteq \Sigma_1 \backslash \Phi_1$, and $V_1 = \{x_1\}$. Q_2 (A_2) has an initial state u_0 and the set of states include u_1, and u_2; $\{f_1, b, c\} \subseteq \Sigma_2 \backslash \Phi_2$, and $V_2 = \{x_2\}$. Q_3 (A_3) has an initial state w_0 and the set of states include w_1, and w_2; $\{c, f_2\} \subseteq \Sigma_1 \backslash \Phi_3$, and $V_3 = \{x_3\}$. The variables x_1, x_2, and x_3 are initially zero. In Q_1, x_1 is set to 2 by the update of the transition labeled by a, since f_1, and f_2 are enabled at s_1; f_1, and f_2 label looping transitions whose update functions decrement x_1 by 1. For simplicity, we omit the update function of the transitions of Q_2 since their stability variables remain zero.

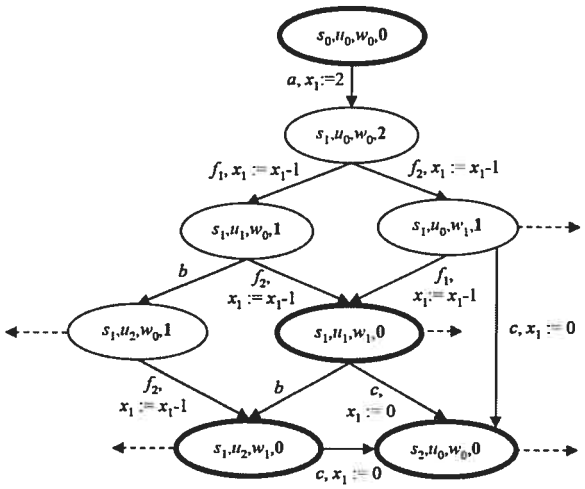


Figure 13. Composition of Q_1, Q_2, and Q_3

The composition automaton is shown in Figure 13. The bold circles are the global stable sates, in which x_1, x_2, and x_3 are zero. The rest of the states are transient, in which the value of x_1 is greater than zero. Note that for simplicity, we do not show the valuation of x_2 and x_3, since they are equal to zero. As opposed to the automata composition of Fig. 3, the composition of extended automata in Figure 13 contains the explicit distinction between stable and transient global states.

5.4 Summary

We elaborated an algorithm to extend our automata model with event tracking variables and updates on them. These variables are essential to designate stable and transient global states in a communicating automata model of a WAUT. The results of this

chapter can be found in [39]. In the next chapter, we solve the problem of property specification in arbitrary scopes of states using LTL.

Chapter 6

LTL Expressiveness: Limitations and Solution

Over the years, it has been realized that the main hurdles in applying model checking for software systems include the following. First, the complexity of modern programming languages and thus software systems is high. Second, it is cumbersome even to the expert, and virtually impossible [6] for the novice, to specify meaningful (often complex) properties using the usual temporal logic formalisms of model checking. This difficulty of expressing properties of the system grows even bigger when it comes to specifying properties of interest on part of a given system while ignoring the rest of it. In this chapter, we address the problem of property specification in LTL assuming that the user is interested in verifying properties over a subset of states while ignoring the valuation of those properties in the remaining states.

6.1 Introduction

In this chapter, we argue that applying abstraction techniques on the system model, where "uninteresting states" are discarded from the system model, and specifying the properties on the resulting model, may not constitute an adequate solution to the problem.

This work proposes a solution to the stated problem that does not require any changes in the system model. The idea is to define specialized operators for the specification of properties in a subset of states of interest. The new operators do not change the expressiveness of LTL, but rather help specifying the properties of interest more intuitively and succinctly.

We first explain the problem related to known difficulties in the specification of properties in LTL formalism. We also describe our solution, namely the syntax and semantics of the new operators and the proof of their correctness, and illustrate the usefulness of our approach with an example. We finally show how our results can be used in our web specification patterns system, WeSPaS, introduced in Chapter 3, as well as in the SPS [88].

6.2 LTL Limitations

Although LTL has been widely considered a natural choice for automata based verification of reactive systems, it suffers from few limitations when it comes to the expressiveness of the language [45]. Over the last two decades, there have been discussions [31,53, 53] on comparing LTL to other formalisms such as CTL, μ-calculus, automata, etc. Each formalism has its own pros and cons in terms of expressiveness and complexity (when executing the verification algorithm). These advantages and disadvantages may not be uniquely identified by different research and industrial communities depending on the variations of their specific needs.

Expressive power of LTL is sufficient for many practical specification tasks, however, specifying non-trivial properties in LTL, as well as in other temporal logics, is often considered difficult even for experts and virtually impossible for novices [6]. One particularly difficult problem is expressing non-trivial properties which are related only to a subset of the states of a system under test. While the problem was partially resolved with templates [27,28], syntax sugar [6], visual tools for property specification such as the Timeline editor [74], and even designated graphical logics for intervals of states [26], it is not yet resolved for more arbitrary state subsets, e.g., defined by a propositional formula. A suitable approach might be to add so-called syntax sugar operators, which allow succinct property specification, while known LTL model checking tools and algorithms still apply. The challenge is to specify properties over certain states that are of interest while ignoring the validity of the property in the remaining states. In other words, one needs to define a scope as an arbitrary set of states over a single path and verify the property over that scope.

Distinction between the states of a system may, for example, depend on the type of actions enabled at each state. This has practical significance since the resulting partition of states could be used to express various levels of granularity at which the behavior of the system is described. Example of state partitioning is stable (sometimes called quiescent) vs. transient (sometimes called intermediate), as described in Chapter 5. A simple example is the property Fp which is valid on a path, but is invalid in designated stable states of the same path.

The property "eventually p on stable states of path" could be reformulated as F($p \wedge$ *stable*), where *stable* is a predicate that identifies stable states. However, for more complicated properties, even for other operators, the solution is not as simple as a conjunction of a predicate with the formula, as we show in Section 3. This problem is mentioned in [28], where the authors stated that Boolean variables could be used in the property specification to distinguish between states and concluded that simple conjunction and disjunction of Booleans with the original property do not serve the purpose.

A straightforward solution to this problem is to remove the "uninteresting" states from the model leaving only the subset of states in which the properties need to be verified. This solution would rely on a projection of a given KS onto a subset of states that are of interest.

Definition 11. *Let* $M = (S, T, S_0, \mathcal{L})$ *and* $M' = (S', T', S_0', \mathcal{L}')$ *be two KSs such that* $S' \subseteq S$. *We say that* M' *is a projection of* M *onto* S', *iff*

1. $T' = \{(s_i, s_k) \mid s_i, s_k \in S'$, *and either* $(s_i, s_k) \in T$ *or there exists a path suffix* $\pi^i = \langle s_i, s_{i+1}, \ldots, s_{k-1}, s_k, \ldots\rangle$, *such that* $s_{i+1}, \ldots, s_{k-1} \notin S'\}$,
2. $S_0' = \{s_i \mid s_i \in S_0 \cap S'$ *or there exists a path* $\pi = \langle s_0, s_1, \ldots, s_{i-1}, s_i, \ldots\rangle$ *of* M, *such that* $s_0 \in S_0 \setminus S'$ *and* $s_0, \ldots, s_{i-1} \notin S'\}$, *and*
3. $\mathcal{L}'(s) = \mathcal{L}(s)$ *for all* $s \in S'$.

Note that if $S' = S$, then $M' = M$.

Then, a standard model checking algorithm [16] could be used to verify the properties on the projection of the model. However, with such a solution, one faces two main problems:

1. If there exists a number of properties each of which concerns a different subset of states, then for each property one has to project the model separately. So the number of models may reach the number of properties.
2. The proposed solution may be not applicable for model checkers, like Spin [44], which use Kripke representation internally, and where the user specifies a modular system in a high level language, such as Promela.

Our solution is to introduce new LTL operators so that properties are verified over an arbitrary subset of states, and at the same time, standard LTL model checking algorithms and tools can still be used. Such a solution does not require any change in the model of the system; it rather helps in expressing properties in question more succinctly and intuitively. As we prove in the next section, the semantics of the new operators follows from the semantics of existing LTL operators.

6.3 Extending LTL with Propositional Scopes

In this section, we discuss how to specify LTL properties that should be verified over arbitrary subsets of the state space of the system under test. However, we first give a definition of a scope. We define a scope of a linear temporal logic formula over a given path as the subset of states on the path where the formula is checked.

Based on this definition, we consider the partition of the state space into in-scope and out-of-scope states. In-scope states are the states of interest, where a given property has to be checked, while ignoring the valuation of the property in the remaining states, which we designate out-of-scope states. For this purpose, we introduce new LTL operators that can be used to formulate properties in the in-scope states. These operators do not extend the LTL formalism; they rather help formulating properties more succinctly. We denote $\Im$ a propositional logic expression that valuates to True in every state where a given property

should be verified. The set of states in which $\Im$ holds constitutes what we call $\Im$-*scope*. Since any LTL property is expressible with the $\neg$, $\wedge$, U, and X operators, we generalize them as the operators $\neg_{\Im}$ (*not in scope*), $\wedge_{\Im}$ (*and in scope*), $U_{\Im}$ (*until in scope*), and $X_{\Im}$ (*next in scope*), and use the obtained operators to derive $F_{\Im}$ (*eventually in scope*) and $G_{\Im}$ (*always in scope*). If $\Im$-scope is the full set of states of the system, those operators coincide with their corresponding LTL counterparts, namely, $\neg$, $\wedge$, U, X, F, and G.

In the following, we formally define the $\Im$-scope operators. However, we first explain their intended semantics informally to clarify the intuition behind each of them. For example, $\neg_{\Im}\varphi$ implies that φ does not hold in the first in-scope state encountered. $\varphi\, U_{\Im}\, \psi$ means that φ holds in all the in-scope states preceding the one in which ψ holds. $X_{\Im}\varphi$ means that φ holds in the next in-scope state after the first such state encountered, and if no in-scope states exist along the path, the property will not hold. $F_{\Im}\varphi$ means that φ eventually holds in an in-scope state irrespective of its validity in out-of-scope states. Similarly, $G_{\Im}\varphi$ means that φ holds in all the in-scope states on the path. Figure 14 shows examples of properties using $\Im$-scope operators.

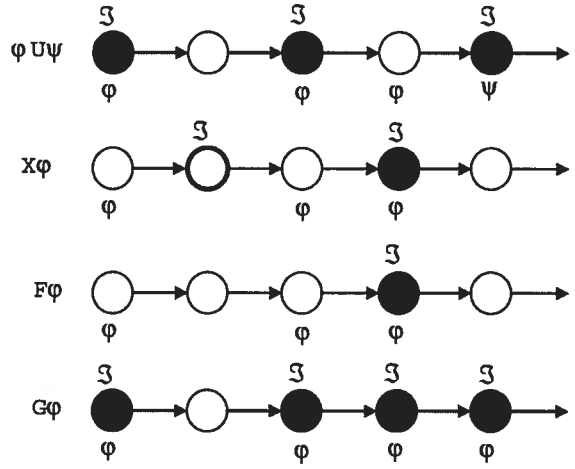


Figure 14. Examples of Properties Using $\Im$-scope Operators.

Definition 12. *Let* $\Im$ *be a propositional logic expression, the operators* $\neg_{\Im}$, $\wedge_{\Im}$, $U_{\Im}$, $X_{\Im}$, $F_{\Im}$, *and* $G_{\Im}$ *are as follows*:

1. $\neg_{\Im} \varphi \stackrel{def}{=} \neg \Im \, U \, (\neg\varphi \wedge \Im)$
2. $\varphi \wedge_{\Im} \psi \stackrel{def}{=} \neg\Im \, U \, ((\varphi \wedge\psi) \wedge \Im)$
3. $\varphi \, U_{\Im} \, \psi \stackrel{def}{=} (\Im \rightarrow \varphi) \, U \, (\psi \wedge \Im)$
4. $X_{\Im} \, \varphi \stackrel{def}{=} \neg\Im \, U \, [\Im \wedge X \, (\neg\Im \, U \, (\Im \wedge \varphi))]$
5. $F_{\Im} \, \varphi \stackrel{def}{=} F \, (\varphi \wedge \Im)$
6. $G_{\Im} \, \varphi \stackrel{def}{=} G \, (\Im \rightarrow \varphi)$

Based on these auxiliary definitions, we introduce the $\Im$-scope operator denoted **In**, with the following recursive definition. The **In** operator simplifies the task of formulating a property within a given scope.

Definition 13. *Let $\Im$ be a propositional logic expression, the $\Im$-scope operator* **In** *is defined as follows:*

1. p **In** $\Im = \neg\Im \, U \, (p \wedge \Im)$
2. $(\neg\varphi)$ **In** $\Im = \neg_{\Im} \, (\varphi$ **In** $\Im)$
3. $(\varphi \wedge \psi)$ **In** $\Im = (\varphi$ **In** $\Im) \wedge_{\Im} (\psi$ **In** $\Im)$
4. $(\varphi \, U \, \psi)$ **In** $\Im = (\varphi$ **In** $\Im) \, U_{\Im} \, (\psi$ **In** $\Im)$
5. $(G \, \varphi)$ **In** $\Im = G_{\Im} \, (\varphi$ **In** $\Im)$
6. $(F \, \varphi)$ **In** $\Im = F_{\Im} \, (\varphi$ **In** $\Im)$
7. $(X \, \varphi)$ **In** $\Im = X_{\Im} \, (\varphi$ **In** $\Im)$

The following lemmas and theorems describe the semantics of the introduced operators.

Lemma 1. $\pi \models p$ **In** $\Im \Leftrightarrow$ *there exists an* i, $0 \leq i < |\pi|$, *such that* $\pi^i \models p$ *and* $\pi^i \models \Im$, *and for all* j, $0 \leq j < i$, $\pi^j \not\models \Im$.

Proof. $\pi \models$ p **In** $\Im \Leftrightarrow$ (According to the definition of p **In** $\Im$), $\pi \models \neg\Im \, U \, (p \wedge \Im)$.

$\Leftrightarrow$ (By semantics of U), there exists an i, $0 \le i < |\pi|$ such that $\pi^i \models (p \wedge \Im)$ and for all j, $0 \le j < i$, $\pi^j \models (\neg\Im)$, and (by semantics of $\wedge$) $\pi^i \models p$ and $\pi^i \models \Im$, (by semantics of $\neg$) $\pi^j \not\models \Im$.

$\Leftrightarrow$ There exists an i, $1 \le i < |\pi|$ such that $\pi^i \models p$ and $\pi^i \models \Im$, and for all j, $0 \le j < i$, $\pi^j \not\models \Im$. QED

Lemma 2. $\pi \models \neg_{\Im} \varphi \Leftrightarrow$ *there exists an* i, $0 \le i < |\pi|$, *such that* $\pi^i \not\models \varphi$ *and* $\pi^i \models \Im$, *and for all* j, $0 \le j < i$, $\pi^j \not\models \Im$.

Proof. Lemma 2 directly follows from Lemma 1. QED

Lemma 3. $\pi \models \varphi \, U_{\Im} \, \psi \Leftrightarrow$ *there exists an* i, $0 \le i < |\pi|$, *such that* $\pi^i \models \psi$ *and* $\pi^i \models \Im$, *and for all* j, $0 \le j < i$ $\pi^j \not\models \Im$ *or* $\pi^j \models \varphi$.

Proof. $\pi \models \varphi \, U_{\Im} \, \psi \Leftrightarrow$ (According to definition of $U_{\Im}$), $\pi \models (\Im \rightarrow \varphi) \, U \, (\psi \wedge \Im)$.

$\Leftrightarrow$ (By semantics of U), there exists an i, $0 \le i < |\pi|$ such that $\pi^i \models (\psi \wedge \Im)$ and for all j, $0 \le j < i$, $\pi^j \models (\Im \rightarrow \varphi)$; and (by semantics of $\wedge$), $\pi^i \models \psi$ and $\pi^i \models \Im$, and (by definition of $\rightarrow$), $\Im \rightarrow \varphi = \neg \Im \vee \varphi$, and (by semantics of $\neg$ and $\vee$) $\pi^j \not\models \Im$ or $\pi^j \models \varphi$.

$\Leftrightarrow$ There exists an i, $0 \le i < |\pi|$ such that $\pi^i \models \psi$ and $\pi^i \models \Im$, and for all j, $0 \le j < i$, $\pi^j \not\models \Im$ or $\pi^j \models \varphi$. QED

Lemma 4. $\pi \models X_{\Im} \varphi \Leftrightarrow$ *there exist* i, k, $0 \le i < k \le |\pi|$, *such that* $\pi^i \models \Im$, $\pi^k \models \Im$ *and* $\pi^k \models \varphi$, *and for all* j, l, $0 \le j < i < l < k$, $\pi^j \not\models \Im$ *and* $\pi^l \not\models \Im$.

Proof. $\pi \models X_{\Im} \varphi \Leftrightarrow \pi \models \neg\Im \, U \, [\Im \wedge X \, (\neg\Im \, U \, (\Im \wedge \varphi))]$.

$\Leftrightarrow$ (According to semantics of U), there exists an i, $0 \leq i < |\pi|$ such that $\pi^i \models [\Im \wedge X (\neg\Im \text{ U } (\Im \wedge \varphi))]$ and for all j, $0 \leq j < i$, $\pi^j \models \neg \Im$; and (by semantics of $\wedge$), $\pi^i \models \Im$ and $\pi^i \models X (\neg\Im \text{ U } (\Im \wedge \varphi))$, and (by semantics of X), $\pi^{i+1} \models (\neg\Im \text{ U } (\Im \wedge \varphi))$; and (by semantics of U), there exists k, $i + 1 \leq k < |\pi|$ such that $\pi^k \models (\Im \wedge \varphi)$ and for all l, $i < l < k$, $\pi^l \models \neg\Im$.

$\Leftrightarrow$ There exist i, k, $0 \leq i < k < |\pi|$, such that $\pi^i \models \Im$, $\pi^k \models \Im$ and $\pi^k \models \varphi$, and for all j, l, $0 \leq j < i < l < k$, $\pi^j \not\models \Im$, and $\pi^l \not\models \Im$. QED

Lemma 5. $\pi \models F_{\Im} \varphi \Leftrightarrow$ *for some* i, $0 \leq i < |\pi|$, $\pi^i \models \varphi$ *and* $\pi^i \models \Im$.

Proof. Directly follows from the semantics of F. QED

Lemma 6. $\pi \models G_{\Im} \varphi \Leftrightarrow$ *for all* i, $0 \leq i < |\pi|$, *where* $\pi^i \models \Im$, $\pi^i \models \varphi$.

Proof. Directly follows from the semantics of G. QED

To demonstrate the correctness of the definitions and semantics of $\Im$-*scope* operators and formulae, we state two theorems in which we claim that if a formula (φ **In** $\Im$) holds in a given path (model) including in-scope and out-of-scope states the corresponding LTL formula φ must hold in the projection of the path (model) including only the in-scope states. To this end, we first define a projection relation based on Definition 11 that removes the out-of-scope states from a path and keeps only the in-scope states.

Definition 14. *Let* $\Im$ *be a propositional logic formula,* $M = (S, T, S_0, \mathcal{L})$ *be a KS,* $S_{\Im} = \{s \in S \mid \Im \text{ is true in } s\}$, *and let* $\pi = \langle s_0, s_1, \ldots \rangle$ *be a path of M. The projection of* π *onto* $S_{\Im}$, *denoted* $\pi_{\downarrow \Im}$, *is the* (*possibly finite*) *subsequence of* π *derived by discarding all states* s_i *from* π *such that* $s_i \notin S_{\Im}$.

Proposition 4. *Let $\mathfrak{I}$ be a propositional logic formula, and $M = (S, T, S_0, \mathcal{L})$ and $M_{\mathfrak{I}} = (S_{\mathfrak{I}}, T_{\mathfrak{I}}, S_{0\mathfrak{I}}, \mathcal{L}_{\mathfrak{I}})$ be two KSs, where $M_{\mathfrak{I}}$ is the projection of M onto $S_{\mathfrak{I}}$, and π be a path of M, then $\pi_{\downarrow\mathfrak{I}} \neq \varepsilon$ is a path of $M_{\mathfrak{I}}$.*

Theorem 4. *For any LTL formula φ and its corresponding formula φ `In` $\mathfrak{I}$, $\pi \models \varphi$ `In` $\mathfrak{I}$ if and only if $\pi_{\downarrow\mathfrak{I}} \models \varphi$.*

Proof. The proof is by induction on the number of operators, n, of a formula φ. For simplicity, we assume that $\mathfrak{I}$ is an atomic proposition. The proof could easily be extended onto any propositional $\mathfrak{I}$. To make the proof more intuitive, we index the formulae with the number of operators that constitute each formula.

Base case. For $n = 0$, where the formula is simply an atomic predicate and φ_0 `In` $\mathfrak{I} = p$ `In` $\mathfrak{I}$, we prove that $\pi \models p$ `In` $\mathfrak{I}$ if and only if $\pi_{\downarrow\mathfrak{I}} \models p$.

p `In` $\mathfrak{I} \Leftrightarrow \pi \models \neg\, \mathfrak{I}\, \mathrm{U}\, (p \wedge \mathfrak{I})$.

$\Leftrightarrow$ (According to Lemma 1) there exists an i, $0 \leq i < |\pi|$, such that $\pi^i \models p$ and $\pi^i \models \mathfrak{I}$, and for all j, $0 \leq j < i$, $\pi^j \not\models \mathfrak{I}$.

$\Leftrightarrow$ There exists an i, $0 \leq i < |\pi|$, such that $p \in \mathcal{L}(s_i)$ and $\mathfrak{I} \in \mathcal{L}(s_i)$, and for all j, $0 \leq j < i$, $\mathfrak{I} \notin \mathcal{L}(s_j)$; and (by Definition 4) s_i is the first state in $\pi^i{\downarrow\mathfrak{I}}$ and for all j, $0 \leq j < i$, s_j is not in $\pi^j{\downarrow\mathfrak{I}}$.

$\Leftrightarrow$ There exists an i, $0 \leq i < |\pi|$, such that $\pi = \langle \ldots, s_i, \ldots \rangle$, $\pi^i{\downarrow\mathfrak{I}} \models p$ and $\pi_{\downarrow\mathfrak{I}} = \pi^i{\downarrow\mathfrak{I}} = \langle s_i, \ldots \rangle$.

$\Leftrightarrow$ (According to the semantics of p) $\pi_{\downarrow\mathfrak{I}} \models p$.

Inductive step. Assume $\pi \models \varphi_m$ `In` $\mathfrak{I}$ if and only if $\pi_{\downarrow\mathfrak{I}} \models \varphi_m$ for all formulae φ_m that consist of m operators, m, $0 \leq m \leq n$, holds. We must show that the equivalence holds as well for $n + 1$. Due to lack of space, we present here the proofs only for the most

complicated cases where the formula φ_{n+1} is of the form χ_u U ψ_v, where χ_u and ψ_v are formulae with u and v operators respectively, such that, $0 \le u \le n$, $0 \le v \le n$, and $u + v = n$, and of the form X ψ_n. The proofs for the remaining LTL operators could be performed in a similar manner.

(1) Here we prove that for all formulae χ_u, ψ_v which together contain n or less operators, $\pi \vDash (\chi_u \text{ U } \psi_v)$ **In** $\Im$ if and only if $\pi_{\downarrow\Im} \vDash \chi_u \text{ U } \psi_v$.

$\pi \vDash (\chi_u \text{ U } \psi_v)$ **In** $\Im \Leftrightarrow \pi \vDash (\chi_u$ **In** $\Im)$ U$_\Im$ $(\psi_v$ **In** $\Im)$.

$\Leftrightarrow$ (According to Lemma 3) there exists an i, $0 \le i < |\pi|$, such that $\pi^i \vDash (\psi_v$ **In** $\Im)$ and $\pi^i \vDash \Im$, and for all j, $0 \le j < i$ $\pi^j \nvDash \Im$ or $\pi^j \vDash (\chi_u$ **In** $\Im)$.

$\Leftrightarrow$ There exists an i, $0 \le i < |\pi|$, such that (by Definition 4) s_i is the first state in $\pi^i{}_{\downarrow\Im}$, and (according to the induction hypothesis) for all i, $0 \le i < |\pi|$, $\pi^i{}_{\downarrow\Im} \vDash \psi_v$; and for all j, $0 \le j < i$, either (by Definition 4) s_j is not in $\pi^i{}_{\downarrow\Im}$, or s_j is a state in $\pi^i{}_{\downarrow\Im}$ and (by induction hypothesis) $\pi^j{}_{\downarrow\Im} \vDash \chi_u$.

$\Leftrightarrow$ There exists an i, $0 \le i < |\pi|$, such that $\pi = \langle \ldots, s_i, \ldots \rangle$, $(\pi^i{}_{\downarrow\Im} \vDash \psi_m$, and $\pi_{\downarrow\Im} = \pi^i{}_{\downarrow\Im} = \langle s_i, \ldots \rangle)$ or $(\pi^i{}_{\downarrow\Im} \vDash \psi_v$, and for all j, $0 \le j < i$, $\pi^j{}_{\downarrow\Im} \vDash \chi_u)$.

$\Leftrightarrow$ (By semantics of U) $\pi_{\downarrow\Im} \vDash \chi_u \text{ U } \psi_v$.

(2) Here we prove that for each formula ψ_n that contain n or less operators, $\pi \vDash$ (X ψ_n) **In** $\Im$ if and only if $\pi_{\downarrow\Im} \vDash$ X ψ_n.

$\pi \vDash (\text{X } \psi_n)$ **In** $\Im \Leftrightarrow \text{X}_\Im$ $(\psi_n$ **In** $\Im)$.

$\Leftrightarrow$ (According to Lemma 4) there exist i, k, $0 \le i < k < |\pi|$, such that $\pi^i \vDash \Im$, $\pi^k \vDash \Im$ and $\pi^k \vDash (\psi_n$ **In** $\Im)$, and for all j, l, $0 \le j < i < l < k$, $\pi^j \nvDash \Im$, and $\pi^l \nvDash \Im$.

$\Leftrightarrow$ There exist i, k, $0 \le i < k < |\pi|$, such that $\Im \in \mathcal{L}(s_i)$, $\Im \in \mathcal{L}(s_k)$, and $\pi^k \models (\psi_n \ \mathbf{In} \ \Im)$, and for all j, $0 \le j < i$, $\Im \notin \mathcal{L}(s_j)$, and for all l, $i+1 \le l < k$, $\Im \notin \mathcal{L}(s_l)$.

$\Leftrightarrow$ There exist i, k, $0 \le i < k < |\pi|$, such that (by Definition 4) s_i is the first state in $\pi^i{\downarrow_\Im}$, s_k is the first state in $\pi^k{\downarrow_\Im}$, such that (by induction hypothesis) $\pi^k{\downarrow_\Im} \models \psi_n$, and for all j, $0 \le j < i$, s_j is not in $\pi^j{\downarrow_\Im}$, and for all l, $i+1 \le l < k$, s_l is not in $\pi^l{\downarrow_\Im}$.

$\Leftrightarrow$ There exist i, k, $0 \le i < k < |\pi|$, such that $\pi = \langle \ldots, s_i, \ldots, s_k, \ldots \rangle$, $\pi{\downarrow_\Im} = \langle s_i, s_k, \ldots \rangle$, and $\pi^k{\downarrow_\Im} \models \psi_n$.

$\Leftrightarrow$ (By semantics of X) $\pi{\downarrow_\Im} \models \mathrm{X}\ \psi_n$.

The base case and the induction step imply that the theorem holds for all cases of n. QED

Given Proposition 4 and Theorem 4, we have the following theorem.

Theorem 5. *Let $\Im$ be a propositional logic formula, and let $M = (S, T, S_0, \mathcal{L})$ and $M_\Im = (S_\Im, T_\Im, S_{0\Im}, \mathcal{L}_\Im)$ be two KSs, $M_\Im$ is the projection of M onto $S_\Im \subseteq S$ such that $\Im$ is true in all $s \in S_\Im$. For any LTL formula φ, $M \models \varphi$ **In** $\Im$ if and only if $M_\Im \models \varphi$.*

Note that Definition 12 and Definition 13 are not the only possible way of expressing propositional scopes in LTL formulae. For example, we surmise that shorter formulae could be obtained with a longer list of rules that differentiate between temporal and propositional terms. These rules are presented in the following definition.

Definition 15. *Let φ and ψ be arbitrary LTL formulae, p, q, and $\Im$ be a propositional logic expression, the $\Im$-scope operator* **In** *is defined as follows:*

1. $p \ \mathbf{In} \ \Im = \neg\Im \ \mathrm{U} \ (p \wedge \Im)$
2. $(\neg\varphi) \ \mathbf{In} \ \Im = \neg_\Im \ (\varphi \ \mathbf{In} \ \Im)$
3. $(p \wedge q) \ \mathbf{In} \ \Im = p \wedge_\Im q$
4. $(p \ \mathrm{U} \ q) \ \mathbf{In} \ \Im = p \ \mathrm{U}_\Im \ q$

5. $(\mathrm{G}\, p)$ **In** $\Im = \mathrm{G}_{\Im}\, p$
6. $(\mathrm{F}\, p)$ **In** $\Im = \mathrm{F}_{\Im}\, p$
7. $(\varphi \wedge \psi)$ **In** $\Im = (\varphi$ **In** $\Im) \wedge_{\Im} (\psi$ **In** $\Im)$
8. $(\varphi \,\mathrm{U}\, \psi)$ **In** $\Im = (\varphi$ **In** $\Im) \,\mathrm{U}_{\Im}\, (\psi$ **In** $\Im)$
9. $(\mathrm{G}\, \varphi)$ **In** $\Im = \mathrm{G}_{\Im}\, (\varphi$ **In** $\Im)$
10. $(\mathrm{F}\, \varphi)$ **In** $\Im = \mathrm{F}_{\Im}\, (\varphi$ **In** $\Im)$
11. $(\mathrm{X}\, \varphi)$ **In** $\Im = \neg\Im \,\mathrm{U}\, (\Im \wedge \mathrm{X}\, (\varphi$ **In** $\Im))$

Translation into automata could be even more efficient, though not all verification tools allow direct specification of properties in automata.

6.4 LTL Scopes in Web Properties

We provide examples of web related properties, taken from our WeSPaS of Chapter 3, which are fine grained with propositional scopes that not only designate stable and transient global states, but also in general scopes designating global states of interest identified by a given proposition.

The following property designate the E-commerce pattern FEPN10 (Section 3.3):

1. *Promotions of certain products are only present either on the Home page or on Shopping pages and their number does not exceed 2.*

This property is to ensure that a promoted product is not oversold. Using standard LTL, the property is written as follows:

$$\mathrm{G}(((\neg \mathit{Home} \wedge \neg \mathit{Shopping}) \rightarrow (\mathit{Promotions} = 0)) \wedge ((\mathit{Home} \vee \mathit{Shopping}) \rightarrow (\mathit{Promotions} \leq 2))) \qquad (6.4.1.1)$$

The global states that concern the property are the ones that designate the home page and the shopping pages. Therefore, the scope of global states is the propositional formula $(\mathit{Home} \vee \mathit{Shopping})$. Using the scope operator the property can be written more intuitively as follows:

$$G(((Promotions \leq 2)\ \mathbf{In}\ (Home \vee Shopping)) \vee (Promotions = 0)) \quad (6.4.1.2)$$

The following property is another example that relates to WAs that comprise highly secure information. It designates the functional security security related pattern FGS7.

2. *The User visits Authentication page then Secure page then returns to Authentication and does this exactly twice.*

In this property, the *User* is able to visit a certain secure page which he is allowed to visit only twice and every time with authentication information. Therefore, given the stated property, we are not interested in any pages other than *Authentication* page and *Secure* page. Using standard LTL, the generalized property is non-trivial and tricky to specify:

$$(\neg Authentication \wedge \neg Secure)\ U\ (Authentication \wedge \neg Secure \wedge X\ (\neg Secure\ U\ (Secure \wedge \neg Authentication \wedge X\ (\neg Authentication\ U\ (Authentication \wedge \neg Secure \wedge X\ (\neg Secure\ U\ (Secure \wedge \neg Authentication \wedge X\ (\neg Authentication\ U\ (Authentication \wedge \neg Secure \wedge X\ (G\ (\neg Secure)))))))))) \quad (6.4.2.1)$$

If we use now the **In** operator, the scope of the property includes global states where the user can be in *Authentication* or in *Secure* pages. Therefore, the scope can be written as: (*Authentication* $\vee$ *Secure*). Now, the property can be written, in a more intuitive and succinct way, with the **In** operator as follows:

$$Authentication \wedge X(Secure \wedge X\ (Authentication \wedge X\ (Secure \wedge X\ (Authentication \wedge G\ (\neg Secure))))))\ \mathbf{In}\ (Authentication + Secure) \quad (6.4.2.2)$$

Assume that the same property has to be verified in a WA that include frames and where only stable global states have to be checked. Without using the **In** operator, the formula in (6.4.2.1) would be even more complex if we take into account stable global states only, while if we use **In** operator, then we reuse formula (6.4.2.2) and modify the scope as follows:

$$Authentication \wedge X\ (Secure \wedge X\ (Authentication \wedge X\ (Secure \wedge X\ (Authentication \wedge G\ (\neg Secure))))))\ \mathbf{In}\ ((Authentication + Secure) \wedge stable) \quad (6.4.2.3)$$

The proposition *stable* is the conjunction of the predicates related to the event tracking variables of the model's corresponding component automata.

6.5 Combining $\mathfrak{I}$-Scope with System of Property Pattern Scopes

Using scopes to limit the domain over which a property is verified was addressed in [27,28] by Dwyer et al. The authors identify several scopes which are used to define the part of a system execution, where a property must hold. A scope is determined by specifying starting and ending state/event for the property. The defined scopes are then used within a system of property specification patterns that are useful for non-experts to read and write formal specifications of systems. We believe that the $\mathfrak{I}$-scope, when combined with the existing scope definitions of the SPS, provides a possibility to further enrich the expressiveness of patterns and make them more useful in practice.

The defined $\mathfrak{I}$-scope can be combined with the pattern scopes introduced in the SPS [27,28] to provide the user of the SPS with more flexibility to specify real world properties. For example, given the Existence pattern in the global scope, we can verify it also in some states of interest defined by a given propositional logic expression $\mathfrak{I}$. Thus the pattern becomes "Exist Globally in $\mathfrak{I}$-scope states" and the corresponding LTL formula is: F ($P \wedge \mathfrak{I}$).

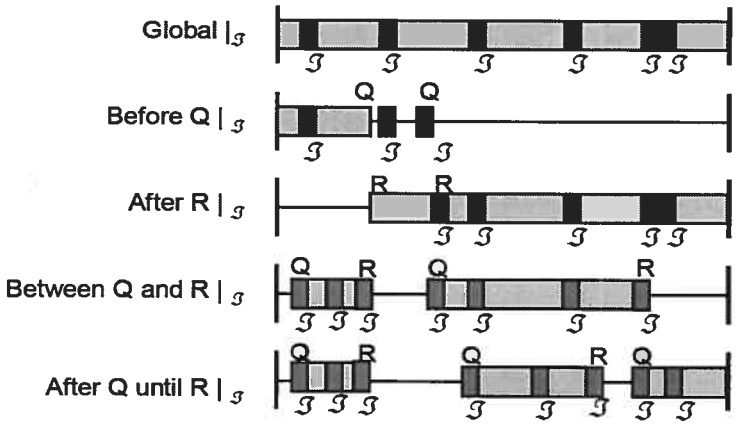


Figure 15. New Scopes

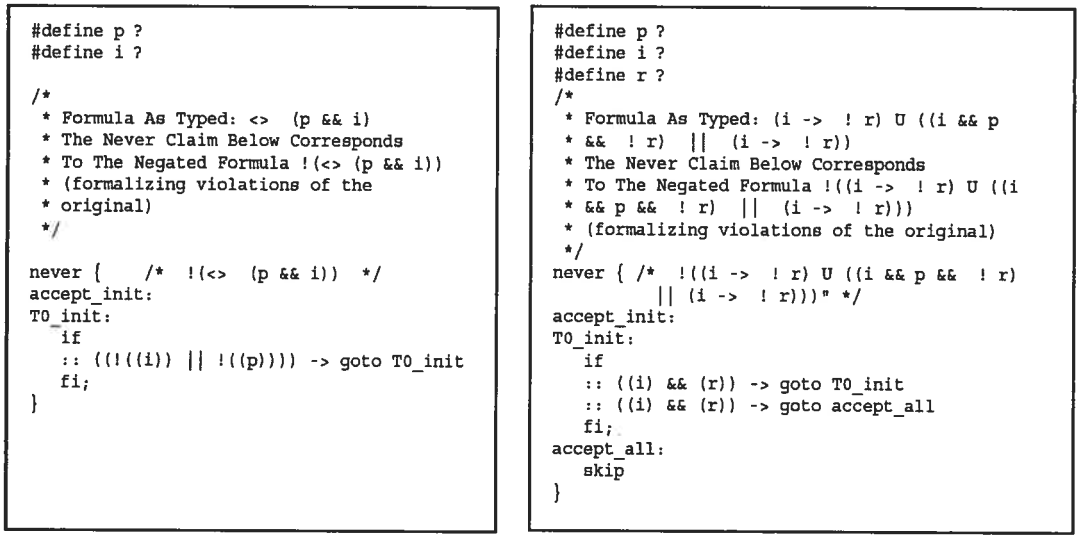
Figure 15 shows the combination of the original scopes (represented in light gray) and the $\mathfrak{I}$-scope. The resulting scopes are shown using dark gray rectangles.

Consequently, we introduce LTL templates for the specification patterns with the modified scopes. These templates can be loaded into the LTL Property Manager of Spin where the propositional expression of the scope can be instantiated in the macro definition of the scope within the templates as needed by the user.

Figure 16 and Figure 17 are examples of the new scopes applied to the patterns Absence, Existence, and Precedence, they also show the resulting templates. Note that new patterns are obtained by rewriting the old patterns using the operator `In` and the proposition $\mathfrak{I}$, and unfolding and simplifying the resulting formulae.

Absence pattern: P is false	
Globally in $\mathfrak{I}$-scope:	$G\,(\mathfrak{I} \rightarrow \neg P)$
Before R in $\mathfrak{I}$-scope:	$F\,(R \wedge \mathfrak{I}) \rightarrow (\mathfrak{I} \rightarrow \neg P)\ U\ (R \wedge \mathfrak{I})$
After Q in $\mathfrak{I}$-scope:	$G\,((\mathfrak{I} \rightarrow Q) \rightarrow G\,(\mathfrak{I} \rightarrow \neg P))$
Between Q and R in $\mathfrak{I}$-scope:	$G\,((\mathfrak{I} \rightarrow Q \wedge \neg R \wedge F\,(R \wedge \mathfrak{I})) \rightarrow (\mathfrak{I} \rightarrow \neg P)\ U\ (\mathfrak{I} \wedge R))$
After Q until R in $\mathfrak{I}$-scope:	$G\,((\mathfrak{I} \rightarrow Q \wedge \neg R) \rightarrow (\mathfrak{I} \rightarrow \neg P)\ W\ (\mathfrak{I} \wedge R))$
Existence pattern: P becomes true	
Globally in $\mathfrak{I}$-scope:	$F\,(P \wedge \mathfrak{I})$
Before R in $\mathfrak{I}$-scope:	$(\mathfrak{I} \rightarrow \neg R)\ U\ ((\mathfrak{I} \wedge P \wedge \neg R) \mid (\mathfrak{I} \rightarrow \neg R))$
After Q in $\mathfrak{I}$-scope:	$G\,(\mathfrak{I} \rightarrow \neg Q) \mid F\,(\mathfrak{I} \wedge Q \wedge F\,(\mathfrak{I} \wedge P))$
Between Q and R in $\mathfrak{I}$-scope:	$G\,((\mathfrak{I} \rightarrow Q \wedge \neg R) \rightarrow (\mathfrak{I} \rightarrow \neg R)\ W\ (\mathfrak{I} \wedge P \wedge \neg R))$
After Q until R in $\mathfrak{I}$-scope:	$G\,((\mathfrak{I} \rightarrow Q \wedge \neg R) \rightarrow (\mathfrak{I} \rightarrow \neg R)\ U\ (P \wedge \neg R \wedge \mathfrak{I}))$
Precedence pattern: S precedes P	
Globally in $\mathfrak{I}$-scope:	$(\mathfrak{I} \rightarrow \neg P)\ W\ (\mathfrak{I} \wedge S)$
Before R in $\mathfrak{I}$-scope:	$F(\mathfrak{I} \wedge R) \rightarrow (\mathfrak{I} \rightarrow P)\ U\ (\mathfrak{I} \wedge (S \mid R))$
After Q in $\mathfrak{I}$-scope:	$G\,(\mathfrak{I} \rightarrow \neg Q) \mid F\,(\mathfrak{I} \wedge Q \wedge (\mathfrak{I} \rightarrow \neg P)\ W\ (\mathfrak{I} \wedge S))$
Between Q and R in $\mathfrak{I}$-scope:	$G\,((\mathfrak{I} \rightarrow Q \wedge \neg R \wedge F\,(R \wedge \mathfrak{I})) \rightarrow (\mathfrak{I} \rightarrow \neg P)\ U\ (\mathfrak{I} \wedge (S \mid R)))$
After Q until R in $\mathfrak{I}$-scope:	$G\,((\mathfrak{I} \rightarrow Q \wedge \neg R) \rightarrow (\mathfrak{I} \rightarrow \neg P)\ W\ (\mathfrak{I} \wedge (S \mid R)))$

Figure 16. Examples of Patterns with new Combined Scopes



```
#define p ?
#define i ?

/*
 * Formula As Typed: <>  (p && i)
 * The Never Claim Below Corresponds
 * To The Negated Formula !(<> (p && i))
 * (formalizing violations of the
 * original)
 */

never {    /*  !(<>  (p && i))  */
accept_init:
T0_init:
   if
   :: (((!((i)) || !((p)))) -> goto T0_init
   fi;
}
```

```
#define p ?
#define i ?
#define r ?
/*
 * Formula As Typed: (i ->  ! r) U ((i && p
 * &&  ! r)  ||  (i ->  ! r))
 * The Never Claim Below Corresponds
 * To The Negated Formula !((i ->  ! r) U ((i
 * && p &&  ! r)  ||  (i ->  ! r)))
 * (formalizing violations of the original)
 */
never { /*  !((i ->  ! r) U ((i && p &&  ! r)
        || (i ->  ! r)))" */
accept_init:
T0_init:
   if
   :: ((i) && (r)) -> goto T0_init
   :: ((i) && (r)) -> goto accept_all
   fi;
accept_all:
   skip
}
```

Figure 17. Existence Pattern Globally in $\mathfrak{I}$-scope, and Before R in $\mathfrak{I}$-scope

6.6 Summary

In this chapter, we solved the problem of property specification in LTL over subsets of states of interest. We introduced specialized operators which help in formulating LTL properties over arbitrary subset of states more intuitively and succinctly. The states of interest are designated by a propositional statement. Moreover, we showed the applicability of our solution to specify web related properties in the context of our framework for run-time verification of WAs. We also discussed how our solution enriches the scopes of the Specification Pattern System, and presented a formulation of the patterns of SPS using the new scopes. The results of this chapter are published in [35,36,37].

In the next chapter, we present the implementation of our approach using Spin model checker. We give an overview of the prototype tool developed for the analysis and modeling of WAs, and demonstrate the effectiveness of our approach using case studies.

Chapter 7

Implementation of the Approach using Spin

After presenting our formal approach to run-time verification of WAUTs, an implementation of this approach is essential as a proof of concept. In this Chapter, we present an implementation of our approach to run-time verification of WAUTs using Spin model checker. We discuss how to represent a browsing session of a WAUT in the form of communicating automata model using Promela, the input language of Spin, and how Spin verifies user defined properties. We also describe the workflow of our approach and the tool implementation for the analysis and modeling of a browsing session of a WAUT. Finally, we present an evaluation of the tool as well as case studies.

7.1 Model Checking of Web Applications

A model of communicating automata of a WAUT can be automatically inferred from a browsing session, and described in Promela. Such a modular representation greatly contributes to the reduction of the state explosion problem during the property verification. The reason is that Spin performs on-the-fly model checking which often yields the verification results before the complete construction of the global state graph resulting from the automata composition.

Consequently, the problem of model checking WAUT models in Spin, can be stated as follows:

1. Model the browsing session of the WAUT in Promela
2. Specify the properties of interest in LTL

3. Verify the Promela model of the WAUT against the LTL properties using the Spin simulator.

Next we describe the automata modeling of a browsing session using Promela's constructs namely processes, their behavior, communication, and data.

7.2 Modeling Browsing Sessions with Promela

In this section, we explain how to specify the system of communicating automata inferred from a browsing session of a WAUT as presented in Chapters 4 and 5, in Promela. However, we first present an overview of the basic components of Promela.

7.2.1 Promela Basic Components

Spin supports a state-based high level language to specify systems descriptions, called Promela (PROcess MEta LAnguage). Promela utilizes three basic components in modeling systems: asynchronous processes, message channels, and data objects [44]. Processes interact via message exchanging using channels, or shared data. Message channels are of two types: FIFO-buffered channels for asynchronous communication where the user defines the size of the channel, and zero-sized channels for synchronous communication, aka rendezvous. Data objects can be defined locally in a process, or globally for all the processes.

Promela uses syntactic constructs from several programming languages. Borrowed from C language, basic data types supported in Promela are *int*, *short*, *byte*, *bool*, and *bit*, as well as operators such as "= =" (equal), "!=" (not equal), "||" (logical or), and "&&" (logical and). The syntax of communication commands is inherited from Hoare's CSP language. The denotation for sending a message *msg* with optional parameters par_1, ..., par_n, over a channel *ch* is *ch*!*msg* (par_1, ..., par_n). The denotation of receiving a message *msg* over channel *ch* is *ch*?*msg* (par_1, ..., par_n). The syntax of conditional constructs and loops are based on Dijkstra's guarded commands as shown in Figure 18.

if	**do**
:: *guard*$_1$ -> *statement*$_1$	:: *guard*$_1$ -> *statement*$_1$
:: *guard*$_2$ -> *statement*$_2$	:: *guard*$_2$ -> *statement*$_2$
.	.
.	.
:: *guard* -> *statement*	:: *guard* -> *statement*
fi	**od**

Figure 18. Promela Conditional and Loop Statements

A guard may be a condition, a communication command, or both. If the condition of a given guard is True and the communication command is not blocked, then the guard is evaluated to True and the corresponding statement is executable. In asynchronous message passing, a send command is blocked if the channel is full, and a receive command is blocked if a channel is empty. In rendezvous, communication is blocked if one of the communicating processes is not ready to send or receive. To execute an **if** statement or a **do** loop, one of the guards, evaluated to True, is non-deterministically selected and the corresponding statement is executed.

7.2.2 Promela Model of Browsing Sessions

To describe the model inferred from a browsing session of a WAUT, we define a Promela system of communicating processes. Intuitively, each local session automaton (corresponding to a window/frame) is mapped to a single process. The communication between processes is realized through messages that represent requests with defined targets, over rendezvous channels. We present here the main features of the mapping of local browsing sessions into Promela:

- Each session automaton corresponds to one `proctype` element.
- State attributes are defined as local variables in each process. These attributes are initially zero. Two other variables are defined: `state`, a state identifier which designates a unique ID for each state, and `transient` to designate stable and transient states in case of WA with frames.

```
active proctype proc1_C2P_Main(){
```

```
byte nLinks,nForms,nFrames,script,status;
byte state;
byte transient = 0;
```

- States of the automaton are designated by a Label element in the corresponding process. The process is initially in its inactive state. So, the process starts by `Inactive` label where the state variables are set to zero.
- Transitions between states are represented as follows. For each state/label, each outgoing transition designates an atomic statement that comprises a `goto` statement in which the label of the destination state is named. The goto statement is preceded by variable assignments to set the attribute values of the destination state. The settings of variables for each state are encoded in macros as follows.

```
#define INACTV d_step {
nLinks=0;nForms=0;nFrames=0;script=0;;status=0;state=0;}
#define M1     d_step {
nLinks=58;nForms=1;nFrames=0;script=0;status=200;state=1
;}
```

- Transitions of a state make part of an `if` statement, where each transition can be executed non-deterministically.

```
Inactive:
 if
 :: atomic {  M1; goto Label1}
 fi;
Label1:
 if
 :: atomic {  M1; goto Label1}
 :: atomic {  M5; goto Label5}
 fi;
```

- Rendezvous events are modeled as sent and received messages over rendezvous channels. The message channel is of size zero and has three parameters that identify a message: a message identifier, the process sending the message, and process receiving the message.

```
chan comMsg = [0] of {byte, mtype, mtype};
comMsg!msg43(Tproc1_C2P_Main, Tproc2_frame2);
comMsg?msg43(Tproc1_C2P_Main, Tproc2_frame2);
```

Note that we use the `atomic` and `d_step` constructs to ensure the atomicity of the transitions between states. Also, in order to designate stable and transient states as described in Chapter 5, we define in each process a local variable, `transient`, which tracks browser triggered events in the process.

For a full Promela representation of a model of a WA, we refer the reader to the example presented in Appendix 2.

7.3 Spin based Analysis and Verification of Web Applications

We discuss in this section the workflow of our approach, and present the implementation of the approach using Spin model checker.

7.3.1 Workflow of the Approach

The approach for analysis and verification of WAs presented in this thesis is instantiated in the workflow presented in Figure 19. The implementation consists of a prototype tool for the analysis and modeling of WAs based on observed browsing sessions, aka execution traces.

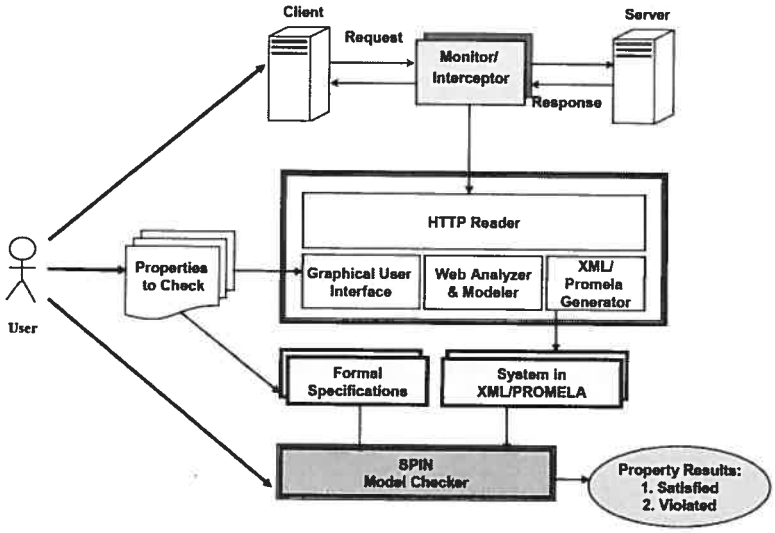


Figure 19. Workflow of the Approach

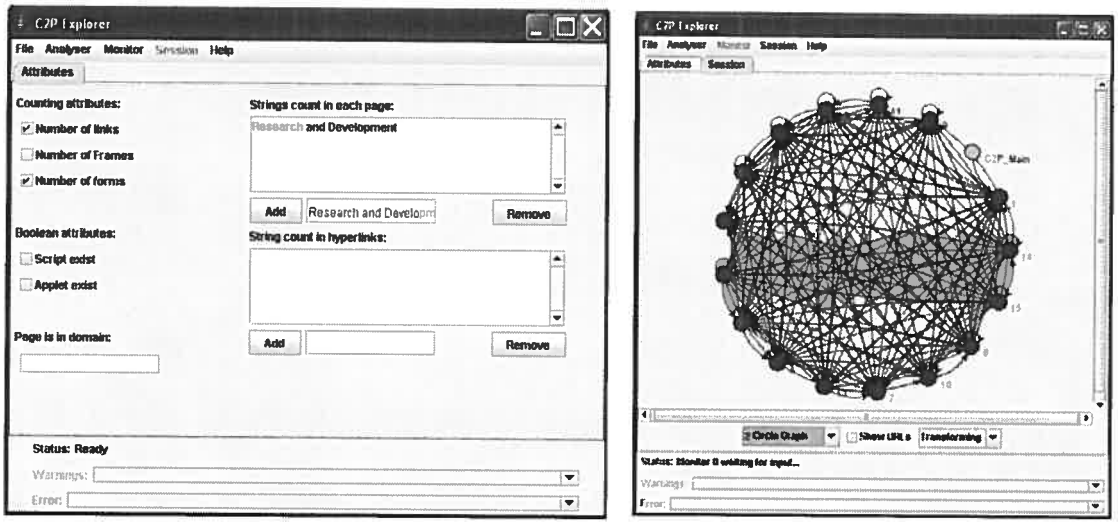
The main components of the implementation are:

- A Monitor/Interceptor. It intercepts HTTP requests and responses during the navigation of the WAUT performed by the user.
- A Web Application Analysis Tool that consists of the following components:
  - Graphical User Interface. It provides a set of attributes from which the user/tester selects the desired ones. These attributes, defined prior to the analysis process, are used in formulating the properties to verify on the application.
  - HTTP Reader. It reads HTTP requests and responses intercepted by the Monitor/Interceptor during the navigation of the WAUT performed by the user.
  - Web Analyzer and Modeler. It takes the intercepted traces as input, analyzes the data in real time (online mode), and builds an internal data structure of the automata model of the browsing session.
  - XML/Promela Generator. It translates the model inferred by the Web Analyzed and Modeler to either Promela or XML-Promela.
- Spin Model Checker. It verifies the properties against the model and generates a counter example if a property is not satisfied.

7.3.2 Web Application Analysis Tool

The WA analysis tool, whose screenshots are shown in Figure 20, is implemented in Java using the Eclipse environment, and has the following features:

- *Property based attribute selection*: through the graphical user interface of the tool, a number of predefined web related attributes that characterize WAs are provided (see Figure 20 (a) where the list is given). The user selected attributes are evaluated in each visited page and are reflected in the automata model. For the time being, the range of attributes allowed in the tool includes integers (such as number of occurrences of certain strings in pages, or depth of frame nesting in windows) and Booleans (such as Script exists or not in web pages).
- *Execution interception and monitoring*: the tool intercepts requests and responses of a WAUT using an open source proxy [87]. The monitoring module can operate in two modes: online and offline mode. In online mode, the monitor reads the executions directly from the proxy server and feeds them to the analysis module. In the offline mode, the monitor registers an execution trace in a log file.
- *Analyzing execution traces and model generation*: the tool parses and analyzes the execution traces and evaluates the user defined attributes in each visited page. An internal data structure of the automata model of the WAUT is built. The model can be generated either in Promela language or XML-Promela.
- *Automata model visualization and statistical data*: the tool has a model visualization feature. The built model of a WAUT can be visualized in two different graphical modes as well as one textual model. In the graphical mode, which is based on existing Java graph libraries, both single automaton, and communicating automata models are visualized, which can be manipulated by the user. For instance, the user can zoom in/out, pick displayed states and drag them, visualize the content of each state, and optionally show/hide transition labels. Also, the tool provides numerical data about the model, namely the number of processes (automata), total number of states, total number of transitions, and the total number of the actual pages visited.



(a) (b)

Figure 20. Tool Screenshots: (a) Attribute Selection, (b) Automata Model Visualization

7.4 Case Studies

We illustrate the applicability of our results using several examples. In these examples, we demonstrate the effectiveness of our modeling approach by verifying intricate properties, such as reachability properties and distributed properties, which are generally difficult to verify. In the first case study, we present models for two WAs with multi-display, and verify properties which could be distributed among frames/windows and are impossible to check without the use of a model checker. In the second case study, we aim at verifying reachability properties. These properties, as explained in Chapters 3 and 4, cannot be checked directly in Spin; instead, they have to be negated, and the violation of their negations in the model of the WAUT, is a proof of the validity of the original properties.

7.4.1 Multi-Display Distributed Properties

We present two case studies of multi-display WAs to verify *distributed properties*, where a distributed property is that of several entities. We argue that these properties cannot be verified without the use of model checker.

7.4.1.1 Three Frames Example

This example is a web site designed and implemented for testing purposes. It aims at serving the checking of certain properties in multi-display WAs. The properties of interest are of the following type (pattern):

- *The combination of certain objects/strings should not occur on any WA page shown to the user.*

In case of multi-display WAs, this type of properties is non-trivial to check since the objects/strings in question could be distributed among the different frames/windows, and for their combination to occur, the user has to navigate the WA following a specific path. We believe that such type of properties cannot be detected without the exhaustive exploration of all the possible paths of the WAUT , which is possible with the use of a model checker.

The example is a web site with a single window and three frames. The web site contains a dozen of web pages. The instantiated property that we aim at verifying is as follows:

- *The combination of the words Montreal, underground, and fire, should not occur throughout the web site.*

While navigating the WA, clicking on all the links, the displayed pages did not show any violation to the stated property. We illustrate the navigation in Figure 21. The links are clicked sequentially in the leftmost frame, while the resulting pages are displayed in the main frame and the uppermost frame. The links clicked are as follows: History, Recent activities, Future plans, Pictures, and Show our motto.

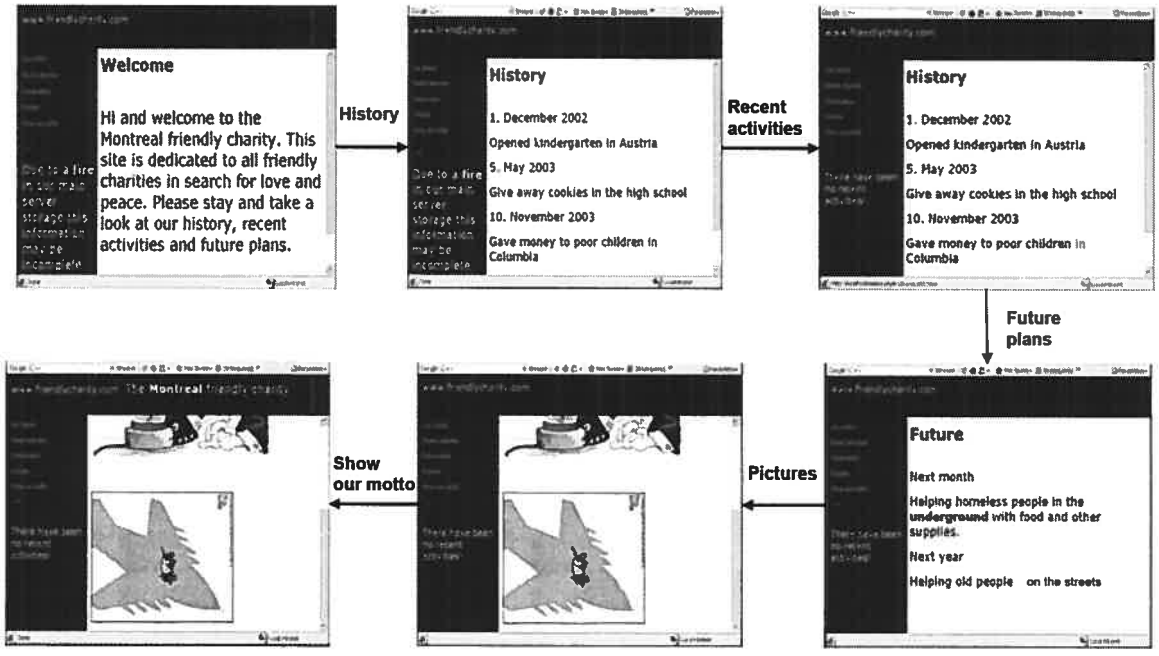


Figure 21. Sequence of Pages Resulting from Navigation of WA

It is clear from the illustrated navigation that, while the three words, Montreal, fire, and underground, do occur individually in various displays, their combination does not occur in any display.

To generate the automata model of the above described WA, we start by inserting the strings in question in the GUI attribute tab of the prototype tool, so that the occurrence of these strings would be counted in every visited page. Next, we explore the site while the tool intercepts the request/response pairs which were continuously analyzed to infer a communicating automata model. Figure 22 shows screenshots of the attribute tab and the communicating automata model produced by the tool. It shows four automata of the main window, and the three frames. For simplicity, we do not show the transition labels. Note that while exploring all the links of the web site in the sequential order, we were not able to reach a display where the property is violated.

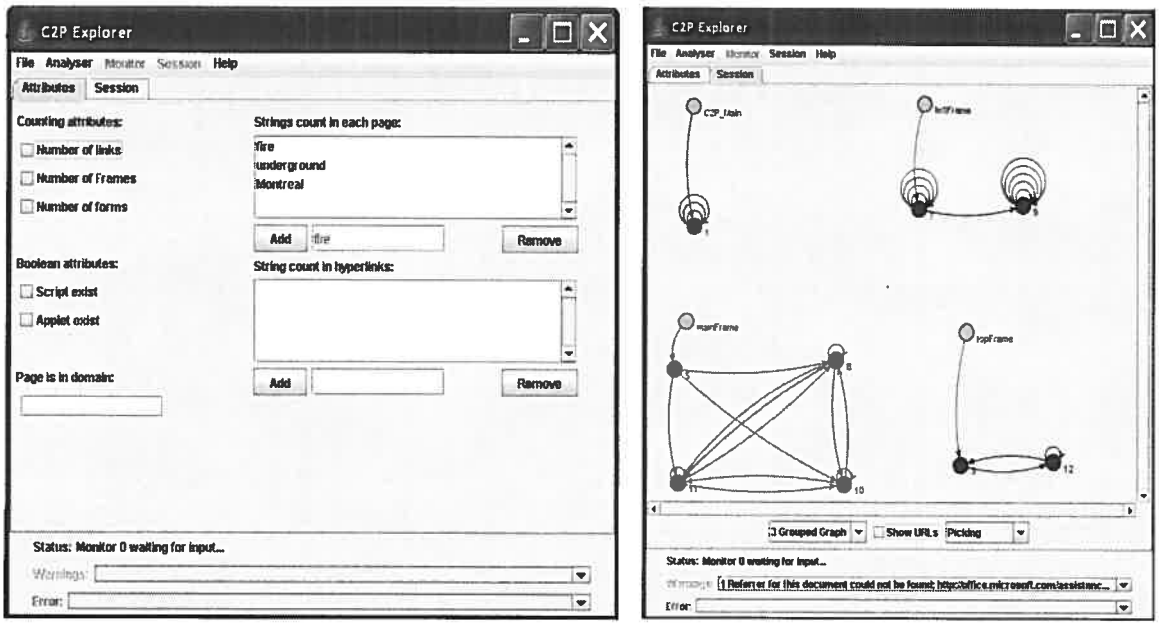


Figure 22. Frames Example: (a) Attribute Selection, (b) Automata Model Visualization

The tool generates the Promela code of the model. When running Spin verifier, the composition of the automata yields a global state graph with 277 states.

We specify the above stated property in LTL as follows:

[] ((*montreal* && *fire* && *underg*) || ((*montreal* && *fire*) || ((*montreal* && *underg*) || ((*underg* && *fire*) || *montreal* || *underg* || *fire*))))

such that *montreal*, *underg*, and *fire* are atomic proposition defined as follows:

- *montreal* designates the proposition that the number of occurrences of the string "Montreal" in all the three frames is equal to zero: (*proc2_topFrame:cntPageStr2* + *proc3_leftFrame:cntPageStr2* + *proc4_mainFrame:cntPageStr2*)= = 0
- *underg* designates the proposition that the number of occurrences of the string "underground" in all the three frames is equal to zero: (*proc2_topFrame:cntPageStr0* + *proc3_leftFrame:cntPageStr0* + *proc4_mainFrame:cntPageStr0*)= = 0

- *fire* designates the proposition that the number of occurrences of the string "fire" in all the three frames is equal to zero: (*proc2_topFrame:cntPageStr1* + *proc3_leftFrame:cntPageStr1* + *proc4_mainFrame:cntPageStr1*)= = 0

When verified in Spin, the property is found to be violated. Spin produces a counter example that indicates the path in the global state graph that violates the property. Figure 23 shows the Message Sequence Chart (MSC) of the counter example.

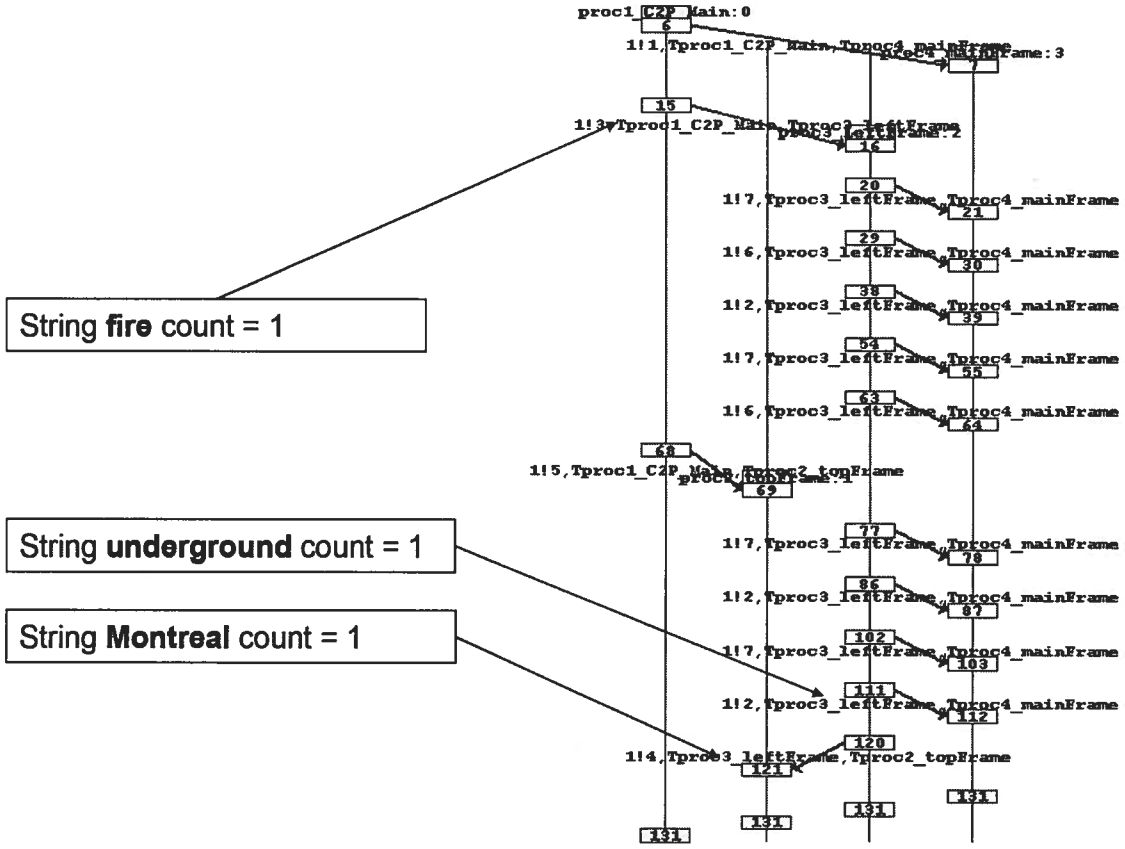


Figure 23. Counter Example Message Sequence Chart of Property

The counter example produced by Spin indicates the sequence of links to be clicked in order to reach a display of the WA where the combination of the three words occurs. As explained earlier, this was not possible by following the user's actions. However, due to our modeling technique of representing the frames as communicating automata, Spin was able to compute all the possible executions and paths that exist in the WA, and thus reporting the violation of the property. Figure 24 illustrates the navigation of the WA manually

reproduced from the counter example. The sequence of links clicked in this case is as follows: Future plans, and Show our motto.

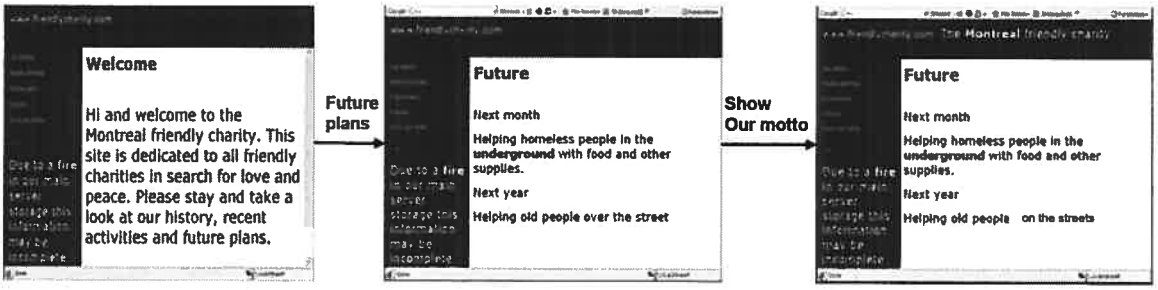


Figure 24. Counter Example Reproduced in the WA

7.4.1.2 Eclipse Consortium Web Application

This example consists of modeling a browsing session of the WA of the Eclipse Consortium, www.eclipse.org, an open platform for tool integration built by a community of tool providers. The corresponding web site shows framed pages and multiple windows.

We navigate the WA while intercepting the request/response pairs using the proxy server. The resulting browsing session contains 56 requests and 56 responses including those of images, icons, and imbedded objects, which are discarded by the tool. The intercepted pairs are continuously fed into the tool, which produces the model of the application in Promela. The model consists of 9 processes reflecting the fact that the application includes 7 frames and 2 windows in which 26 distinct web pages were visited. The frames are within the main browser's window and the second independent window has no frames within it. The processes in the Promela model could also be visualized as automata by Spin. As an illustration, we show the automata corresponding to the independent window in Figure 25, respectively. Note that the transition labels are not shown in the automaton for simplicity.

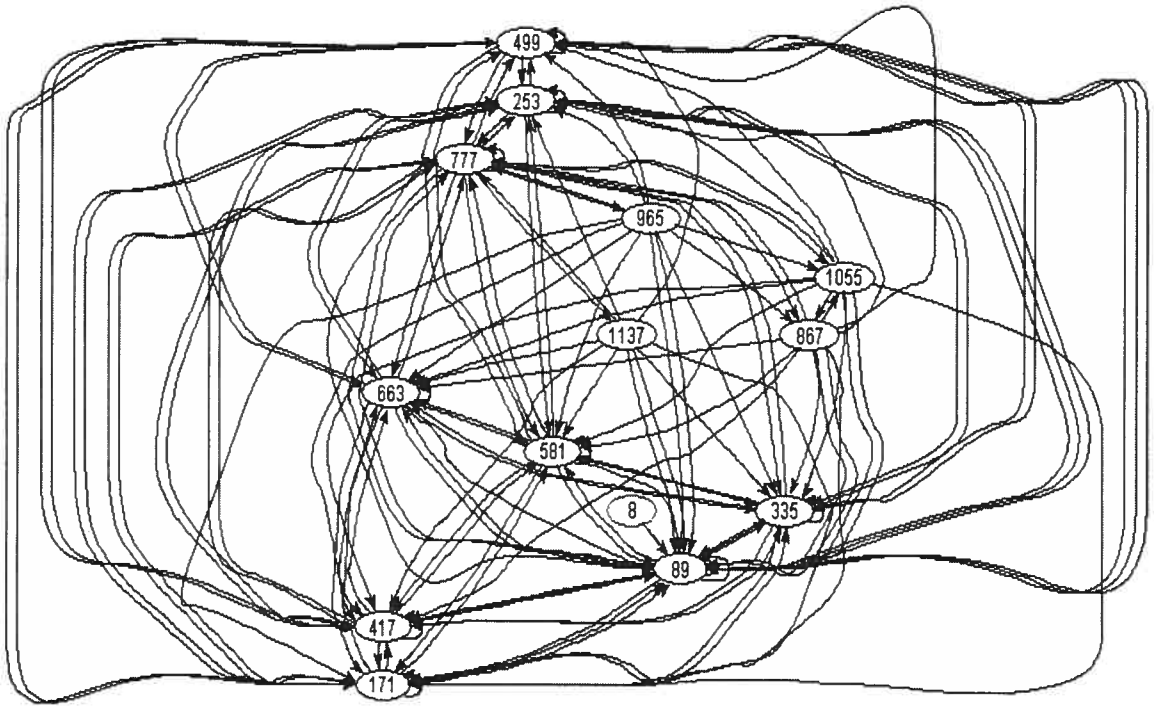


Figure 25. Automaton Corresponding to the Window *blank0*

Finally, Spin verifier performs an exhaustive simulation of the model leading to a global state graph of the behavior of the application. Figure 26 shows the results of the simulation. The global state graph of our model consists of 847 states and 9652 transitions (stored + matched).

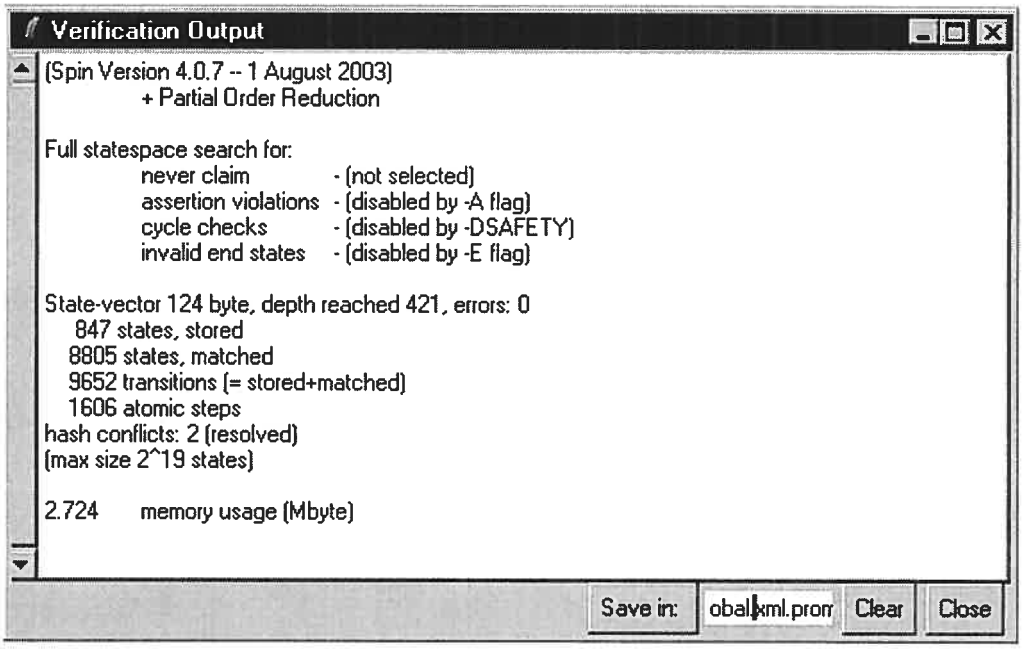


Figure 26. Output of Exhaustive Simulation in Spin

Properties

As an example, we verify three properties:

- *Number of links in concurrently displayed pages should be balanced.*

The first property requires that in the window *mainW*, and thus the frames within it, and the window *blank0*, the number of links in the displayed pages should be balanced, i.e., the difference between the number of links in the displayed pages in the two windows should not exceed a certain number which we fix to 15. This global property requires the exploration of all possible interleaving executions of the transitions of the automata of all the entities of the WAUT, namely, the main window *mainW* and the frames displayed with it, and the independent window *blank0*.

- *Frames having same names are not simultaneously active.*

The second property ensures the absence of a frame error where frames having the same name should not be displayed simultaneously; otherwise, clicking links with targets could result in erroneous display of pages.

- *Page X is reachable from Home page without going through page Y.*

The third property is a reachability property. When instantiated, the property requires that given three web pages, *program*, *conference*, and *home_main*, there exists at least a path where page *program* is reachable from page *home_main* without going through page *conference*. Note that pages *program* and *conference* are loaded in the independent window *blank0* and page *home_main* is loaded in the frame *main_0*.

Verification results

The first property should be checked only in stable states; otherwise, if it is checked in all the states of the model, the verification result would not be accurate. Therefore, we use the scope operator **`In`** to designate the scope of stable states of the model. In this case, we define the scope as the proposition *stable* which evaluates to the following expression: (*banner0:transient* = = 0 && *nav0:transient* = = 0 && main*0:transient* = = 0 && *nav0:transient* = = 0 && *banner5:transient* = = 0 && *home_nav5:transient* = = 0 && main*5:transient* = = 0 && *nav5:transient* = = 0), such that the variables are event tracking variable in each process. Then, the property is formulated in LTL as follows: [] (p || q) **`In`** *stable*,

where p and q are predicates such that p = nLinks2 - (nLinks1 + nLinks_banner0 + nLinks_nav0 + nLinks_main0) <= 15, and q = nLinks2 - (nLinks1 + nLinks_banner5 + nLinks_home_nav5 + nLinks_nav5 + nLinks_main5) <= 15.

Each variable in these predicates is associated to a process and represents a page attribute that counts the number of links in the page. *nLinks2* is associated to the process of *blank0*, *nLinks1* to the process of *mainW*, and the rest of the variables are associated to the processes of the frames. This property is not satisfied in the model and the verification result produces a counter example simulating a trace that violates the property.

The second property is formulated in LTL as follows: [] p,

where p = *duplicateFrames_mainW* = = 0 such that *duplicateFrames_mainW* is a Boolean variable that is set to true if two frames having same name are active simultaneously. This property holds in our model.

To verify the third property, we negate it, as explained in Chapter 4, and check if its negation does not hold in the model. The negation of the property becomes: on all paths from page *home_main* to page *program*, page *conference* is present. We use the web property pattern FGR9 from our repository WeSPaS [40] to formulate this property as follows: [] (*home_main* && ! *program* → ((! *program*) U ((*conference* && ! *program*) || [] (! *program*))))

The negation of the property holds in the model. In other words, there is no path from page *home_main* to page *program* where page *conference* is absent. Therefore, page *program* is never reachable from *home_main* without going through *conference* .Thus, the original property does not hold in the model.

Actually, this property stems from the fact that during the browsing of the eclipse WA, the page *program* was reached only after visiting the page *conference.* Therefore, we wanted to check whether this scenario is true on all possible navigation paths.

7.4.2 Reachability Properties

Reachability properties have to be checked in some paths of a given model. However, unlike branching temporal logics, LTL is a linear logic whose semantics do not include quantification over paths. Therefore, in LTL every formula is by default specified universally on all the paths of a given system model.

The following case study is a WA of a web site for world classical music WA called Beethoven (www.beethoven.com). This WA is single window which contains static and dynamic web pages generated by form submitting using search functions and shopping pages. We navigate 39 pages of the WA. The model produced by the prototype tool contains 26 states (since several pages could be mapped to the same state) and 183 transitions. Figure 27 shows the produced automata model.

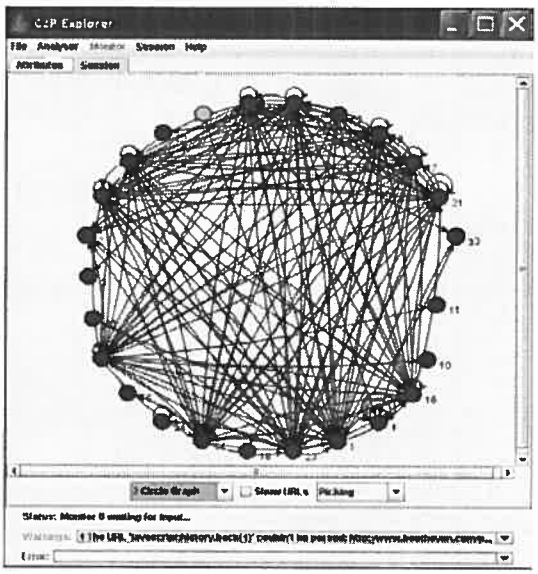


Figure 27. Automata Model of Beethoven WA

Properties

The properties to verify are as follows:

The first property ensures that the word music is not overly used throughout the WA by limiting the number of occurrences to six.

- *The number of occurrences of the string music in every page is strictly less than 6.*

The second property is an example of properties that ensure the reachability of a certain page from another page without always going back to home page.

- *page schedule is reachable from explore page without going through home page.*

The third property is a more generalized case of the second one. It ensures that a certain page is reachable from any other page without always going back to home page.

- *There is at least a page from which page schedule is reachable without going through the home page.*

The fourth property is similar to the second one, but the reachability is related to a group of pages having string music occurrence less than three.

- *There exists at least one page with occurrence of string music < 3 reachable from Explore page without going through the Home Page.*

Verification Results

To verify those properties, we negate the ones that are of reachability type. The results of verification are as follows:

The first property is formulated in LTL: G (*num_music* < 6), where *num_music* is an integer attribute that counts the number of string *music* in each page. This property is violated in the model extracted where Spin produces a counter example leading to a page which has the string count equals to 8.

In order to verify the second property, it should be negated as follows: *On all the paths from explore page to schedule page home page is present.* We use the web pattern FGR9 of our repository WeSPaS [40]: *ExistenceBetween* (*home*, *explore*, *schedule*). The corresponding LTL formula is:

> [] (*explore* && !*schedule* → ((!*schedule*) U ((*home* && !*schedule*) || [] (!*schedule*)))), such that *explore*, *schedule*, and *home* are atomic propositions which designate the corresponding pages.

When verifying this formula, Spin returns the valid result. This implies that the original property is invalid and that page Schedule is not reachable from Explore page without going back to home page.

The third property should also be negated as follows: *On all the paths to schedule page, the home page is present*. Similar to the second property, we use the web pattern FGR9: *ExistenceBetween* (*home*, (!*home* && !*schedule*), *schedule*). The LTL formula is:

> ([] ((!*home*) && !*schedule* → ((! *schedule*) U ((*home* && !*schedule*) || [] (!*schedule*))))), such that *schedule*, and *home* are atomic propositions which designate the corresponding pages.

The verification results of this property are similar to the ones of the second property. Therefore, the page schedule is not reachable from any other page without going back to home page.

The fourth property is also negated as follows: On all the paths from Explore page to pages where occurrence of string music < 3 the Home Page is present. Using the pattern FGR9: ExistenceBetween (home, explore, (num_music < 3 && !explore)), The LTL formula is:

> [] (*explore* && !((*num_music* < 3) && !*explore*) → ((! ((*num_music* < 3) && !*explore*)) U (*home* || [] (!(*minocc* && !*explore*))))), such that *explore*, and *home* are atomic propositions which designate the corresponding pages, and *num_music* the string count of the word music.

This property is found to be invalid in the model. Spin produces a counter example showing a trace from explore page state to a state of *num_music* = 2 without having home page state present. This implies that the original property is valid in the model and that there exists at least one path from explore page to page of music string counter less than three without visiting home page.

7.5 Evaluation and Discussion

After presenting case studies using the proposed implementation, we discuss in this section the scalability of the web analysis tool. We also sum up verification of properties in the case studies.

7.5.1 Tool Performance

To evaluate the prototype tool, we conducted several experiments using several WAs which include static and dynamic pages; eight of those applications are single window and five are multi-display applications.

Examples of the WAs that are analyzed are shown in Table 11.

Name	URL	Number of windows/frames
CRIM research center	www.crim.ca	1 window
University of Montreal	www.umontreal.ca	1 window
IEEE Computer Society Digital Library	www.computer.org	2 window
Concordia University	www.concordia.ca	4 windows
Eclipse Consortium	www.eclipse.org	2 windows – 7 frames
Berkeley University of California	www.berkeley.edu	1 window – 13 frames

Table 11. WAs Tested using the Prototype Tool.

The aim is to verify properties of WAUTs based on the user's browsing independently of any navigational aids, such as the back and forward buttons or the browser's history and bookmarks. The models of the first two WAUT are produced while the crawler automatically navigates through the applications. The rest are produced while manually navigating through the applications. An open source crawler [91] is used to exhaustively navigate through static applications, and which are single window, and thus the complete models of those applications are built by the prototype tool. Using the developed prototype tool, property related attributes are specified, execution traces of those WAUTs are then analyzed, and corresponding automata models are constructed. The tool then exports those models into Promela. Table 12 shows for each WA, characteristics of the corresponding model, and the average time to build the models. The number of global states and transitions for the WAUT, which are multi-display, are obtained using Spin verifier. Note that the time needed to produce the models depends on the following parameters: the server response time, the size of the memory of the operating machine, the size of web pages visited, and the number of attributes evaluated in those pages.

URL	Automata Model	Time
www.crim.ca	1 automaton, 294 states, 3879 transitions	2 min, 32 sec.
www.umontreal.ca	1 automaton, 267 states, 1657 transitions	4 min, 51 sec.
www.computer.org	2 automata, 36 global states, 213 transitions	1 min.
www.concordia.ca	4 automata, 2049 global states, 28204 transitions	1 min.
www.eclipse.org	9 automata, 847 global states, 9652 transitions	1 min, 20 sec.
www.berkeley.edu	14 automata, 19802 global states, 72801 transitions	2 min.

Table 12. Models Generated using the Prototype Tool.

The illustrated experiments show the efficiency and scalability of the prototype tool in infering models of relatively large WAs.

7.5.2 Property Verification

Various properties, verified on the models of the considered WAUTs, include reachability, frame related, security based, etc. These are taken from our Web Specification Pattern System and here we list few of them:

1. Non-functional:
   1.1. *Broken links and deadlocks are absent.*
   1.2. *Number of links in each display (single or multi) should not exceed a certain threshold (depends on size of application).*
   1.3. *Number of images in each display (single or multi) should not exceed a certain threshold (depends on size of application).*
   1.4. *Number of links in each display (single or multi) is balanced.*
   1.5. *Combinations of certain words/objects are absent.*
2. Functional:
   2.1. *Home page is reachable from every other page.*
   2.2. *Home page is reachable from every other page without going through a certain page X.*
   2.3. *Secure pages are not reachable without authentication process.*
   2.4. *In e-commerce applications, promotions of certain products are only present either on the Home page or on Shopping pages and, for each page, the number of promotions does not exceed two.*
   2.5. *Privacy policy page in e-commerce applications is reachable from every page.*
   2.6. *Secure pages are accessed a bounded number of times, and each time with authentication.*

The results of the verification of properties on the models of the WAUTs showed that many of the tested properties have been violated. The following (Table 13) shows the number of WAUTs which violated the properties described above. The results are categorized according to the type of the application, be it single window and multi-display.

	Property										
	1.1	**1.2**	**1.3**	**1.4**	**1.5**	**2.1**	**2.2**	**2.3**	**2.4**	**2.5**	**2.6**
Single window WA	2	2	1	2	0	0	3	0	1	1	0
Multi-display WA	1	5	5	5	3	0	4	1	n/a	n/a	n/a

Table 13. Property Violations in the Models of the WAUTs.

In summary, the WAs of small sizes, which often have a simple structure, had a limited number of property violations. Large WAs, which are usually developed using automated tools, exhibited also a small number of violations. The medium-size applications are the ones found to have the largest number of property violations, especially violations of reachability properties.

On the other hand, multi-display WAs especially with frames, irrespectively of their size or types of properties, were found to have more property violations than single window WAs, even with the simplest properties. Namely, most of the non-functional properties such as, *Combinations of certain words/objects are absent*, and *Number of links in each display (single or multi) is balanced*, though straightforward to check in single window WAs, were violated in most of the tested WAs with frames. This is due to the complex nature of those applications and the concurrent behavior they exhibit.

Also, we do not claim that all types of verified properties are preserved also in the actual WAs, as discussed in Chapter 4. Safety properties violated in the WAs models are said to be also violated in the actual WAs. However, this cannot be claimed for all liveness properties. Only violated liveness properties that has an infinite counter example can be said to be preserved in the original WAs.

7.6 Summary

In this chapter, we presented our implementation of the proposed approach using Spin model checker. We explained how our web models are mapped into Promela. We also presented the workflow of the approach as well as the main features and components of the prototype tool which analyzes execution traces of a WAUT and infers a communicating automata model. We demonstrated the applicability of our approach and implementation using several case studies where we verified intricate properties. Finally, we discussed the

scalability and performance of the prototype tool as well as some properties verified on models of WAs. Parts of this chapter are published in [39,41].

In the next chapter, we conclude the work of this thesis and discuss several directions of future work.

Chapter 8

Conclusions and Future Work

In a recent survey published by Queen's university, research work in the area of analysis models and methods in website verification and testing are evaluated and compared. The survey concludes that this filed of research is still in its infancy. Such a conclusion indicates that the field of formal approaches to analyzing and verifying WAs has many prospects to the contribution to the analysis and verification of WAs. In this chapter we conclude the work presented in this thesis and discuss various venues for potential future work.

8.1 Conclusions

The research work presented in this thesis includes three main contributions:

1. Methods for run-time verification of WA.
2. Extension of Linear Temporal Logic with scopes.
3. Building a system of web property patterns.

In the first contribution, we proposed a framework to formally model WAs for the purpose of run-time verification using model checking. We followed the dynamic (black-box based) approach, where we executed the application under test (using navigation and form filling), and observed the external behavior of the application by intercepting execution traces with a proxy server. We offered procedures for converting the observed behavior, which we call a browsing session, into a system of communicating automata. We elaborated a procedure for extending a communicating automata model of WAs where stability of states is encoded into auxiliary (event tracking) variables, which can also find useful applications outside of the WAs domain. Our approach was implemented using the model checker Spin. We presented the developed prototype tool that monitors the execution of a WAUT, analyzes the collected traces, and infers a communicating automata model

which can be mapped into Promela, or XML-Promela. We applied our approach to various case studies of WAs that contain static and dynamic pages, single window and multi-display, and illustrated the efficiency and scalability of the implementation.

In the second contribution, we addressed the problem of property specification in LTL over a subset of the states of a given system by introducing specialized operators in LTL using scopes. We believe that the new operators can be used in models that exhibit both infinite and finite behaviors and are useful in various application domains. The scope extended LTL was used to specify web properties in WAs that had multiple frames where those properties had to be verified in only stable/transient states. The scope extended LTL was also used in other properties where they had to be checked in only certain states of interest. We also suggested an enrichment of the Specification Pattern System of Dwyer et al. where we combined the SPS's scopes with our arbitrary scopes of states, thus doubling the number of the SPS's scopes.

In the third contribution, we developed a Web Specification Pattern System to alleviate the hurdle for web users and testers of formally specifying web properties. Based on various quality attributes that are necessary for the quality assurance of WAs, we surveyed a range of resources in industry and research that set numerous requirements to meet those quality attributes. We collected 120 requirements and built the WeSPaS, which is a repository of web patterns, categorized into functional and non-functional patterns. Each web pattern includes the description of a quality requirement, a unique ID reflecting its category, the involved page attributes, the corresponding LTL formula, and its source.

8.2 Future Work

We present some ideas of how the theoretical and practical results of this thesis can be further improved as well as new potential contributions.

8.2.1 Formal Model and Implementation

First, in the framework presented in this thesis, whenever the verified properties are invalid in the models of WAUTs, Spin produces a counter example that can be visualized either textually or in the form of MSC allowing the user to trace back to the state that has violated the verified property. However, the user has to manually trace back to the actual page of the WAUT where the property is violated which is a daunting task in case of large WAUTs. Therefore, as a future work, a methodology could be investigated and implemented to decipher the Spin verification results and map them back to the actual WAUT. More precisely, the counter example trace produced by Spin verifier can be reconstructed into a trace of hyperlinks or requests which can be played back using an instrumented version of an open source browser.

Second, the developed prototype tool is limited to a range of attributes offered to the user to choose from. Also, the user has to manually specify LTL related properties to be verified using Spin. In the future, further dynamic means can be developed in the tool to accept any attribute the user needs to evaluate in visited pages, such as the use of XPath expressions. This would allow the user to dynamically choose the attributes of interest in each page in the form of an XML-like expression that specifies the related HTML tags as well as the related content. On the other hand, currently, to navigate pages with forms, the user intervention is needed to manually populate the forms. As a future improvement of the tool, a partial automation of populating forms can be added to the tool where forms with option controls, and lists can be analyzed.

8.2.2 LTL Extensions with Scopes

Possible future work in the field of extending LTL with scopes would be extending our ideas toward temporal scopes and studying properties of the resulting logics. However, this necessitates performing more case studies to justify the extension toward temporal scopes. Possibly, this work could contribute to the improvement and elucidation of existing

specification patterns, e.g., in relation to mixed open/closed delimiters, mixed state/event properties etc.

Allowing temporal scopes introduces several issues that need to be dealt with. First, we intend to develop algorithms and tools for translation of formulas with `In` operator into automata. Another option is to develop a tool that translates any given LTL property that includes scope operators into standard LTL and applies optimization and simplification rules on the resulting LTL formula. While transformation into automata could be more practical and convenient for users of tools that support automata properties/observers, such as Spin never claims or Object Geode Goal, there are tools, such as NuSMV, that support LTL, but not observers. Second, using temporal scopes, we intend to designate the states of interest with a temporal formula. This can save us from introducing designated predicates and variables distinguishing states in scope from the others, e.g., transient and stable states of the system.

8.2.3 System of Web Specification Patterns

Our developed system of web specification patterns is not yet implemented. As a future development, a tool support can be developed and integrated with our web analysis prototype tool. This tool support would allow the user to easily browse through the patterns, choose a particular pattern, and use its LTL formula for model checking. This opens the door to another direction of making the WeSPaS a public repository where other researchers and specialists can present their contributions.

8.2.4 Addressing Emerging Web Technologies

The web is rapidly evolving and new technologies are emerging. As a future work of this thesis, we foresee the extension of our approach and framework to address web services composition, and Ajax technology.

Web services technologies have recently emerged as a standard to integrate disparate applications and systems promoting dynamic interoperability of highly distributed and heterogeneous web-hosted services. We propose as future work to extend our

framework and prototype tool to fully automate the run-time verification of business properties of web service composition.

On the other hand, Ajax is a recent promising technology that has been employed by a few enterprises such as Google. Ajax stands for Asynchronous Java + XML; it is essentially a generic technology (model) to update elements of Web application scripts, rather than the whole page, based on asynchronous java script generated requests and server responses. Our framework could be extended to address WAs that use Ajax based technology.

Bibliography

1. Alalfi M, Cordy JR, and Dean TR. *A Survey of Analysis Models and Methods in Website Verification and Testing*. Technical Report [2007-532], School of Computing, Queen's University. Kingston, Ontario, Canada, February 2007.
2. Andrews A, Offutt J, Alexander R. Testing Web Applications by Modeling with FSMs. *Software Systems and Modeling*, July 2005, 4(3):326-345.
3. Artho C, Schuppan V, Biere A, Eugster P, Baur M, Zweimüller B. JNuke: Efficient Dynamic Analysis for Java. *16th International Conference on Computer Aided Verification*, Boston, USA, July 2004.
4. Barringer H, Goldberg A, Havelund K, Sen K. *Eagle Does Space Efficient LTL Monitoring*. Technical Report, CSPP-25, University of Manchester, Department of Computer Science, October 2003.
5. Becker SA, Hevner AR. A White Box Analysis of Concurrent System Designs. *10th Annual International Phoenix Conference on Computers and Communications*, p. 332-338, Scottsdale, AZ, USA, 1991.
6. Beer I, Ben-David S, Eisner C. The Temporal Logic Sugar. *13th Int. Conference on Computer Aided Verification*, LNCS, 2102: 363-367, Paris, France, July 18-23, 2001.
7. Benedikt M, Freire J, Godefroid P. VeriWeb: Automatically Testing Dynamic Web Sites. *11th International World Wide Web Conference*, Hawai, U.S.A, 7-11 May 2002.
8. Berard B, Bidoit M, Finkel A, Laroussinie F, Petit A, Petrucci L, Schnoebelen P, McKenzie P. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer-Verlag, Berlin 2001.
9. Boldyreff C. and Kewish R. Reverse Engineering to Achieve Maintainable WWW Sites. *Proc. of the 8th Working Conference on Reverse Engineering*, pp. 249 – 257. Stuttgart, Germany, October 2001.
10. Boroday S, Petrenko A, Sing J, Hallal H. Dynamic Analysis of Java Applications for Multi Threaded Anti Patterns. *3rd International Workshops on Dynamics Analysis*, St-Louis, MI, USA, May 17, 2005.

11. Chaki S, Clarke EM, Ouaknine J, Sharygina N, Sinha N. State/Event-based Software Model Checking. *4th Int. Conference on Integrated Formal Methods*, LNCS, v. 2999, Canterbury, England, April 2004.
12. Chechik M, Paun D. Events in Property Patterns. *6th Int. SPIN Workshop on Theoretical and Practical Aspects of SPIN Model Checking*, LNCS, 1680: 154-167, September 1999.
13. Chechik M, Devereux B, Easterbrook S, and Gurfinkel A. Multi-valued Symbolic Model-Checking. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(4): 371 – 408, October 2003.
14. Cheng AMK. *Real-time systems: scheduling, analysis, and verification*, Wiley-Interscience, 2002.
15. Clarke EM, Grumberg O, Long DE. Model checking and abstraction. *ACM transactions on programming languages and systems*, 16(5):1512-1542, 1994.
16. Clarke EM, Grumberg O, Peled DA. *Model Checking*. MIT Press, 2000.
17. Conallen J. Modeling Web Application Architectures, with UML. *Communications of the ACM*, 2(10), October 1999.
18. Corbett JC, Dwyer MB, Hatcliff J, Laubach S, Pasareanu CS, Robby, Zheng H. Bandera: Extracting Finite-state Models from Java Source Code. *22nd International Conference on Software Engineering*, Limerick, Ireland, pp. 439-448, 2000.
19. Corbett JC, Dwyer MB, Hatcliff J, Robby. A Language Framework for Expressing Checkable Properties of Dynamic Software. *SPIN Software Model Checking Workshop*, LNCS, Springer-Verlag, Aug, 2000.
20. Corbett JC, Hatcliff MB, Laubach J, Pasareanu S, and Zheng C. Bandera: extracting finite-state models from Java source code. *Proceedings of the 22nd Int. Conference on Software Engineering*, pp. 439-448. Limerick, Ireland, June 2000.
21. De Alfaro L, Henziger TA, Mang FYC. MCWEB: A Model-Checking Tool for Web Site Debugging. *10th WWW Conference*, Hong Kong, 2001.
22. De Alfaro L. Model Checking the World Wide Web. *13th International Conference on Computer Aided Verification*, Paris, France, July 2001.

23. De Oliveira MCF, Turine MAS, and Masiero PC. A Statechart-Based Model for Hypermedia Applications. *ACM Transactions on Information Systems* 19(1): 28-52, 2001.

24. Di Sciascio E, Donini FM, Mongiello M, Piscitelli G. Web applications design and maintenance using Symbolic Model Checking. *7th European Conference On Software Maintenance And Reengineering*, Italy, March 26-28, 2003.

25. Di Sciascio E, Donini FM, Mongiello M, Totaro R, Castelluccia D. Design Verification of Web Applications Using Symbolic Model Checking. *5th Int. Conference on Web Engineering*, LNCS 3579, Sydney, Australia, 2005.

26. Dillon LK, Kutty G, Moser LE. A Graphical Interval Logic for Specifying Concurrent Systems. *ACM Transactions on Software Engineering and Methodology*, 3(2): 131-165, April 1994.

27. Dwyer M, Avrunin GS, Corbett JC. Patterns in Property Specifications for Finite-state Verification. *21st Int. Conference on Software Engineering*, May, 1999.

28. Dwyer M, Avrunin GS, Corbett JC. Property Specification Patterns for Finite-state Verification. *2nd Workshop on Formal Methods in Software Practice*, March, 1998.

29. Dwyer MB and Clarke LA. Data flow analysis for verifying properties of concurrent programs. *Proceedings of the 2nd ACM SIGSOFT symposium on Foundations of software engineering*, pp. 62-75, 1994.

30. Eisner C, Fisman D, Havlicek J, Lustig Y, McIsaac A, and Campenhout DV. Reasoning with Temporal Logic on Truncated Paths. *Proceedings of 15th Computer-Aided Verification (CAV 2003)*, Boulder, Colorado, USA, July 8-12, 2003.

31. Emerson E, Halpern J. 'sometimes' and 'not never' Revisited: on Branching versus Linear Temporal Logic. *Journal of the ACM*, 33(1): 151-178, January 1986.

32. Gamma E, Helm R, Johnson R, and Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

33. Graham S. *HTML Sourcebook, A Complete Guide to HTML 3.0*. John Wiley & Sons, Inc., 1996.

34. Hallal H, Petrenko A, Ulrich A, Boroday S. Using SDL Tools to Test Properties of Distributed Systems. *Formal Approches to Testing of Software*, Workshop of the Int. Conference on Concurrency Theory, Aalborg, Denmark, pp. 125-140, August 21-24, 2001.

35. Haydar M, Boroday S, Petrenko A, Sahraoui H. *Adding Propositional Scopes to Linear Temporal Logic*. Technical Report [CRIM 05/05-06], Centre de Recherche Informatique de Montreal, May 2005.

36. Haydar M, Boroday S, Petrenko P, Sahraoui H. Propositional Scopes in Linear Temporal Logic. *5th Int. Conference on Novelles Technologies de la Repartition*, Gatineau, Canada, August 30-September 1, 2005.

37. Haydar, M., Boroday, S., Petrenko, A. and H. Sahraoui. "Properties and Scopes in Web Model Checking" In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005).* Long Beach, California, USA, November 7-11, 2005.

38. Haydar M, Malak G, Sahraoui H, Petrenko A, and Boroday S. Anomaly Detection and Quality Evaluation of Web Applications. Accepted for publication in *Coral Calero, Mª Ángeles Moraga, and Mario Piattini, Handbook of Research on Web Information Systems Quality*. IGI Global, 2007.

39. Haydar M, Petrenko A, Boroday S, and Sahraoui H. A Formal Approach for Run-Time Verification of Web Applications using Scope-extended LTL. *Journal of Software Testing, Verification and Reliability*, Wiley, 2007 (under review).

40. Haydar M and Sahraoui H. *WeSPaS: A Specification Pattern System for Web Verification*. Technical Report [CRIM 07/10-17], Centre de Recherche Informatique de Montreal, July 2007.

41. Haydar M, Petrenko A, Sahraoui H. Formal Verification of Web Applications Modeled by Communicating Automata. *24th IFIP WG 6.1 IFIP Int. Conference on Formal Techniques for Networked and Distributed Systems*, LNCS, Spain, 3235:115-132, September 2004.

42. Haydar M. Formal Framework for Automated Analysis and Verification of Web-based Applications. *19th IEEE Int. Conference on Automated Software Engineering*, Linz, Austria, September 20-24, 2004.
43. Henriksen JG, and Thiagarajan PS. Dynamic Linear Time Temporal Logic. In *Annals of Pure and Applied logic*, 96(1-3): 187–207, 1999.
44. Holzmann GJ. *The Spin Model Checker, Primer and Reference Manual*. Addison-Wesley, 2003.
45. Huth MRA, Ryan MD. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2000.
46. Jard C, Jourdan GV, Jeron T, Rampon JX, and Irisa R. A general approach to trace-checking in distributed computing Systems. *Proceedings of the 14th Int. Conference on Distributed Computing Systems*, pp. 396-403, 1994.
47. Kan S. *Metrics and Models in Software Quality Engineering*, 2nd ed. Pearson, 2002.
48. Kupferman O, Vardi MY. Model Checking of Safety Properties. *Proceedings of 11th Computer-Aided Verification (CAV 2003),* Trento, Italy, July 6-10, 1999.
49. Koster M. Robots in the web: threat or treat? *ConneXions*, April, 1995, 9(4).
50. Krishnamurthy B, Rexford J. *Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement*. Addison-Wesley, 2001
51. Kroening D, Groce A, Clarke EM. Counterexample guided abstraction refinement via program execution, *6th International Conference on Formal Engineering Methods*, Springer, pp. 224-238, November 2004.
52. Kupferman O. Augmenting branching temporal logics with existential quantification over atomic propositions. *Journal of Logic and Computation*, 9(2): 135-147, April 1999.
53. Lamport L. Sometimes is sometimes 'not never'. *7th ACM Symposium on Principles of Programming Languages*, pp. 174-185, January 1980.
54. Latvala T. *On model checking safety properties*. Technical Report HUT-TCSA76, Helsinki University of Technology, 2002.

55. Larsen K, Larsson GF, Pettersson P, and Yi W. Efficient verification of real-time systems: Compact data structure and state-space reduction. *Proceedings of the 18th IEEE Real-Time Systems Symposium*: 14-24, 1997.

56. Leung K, Hui LCK, Yiu SM, and Tang RWM. Modeling Web Navigation by Statechart. *The 24th Int. Computer Software and Applications Conference (COMPSAC'00)*, pp. 41-47, Taipei, Taiwan, 2000.

57. Luotonen A. *Web Proxy Servers*. Prentice Hall Inc. Upper Saddle River, NJ, USA, 1998.

58. Malak, G. Évaluation de la Qualité des Applications Web: Approche Probabiliste. *Departement d'informatique et de recherches operationnelles*. Montreal, Universite de Montreal. Ph.D. Thesis (in progress), 2007.

59. Manna Z, Wolper P. Synthesis of Communicating Processes from Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 6(1): 68-93, January 1984.

60. Mansurov N, Probert R. Dynamic Scenario-Based Approach to Re-Engineering of Legacy Telecommunication Systems. *9th SDL Forum*, pp. 325-341, Montreal, 21–25 June 1999.

61. Nguyen HQ. *Testing applications on the Web: test planning for Internet-based systems*. J. Wiley, 2001.

62. Offutt J. Quality attributes of Web software applications. *IEEE Software,* 19(2): 25-32, 2002.

63. Paganelli L and Paterno F. A Tool for Creating Design Models from Web Site Code. *International Journal of Software Engineering and Knowledge Engineering* 13(2): 169-189, 2003.

64. Paulo FB, Masiero PC, and De Oliveira MCF. Hypercharts: extended statecharts to support hypermedia specification. *IEEE Transactions on Software Engineering* 25(1): 33-49, 1999.

65. Paulo FB, Turine MAS, De Oliveira MCF, and Masiero PC. XHMBS: a formal model to support hypermedia specification. *Proceedings of the 9th ACM conference on*

Hypertext and hypermedia: structure in hypermedia systems: links, objects, time and space: 161-170, 1998.

66. Paun D, Chechick M. Events in Linear-Time Properties. *4th IEEE Int. Symposium on Requirements Engineering*, June 1999.
67. Pereira A, Song M, Gorgulho G, Meira Jr. W, Campos S. A Formal Methodology to Specify E-commerce Systems. *4th International Conference on Formal Engineering Methods*, Shanghai, China, October 21-25, 2002.
68. Pertet S, Narasimhan P. *Causes of Failure in Web Applications.* Technical Report [CMU-PDL-05-109], Parallel Data Laboratory. Carnegie Mellon University, Pittsburgh, U.S.A, December, 2005.
69. Petrenko A, Boroday S, Groz R. Confirming Configurations in EFSM Testing. *IEEE Transactions on Software Engineering*, 30(1): 29-42, January 2004.
70. Plaza J. Logics of Public Communications.Proc. of *the 4th International Symposium on Methodologies for Intelligent Systems* (M.L. Emrich, M.S. Pfeifer, M. Hadzikadic, Z.W. Ras, eds.): pp. 201-216., North-Holland, 1989.
71. Pnueli, A. The Temporal Logic of Programs. *18th IEEE Symposium on Foundations of Computer Science*, pp46-57, 1977.
72. Ricca F, Tonella P. Analysis and Testing of Web Applications. *International Conference on Software Engineering*, Toronto, Ontario, Canada, pp. 25-34, May 12-19, 2001.
73. Ricca F, Tonella P. Web Site Analysis: Structure and Evolution. *International Conference on Software Maintenance*, San Jose, California, USA, pp. 76-86, October 11-14, 2000.
74. Smith MH, Holzmann GJ, Etessami K. Events and Constraints: a Graphical Editor for Capturing Logic Properties of Programs. *5th Int. Symposium on Requirements Engineering*, August 2001.
75. Smith RL, Avrunin GS, Clarke LA, Osterweil LJ. PROPEL: an Approach Supporting Property Elucidation. *24th Int. Conference on Software Engineering*, pp. 11-21, Orlando, Florida, 2002.

76. Stotts PD, Cabarrus CR. Hyperdocuments as Automata: Verification of Trace-Based Browsing Properties by Model Checking. *ACM Transactions on Information Systems*, 16(1): 1-30, January 1998.

77. Stotts PD, Navon J. Model Checking CobWeb Protocols for Verification of HTML Frames Behavior. *11th WWW Conference*, Hawai, U.S.A., May 2002.

78. Systä T. *Static and Dynamic Reverse Engineering Techniques for Java Software Systems*. Ph.D. dissertation, Dept. of Computer and Information Sciences, University of Tampere, 2000.

79. Temesis - Qualité, et accessibilité des sites Internet. Opquast: Quality Best Practices for On-line Services. Accessed in February 2005, from http://en.opquast.com/.

80. Tonella P, Ricca F. Dynamic Model Extraction and Statistical Analysis of Web Applications. *International Workshop on Web Site Evolution*, Montreal, Canada, pp. 43-52, October 2, 2002.

81. Vanderdonckt, J, Bouillon L, and Souchon N. Flexible Reverse Engineering of Web Pages with VAQUISTA. *Proceedings of the 8th IEEE Working Conference on Reverse Engineering*. Stuttgart, IEEE Press, pp. 241-248, October 2-5, 2001.

82. Vardi MY. Branching vs. Linear Time: Final Showdown. Proc. of *7th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems* (TACAS 2001), LNCS, 2031:1–22, Italy, April 2001.

83. West CH. An Automated Technique of Communication Protocols Validation. *IEEE Transactions on Communication*, 26: 1271-1275, 1978.

84. Wu Y, Offutt J. *Modeling and Testing Web-based Applications*. GMU ISE Technical Report, ISE-TR-02-08, November 2002.

85. A glossary of World Wide Web Terminology. Accessed in 2002, from http://personal.umich.edu/~zoe/Glossary.html.

86. Online Dictionary and Search Engine for Computer and Internet Technology. Accessed in 2002, from http://www.pcwebopedia.com/.

87. SOLEX, Web Application Testing with Eclipse, Accessed in 2005, from http://solex.sourceforge.net/.

88. The Specification Patterns System. Accessed in 2002, from http://patterns.projects.cis.ksu.edu/.
89. W3C World Wide Web Consortium. Accessed in 2002, from http://www.w3.org.
90. Web Site Test Tools and Site Management Tools. Accessed in 2002, from http://www.softwareqatest.com/qatweb1.html.
91. WebSPHINX: A Personal, Customizable Web Crawler. Accessed in 2005, from http://www.cs.cmu.edu/~rcm/websphinx.
92. IBM. Web Design Guidelines. Accessed in 2005, from http://www-306.ibm.com/ibm/easy/eou_ext.nsf/publish/611.
93. Bell Laboratories. VeriSoft. Accessed in 2002, from http://www1.bell-labs.com/project/verisoft/.
94. HTML TIDY. Accessed in 2002 from http://www.w3.org/People/Raggett/tidy/.
95. Microsoft Application Center Test. Accessed in 2002, from http://msdn.microsoft.com/library/default.asp?url=/library/enus/act/htm/actml_main.asp .
96. IEEE Computer Society. Software Reengineering Bibliography. Accessed in October 2002, from http://www.informatik.uni-stuttgart.de/ifi/ps/reengineering.

Appendix 1 – XML Model of a WA with Frames

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE model PUBLIC "-//W3C//DTD ModelCheckerML 1.0//EN"
                       "promela.dtd">
<!--Created Tue May 08 13:12:32 EDT 2007-->
<model>
    <directive name="M9" type="MACRO">d_step {
cntPageStr0=0;cntPageStr1=0;cntPageStr2=0;status=200;state=9;
}</directive>
    <directive name="M7" type="MACRO">d_step {
cntPageStr0=1;cntPageStr1=0;cntPageStr2=0;status=200;state=7;
}</directive>
    <directive name="M11" type="MACRO">d_step {
cntPageStr0=0;cntPageStr1=0;cntPageStr2=0;status=200;state=11;
}</directive>
    <directive name="M10" type="MACRO">d_step {
cntPageStr0=0;cntPageStr1=1;cntPageStr2=0;status=200;state=10;
}</directive>
    <directive name="M8" type="MACRO">d_step {
cntPageStr0=0;cntPageStr1=0;cntPageStr2=0;status=200;state=8;
}</directive>
    <directive name="M5" type="MACRO">d_step {
cntPageStr0=0;cntPageStr1=0;cntPageStr2=1;status=200;state=5;
}</directive>
    <directive name="M12" type="MACRO">d_step {
cntPageStr0=0;cntPageStr1=0;cntPageStr2=1;status=200;state=12;
}</directive>
    <directive name="M3" type="MACRO">d_step {
cntPageStr0=0;cntPageStr1=0;cntPageStr2=0;status=200;state=3;
}</directive>
    <directive name="M1" type="MACRO">d_step {
cntPageStr0=0;cntPageStr1=0;cntPageStr2=0;status=200;state=1;
}</directive>
    <directive name="INACTV" type="MACRO">d_step {
cntPageStr0=0;cntPageStr1=0;cntPageStr2=0;status=0;state=0; }</directive>
    <directive name="msg3" type="MACRO">1</directive>
    <directive name="msg6" type="MACRO">2</directive>
    <directive name="msg2" type="MACRO">3</directive>
    <directive name="msg4" type="MACRO">4</directive>
    <directive name="msg1" type="MACRO">5</directive>
    <directive name="msg7" type="MACRO">6</directive>
    <directive name="msg8" type="MACRO">7</directive>
    <declaration typename="MTYPE">
        <enumtype>
            <enumelement name="Tproc1_C2P_Main"/>
            <enumelement name="Tproc2_topFrame"/>
            <enumelement name="Tproc3_mainFrame"/>
            <enumelement name="Tproc4_leftFrame"/>
        </enumtype>
    </declaration>
    <declaration typename="BYTE" vis="HIDDEN">
```

```
        <var name="active2_topFrame"/>
        <var name="active3_mainFrame"/>
        <var name="active4_leftFrame"/>
    </declaration>
    <declaration typename="CHANNEL">
        <var name="comMsg">
            <messages number="0" synchronous="yes">
                <field type="BYTE"/>
                <field type="MTYPE"/>
                <field type="MTYPE"/>
            </messages>
        </var>
    </declaration>
    <declaration typename="CHANNEL">
        <var name="comIdle">
            <messages number="0" synchronous="yes">
                <field type="MTYPE"/>
            </messages>
        </var>
    </declaration>
    <proc instances="1" name="proc1_C2P_Main" type="PROCTYPE">
        <body>
            <declaration typename="BYTE">
                <var name="cntPageStr0"/>
                <var name="cntPageStr1"/>
                <var name="cntPageStr2"/>
                <var name="status"/>
                <var name="state"/>
            </declaration>
            <declaration typename="BYTE">
                <var name="transient" value="0"/>
            </declaration>
            <labeled_statement label_name="Inactive">
                <if>
                    <option>
                        <atomic>
                            <!--Transition 1 :
http://localhost/webexample1/index.html-->
                            <expression type="ASGN">
                                <left>
                                    <var name="transient"/>
                                </left>
                                <right>
                                    <const>3</const>
                                </right>
                            </expression>
                            <const>M1</const>
                            <goto label_name="Label1"/>
                        </atomic>
                    </option>
                </if>
            </labeled_statement>
            <labeled_statement label_name="Label1">
                <if>
```

```
                    <option>
                        <atomic>
                            <!--Transition 5 :
http://localhost/webexample1/frameMain.html-->
                            <send mtype="fifo_send">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <send_arguments>
                                    <const>msg2</const>
                                    <const>Tproc1_C2P_Main</const>
                                    <const>Tproc3_mainFrame</const>
                                </send_arguments>
                            </send>
                            <expression type="ASGN">
                                <left>
                                    <var name="transient"/>
                                </left>
                                <right>
                                    <expression type="SUBST">
                                    <left>
                                    <var name="transient"/>
                                    </left>
                                    <right>
                                    <const>1</const>
                                    </right>
                                    </expression>
                                </right>
                            </expression>
                            <const>M1</const>
                            <goto label_name="Label1"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 7 :
http://localhost/webexample1/frameLeft.html-->
                            <send mtype="fifo_send">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <send_arguments>
                                    <const>msg3</const>
                                    <const>Tproc1_C2P_Main</const>
                                    <const>Tproc4_leftFrame</const>
                                </send_arguments>
                            </send>
                            <expression type="ASGN">
                                <left>
                                    <var name="transient"/>
                                </left>
                                <right>
                                    <expression type="SUBST">
                                    <left>
```

```
                                        <var name="transient"/>
                                        </left>
                                        <right>
                                        <const>1</const>
                                        </right>
                                        </expression>
                                    </right>
                                </expression>
                                <const>M1</const>
                                <goto label_name="Label1"/>
                            </atomic>
                        </option>
                        <option>
                            <atomic>
                                <!--Transition 3 :
http://localhost/webexample1/frameTop.html-->
                                <send mtype="fifo_send">
                                    <channel>
                                        <var name="comMsg"/>
                                    </channel>
                                    <send_arguments>
                                        <const>msg1</const>
                                        <const>Tproc1_C2P_Main</const>
                                        <const>Tproc2_topFrame</const>
                                    </send_arguments>
                                </send>
                                <expression type="ASGN">
                                    <left>
                                        <var name="transient"/>
                                    </left>
                                    <right>
                                        <expression type="SUBST">
                                        <left>
                                        <var name="transient"/>
                                        </left>
                                        <right>
                                        <const>1</const>
                                        </right>
                                        </expression>
                                    </right>
                                </expression>
                                <const>M1</const>
                                <goto label_name="Label1"/>
                            </atomic>
                        </option>
                    </if>
                </labeled_statement>
            </body>
        </proc>
        <proc instances="1" name="proc2_topFrame" type="PROCTYPE">
            <body>
                <declaration typename="BYTE">
                    <var name="cntPageStr0"/>
                    <var name="cntPageStr1"/>
```

```
                <var name="cntPageStr2"/>
                <var name="status"/>
                <var name="state"/>
            </declaration>
            <declaration typename="BYTE">
                <var name="transient" value="0"/>
            </declaration>
            <labeled_statement label_name="Inactive">
                <if>
                    <option>
                        <atomic>
                            <!--IDLE-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comIdle"/>
                                </channel>
                                <receive_arguments>
                                    <const>Tproc2_topFrame</const>
                                </receive_arguments>
                            </receive>
                            <expression type="ASGN">
                                <left>
                                    <var name="active2_topFrame"/>
                                </left>
                                <right>
                                    <const>0</const>
                                </right>
                            </expression>
                            <const>INACTV</const>
                            <goto label_name="Inactive"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 3 :
http://localhost/webexample1/frameTop.html-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <receive_arguments>
                                    <const>msg1</const>
                                    <const>Tproc1_C2P_Main</const>
                                    <const>Tproc2_topFrame</const>
                                </receive_arguments>
                            </receive>
                            <expression type="ASGN">
                                <left>
                                    <var name="active2_topFrame"/>
                                </left>
                                <right>
                                    <const>1</const>
                                </right>
                            </expression>
```

```
                            <const>M3</const>
                            <goto label_name="Label3"/>
                        </atomic>
                    </option>
                </if>
            </labeled_statement>
            <labeled_statement label_name="Label3">
                <if>
                    <option>
                        <atomic>
                            <!--IDLE-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comIdle"/>
                                </channel>
                                <receive_arguments>
                                    <const>Tproc2_topFrame</const>
                                </receive_arguments>
                            </receive>
                            <expression type="ASGN">
                                <left>
                                    <var name="active2_topFrame"/>
                                </left>
                                <right>
                                    <const>0</const>
                                </right>
                            </expression>
                            <const>INACTV</const>
                            <goto label_name="Inactive"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 12 :
http://localhost/webexample1/frameMotto.html-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <receive_arguments>
                                    <const>msg8</const>
                                    <const>Tproc4_leftFrame</const>
                                    <const>Tproc2_topFrame</const>
                                </receive_arguments>
                            </receive>
                            <const>M12</const>
                            <goto label_name="Label12"/>
                        </atomic>
                    </option>
                </if>
            </labeled_statement>
            <labeled_statement label_name="Label12">
                <if>
                    <option>
```

```
                        <atomic>
                            <!--IDLE-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comIdle"/>
                                </channel>
                                <receive_arguments>
                                    <const>Tproc2_topFrame</const>
                                </receive_arguments>
                            </receive>
                            <expression type="ASGN">
                                <left>
                                    <var name="active2_topFrame"/>
                                </left>
                                <right>
                                    <const>0</const>
                                </right>
                            </expression>
                            <const>INACTV</const>
                            <goto label_name="Inactive"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 3 :
http://office.microsoft.com/assistance/slcache.aspx?Group=1&lidhelp=0409&
liduser=0C0C&lidui=0409&Ver=1-->
                            <const>M3</const>
                            <goto label_name="Label3"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 12 :
http://localhost/webexample1/frameMotto.html-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <receive_arguments>
                                    <const>msg8</const>
                                    <const>Tproc4_leftFrame</const>
                                    <const>Tproc2_topFrame</const>
                                </receive_arguments>
                            </receive>
                            <const>M12</const>
                            <goto label_name="Label12"/>
                        </atomic>
                    </option>
                </if>
            </labeled_statement>
        </body>
    </proc>
    <proc instances="1" name="proc3_mainFrame" type="PROCTYPE">
```

```
        <body>
            <declaration typename="BYTE">
                <var name="cntPageStr0"/>
                <var name="cntPageStr1"/>
                <var name="cntPageStr2"/>
                <var name="status"/>
                <var name="state"/>
            </declaration>
            <declaration typename="BYTE">
                <var name="transient" value="0"/>
            </declaration>
            <labeled_statement label_name="Inactive">
                <if>
                    <option>
                        <atomic>
                            <!--IDLE-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comIdle"/>
                                </channel>
                                <receive_arguments>
                                    <const>Tproc3_mainFrame</const>
                                </receive_arguments>
                            </receive>
                            <expression type="ASGN">
                                <left>
                                    <var name="active3_mainFrame"/>
                                </left>
                                <right>
                                    <const>0</const>
                                </right>
                            </expression>
                            <const>INACTV</const>
                            <goto label_name="Inactive"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 5 :
http://localhost/webexample1/frameMain.html-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <receive_arguments>
                                    <const>msg2</const>
                                    <const>Tproc1_C2P_Main</const>
                                    <const>Tproc3_mainFrame</const>
                                </receive_arguments>
                            </receive>
                            <expression type="ASGN">
                                <left>
                                    <var name="active3_mainFrame"/>
                                </left>
```

```
                                <right>
                                    <const>1</const>
                                </right>
                            </expression>
                            <const>M5</const>
                            <goto label_name="Label5"/>
                        </atomic>
                    </option>
                </if>
            </labeled_statement>
            <labeled_statement label_name="Label5">
                <if>
                    <option>
                        <atomic>
                            <!--IDLE-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comIdle"/>
                                </channel>
                                <receive_arguments>
                                    <const>Tproc3_mainFrame</const>
                                </receive_arguments>
                            </receive>
                            <expression type="ASGN">
                                <left>
                                    <var name="active3_mainFrame"/>
                                </left>
                                <right>
                                    <const>0</const>
                                </right>
                            </expression>
                            <const>INACTV</const>
                            <goto label_name="Inactive"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 10 :
http://localhost/webexample1/frameFuture.html-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <receive_arguments>
                                    <const>msg6</const>
                                    <const>Tproc4_leftFrame</const>
                                    <const>Tproc3_mainFrame</const>
                                </receive_arguments>
                            </receive>
                            <const>M10</const>
                            <goto label_name="Label10"/>
                        </atomic>
                    </option>
                    <option>
```

```
                        <atomic>
                            <!--Transition 11 :
http://localhost/webexample1/framePicture.html-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <receive_arguments>
                                    <const>msg7</const>
                                    <const>Tproc4_leftFrame</const>
                                    <const>Tproc3_mainFrame</const>
                                </receive_arguments>
                            </receive>
                            <const>M11</const>
                            <goto label_name="Label11"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 8 :
http://localhost/webexample1/frameHistory.html-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <receive_arguments>
                                    <const>msg4</const>
                                    <const>Tproc4_leftFrame</const>
                                    <const>Tproc3_mainFrame</const>
                                </receive_arguments>
                            </receive>
                            <const>M8</const>
                            <goto label_name="Label8"/>
                        </atomic>
                    </option>
                </if>
            </labeled_statement>
            <labeled_statement label_name="Label8">
                <if>
                    <option>
                        <atomic>
                            <!--IDLE-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comIdle"/>
                                </channel>
                                <receive_arguments>
                                    <const>Tproc3_mainFrame</const>
                                </receive_arguments>
                            </receive>
                            <expression type="ASGN">
                                <left>
                                    <var name="active3_mainFrame"/>
                                </left>
```

```
                                <right>
                                    <const>0</const>
                                </right>
                            </expression>
                            <const>INACTV</const>
                            <goto label_name="Inactive"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 8 :
http://localhost/webexample1/frameHistory.html-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <receive_arguments>
                                    <const>msg4</const>
                                    <const>Tproc4_leftFrame</const>
                                    <const>Tproc3_mainFrame</const>
                                </receive_arguments>
                            </receive>
                            <const>M8</const>
                            <goto label_name="Label8"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 11 :
http://localhost/webexample1/framePicture.html-->
                            <const>M11</const>
                            <goto label_name="Label11"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 10 :
http://localhost/webexample1/frameFuture.html-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <receive_arguments>
                                    <const>msg6</const>
                                    <const>Tproc4_leftFrame</const>
                                    <const>Tproc3_mainFrame</const>
                                </receive_arguments>
                            </receive>
                            <const>M10</const>
                            <goto label_name="Label10"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
```

```
                            <!--Transition 11 :
http://localhost/webexample1/framePicture.html-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <receive_arguments>
                                    <const>msg7</const>
                                    <const>Tproc4_leftFrame</const>
                                    <const>Tproc3_mainFrame</const>
                                </receive_arguments>
                            </receive>
                            <const>M11</const>
                            <goto label_name="Label11"/>
                        </atomic>
                    </option>
                </if>
            </labeled_statement>
            <labeled_statement label_name="Label10">
                <if>
                    <option>
                        <atomic>
                            <!--IDLE-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comIdle"/>
                                </channel>
                                <receive_arguments>
                                    <const>Tproc3_mainFrame</const>
                                </receive_arguments>
                            </receive>
                            <expression type="ASGN">
                                <left>
                                    <var name="active3_mainFrame"/>
                                </left>
                                <right>
                                    <const>0</const>
                                </right>
                            </expression>
                            <const>INACTV</const>
                            <goto label_name="Inactive"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 11 :
http://localhost/webexample1/framePicture.html-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <receive_arguments>
                                    <const>msg7</const>
                                    <const>Tproc4_leftFrame</const>
```

```
                                    <const>Tproc3_mainFrame</const>
                                </receive_arguments>
                            </receive>
                            <const>M11</const>
                            <goto label_name="Label11"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 10 :
http://localhost/webexample1/frameFuture.html-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <receive_arguments>
                                    <const>msg6</const>
                                    <const>Tproc4_leftFrame</const>
                                    <const>Tproc3_mainFrame</const>
                                </receive_arguments>
                            </receive>
                            <const>M10</const>
                            <goto label_name="Label10"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 8 :
http://localhost/webexample1/frameHistory.html-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <receive_arguments>
                                    <const>msg4</const>
                                    <const>Tproc4_leftFrame</const>
                                    <const>Tproc3_mainFrame</const>
                                </receive_arguments>
                            </receive>
                            <const>M8</const>
                            <goto label_name="Label8"/>
                        </atomic>
                    </option>
                </if>
            </labeled_statement>
            <labeled_statement label_name="Label11">
                <if>
                    <option>
                        <atomic>
                            <!--IDLE-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comIdle"/>
                                </channel>
```

```
                                <receive_arguments>
                                    <const>Tproc3_mainFrame</const>
                                </receive_arguments>
                            </receive>
                            <expression type="ASGN">
                                <left>
                                    <var name="active3_mainFrame"/>
                                </left>
                                <right>
                                    <const>0</const>
                                </right>
                            </expression>
                            <const>INACTV</const>
                            <goto label_name="Inactive"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 11 :
http://localhost/webexample1/framePicture.html-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <receive_arguments>
                                    <const>msg7</const>
                                    <const>Tproc4_leftFrame</const>
                                    <const>Tproc3_mainFrame</const>
                                </receive_arguments>
                            </receive>
                            <const>M11</const>
                            <goto label_name="Label11"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 8 :
http://localhost/webexample1/frameHistory.html-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <receive_arguments>
                                    <const>msg4</const>
                                    <const>Tproc4_leftFrame</const>
                                    <const>Tproc3_mainFrame</const>
                                </receive_arguments>
                            </receive>
                            <const>M8</const>
                            <goto label_name="Label8"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
```

```
                            <!--Transition 10 :
http://localhost/webexample1/frameFuture.html-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <receive_arguments>
                                    <const>msg6</const>
                                    <const>Tproc4_leftFrame</const>
                                    <const>Tproc3_mainFrame</const>
                                </receive_arguments>
                            </receive>
                            <const>M10</const>
                            <goto label_name="Label10"/>
                        </atomic>
                    </option>
                </if>
            </labeled_statement>
        </body>
    </proc>
    <proc instances="1" name="proc4_leftFrame" type="PROCTYPE">
        <body>
            <declaration typename="BYTE">
                <var name="cntPageStr0"/>
                <var name="cntPageStr1"/>
                <var name="cntPageStr2"/>
                <var name="status"/>
                <var name="state"/>
            </declaration>
            <declaration typename="BYTE">
                <var name="transient" value="0"/>
            </declaration>
            <labeled_statement label_name="Inactive">
                <if>
                    <option>
                        <atomic>
                            <!--IDLE-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comIdle"/>
                                </channel>
                                <receive_arguments>
                                    <const>Tproc4_leftFrame</const>
                                </receive_arguments>
                            </receive>
                            <expression type="ASGN">
                                <left>
                                    <var name="active4_leftFrame"/>
                                </left>
                                <right>
                                    <const>0</const>
                                </right>
                            </expression>
                            <const>INACTV</const>
```

```
                        <goto label_name="Inactive"/>
                    </atomic>
                </option>
                <option>
                    <atomic>
                        <!--Transition 7 :
http://localhost/webexample1/frameLeft.html-->
                        <receive rtype="normal">
                            <channel>
                                <var name="comMsg"/>
                            </channel>
                            <receive_arguments>
                                <const>msg3</const>
                                <const>Tproc1_C2P_Main</const>
                                <const>Tproc4_leftFrame</const>
                            </receive_arguments>
                        </receive>
                        <expression type="ASGN">
                            <left>
                                <var name="active4_leftFrame"/>
                            </left>
                            <right>
                                <const>1</const>
                            </right>
                        </expression>
                        <const>M7</const>
                        <goto label_name="Label7"/>
                    </atomic>
                </option>
            </if>
        </labeled_statement>
        <labeled_statement label_name="Label7">
            <if>
                <option>
                    <atomic>
                        <!--IDLE-->
                        <receive rtype="normal">
                            <channel>
                                <var name="comIdle"/>
                            </channel>
                            <receive_arguments>
                                <const>Tproc4_leftFrame</const>
                            </receive_arguments>
                        </receive>
                        <expression type="ASGN">
                            <left>
                                <var name="active4_leftFrame"/>
                            </left>
                            <right>
                                <const>0</const>
                            </right>
                        </expression>
                        <const>INACTV</const>
                        <goto label_name="Inactive"/>
```

```
                    </atomic>
                </option>
                <option>
                    <atomic>
                        <!--Transition 11 :
http://localhost/webexample1/framePicture.html-->
                        <send mtype="fifo_send">
                            <channel>
                                <var name="comMsg"/>
                            </channel>
                            <send_arguments>
                                <const>msg7</const>
                                <const>Tproc4_leftFrame</const>
                                <const>Tproc3_mainFrame</const>
                            </send_arguments>
                        </send>
                        <const>M7</const>
                        <goto label_name="Label7"/>
                    </atomic>
                </option>
                <option>
                    <atomic>
                        <!--Transition 10 :
http://localhost/webexample1/frameFuture.html-->
                        <send mtype="fifo_send">
                            <channel>
                                <var name="comMsg"/>
                            </channel>
                            <send_arguments>
                                <const>msg6</const>
                                <const>Tproc4_leftFrame</const>
                                <const>Tproc3_mainFrame</const>
                            </send_arguments>
                        </send>
                        <const>M7</const>
                        <goto label_name="Label7"/>
                    </atomic>
                </option>
                <option>
                    <atomic>
                        <!--Transition 8 :
http://localhost/webexample1/frameHistory.html-->
                        <send mtype="fifo_send">
                            <channel>
                                <var name="comMsg"/>
                            </channel>
                            <send_arguments>
                                <const>msg4</const>
                                <const>Tproc4_leftFrame</const>
                                <const>Tproc3_mainFrame</const>
                            </send_arguments>
                        </send>
                        <const>M7</const>
                        <goto label_name="Label7"/>
```

```
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 9 :
http://localhost/webexample1/frameLeft2.html-->
                            <const>M9</const>
                            <goto label_name="Label9"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 12 :
http://localhost/webexample1/frameMotto.html-->
                            <send mtype="fifo_send">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <send_arguments>
                                    <const>msg8</const>
                                    <const>Tproc4_leftFrame</const>
                                    <const>Tproc2_topFrame</const>
                                </send_arguments>
                            </send>
                            <const>M7</const>
                            <goto label_name="Label7"/>
                        </atomic>
                    </option>
                </if>
            </labeled_statement>
            <labeled_statement label_name="Label9">
                <if>
                    <option>
                        <atomic>
                            <!--IDLE-->
                            <receive rtype="normal">
                                <channel>
                                    <var name="comIdle"/>
                                </channel>
                                <receive_arguments>
                                    <const>Tproc4_leftFrame</const>
                                </receive_arguments>
                            </receive>
                            <expression type="ASGN">
                                <left>
                                    <var name="active4_leftFrame"/>
                                </left>
                                <right>
                                    <const>0</const>
                                </right>
                            </expression>
                            <const>INACTV</const>
                            <goto label_name="Inactive"/>
                        </atomic>
```

```
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 11 :
http://localhost/webexample1/framePicture.html-->
                            <send mtype="fifo_send">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <send_arguments>
                                    <const>msg7</const>
                                    <const>Tproc4_leftFrame</const>
                                    <const>Tproc3_mainFrame</const>
                                </send_arguments>
                            </send>
                            <const>M9</const>
                            <goto label_name="Label9"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 8 :
http://localhost/webexample1/frameHistory.html-->
                            <send mtype="fifo_send">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <send_arguments>
                                    <const>msg4</const>
                                    <const>Tproc4_leftFrame</const>
                                    <const>Tproc3_mainFrame</const>
                                </send_arguments>
                            </send>
                            <const>M9</const>
                            <goto label_name="Label9"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 9 :
http://localhost/webexample1/frameLeft2.html-->
                            <const>M9</const>
                            <goto label_name="Label9"/>
                        </atomic>
                    </option>
                    <option>
                        <atomic>
                            <!--Transition 10 :
http://localhost/webexample1/frameFuture.html-->
                            <send mtype="fifo_send">
                                <channel>
                                    <var name="comMsg"/>
                                </channel>
                                <send_arguments>
```

```
                                        <const>msg6</const>
                                        <const>Tproc4_leftFrame</const>
                                        <const>Tproc3_mainFrame</const>
                                    </send_arguments>
                                </send>
                                <const>M9</const>
                                <goto label_name="Label9"/>
                            </atomic>
                        </option>
                        <option>
                            <atomic>
                                <!--Transition 12 :
http://localhost/webexample1/frameMotto.html-->
                                <send mtype="fifo_send">
                                    <channel>
                                        <var name="comMsg"/>
                                    </channel>
                                    <send_arguments>
                                        <const>msg8</const>
                                        <const>Tproc4_leftFrame</const>
                                        <const>Tproc2_topFrame</const>
                                    </send_arguments>
                                </send>
                                <const>M9</const>
                                <goto label_name="Label9"/>
                            </atomic>
                        </option>
                    </if>
                </labeled_statement>
            </body>
        </proc>
</model>
```

Appendix 2 – Promela Model of WA with Frames

```
/* Created Tue May 08 13:12:09 EDT 2007*/

#define INACTV    d_step {
cntPageStr0=0;cntPageStr1=0;cntPageStr2=0;status=0;state=0; }
#define M1        d_step {
cntPageStr0=0;cntPageStr1=0;cntPageStr2=0;status=200;state=1; }
#define M3        d_step {
cntPageStr0=0;cntPageStr1=0;cntPageStr2=0;status=200;state=3; }
#define M12       d_step {
cntPageStr0=0;cntPageStr1=0;cntPageStr2=1;status=200;state=12; }
#define M5        d_step {
cntPageStr0=0;cntPageStr1=0;cntPageStr2=1;status=200;state=5; }
#define M8        d_step {
cntPageStr0=0;cntPageStr1=0;cntPageStr2=0;status=200;state=8; }
#define M10       d_step {
cntPageStr0=0;cntPageStr1=1;cntPageStr2=0;status=200;state=10; }
#define M11       d_step {
cntPageStr0=0;cntPageStr1=0;cntPageStr2=0;status=200;state=11; }
#define M7        d_step {
cntPageStr0=1;cntPageStr1=0;cntPageStr2=0;status=200;state=7; }
#define M9        d_step {
cntPageStr0=0;cntPageStr1=0;cntPageStr2=0;status=200;state=9; }

#define msg3      1
#define msg6      2
#define msg2      3
#define msg4      4
#define msg1      5
#define msg7      6
#define msg8      7

mtype = {Tproc1_C2P_Main, Tproc2_topFrame, Tproc3_mainFrame,
Tproc4_leftFrame};

hidden byte active2_topFrame, active3_mainFrame, active4_leftFrame;

chan comMsg = [0] of {byte, mtype, mtype};

chan comIdle = [0] of {mtype};


active proctype proc1_C2P_Main()
{
      byte cntPageStr0,cntPageStr1,cntPageStr2,status;
      byte state;
      byte transient = 0;
Inactive:
      if

      /*1 : http://localhost/webexample1/index.html*/
      :: atomic { transient=3; M1; goto Label1}
      fi;
Label1:
```

```
      if

      /*5 : http://localhost/webexample1/frameMain.html*/
      :: atomic { comMsg!msg2(Tproc1_C2P_Main,Tproc3_mainFrame);
transient=transient-1; M1; goto Label1}

      /*7 : http://localhost/webexample1/frameLeft.html*/
      :: atomic { comMsg!msg3(Tproc1_C2P_Main,Tproc4_leftFrame);
transient=transient-1; M1; goto Label1}

      /*3 : http://localhost/webexample1/frameTop.html*/
      :: atomic { comMsg!msg1(Tproc1_C2P_Main,Tproc2_topFrame);
transient=transient-1; M1; goto Label1}
      fi;
}


active proctype proc2_topFrame()
{
      byte cntPageStr0,cntPageStr1,cntPageStr2,status;
      byte state;
      byte transient = 0;
Inactive:
      if
      /*IDLE*/
      ::atomic { comIdle?Tproc2_topFrame; active2_topFrame=0; INACTV;
goto Inactive}

      /*3 : http://localhost/webexample1/frameTop.html*/
      :: atomic { comMsg?msg1(Tproc1_C2P_Main,Tproc2_topFrame);
active2_topFrame=1; M3; goto Label3}
      fi;
Label3:
      if
      /*IDLE*/
      ::atomic { comIdle?Tproc2_topFrame; active2_topFrame=0; INACTV;
goto Inactive}

      /*12 : http://localhost/webexample1/frameMotto.html*/
      :: atomic { comMsg?msg8(Tproc4_leftFrame,Tproc2_topFrame); M12;
goto Label12}
      fi;
Label12:
      if
      /*IDLE*/
      ::atomic { comIdle?Tproc2_topFrame; active2_topFrame=0; INACTV;
goto Inactive}

      /*3 :
http://office.microsoft.com/assistance/slcache.aspx?Group=1&lidhelp=040
9&liduser=0C0C&lidui=0409&Ver=1*/
      :: atomic {  M3; goto Label3}

      /*12 : http://localhost/webexample1/frameMotto.html*/
      :: atomic { comMsg?msg8(Tproc4_leftFrame,Tproc2_topFrame); M12;
goto Label12}
      fi;
```

```
}


active proctype proc3_mainFrame()
{
      byte cntPageStr0,cntPageStr1,cntPageStr2,status;
      byte state;
      byte transient = 0;
Inactive:
      if
      /*IDLE*/
      ::atomic { comIdle?Tproc3_mainFrame; active3_mainFrame=0; INACTV;
goto Inactive}

      /*5 : http://localhost/webexample1/frameMain.html*/
      :: atomic { comMsg?msg2(Tproc1_C2P_Main,Tproc3_mainFrame);
active3_mainFrame=1; M5; goto Label5}
      fi;
Label5:
      if
      /*IDLE*/
      ::atomic { comIdle?Tproc3_mainFrame; active3_mainFrame=0; INACTV;
goto Inactive}

      /*10 : http://localhost/webexample1/frameFuture.html*/
      :: atomic { comMsg?msg6(Tproc4_leftFrame,Tproc3_mainFrame); M10;
goto Label10}

      /*11 : http://localhost/webexample1/framePicture.html*/
      :: atomic { comMsg?msg7(Tproc4_leftFrame,Tproc3_mainFrame); M11;
goto Label11}

      /*8 : http://localhost/webexample1/frameHistory.html*/
      :: atomic { comMsg?msg4(Tproc4_leftFrame,Tproc3_mainFrame); M8;
goto Label8}
      fi;
Label8:
      if
      /*IDLE*/
      ::atomic { comIdle?Tproc3_mainFrame; active3_mainFrame=0; INACTV;
goto Inactive}

      /*8 : http://localhost/webexample1/frameHistory.html*/
      :: atomic { comMsg?msg4(Tproc4_leftFrame,Tproc3_mainFrame); M8;
goto Label8}

      /*11 : http://localhost/webexample1/framePicture.html*/
      :: atomic {  M11; goto Label11}

      /*10 : http://localhost/webexample1/frameFuture.html*/
      :: atomic { comMsg?msg6(Tproc4_leftFrame,Tproc3_mainFrame); M10;
goto Label10}

      /*11 : http://localhost/webexample1/framePicture.html*/
      :: atomic { comMsg?msg7(Tproc4_leftFrame,Tproc3_mainFrame); M11;
goto Label11}
      fi;
```

```
Label10:
      if
      /*IDLE*/
      ::atomic { comIdle?Tproc3_mainFrame; active3_mainFrame=0; INACTV;
goto Inactive}

      /*11 : http://localhost/webexample1/framePicture.html*/
      :: atomic { comMsg?msg7(Tproc4_leftFrame,Tproc3_mainFrame); M11;
goto Label11}

      /*10 : http://localhost/webexample1/frameFuture.html*/
      :: atomic { comMsg?msg6(Tproc4_leftFrame,Tproc3_mainFrame); M10;
goto Label10}

      /*8 : http://localhost/webexample1/frameHistory.html*/
      :: atomic { comMsg?msg4(Tproc4_leftFrame,Tproc3_mainFrame); M8;
goto Label8}
      fi;
Label11:
      if
      /*IDLE*/
      ::atomic { comIdle?Tproc3_mainFrame; active3_mainFrame=0; INACTV;
goto Inactive}

      /*11 : http://localhost/webexample1/framePicture.html*/
      :: atomic { comMsg?msg7(Tproc4_leftFrame,Tproc3_mainFrame); M11;
goto Label11}

      /*8 : http://localhost/webexample1/frameHistory.html*/
      :: atomic { comMsg?msg4(Tproc4_leftFrame,Tproc3_mainFrame); M8;
goto Label8}

      /*10 : http://localhost/webexample1/frameFuture.html*/
      :: atomic { comMsg?msg6(Tproc4_leftFrame,Tproc3_mainFrame); M10;
goto Label10}
      fi;
}


active proctype proc4_leftFrame()
{
      byte cntPageStr0,cntPageStr1,cntPageStr2,status;
      byte state;
      byte transient = 0;
Inactive:
      if
      /*IDLE*/
      ::atomic { comIdle?Tproc4_leftFrame; active4_leftFrame=0; INACTV;
goto Inactive}

      /*7 : http://localhost/webexample1/frameLeft.html*/
      :: atomic { comMsg?msg3(Tproc1_C2P_Main,Tproc4_leftFrame);
active4_leftFrame=1; M7; goto Label7}
      fi;
Label7:
      if
      /*IDLE*/
```

```
      ::atomic { comIdle?Tproc4_leftFrame; active4_leftFrame=0; INACTV;
goto Inactive}

      /*11 : http://localhost/webexample1/framePicture.html*/
      :: atomic { comMsg!msg7(Tproc4_leftFrame,Tproc3_mainFrame); M7;
goto Label7}

      /*10 : http://localhost/webexample1/frameFuture.html*/
      :: atomic { comMsg!msg6(Tproc4_leftFrame,Tproc3_mainFrame); M7;
goto Label7}

      /*8 : http://localhost/webexample1/frameHistory.html*/
      :: atomic { comMsg!msg4(Tproc4_leftFrame,Tproc3_mainFrame); M7;
goto Label7}

      /*9 : http://localhost/webexample1/frameLeft2.html*/
      :: atomic {  M9; goto Label9}

      /*12 : http://localhost/webexample1/frameMotto.html*/
      :: atomic { comMsg!msg8(Tproc4_leftFrame,Tproc2_topFrame); M7;
goto Label7}
      fi;
Label9:
      if
      /*IDLE*/
      ::atomic { comIdle?Tproc4_leftFrame; active4_leftFrame=0; INACTV;
goto Inactive}

      /*11 : http://localhost/webexample1/framePicture.html*/
      :: atomic { comMsg!msg7(Tproc4_leftFrame,Tproc3_mainFrame); M9;
goto Label9}

      /*8 : http://localhost/webexample1/frameHistory.html*/
      :: atomic { comMsg!msg4(Tproc4_leftFrame,Tproc3_mainFrame); M9;
goto Label9}

      /*9 : http://localhost/webexample1/frameLeft2.html*/
      :: atomic {  M9; goto Label9}

      /*10 : http://localhost/webexample1/frameFuture.html*/
      :: atomic { comMsg!msg6(Tproc4_leftFrame,Tproc3_mainFrame); M9;
goto Label9}

      /*12 : http://localhost/webexample1/frameMotto.html*/
      :: atomic { comMsg!msg8(Tproc4_leftFrame,Tproc2_topFrame); M9;
goto Label9}
      fi;
}
```