Evaluation of Web Applications Through Simulation of Web Designs

by

Pedro Alexandre Ferreira Teixeira Peixoto,

Thesis

Thesis submitted for the degree of

Doctor of Philosophy

University of Technology, Sydney

2006

CERTIFICATE OF AUTHORSHIP/ORIGINALITY

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledge within the text.

I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Signature of Candidate

Acknowledgments

I would like to thank all the people that helped and supported me during the last years, without whom this work would not have been possible.

I would firstly like to thank my supervisors, Dr. KK Fung and Prof. David Lowe, for their advice, feedback, and continuous support throughout these last years. The completion of this thesis would not have been possible without their help.

I would like to thank the Portuguese Government, in particular to the 'Fundacao para a Ciencia e Tecnologia (FCT)', which generously offered me a Ph.D. scholarship to study at UTS. My research would not have been possible without their support.

Thank you also to Pat Skinner for proofreading the thesis.

I would like to thank all my friends and flatmates in Australia who always supported me through all the difficult times. Their friendship made me feel that I was not alone. I would also like to thank all my friends in Portugal, who never let me feel I was on the other side of their world.

I would like to thank my mother, father, sister, and remaining family, for their love and understanding during this long journey. Lastly, I would like to dedicate this work to Beatriz, my newborn niece, for whom the world has more than one explanation and life is an ongoing thesis.

Contents

List of Figures

List of Tables

Glossary

ACM – Association for Computing Machinery

Functional requirement – A description of a function a system or system component must be able to perform (*IEEE standard glossary of software engineering terminology – 610.12-1990* 1990)

Functional Content – Measure used in this research for the evaluation of each treatment and defined as the observable factors a treatment provides and their structure and value

Functional Information – Measure used in this research for the evaluation of each treatment and defined as the level of contribution of the observable factors for the evaluation of functional requirements

HDL – Hardware Description Language (Garzotto et al. 1991*b*)

HDM – The Hypertext Design Model (Garzotto et al. 1991*b*)

IEEE – Institute of Electrical and Electronics Engineers

OOHDM – The Object-Oriented Hypermedia Design Model (Schwabe and Rossi 1995*a*)

Prototyping – A development technique that uses a preliminary version of part or all of the software product for user feedback, feasibility evaluation, or other issues supporting the development process (*IEEE standard glossary of software engineering terminology – 610.12-1990* 1990)

RMM – The Relationship Management Methodology (Isakowitz et al. 1995)

Simulation – "(1) A model that behaves or operates like a given system when provided with a set of controlled inputs. (2) The process of developing or using a model as in (1)." (*IEEE standard glossary of software engineering terminology – 610.12-1990* 1990)

Software Development Cycle – The period of time ranging from the start of the software product project to its delivery (*IEEE standard glossary of software engineering terminology – 610.12-1990* 1990)

Software Life Cycle – The period of time ranging from the conceptualization of the software product project to when it is no longer available (*IEEE standard glossary of software engineering terminology – 610.12-1990* 1990)

SRS – System Requirements Specification

UIM – The proposed User Interaction Model

UML – The Unified modelling Language (OMG 2005)

Use cases – capture *who* (actor) does *what* (interaction) with the system, for what *purpose* (goal), without dealing with system internals. A complete set of use cases specifies all the different ways to use the system, and therefore defines all behaviour required of the system, bounding the scope of the system (Malan and Bredemeyer 2001)

Validation – Confirmation by examination and provisions of objective evidence that the particular requirements for a specific intended use are fulfilled (*IEEE standard for software verification and validation – 1012-1998* 1998)

Verification – Confirmation by examination and provisions of objective evidence that specified requirements have been fulfilled (*IEEE standard for software verification and validation – 1012-1998* 1998)

VHDL – VHSIC Hardware Description Language (VHSIC being an acronym for Very High-speed Integrated Circuit) (*IEEE Standard VHDL Language Reference Manual – 1076* 2002)

WAD – A Web Application Design. A high-level description of how a system is organized and operates, usually using one or more Web Application Design Models. It shows the objects or object classes in a system and, where appropriate, the relationships between these entities (Sommerville 2004)

WADM – A Web Application Design Model. Allows the representation of the result of the design activity by providing a framework or language which can be used to compare and document the specifications of applications (Lowe and Hall 1999)

Web Engineering – Web engineering is the establishment and use of sound scientific, engineering and management principles and disciplined and systematic approaches to the successful development, deployment and maintenance of high quality Web-based systems and applications (Murugesan et al. 1999)

WebML – The Web Modeling Language (Ceri et al. 2000)

WDL – The developed Web-design Description Language

WSM – The proposed Web-design Simulation Model

Abstract

Evaluation of Web Applications Through Simulation of Web Designs

The development of Web applications continues to pose numerous difficulties for Web developers due to the inherent complexity of the projects. Although methodologies have been proposed to tackle the development of these projects, they are especially concerned with setting guidelines and defining tasks to better structure the design phase. For this purpose, several design models have been developed and used in the design of Web applications, providing a suitable level of abstraction and independence from a specific implementation. However, the other phases of the Software Development Cycle have not received the same level of attention from researchers. In particular, the test phase is lacking in theory and tools to effectively and efficiently verify the project requirements. Evaluation of the functional requirements of a system under development is commonly done by its partial implementation and test. This requires the development and coding of a prototype of the system to be able to verify the design. Furthermore, this prototyping effort could be partially or totally in vain if tests find that the design does not meet the intended requirements.

This research argues that it is possible to simulate Web application design models for the verification of functional requirements. Furthermore, it claims that simulation is able to provide as much functional information as an implementation would. The research proposes a multi-layer Web-design Simulation Model, which was developed to enable simulation of Web application designs and takes into consideration developers' key design concerns. Furthermore, a Web-design Description Language was especially developed to provide meaningful simulation of design models. It borrows concepts from the hardware engineering field where simulation is extensively used for design verification. By performing simulation directly on the designs, the need for prototyping for functional evaluation is reduced or no longer necessary and verification of the requirements can be performed as soon as a design is available. This has the potential to contribute to a faster Software Development Cycle of Web applications.

To prove the feasibility of the simulation and the meaningfulness of its application, an experiment on a selected Web application design was conducted. This entailed a comparison between the implementation and simulation results for the functional requirements evaluation. The comparison was performed by assessing the functional content and information of the results that both methods provided. The comparison showed that, although both are suitable for verification of functional requirements, the proposed Simulation Model provides additional functional information and a more intuitive analysis for the evaluation of Web application designs.

Chapter 1

Introduction

The innovative and ubiquitous medium that the World Wide Web constitutes, provides Web developers with prodigious support for information sharing and Web applications development. However, along with the advent of the Internet a multitude of new types of development problems arose. Furthermore, the complexity that Web applications development has reached is far greater than was foreseen only a decade ago (Ginige and Murugesan 2001). It is argued that having the content so inextricably intertwined with the functionalities of the applications poses new challenges to developers (Pressman 2000).

For a certain time, and probably due in part to the immaturity of the field, an *ad hoc* development approach prevailed. A team would be assembled with all those involved contributing to the project with their heterogeneous knowledge, backgrounds, and skills (Lang and Fitzgerald 2005). The development frenzy and the short time-frames in the early years of Web application development projects have surely contributed to this approach, preventing more scientific planning. Web applications evolved from Web sites and differ from them in the ability a user has to alter the state of the business logic on the server (Conallen 2000, p. 10). Without suitable engineering-based methodologies, models and tools, the development of a

Web application strongly relied on the developers' innate ability and previous work experience to carry out the project. As Isakowitz et al. (1995) once noted, hypermedia design was "more of an art than a science". It worsened when new technologies were later introduced, enabling dynamic hypermedia, server and client-side scripts, back-office integration, e-commerce, and e-business. Clearly, the complexity of the new field could not be addressed by just the skills of the developers, and the adoption of a set of good practices was needed. As a consequence, a paradigm shift was advocated and various models, methodologies, and tools started to emerge, of which some examples are the Hypertext Design Model (Garzotto et al. 1991*b*, Garzotto et al. 1993, Esprit 1990), the Object-Oriented Hypermedia Design Model (Schwabe and Rossi 1995*a*, Schwabe et al. 1996, Rossi et al. 2001), and the Relationship Management Methodology (Isakowitz et al. 1995).

Models are extremely useful when requiring an abstract view of a system. They allow developers to focus on a specific aspect or from a particular perspective of the system, hiding complexities and breaking down the system into manageable parts (Johnson and Henderson 2002). Therefore, the development and adoption of models to tackle Web application complexities constitute no surprise. Similarly, development methodologies arose to tackle the complexity of software development projects. However, unlike models, they seek a more global perspective and better understanding of the entire development project, by setting tasks and guidelines that aid engineers throughout the project, ensuring high-quality systems.

In the development of software products developers follow a number of tasks that need to be performed if a high-quality end product is to be achieved. In the software engineering field, the period of time from starting to develop a software product till its delivery is called Software Development Cycle, which, if including the operation, maintenance and, sometimes, retirement phase, is called Software Life Cycle (*IEEE standard glossary of software engineering terminology – 610.12-*

1990 1990). By modelling the Software Development Cycle, the consistency and structure of each activity within the project can be supported. There are several valuable reasons for using Software Development Cycle models in a development project. Examples of these include: achieving a high quality product; controlling the project's schedule; and minimising project cost slips. These reasons alone make the adoption of a Software Development Cycle model indispensable in a large and complex software development project. A large number of models have been proposed to tackle software development. The two most well known of these are the Waterfall model (Royce 1970) proposed by Royce, and the Spiral model by Boehm (Boehm 1986). But whatever the adopted model, there are a number of phases that they all share and address. These phases investigate and describe what functionalities the final product should implement, how they will be implemented, and how they will be validated and verified. These phases entail and are usually referred to as: requirements analysis, design, coding, testing, system delivery, operation, and maintenance (Pfleeger 2001, p. 47).

To cope with the new challenges that Web application development has brought, a new branch of engineering emerged in the late 1990's – Web Engineering (Murugesan et al. 1999). It borrows concepts, methods and methodologies from the field of computer and software engineering and utilises them in the field of Web application development. Web engineering states that in order to achieve a high-quality Web application, the development cycle must be tailored to tackle the complexities which Web development poses, namely requirements elicitation and analysis, development methodologies, maintainability, scalability, testing and documentation (Ginige and Murugesan 2001, Ginige 2002*a*).

The development cycle of a Web application usually follows a mix of incremental and iterative phases (Pressman 2000). After the definition of the system's services, constraints and goals, during the requirements analysis and definition phase, a suitable design is elaborated which, it is thought, will address and meet all or at least a significant number of the client requirements. This design will lead to a partial implementation or prototype, which is thoroughly tested. The testing phase objectives are to verify and validate the intended functional requirements. Depending on the results of the testing phase, the development may go back to the analysis and design phase for further refinement, and a new development cycle begins. The decision to stop the development cycle and deliver the final product is taken based on a number of factors, such as a cost-benefit analysis of continuing the development, budget and time-to-delivery constraints.

Much research has been done in the analysis and design phases, which has led to a rich body of research literature. However, testing, which is one of the most important procedures to verify and validate the application, has not received the same level of attention by the research community. This is bewildering, since testing typically accounts for a substantial proportion of the total cost of developing a software product (Harrold 2000, Moriguchi 1996). In a recent survey carried out by the Business Internet Group of San Francisco it was found that a large number of federal Web sites in the US showed some sort of failure within the first 15 minutes of a typical user visit (BIGSF 2003). Furthermore, another survey conducted by the Cutter Consortium (Epner 2000) found that nearly half of the projects did not meet the intended functionalities.

When compared with other engineering development fields, Web applications developers lack suitable tools and sound methods from which to choose for the testing phase. One meaningful case of an engineering field where such tools and methods exist and are used is the hardware development field, in particular in the microchip design and development. One especially important method which is used for the verification of microchip designs is the use of simulation techniques. Simulation addresses the issue of whether the product being built is the right one and if it is being correctly designed. Furthermore, simulation is performed on the design representing the targeted hardware system without the need to build any physical device. Hardware description languages (HDLs) have brought a formal and systematic approach to the microchip design process, offering simulation and synthesis capabilities, thus reducing the complexity of management and development times.

Simulation is an intensively used technique in all fields of engineering. Its importance and contribution for a better understanding of a system is clearly stated by Shannon, who defines simulation as:

DEFINITION 1 *Simulation: "the process of designing a model of a real system and conducting experiments with this model for the purpose of understanding the behaviour of the system and/or evaluating various strategies for the operating of the system" (Shannon 1998).*

It is these *experiments*, *understanding* and *evaluation* of a model that give simulation its importance. Simulating Web application design models allows the evaluation of their functionalities and the assessment of different system designs to be performed. This, in turn, has the potential to improve design error detection, contributing to a decrease in development time and assuring a high-quality end product. Since the design phase is a necessary stage in a well-planned Web application development project, the ability to simulate the design model results in a double benefit: firstly, there is no need to partially implement the design (prototyping) to evaluate its functionalities; secondly, since design is one of the first phases to be carried out in a development project, design errors can be promptly detected. This research argues that simulation of Web applications design models is not only feasible, but also useful for functional requirements evaluation.

1.1 Focus and Purpose

Many Web applications have grown to be so large and complex that empirical and *ad-hoc* methods of Web development are no longer capable of delivering high-quality nor on-time development of Web applications. New testing models and tools should permit an easy and agile verification of the design requirements without the need for partial implementations, as is common practice in other fields of engineering, with the notable example of hardware development. Computer-assisted simulation and testing have been successfully used in hardware development for years, reassuring developers that the future system meets all the intended requirements.

There has been much research into the simulation of how Web applications react under a certain number of constraints, such as network loads, simultaneous user access, or page loading times. There has, however, been little consideration given to the utilisation of simulation for the functional evaluation of the design itself. As with the hardware development field, this would assist developers in observing how the future application reacts to stimuli in different scenarios. Furthermore, since no prototypes are needed for testing purposes, simulation can start as soon as a design is available and potential design errors can be discovered early.

The present research focuses on the testing phase of the development of Web applications, in particular, on the verification of the functional requirements of designs. Furthermore, simulation will be used to assess these requirements in a similar manner to how it is used in the assessment of hardware designs. For this purpose, the research has developed a suitable model of Web applications that supports their simulation. Additionally, the development of a Web-design description language that supports functional evaluation will complement the simulation model and define the simulation framework. Finally, the computer implementation of the simulation framework will provide developers with a powerful tool for the functional evaluation of Web applications.

1.2 Significance to the Area

The proposed goal of formally describing a complex Web application with the objective of simulating its functionalities for evaluation purposes is enticing. Simulation for this purpose has not yet been explored; however, we argue that the field has now reached a level of maturity that allows this to occur. Identification of the requirements a Web application design model must possess to enable simulation and the definition of a suitable Web application simulation model, allow simulation techniques to be used for functional evaluation of Web application designs.

The development of Web applications can represent a large investment of time and resources, with the testing phase of the development cycle significantly contributing to it. Simulation has been successfully and intensely used in the development and testing of complex electronic digital designs in the hardware engineering field. Similarly, by providing a powerful testing technique of Web designs, simulation has the potential to improve the testing phase, eliminate the need for prototyping for functional evaluation purposes and, inherently, reduce the development time. Furthermore, simulation allows extensive functional evaluation to be carried out, potentially leading to a high quality final product. From an economic point of view, reliable code and faster deployment time are synonyms for higher customer satisfaction and reduced development costs. Costs required to correct errors are directly proportional to the phase in which they are discovered, since they become increasingly entangled into the code in later phases (Moriguchi 1996, p. 197); the earlier they are found and corrected, the less they contribute to the overall development cost.

1.3 Contributions

Upon completion, the contributions of the research to the Web development field will be substantial. One of its contribution is the definition of a simulation model that addresses the key development concerns of Web engineers by defining a four-layer model, with each layer responsible for a specific and well-determined aspect of a Web application. This multi-layer simulation model allows an in-depth and meaningful evaluation of Web application design functionalities.

The development of a suitable description language that complements the simulation model, allows the representation of the key functionalities and structure of a Web application design in a formal manner. This language emulates the behaviour and structure of a Web application in such a way that functional verification by simulation techniques is made possible. Furthermore, scenario- and goal-based evaluation of the Web application functionalities are made straightforward, systematic, and less strenuous, since there is no need for partial implementations of the design.

There is little doubt that Web-based applications will continue to increase in complexity. New technologies will continue to emerge, supporting new features and services. It is then of paramount importance that Web engineers have available powerful auxiliary testing methods that can aid in Web application development projects. By supporting the testing phase, simulation of design models will enable developers to explore new and more intricate designs of Web applications, and contribute to achieving a higher quality of the end product. It is believed that the impact of this research will be similar to what occurred in the hardware design field, where descriptions languages have allowed developers to explore increasingly more complex techniques, designs and system architectures.

As a result of the present research the author has made a number of public presentations and some of his work has been published in peer-reviewed international conference proceedings (Peixoto et al. 2004*a*, Peixoto et al. 2004*b*, Peixoto 2005).

1.4 Outline of the Thesis

This thesis is structured in eight chapters and five appendices. In Chapter 2, a review of the relevant literature is made and the hypothesis of the research is stated. Chapter 3 describes the research methodology required to prove the hypothesis. Chapter 4 proposes a Web-design Simulation Model for Web applications, and the developed Web-design Description Language is presented in Chapter 5. Next, the tool based on the developed simulation framework is described in Chapter 6, and Chapter 7 presents the results of an experiment using the simulation tool. Chapter 8 draws the conclusions and proposes possible future work. The five Appendices present respectively the Web-design Description Language syntax, the syntax of the allowed set of simulation stimuli, the simulation tool database structure and commands, the Web application design used for the experimental testing of the hypothesis and, finally, the results of the simulation of the experiment's design.

Chapter 2

Literature Review

This chapter discusses related research that supports and elucidates the topic of the present thesis. The issues that are under examination cover all the aspects that this research directly or indirectly addresses. The following sections take an in-depth look into the topics of software engineering, software development models and methodologies, software quality, Web engineering, Web development models and methodologies, testing and simulation.

2.1 Software Engineering

It can be argued that the development of Web applications form a case of the development of a software product (Glass 2003, Holck 2003, Kappel et al. 2004). Therefore, this chapter starts by investigating the software engineering literature, with particular emphasis on the development of software products. These classic or traditional approaches, based on several decades of sound research and experience, will allow some conclusions to be drawn regarding properties and objectives that both software and Web development projects share and pursue.

Software engineering is the branch of systems engineering that facilitates the development of large and complex software systems (Finkelstein and Kramer 2000). It defines and models the phases that a project should go through to successfully achieve a high-quality product within the predetermined time-frame and budget. These models are especially concerned with the development of software products, and they include a set of guidelines and activities for managing and controlling the design and building phases of a software system. This forms part of what is often referred to as the Software Life Cycle Methodology, which Bosch et al. define as consisting of "a collection of tools, techniques, and methods which provides roles and guidelines for ordering and controlling the actions and decisions of project participants during the software life cycle", whereas the Software Life Cycle can be defined as "the time required to define, develop, test, deliver, operate and maintain a (software) system" (Bosch et al. 1982). These phases usually entail: (Pfleeger 2001, p. 47)(Moriguchi 1996, p. 182):

- Requirements analysis and definition
- System/Software/Program/Module design
- Coding/Debug
- Module/Program/Software functional/System test
- System delivery
- Operation
- Maintenance

The requirements analysis phase basically consists of the study of the user needs, culminating in the definition of the system, hardware, or software requirements (*IEEE standard glossary of software engineering terminology - 610.12-1990* 1990). It is a contract between a customer and the development team, often resulting in a written document called a System Requirements Specification (SRS). The SRS describes what the system should be able to accomplish and under what conditions it will operate. There are several advantages of using a well-written SRS during software development, namely: it establishes an agreement between customers and suppliers on the functionalities of the software product; potentially leads to a reduction of the development effort; provides a good estimation of the costs and duration of the project; provides a baseline for validation and verification; eases software transfer to new users and machines; and provides a basis for later refinement of the product (*IEEE guide to software requirements specifications* 1984). It is, therefore, important that the SRS be written in an unambiguous manner, be complete, verifiable, consistent, modifiable, traceable, and be used during all of the phases of the project.

These requirements describe the functionalities that the software product should implement – the functional requirements – as well the restrictions and external constraints which the final product must obey – the non-functional requirements. Formally, the definition of requirements may be expressed as a set of descriptions of the system behaviour, application domain information, constraints and restrictions of the operation, and the properties and attributes the product shall possess (Kotonya and Sommerville 1998). Furthermore, the functional requirements specify the stimuli to the system, the responses from it, and the "behavioural relationships between them" (Young 2004).

The design phase consists of the conceptualisation of the product from the requirements. During this phase, which is sometimes named "programming-in-the-large" (DeRemer and Kron 1975), the system is partitioned into independent operation units. The scope of these units depends on the level of abstraction of the design. When the product design is available, the actual coding and implementation may commence, sometimes referred to as "programming-in-the-small" (Favre 1997, Cazzola et al. 1998).

The next phase deals with the important issue of testing the product. This phase accounts for a significant amount of the total development cost (Moriguchi 1996, p. 198). For this reason, testing may and should begin early in the Software Life Cycle and not only after the final implementation of the software product. The sooner requirements, design and implementation errors are discovered, the less impact they will have on the overall project duration. The earlier in the Software Life Cycle the errors are identified, the less expensive correcting them becomes.

The remaining phases consist of the delivery of the system, its operation, and maintenance. System delivery can be as simple as handing over the product, or as complex as encompassing training and providing documentation. A successful system delivery will allow users to confidently operate the system to its full extent and capabilities, leading to a high client and user satisfaction level. Operation is considered as the phase in which the system is being used by the intended audience in a real environment. And finally, maintenance is any task that involves changing the system after it has commenced operation.

The above-mentioned stages set guidelines for the different phases a software product is likely to go through. They describe what is expected in each phase, namely, what requisites are needed to perform that specific phase, and what results should be expected at its completion. They do not, however, impose a structurally rigid format on the Software Life Cycle. How and when each phase is addressed during the Software Life Cycle, and if it is an interactive, recursive or incremental cycle, depends on the software process model. These models organise all phases into a development process, and, as such, describe in an orderly manner when each phase shall begin and what input and output are expected. Numerous software process models have been proposed, with each having its own advantages and pitfalls.

2.1.1 Software Process Models

There are several reasons for modelling a software development process. It helps the development team to understand what is required in each phase and how close the current development is from those goals. The following paragraphs describe the most common models found in the literature.

The Waterfall Model

One of the first proposals of software process modelling was the Waterfall model (Royce 1970). Based on his own experience with large software development projects, Royce put forward a model that encompasses the main phases often found in such projects (see Figure 2.1). The Waterfall model was extensively used by the United States Department of Defense (DoD) for several years, and still is for some projects. It is a simple yet comprehensive description of the main tasks involved in software development projects. Each step has its input from the previous step and produces the input for the following step. It is a simple cascade of steps, but even Royce believes that a real software project is not as linear as the model describes.

In reality, iteration among steps occurs frequently in software development. Problems with the Waterfall model have been extensively discussed in the literature (Boehm 1988, Clear 2003, Laplante and Neill 2004, Guimaraes and Vilela 2005). Usually these refer to the inherent iterative nature of software development, which the linearity of the model fails to adequately address. Furthermore, it is only at a very late stage that a running application is delivered and tested, resulting in an equally late evaluation. This usually leads to serious issues if it is found that requirements need to be changed or that the implementation does not fully or correctly implement them.

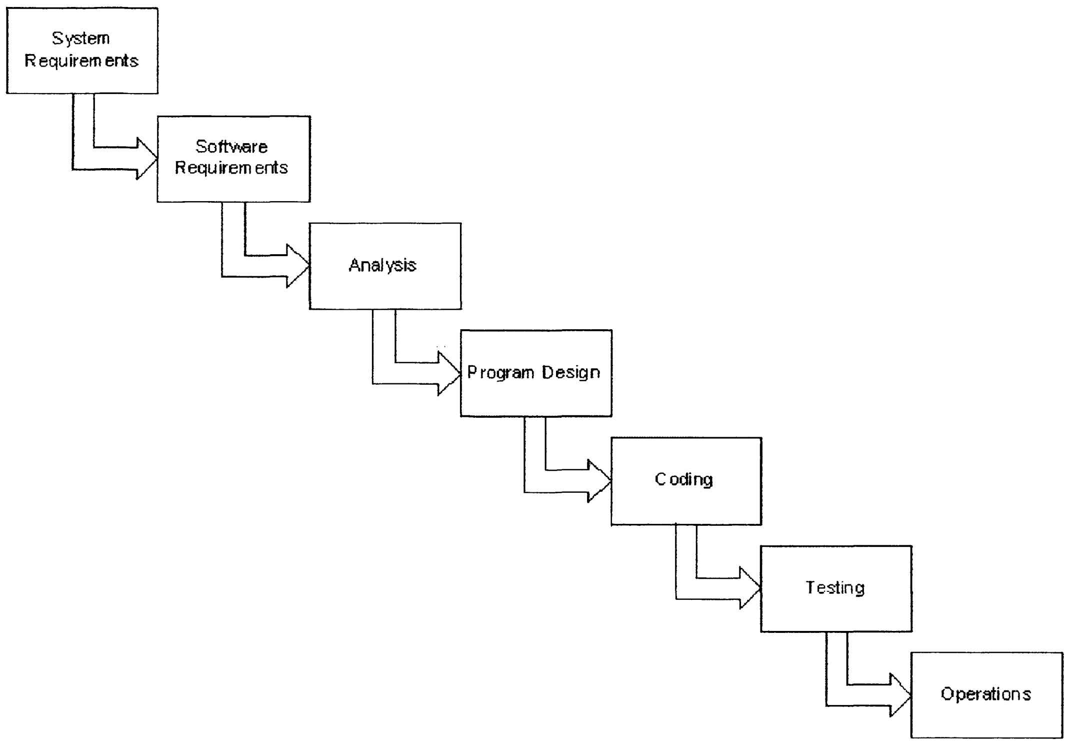


Figure 2.1: The Waterfall Model (Royce 1970)

The Spiral Model

Boehm proposed a model which has *risk* as an intrinsic characteristic of software development (Boehm 1986, Boehm 1988). Risks, in this context, can be informally defined as "something that can go wrong" (Sommerville 2004, p.73). These risks often have an impact on the costs and schedule of the development, therefore risk minimisation is an important activity. In this model there are four distinct cycles: concept, requirements definition, system design, and implementation. Each cycle of the spiral involves four activities shown as the four sectors in Figure 2.2: determination of objectives, risk assessment and reduction, development and verification, and planning.

Although addressing the issue of risks as part of the development process,

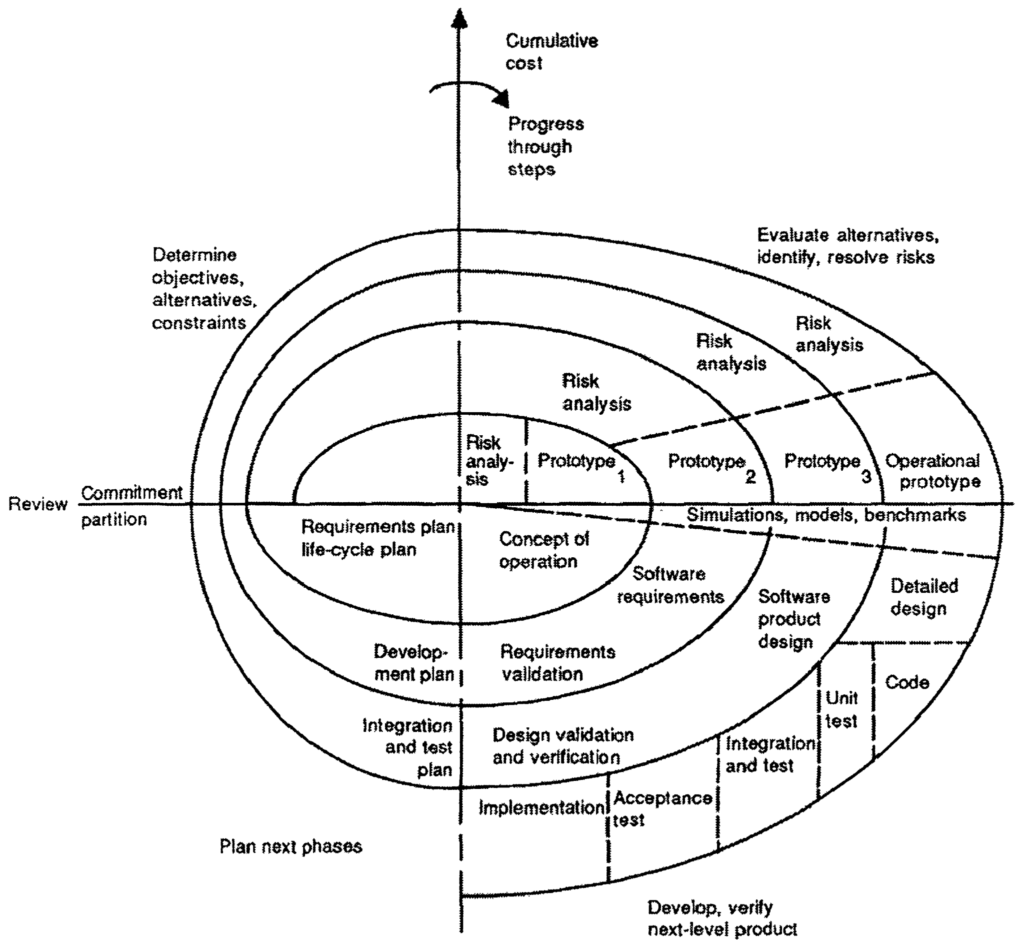


Figure 2.2: The Spiral Model (Boehm 1988)

therefore contributing to minimise costs and duration of the project due to them, the Spiral model has two main problems: first, some unnecessary overheads are introduced in small projects, and second, the model does not directly address the issue of maintenance (Lowe and Hall 1999, p. 243).

2.1.2 Testing for Quality

Testing is the stage of the Software Development Cycle in which the product is Verified and Validated (V&V). It ensures that the software product meets the specifications and the expected functionalities. Definition of these processes can be found in the *IEEE standard for software verification and validation – 1012-1998* where they are expressed as:

DEFINITION 2 *Validation – "Confirmation by examination and provisions of objective evidence that the particular requirements for a specific intended use are fulfilled."*

DEFINITION 3 *Verification – "Confirmation by examination and provisions of objective evidence that specified requirements have been fulfilled."*

Informally, Validation addresses the question 'Are we building the right product?', therefore evaluating if the system meets all the needs and expectations of the user. Verification asks the question 'Are we building the product right?', therefore it evaluates if the system meets its specifications. There are two approaches to Verification and Validation: *Software inspections* and *Software testing* (Sommerville 2004, p. 517). The former checks the system representation by looking into the requirements specification document, design models and the actual coding. These inspections are performed without executing the system even if some of the inspections are fully automated, such as when using tools that automatically perform an analysis of the source code; for this reason they are called *static V&V techniques. Software testing* involves the execution of the system's partial or complete implementation and, due to its nature, it is called a *dynamic technique.* It entails the execution of the system implementation with test data and posterior analysis of its outputs and behaviour. Since it relies on an executable source code, this technique can only be carried out when an implementation is available.

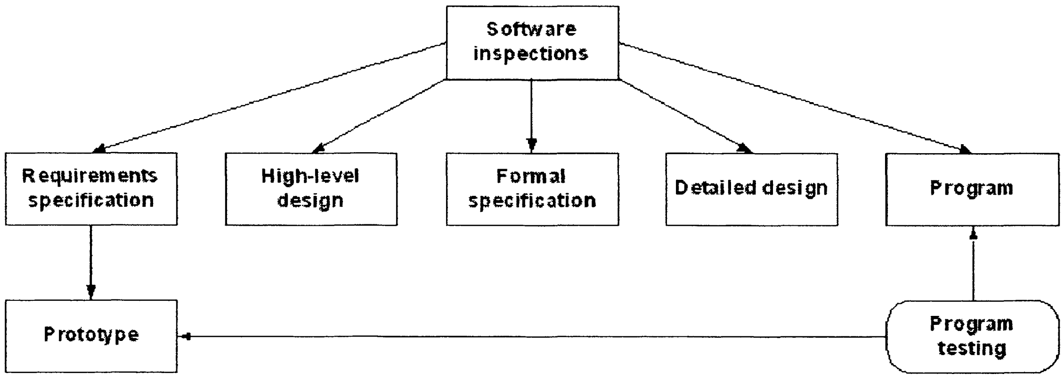


Figure 2.3: Verification and Validation of Software Products (Sommerville 2004 p. 517)

Figure 2.3 shows that software inspections and testing are two complementary approaches to system checking and analysis. As seen in the figure, testing can only be performed when a prototype or an executable program is available. On the other hand, software inspections can be performed at all stages of the software development process. Sommerville argues that there are three advantages of inspection over testing, namely: errors found during testing can hide other errors; inspections can be executed on partial system versions; and other quality attributes such as compliance with standards, portability, and maintainability, are also possible to be considered by inspection techniques. The effectiveness of software inspections over testing is supported by a number of studies found in the literature. Examples of these studies can be found in Eagan (1986) and Selby et al. (1987).

There are many methods and techniques to promote and assure high-quality software products. One of them is by evaluating[1] the software product against the requirements specification (Harrold 2000). Although it is difficult to define the concept of a high-quality software product, it has been traditionally described as the property a developed product exhibits when it meets its specifications (Crosby 1979). Researchers have long been arguing that testing should not be a unique stage

[1]In this research, the term 'evaluation' is used as having the meaning 'qualitative verification'.

performed at the end of the implementation but a process that should be carried out through the entire Software Development Cycle, strongly interacting with the remaining development phases (Gelperin and Hetzel 1988). In fact, some developers argue that "testing is never finished, only abandoned" (Reasoning 2003).

2.1.3 Simulation as a Verification and Validation Process

Simulation has long been proposed and used in the engineering field for software inspection purposes (Adrion et al. 1982, Zhu et al. 1997). Simulation is performed on a specification or design of the system and not on some partial or complete implementation. This means that testing may commence during the design phase, without having to wait for the implementation phase to produce some prototype. By performing simulation, potentially faulty behaviour of systems can be observed and system outputs can be compared against the expected results. A definition of simulation is given by IEEE as:

DEFINITION 4 *Simulation – "(1) A model that behaves or operates like a given system when provided with a set of controlled inputs. (2) The process of developing or using a model as in (1)" (*IEEE standard glossary of software engineering terminology – 610.12-1990 *1990).*

In his work, Shannon (1998) enumerates several advantages of the use of simulation, namely:

- Testing of new designs without having to implement them
- Investigating new procedures and information flow without any disrupt on the ongoing operations
- Identification of bottlenecks and testing of possible solutions
- Test hypothesis about how and why certain phenomena occur

- Control of the time, and as such, the possibility of simulating the system as fast or as slow as desired
- Better understanding of how the system works and which are the most important variables to performance
- The ability to experiment with new and unfamiliar situations and scenarios.

These benefits led simulation to become one of the most interesting techniques for developing and experimenting with new models in engineering. Discrete event simulation, a particular case of simulation, assumes that output variables may only take discrete values. Banks and Carson (1986) define this type of simulation as "one in which the state of the system changes only at a set of discrete points in time called event times". The simulation is performed by increasing the simulated time and changing the state of the system according to its model. Contrasting with this type of simulation is the continuous event simulation. In a continuous event simulation, the system may change its state continuously over time.

What engineers are searching for is how the system behaves and what its outputs are due to a given stimuli. With simulation, precisely repeatable experiments and analysis of the inputs versus outputs are possible. One other advantage is that "real-world" experiments are often impossible, undesirable, or too expensive to carry out. Different scenarios may be simulated and faulty situations can be replicated, contributing to a better understanding of the system's behaviour.

The use of simulation techniques for verification and validation purposes of systems have for long been proposed and many examples can be found in the literature body. Of particular interest for this thesis is the use of simulation techniques for the verification of real-time systems. A Real-Time System (RTS) may be defined as one system in which its correctness depends on the logical result of computation and on the time at which these results are produced (Nissanke 1997, p. 2). This definition closely resembles a Web Application operation, since its results are dependent not only of a logical result but on the time at which these results are computed.

Simulation testing of real-time systems and software provides many benefits such as cost savings and error detection. Examples of using simulation as a testing technique for a RTS can be found in Pasahow (1973), where a simulation structural framework for computer-aided testing of software programs in an interactive real-time environment is described; in (Henry et al. 2003) where simulation testing of real-time software is used for early error detection and hardware/software integration testing; and in (Evanco and Yang 1992), where Petri nets – which are a graphical and mathematical modeling tool which can be aplied to many systems (Murata 1989) – are used for simulation purposes.

One other area where simulation has been used as a verification and validation tool is in the design of user interfaces. Laakso and Laakso (2003), for example, describe a model in which the user interface is designed at the begining of the project enabling early testing of the system's suitability for its intended use. Functional design and implementation is left to later stages in the project, after the user interface specification has been tested. Testing of the user interface, as proposed by the authors, is performed by the use of simulation techniques. The authors argue that by simulating the user interface and system's applicability to the intended use at an early stage of the development project and by leaving functional design and implementation to later stages, any eventual changes needed will still be easy to make.

Simulation consists of the design of a dynamic model of a dynamic system in order to better understand the system's behaviour or to evaluate the operation of the system in various scenarios (Ingalls 2002). In order to achieve an accurate representation of the system to be simulated, a simulation study should address several phases and processes. Balci (1987) proposes a simulation study methodology consisting of ten phases and processes. These phases are: (1)Communicated Problem; (2)Formulated Problem; (3)Proposed Solution Technique; (4)System and Objectives Definition; (5)Conceptual Model; (6)Communicative Model; (7)Programmed Model; (8)Experimental Model; (9)Simulation Results; and (10)Integrated Decision Support. Ten processes link these ten phases, and address each step of the study. Furthermore, Balci proposes Verification and Validation techniques to be used throughout the study, in order to ensure credible simulation results (Balci 1994, Balci 1995, Balci 1997). This methodology will be systematically used throughout the present research, in order to achieve a high quality, accurate and credible simulation model.

2.2 Web Engineering

Web Engineering emerged to tackle Web applications development from scientific and engineering perspectives. It recognises these development projects as a particular case of software development. Hence, it borrows concepts, techniques, models, and methodologies from the software engineering field and adapts them to this new type of project. Web engineering has been trying to address the problems that have arisen from these projects with a systematic and disciplined approach for developing, documenting and maintaining Web applications (Costagliola et al. 2002). Web engineering may be defined as:

DEFINITION 5 *Web Engineering (WebE): "Web engineering is the establishment and use of sound scientific, engineering and management principles and disciplined and systematic approaches to the successful development, deployment and maintenance of high quality Web-based systems and applications" (Murugesan et al. 2001).*

Web engineering tries to cover a wide range of areas, including requirements specification and analysis, Web-based system analysis and design, Web-based system development methodologies and techniques, testing, verification and validation techniques and tools, quality assessment, control and assurance, development models, Web project management, user-centric development and user modelling.

2.2.1 Difference from Traditional Software Development

Although Web application development is usually viewed as a form of software development, there are some remarkable differences between Web development and other forms of software development (Ginige and Murugesan 2001). These prevent or make it difficult to use many of the traditional approaches. Firstly, the short development times and product life cycles are often an obstacle to the use of conventional software principles, which are considered too time consuming to implement and follow. Developers do not have the time to deal with them and a more agile approach is usually adopted (Pressman 2000, Reifer 2000). The short time-frames of Web projects demand a light development methodology, otherwise there is a high probability that these methodologies will simply not be used.

Secondly, users must be acknowledged as being an important part throughout the Software Life Cycle (Ginige 2002*b*). How users will access the content, the paths followed and actions performed may very well influence how the Web application should be designed and implemented.

Thirdly, due to the variety of the backgrounds and expertise of the people involved in the development team, there will be inevitable misunderstandings and misinterpretations of the intended goals and processes, which may occur at any stage of the Software Development Cycle. Although software engineering may suffer from the same problem, in the Web domain it assumes greater relevance because of the high heterogeneity level of the team (Pressman 1998, Hansen and Deshpande 1997).

Lastly, there usually is some fuzziness in the definition of the problem, especially concerning the analysis of the requirements (Bolchini and Randazzo 2005).

This implies that the development method must be able to quickly and adequately respond to any changes in the requirements. Often requirements change during the product's life cycle and the development method must possess a rapid and appropriate reaction to minimise its implications on the project.

As Coda et al. (1998) have stated, much has been achieved in the software development field, which provides technologies, methodologies, and tools for the delivering of high-quality products in a timely and cost-effective manner. To achieve similar goals and to promote the maturity of the Web application development field, one has to follow a comparable approach. The following sections present the most innovative and relevant models and methodologies used in the development of Web applications found in the literature.

2.2.2 Methodologies

Until recently, Web applications were often developed without appropriate methodological support or tools, solely based on the common sense, skills and knowledge of the individuals involved, much like the early years of software development (Coda et al. 1998, Pressman 2000, Barry and Lang 2001, Kappel et al. 2004, Rosson et al. 2005). The current trend, however, is to use the expertise gained with software development projects and adjust them to the Web development paradigm.

Object-Oriented Hypermedia Design Method (OOHDM)

The Object-Oriented Hypermedia Design Method is a model-based methodological approach for the design of hypermedia applications. It consists of four different activities used in a "mix of iterative, incremental and prototype-based styles of development" (Schwabe and Rossi 1995*a*, Schwabe and Rossi 1995*b*), namely:

1. Conceptual model design

2. Navigational design

3. Abstract Interface design

4. Implementation

During the conceptual model design phase, an object-oriented model of the application domain is built. This results in a conceptual schema which entails a set of classes, relationships, and sub-systems, capturing the domain semantics in such a manner that it remains independent from the types of users and tasks of the system. During this activity, OOHDM does not impose a method to use, instead it refers to well-known object-oriented modelling principles such as the Object modelling Technique (OMT) (Rumbaugh et al. 1991).

The navigational design activity takes place when the definition of the navigation structure in terms of navigational contexts has been established. Navigational contexts consist of navigational classes such as nodes, links, indexes and guided tours. These are dependent on the intended users and different navigational models may be developed for the same conceptual model, which allows different views for different users.

During the abstract interface design activity, the interface is designed. However, to achieve independence from the implementation environment, the model is purposely abstract. One of its main tasks is to determine which interface objects will be displayed to the user, and "the way in which different navigational objects will appear" (Schwabe et al. 1996).

Lastly, the implementation activity maps the interface objects into implementation objects. Unlike the three activities before, this time the runtime environment is taken into account. Although traditionally manually coded, some attempts to automate this task have been proposed, such as the OOHDM-Web environment (Schwabe et al. 1999) which relies heavily on templates and function libraries. According to Schwabe et al. (1996) the main contribution of OOHDM is how it actually structures the design process. In fact, the final implementation of the web application is achieved by several incremental iterations through the first three activities.

Relationship Management Methodology (RMM)

RMM is a methodology which consists of several steps for aiding the design of large hypermedia applications, which supports the process of development and maintenance (Isakowitz et al. 1995). The seven steps are:

1. Entity-Relationship Design
2. Slice Design
3. Navigational Design
4. Conversion Protocol Design
5. User-Interface Design
6. Runtime behaviour Design
7. Construction and Testing

The Entity-Relationship (E-R) Design forms the basis upon which the hypermedia application is built, and E-R diagrams are drawn to expose the relevant information of the application. At the end of the Navigation Design stage, these diagrams are used to produce a Relationship Management Data Model diagram (RMDM) which is the cornerstone of the whole methodology. It provides the means to describe the information and navigation of Web applications. Furthermore, RMM separates the data, structure and user-interface aspects of a Web application, contributing to its easier design, development and maintenance. The main contribution of RMM lies in the logical layer between the presentation layer (where the data is displayed) and the storage layer (where the information is physically organised).

The original proposed RMM had severe restrictions concerning the combination of different entities in a single screen, which was unable to model the content of complex Web pages. Another limitation concerned the re-use of low-level designs which was prevented by a forced top-down design methodology. These concerns were addressed at a later stage, resulting in RMM Extended (Isakowitz et al. 1998).

Summary

Although methodologies have been proposed to tackle the complexity of Web applications development, there is always a need to test the application to assess its conformance to the proposed functional requirements (Hieatt and Mee 2002, Lucca et al. 2002, Elbaum et al. 2003). In fact, most of the researchers advocate a cyclical and incremental Software Development Cycle, with several iterations through the analysis, design, implementation, and testing phases (Schwabe and Rossi 1995*b*, Isakowitz et al. 1995). Furthermore, Nanard and Nanard (1995) point out the special need of the Web development cycle for prototyping and intensive testing. Therefore, functional requirements are tested based on these prototypes and not on the design models themselves which, supposedly, are the ones that should capture them. Simulation, on the other hand, would verify the requirements based on the design models and not on prototypes which can, themselves, contain errors, misconceptions and faulty implementations. Furthermore, if the requirements analysis phase is not properly conducted, only after the implementation of a prototype can developers perceive faulty conceptual behaviour. If this scenario occurs, only on the next iteration and cycle can analysis, design, and implementation errors be corrected. Simulation, by not relying on prototypes, focuses on the verification of the requirements during the design phase where they are captured. By closely grouping design and testing phases without an intermediate implementation phase, simulation provides developers with a powerful technique which potentially leads to a faster development cycle and high-quality products.

2.2.3 Models

Whatever methodology is chosen for a given Web application development project, developers rely on Web application models for the design phase. These allow a division of concerns of the different elements involved in the application, providing suitable abstract notations without committing to a specific implementation. At the end of this phase, a design of the application is produced and implementation may begin. There is an abundance of design models, each with their own particular characteristics. The next sections describe some of the most well known design models found in the literature.

Dexter Hypertext Reference Model

As Halasz and Schwartz (1994) note, this model allows a comparison of the characteristics and functionalities of Hypertext applications. The Dexter hypertext reference model considers three layers of a Hypertext application: (1) the run-time layer, (2) the storage layer and (3) the within-component layer. The model's main focus is on the middle layer – the storage layer. The within-component layer is concerned with the content and structure within the components of the application. However, this layer is not elaborated in the Dexter model. The broad range of possible content and structure that can be included in components is such that this model does not attempt to tackle them (Halasz and Schwartz 1994). In fact, the model treats the components structure as being outside the hypertext model, and expects other models more suitable to describe the structure of applications, documents or data types to be used in conjunction with it, the result being that text, graphics or animation are treated as generic data. The storage layer assembles the components to construct a hypertext network. The storage and within-component layers consider the hypertext as a passive data structure. To add dynamic behaviour, a third layer is provided – the run-time layer. The run-time layer supports the user with "tools to access, view and manipulate the network structure" (Halasz and Schwartz 1994). This layer enables the user to access view and edit hypertext. Once again, the model does not attempt to provide a detailed description of the methods involved in such operations. As a result, the model is not able to precisely describe and handle the user's interaction with the hypertext. This is an important issue when considering model simulation, since user interaction is the main provider of stimuli that drives the simulation.

Hypermedia Design Model (HDM)

One of the most prominent design models is the Hypermedia Design Model (HDM) developed by Garzotto, Paolini and Schwabe (1991*b*). It is, in fact, the basis for two important methodologies described earlier in this chapter – the Relationship Management Methodology and the Object-Oriented Hypermedia Design Model.

This model was part of an European project – the HYTEA project (Esprit 1990) – carried by the ESPRIT consortium whose objective was an engineering approach for authoring tools for hypertext development, thus the acronym **HY**per**T**ext **E**nvironment for **A**uthoring. The model claims that "systematic and rational structural decisions about the hypertext should be made before the actual hypertext is ever written, so that coherent and expressive hypertext webs can be designed instead of added-on." (Garzotto et al. 1991*b*). It borrows the concept of the entity-relationship model from the database development field and extends it to a hierarchical organisation. The approach is a design model used to describe hypertext applications, consisting of basic blocks called *entities* that may be structured in *components* and *sub-components*.

HDM uses the concepts of authoring-in-the-large and authoring-in-the-small which can be found in the software engineering discipline. There are two main tasks to perform in the authoring-in-the-large phase: the global and the instantiation tasks (Garzotto et al. 1991*a*). The global tasks are concerned with *classes* of information elements and, at the end, will result in an application schema, which represents a high-level specification of the application's main features. The interconnection tasks will examine which information is represented by the classes and how they are connected. The main goal of authoring-in-the-large is to define the topology of the Web application before the hypertext is written. Authoring-in-the-small, as the authors define it, refers to local tasks which deals with the actual information and its specific look-and-feel implementation being presented to the reader.

HDM was developed as an aid for the authoring-in-the-large development phase. The authors argue that authoring-in-the-small is strongly dependent on the application's specific characteristics and implementation, whereas authoring-in-the-large displays similarities when used with different applications in a given domain. Hence, the authoring-in-the-large phase must be addressed with high-level notation and primitives as opposed to the traditional node-and-link approach. One of the HDM goals is to free the developer from the implementation details, focusing on the high-level issues of the design abstraction. Another goal is to identify, model, and promote reusability of the most common hypertext application patterns.

As two of the aims of HDM is system independence and high-level abstraction, the relation between the implemented Web application and the model is somewhat loose, leaving some of the implementation issues to the browsing semantics being used on the target system. This loose coupling between HDM and the targeted browser semantics, concerning visualisation and the dynamic properties of the hypertext, leaves three important aspects open to the designer:

1. What and how are the objects perceived by the user?

2. Which and how are links visible and navigable?

3. What behaviour do links possess?

As an example, a single-source-multiple-destination link mechanism does not necessarily translate to a multiple active window display. By not defining and imposing constraints on the presentation of Web applications (although considering but not incorporating some conceptual layout issues), HDM does not address some of the authoring-in-the-small aspects, such as user interaction and user profiles, to name a few. Furthermore, the entities used in HDM do not implement a well-defined interface schema and the interconnection between semantics and syntax is fairly informal.

Unified modelling Language (UML)

The Object Management Group (OMG 2005) defines the Unified Modeling Language as "a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems" (OMG 2004, p. 2). UML is a modelling language which supports object-oriented design and analysis.

UML is intended to be scalable and suitable across different domains. As a result, there have been attempts to utilise it in the Web application development field. For instance, in Baresi et al. (2001) UML and HDM are combined to produce a Web development framework – UML's Use Cases diagrams and UML's Dynamic diagrams are extended and used to describe, respectively, operational and navigational requirements, and to model Web operations. Another example is the Web Application Extensions to UML (WAE), where Conallen proposes an extension to the UML core to tackle the Web applications specificities (Conallen 1999, Conallen 2000). This extension has brought formalism into the development process. However, the Web Application Extensions mainly focus on the architectural and implementation issues of a Web application, leaving the remaining navigational and presentation aspects with less support (de Koch 2001, p. 134). Moreover, WAE does not consider content, navigation, and presentation as three distinct aspects of a Web application, and hence does not contribute to the separation of concerns which is normal practice in a project of such a nature.

A very promising field is that of executable UML, which may be defined as a particular usage (or *profile*) of UML that allows developers to define the behaviour of a single subject matter in such detail that it can be executed (Mellor and Balcer 2002, p. 7). An executable UML specification entails a set of models represented and expressed as UML diagrams, that describe and define the conceptualization and behaviour of the system under study. An executable UML specification comprises three fundamental models: a class model expressed using UML class diagrams, the state machines for the classes expressed using UML statechart diagrams, and the states' procedures or set of actions expressed using an action language.

Executable UML relies on the action semantics specification for UML (OMG 2002). This specification defines the semantics of the actions, but not their syntax. The goal of action semantics for UML is to make UML modeling executable, thus allowing designers to test and verify the models, and even to generate 100% of the code (Sunye et al. 2001). By making UML models executable, simulation for verification purposes of UML models can be achieved. Transposing these concepts into the Web Application field by meging action semantics and Web Application Extensions to UML is just one step ahead and a very promising field.

WebML – The Web Modeling Language

WebML can be informally defined as "a notation for visually specifying complex Web sites at the conceptual level" (Bongio et al. 2001). It uses a graphical notation and XML syntax and consists of four models: the Data model, Hypertext model, the Content Management model, and the Advanced Hypertext model (Ceri et al. 2000).

WebML is especially well suited for the conceptual modelling of *data-intensive* Web applications (Ceri, Fraternali, Bongio, Brambilla, Comai and Matera 2002, Ceri, Fraternali and Matera 2002). One feature worth noting is the support for user modelling, with the notion of group and user, enabling Web applications to incorporate personalised user profiles.

WebML does not propose another Data model, but follows the well-known Entity-Relationship (E-R) notation. It is by using the E-R notation that the structure of the data used by the Web application is defined. The Hypertext model aims at defining the front-end interface of the Web application, and it is composed of several design elements such as pages, links and units. These units access the underlying data and map it onto the pages. Therefore, every WebML design element that has a graphic representation and that will be displayed to the user belongs to this model. On the other hand, the Content Management model defines the design elements that manipulate the application data. Operations such as creating and deleting data are modelled by units belonging to this model. Finally, the Advanced Hypertext model describes how everything can be combined into an executable Web application. It defines the manner in which the several design units can be mapped to an implementation.

WebML's main objectives are the high-level description of a Web application structure, separation of information content from presentation and navigation concerns, automatic code construction based on the design, and personalisation policies (WebRatio 2001). Furthermore, a tool – WebRatio® – that supports automatic code synthesis has been developed which highlights the design formalism (WebRatio 2005).

Enhanced Object-Relationship Model (EORM)

The Object-Relationship model was first proposed by Rumbaugh (1987), whereby n-ary relations are defined over objects. This model is a mix of the object-oriented model with the entity-relationship model found in database theory and, as the author states, "is particularly useful for designing and partitioning systems of interrelated objects" (Rumbaugh 1987). The Enhanced Object-Relationship Model (EORM) is a refined object-relationship model, wherein concepts such as attributes and behaviour were added (Lange 1994). The core aspect of this model is to explicitly attach semantic meaning to the relations of an object-oriented model. The motivation for the EORM work was the disregard of classical object-oriented methods for the object-interaction aspects.

It consists of three distinct frameworks, namely, the Class framework, in which the different problem domain classes are identified and refined – that is, attributes and operations specified, and inheritance determined; the Composition framework, consisting of a reusable library of link class definitions, with two associated activities – composition identification and composition refinement; and the GUI framework which consists of a reusable library of presentation definitions, and where activities such as presentation, window identification, and mapping of classes and compositions to presentations are performed. Furthermore, a support tool has been implemented and provides graphical building and automates code generation for prototyping purposes (Lange 1993). However, presentation of the model in browser windows is not one of the foci of this model (Lange 1994, p. 373). As such, complex window layout is not considered and user interaction with the graphical interface is neglected.

HadeZ

HadeZ is a object-oriented language, with a formal syntax and semantics, based on the Z and Z++ specification languages (Germán 2000). It aims to specify hypertext designs without concern for how it must be implemented. Furthermore, HadeZ is process independent and may be used with EORM, HDM, OOHDM or RMM. It divides the specification into three parts: conceptual schema, structural schema, and perspective schema. The most interesting characteristics of this language are the support of property verification of the specification and the possibility of reusing the design.

Although, the use of formal languages allow automatic verification of specifications, there are no current automatic code construction tools. In fact, the author states that it might be a task impossible to fully achieve due to a HadeZ specification being implementation dependent (Germán 2000, p.163).

W2000

The W2000 framework was first proposed by Baresi, Garzotto and Paolini. The authors argue that Web application modeling can be divided into two main aspects: modeling the hypermedia and modeling of the operations (Baresi, Garzotto and Paolini 2000). Furthermore, they claim that it is the integration of both these aspects that result in the conceptual modeling of Web applications. The W2000 framework extends the latest version of HDM – the HDM2000 (Baresi, Garzotto, Paolini and Valenti 2000) – with UML. According to the W2000 framework, the design of a Web application is divided into four dimensions: Information and Access Structures design, Operations and Business Process design, Navigation design, and Presentation design (Perrone et al. 2005). One of the most distinguishing features of W2000 is the explicit modeling of the operations present in a Web application. Web applications are modeled from not only the navigation and information perspectives, but also from an operations perspective. This Operation perspective allows designers to define single operations invoked by users and the logical transactions of the services provided by the application.

The World Wide Web Design Technique (W3DT)

Bichler and Nusser proposed the World Wide Web Design Technique (W3DT) as a new approach for the design of Web-based hypermedia applications. It consists of two activities: the graphical modeling of the structure of the Web site; and the computer generation of a running prototype of the system. The authors argue that the W3DT approach differs substantially from other design models, since W3DT aims to be a "design environment for the design of large-scale Web applications" (Bichler and Nusser 1996) and that its design primitives are based on the HTML functionalities. The basic primitives used are: *diagram*, *layout*, *link*, and *page*, which can be dynamically generated as result of script execution. These basic primitives are grouped into *Sites*, which forms the Web application design. The second activity – the automatic generation of the code – is provided by the WebDesigner Computer-Aided Software Engineering (CASE) tool. This tool allows Web developers to generate the necessary code from a W3DT model at any time. It also provides support for the design process through a graphical editor, which allows inspection and setting of the different primitives attributes and layout association to pages. Although, W3DT may be suitable for the 'authoring-in-the-large' of hypermedia, it does not model the Web application functionalities. Therefore, it does not provide enough functional information for the evaluation of the functional requirements of Web applications.

The Object-Oriented Method (OO-Method) and the Object-Oriented-Hypermedia (OO-H)

The Object-Oriented Method (OO-Method) was proposed by Pastor et al.. The authors argue that it combines the "conventional" Object-Oriented methodologies with Object-Oriented formal specification languages, in order to take advantage of the OO methodologies coming from the industrial context, and the OO formal languages which help to eliminate ambiguities of the elements. The OO-Method consists of two components: the conceptual model and the execution model. The conceptual model, which is further divided into the object model, the functional model, and the dynamic model, attempts to obtain a definition of the system without considering the implementation aspects – hence 'conceptual model'. The execution model defines the implementation-dependent features of the future system, which when used in conjunction with the conceptual model and a CASE tool produces the 'functional software prototype' of the system. The formalism of the conceptual model specifications and the existence of a tool for the automatic construction of the code are good indicators of the suitability of this model for simulation. However, attempts at using simulation for functional evaluation of the resulting OO-Method designs have not yet been reported.

Gomez et al. proposed the Object-Oriented-Hypermedia as an unified approach to the representation of the structure, behaviour, and presentation of Web systems (Gomez et al. 2001). The authors claim that its main contribution is the definition of a framework which is standards-based and captures the relevant properties of the modelling and implementation processes of Web application interfaces. It does not attempt to model Web applications but rather to provide developers with the semantics and notation for developing Web-based interfaces, which will be later integrated with pre-existent application modules. A CASE tool complements this approach, and generates the necessary code from the defined model. Through

OO-H, device-independent front-end specifications can be obtained. However, the OO-H does not attempt to model the underlying application, but to allow the development of web-based interfaces for existing OO-Method applications (Cachero et al. 2000).

Summary

Some of the existing Web application design models found in literature were described. There are, however, many more design models. Christodoulou et al. (1998), for example, classify existing hypermedia application development and management systems into one of six approaches, namely: object-oriented, entity-relationship, component-based, hybrid, open hypermedia, and others. In their work, the comparison is made through the evaluation of several criteria, such as: conceptual data model design, abstract navigation model design, user-interface and run-time behaviour design, implementation issues, and evaluation of existing implementation environments and tools. Conclusions of the study indicate that no single methodology covers all the criteria efficiently, and that selection of a particular methodology for hypermedia application development should be made on a case by case basis. Furthermore, one of the evaluation criteria – the *Quality testing* criteria, which assesses how much the development tool assist Web developers in evaluating the design – was found to be one of the criteria having the lowest scores. This is an indication of the lack of support of the existing development environments for the testing phase of applications. Evolution of the different design methods found in the literature can be represented in a diagram (Lang 2002), where the different methods and their relation are shown (see Figure 2.4)[2]. This diagram suggests a continuous evolution of the theory of design models which, based on the number of recent publications

[2]Note: the View-based Hypermedia Design Methodology (VHDM) proposed by Lee, Kim, Kim and Cho (1999) enhances the existing RMM; the Scenario-based Object-oriented Hypermedia Design Methodology (SOHDM) Lee, Lee and Yoo (1999) adopts a Object-Oriented technologies to deal with rich semantics.

in this area, is not expected to slow down soon.

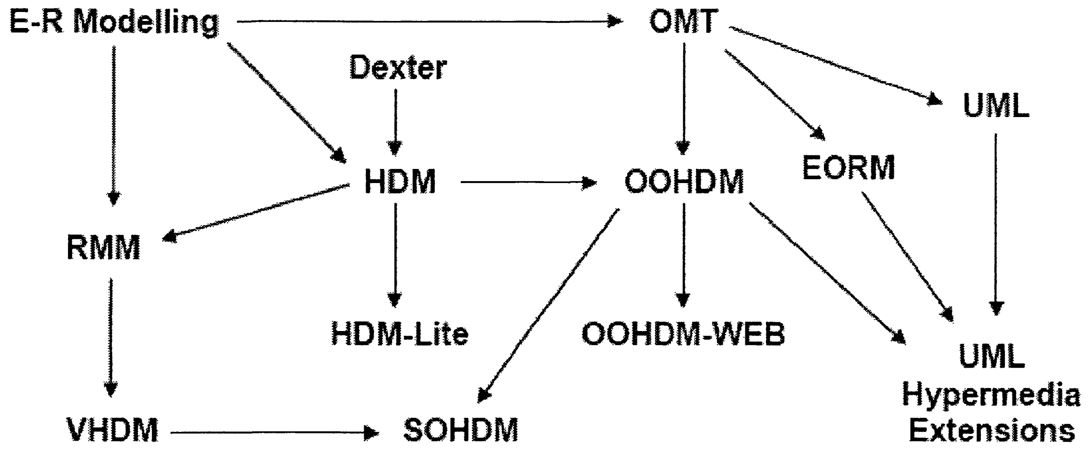


Figure 2.4: Evolution of Hypermedia Development Methods (Lang 2002)

Design models translate the functional requirements into an abstraction of the intended system, without committing to a specific implementation and technology. Therefore, the functionalities of a system will be (or should be) represented by the design. Even though carefully planned designs may present few conceptual errors, there are those that will only be discovered when testing a prototype. This happens due to the fuzziness of requirements and the complexity of Web applications. Furthermore, how the system will operate in a real-world situation is not easy to determine without observing its behaviour. It is not only due to the fact that most of the design models do not suitably capture system behaviour in a multi-session environment; they also do not properly represent how the system will work in a multi-user scenario. Although some design models consider personalisation (such is the case of WebML), how the application will work in a real-world situation is only effectively evaluated during the testing of a prototype.

Some of the described design models have an associated CASE tool which generate the necessary code. This generated code usually consists of skeletons which need to have detail added before they become fully functional. However, thorough testing of complex Web applications usually requires other tools to automatically perform the necessary tests. These testing tools usually use the generated code to evaluate the functional requirements of the application. Therefore, it is necessary to follow several steps before the functional requirements can be verified. On the other hand, most of the design models do not have a CASE tool for the code construction. In this case, Web developers usually rely on prototyping in order to evaluate the application's functionalities, resulting in a lengthy development cycle. This is where simulation can be of assistance.

Simulation is, by definition, a technique for evaluating how a model behaves in a given scenario. If developers could observe how the design model will functioning in a "close to the real-world" situation, it would provide confidence that implementation of that particular design would be likely to meet the proposed functional requirements. However, although some of the design models already have a companion tool for aiding design (as RationalRose for UML (IBM 2005)) or for automatic code synthesis (such as WebRatio for WebML (WebRatio 2005)), simulation has not yet been considered, and testing by prototyping is still the norm.

2.2.4 The Testing Phase

Even if the analysis and requirements gathering phase of a Web application development has been carefully planned and systematically carried out, design errors are bound to happen. This is often due to the ambiguity of the requirements. Furthermore, if the design model contains errors, its implementation will inevitably not meet the intended requirements. However, irrespective of the source of these errors, verification of the system requirements is an necessary task to perform if a high-quality product is desired.

Functional testing can be roughly split into two types: black box and white box. Black box testing does not have access to the internal procedures of individual modules, usually thought of as units with which the developer knows how to interact but of which he/she cannot observe the internal mechanisms. On the other hand, white box testing allows testers to observe the code and probe internal signals and variable values. Testing can also be categorised in other ways. An example of this is based on the scale of unit being tested: unit testing, integration testing, system testing, and performance testing. Unit testing is concerned with the code and usually begins as soon as implementation begins; integration testing tests how the individual units work together; system testing tests the entire functionalities of the system; and performance testing is concerned with the time or cost associated with functions (Ash 2003).

There are several commercial and open-source tools available to test implementations of Web applications (Hower 2005). Most of them focus on verification techniques of the implemented code, such as HTML and link checkers, and performance issues, and few are used to verify its functionalities (Elbaum et al. 2005). Examples of *functional* testing tools can be found in jUnit (Gamma and Beck 2005), WebTest (Canoo 2005), WebInject (Goldberg 2005), JStudio (Soft. 2005), AppPerfect (AppPerfect Co. 2005), and HttpUnit (Gold 2004). These tools test an implementation by issuing HTTP requests to the Web server emulating user interaction, and inspecting and processing its responses (Ricca and Tonella 2005). Although constituting interesting automated approaches, Kung et al. (2000*a*) argue that some of these tools do not provide behavioural information about Web applications.

Using implementations to test Web application functionalities requires partial coding of the design. Testing is performed by defining a sequence of pages to be visited and input values for form pages (Ricca and Tonella 2001). The developer navigates through the Web application, and tests its functionalities by entering data on form pages and submitting it to the server. This produces two types of observable evidence: a sequence of navigated Web pages and content changes in the application database. The developer can then evaluate the tests by comparing the obtained results with what was expected.

Testing Web applications should start as early as possible during the Software Development Cycle (Hieatt and Mee 2002, Sommerville 2004). This not only makes the process of finding errors and correcting them easier, but also improves the quality of the process and reduces the overall development time. However, testing the functional requirements is not possible until some sort of a prototype is implemented. This requires that, after the design phase, the developer produces a partial implementation of the application, which can then be tested.

2.2.5 Simulating for Testing

Simulation of Web applications is a rather new topic in the Web Engineering field. So far research has been focusing on a specific perspective of the technique. One area in which simulation for evaluation purposes has been intensely used is the assessment of Web servers' performance under selected working conditions. For example, simulation is used to support: Web server load-balance techniques (Bryhni et al. 2000); evaluating the extent to which dynamic Web pages degrade Web server performance (Iyengar et al. 1997); evaluating the implications of different Web proxy caching techniques in user-perceived latency (Feldmann et al. 1999, Rosu et al. 2000); and evaluating resource utilisation and response times of Web servers (Wells et al. 2001).

However, there is a gap in the literature regarding the simulation of Web applications design models for testing purposes. This type of simulation is not concerned with performance evaluation, but with the verification of the functional requirements of designs. A Web application resembles a state machine in which variables may take on only discrete values, and also, at any point in time, the system is found in one of a well-known set of states. This complies with the definition of a *discrete event system*. A definition of a discrete event system is provided by Cassandras and Lafortune (1999) as a system whose state space is described by a discrete set, and the transitions – called "*events*" – between states occur only at discrete points in time. The present research argues that, by considering a Web application as a discrete event system, simulation of its functionalities is not only feasible but achievable. Examples of how it was and how it can now be achieved in Web development exist in a closely related field – the hardware development field.

The modeling process for simulation usually follows a set of key activities. Withers et al. (1993) in their work propose a model of this modeling process, in which the main activities identified are: understanding the system and customer; produce a conceptual model; produce a model; use the model; and assess the use of the model. Computer simulation requires that a conceptual model – the model of a system to be simulated – be translated into a suitable and executable model. This process is called *Model Translation* (Shannon 1992, Banks 1999, Banks 2000), and results in what Balci (1994) and Nance (1983) call the *Programmed Model* (Nance 1983, Balci 1990). To simulate Web application design models, a similar model translation process must be performed. Therefore, ambiguities in the interpretation of Web application design models should be kept to a minimum, which is to say that simulation requires a high degree of formalism from the design models. However, the existing design models were designed to tackle the complexities of a Web application and not to promote their simulation, and many of them do not possess a sufficient degree of formalism in their definition to enable simulation. Additionally, Web application design models are usually a high-level abstraction of a system, and they do not address, by their very definition, implementation issues. Therefore, if the purpose of simulating Web application design models is to evaluate their functionalities in the same manner prototyping does, the design models have to be translated into a simulation model capable of bridging the gap between design and implementation, without actually committing to a platform. This translation of design models into an intermediate simulation model has a double benefit: it enables simulation, and allow heterogeneous Web application design models to be simulated. Therefore, no matter which design model is used during the design phase, simulation for functional evaluation is made possible. The validity of this approach is supported by research indicating that no single Web application design model effectively addresses all possible designs. Furthermore, some projects may even use different design models to tackle different parts of the design. By using a simulation model that is not tied to a specific design model, a wide range of design models is covered, potentially contributing to a higher level of acceptance by the Web developers' community.

2.3 Hardware Description Languages

Description languages have long been used in hardware development, especially in digital electronics. The need for a tool that could assist designers with simulations and tests, prior to the system being built, had become evident with the increasing complexity of digital systems. One such examples is the successful VHDL language. This stands for VHSIC Hardware Description Language (VHSIC being an acronym for Very High-speed Integrated Circuit) and became an IEEE standard in 1987, after the United States Department of Defense (DoD) sponsored a program for the development of high-speed digital circuits (Yalamanchili 2001).

There are several factors that have contributed to the enormous success of VHDL. Firstly, the ability of the language to describe a system with different levels of abstraction and views provides a portable and standard means to convey information. Furthermore, VHDL is not technology-dependent, which means that once a system is described it can be synthesised, targeting different platforms and still being functionally equivalent. Moreover, the capability of easily simulating the system with nothing more than its VHDL description allows for testing of the design before any implementation whatsoever. VHDL is structured around two main constructs: (1) the *entity* and (2) the *architecture.* The entity construct defines the input and output ports of a component; the architecture block determines the behaviour of the component. Through separation of these two concepts – structure and behaviour – using a formal language, VHDL is able to perform simulation of the hardware design for the verification of its functional requirements.

Figure 2.5 shows the process of developing a hardware digital system using VHDL. The first step in the process is the definition of the system's specification by determining the system's functional aspects and time constraints. A VHDL model is obtained from this specification, and simulation of the model is now made possible. Simulation is used to verify that the system shows the expected behaviour by using a suitable testbench. If verified, the system is then synthesized. This requires a library of gates, which includes the definition of parameters such as inputs, outputs, and time delays, of several Integrated Circuits. The synthesis step of the VHDL model results in a list of gates to be used in the final product. After the synthesis, simulation is once again used to verify the design, but now gate and propagation delays are taken into account. If the verification succeeds, a layout of the intended system is obtained. Implementation may start as soon as engineers are satisfied with the model's behaviour and performance.

The separation of concerns in VHDL – behavioural, structural and physical implementation – has a correspondence with Web applications, where the behavioural aspect relates to *what* functionalities an application implements, the structural aspect to *how* and *which* components should be implemented, and the physical aspect to *where* it should be implemented. The advantages of a similar tool for web development are easily understandable. Aims such as simulation and automatic synthesis of the code are highly desirable. If such a Web description language ex-

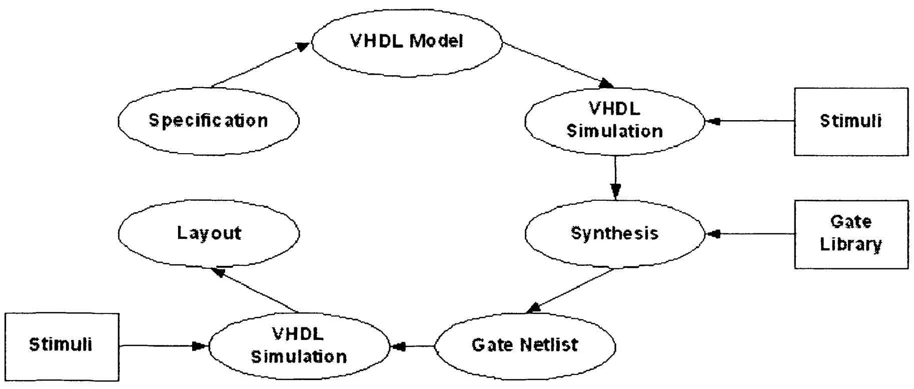


Figure 2.5: Simulation and Synthesis of VHDL Models

isted, Web applications could be tested during the design phase without needing the support of partial implementations.

2.4 How the Present Research Differs from Existing Work

Web application development usually consists of analysis, design, implementation and testing phases, all contributing to achieving a high-quality end-product. The objective of the models and methodologies described earlier in this chapter is to serve as design abstractions and guidelines for the development of Web applications. However, as Germán et al. (1998) noted, these are usually not formally specified. Although the literature offers a great variety of methodologies and models for the development of Web applications, the testing phase is still mostly based on partial implementations of the design. This means that testing cannot start until at least a prototype is built during the implementation phase. Furthermore, developers rely on the produced code, which can itself contain errors, to test the functional requirements of the project.

Other fields of engineering have been successfully using simulation as a testing technique for verification of the system's functional requirements. In particular, the hardware development field has provided developers with the needed theory, models, and tools for that purpose. This allows them to design and test extremely complex systems without the need for prototyping. Consequently, the development cycle does not include the implementation phase until much later in the cycle, when developers are sufficiently satisfied with the designed behaviour of the system. This, inevitably, leads to a faster and less costly development cycle.

At the present state-of-the-art, the only type of simulation for testing purposes that is performed in the Web field concerns Web servers' performance issues. This type of simulation aims at evaluating the performance aspects of Web applications and the servers' systems where they are being executed, namely, analysis of the response time of the Web application subjected to a set of user-like stimuli under specific load, traffic and hardware conditions. However, there is currently no reported research on a framework for the functional simulation of Web application design models.

In order to develop a simulation model, a systematic analysis of the requirements that make simulation possible is needed. The myriad of components, their heterogeneous data interface and data exchange, and the complexity of the user interaction with the application are indeed difficulties that must be tackled before simulation can be achieved. This research argues that the Web application development field has come to a maturity level that enables such a research project. Similarly to what has happened in the hardware design field, where description languages have reached a high maturity level, this research aims to take the testing phase of the Web application development to a new level, where simulation allows effortless, implementation-free, and meaningful verification of the functional requirements. This will make prototyping for functional evaluation unnecessary, bypassing the implementation phase until a high degree of confidence that the design addresses

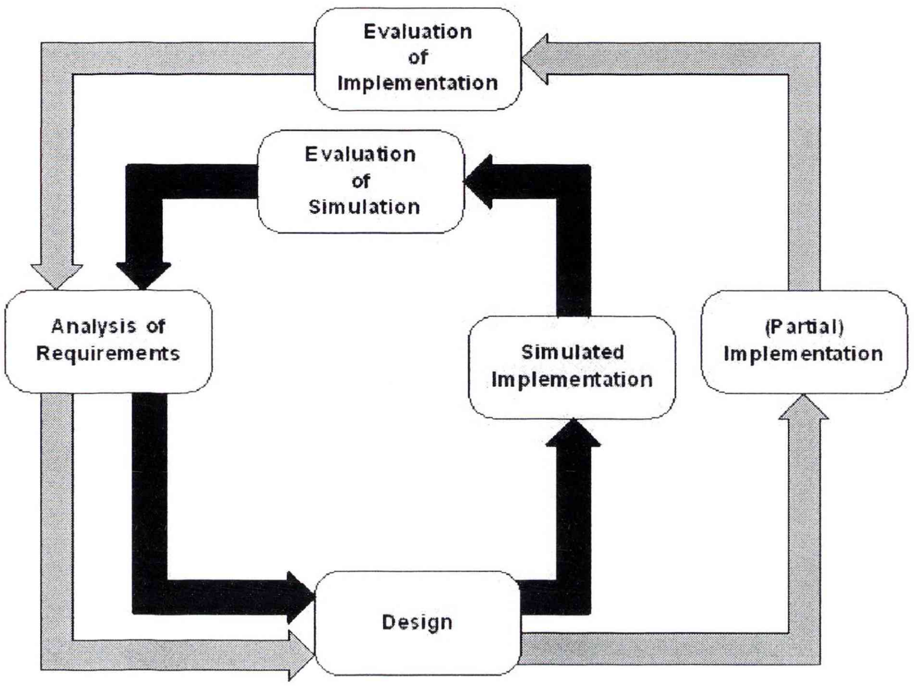


Figure 2.6: The Bypassed Implementation Phase

all the intended functional requirements is reached. Figure 2.6 shows a 'traditional' cycle of software development (represented by the outer gray arrows), and the proposed bypassed prototyping phase through the use of simulation (represented by the inner black arrows). It is not a new design methodology or design model that is targeted; it is the identification and analysis of the requirements that Web application design models must possess in order to enable and support the simulation of Web application designs. Ultimately, with those requirements clearly identified, it is our belief that design models can use them in order to promote simulation, thus enormously contributing to aiding and simplifying the development process.

Summarising, the hypothesis of the present research can be stated as:

"It is possible to simulate Web applications based on their designs. Further, the simulation provides similar information with regard to evaluation of functional requirements as would an evaluation based on an actual implementation."

Chapter 3

Methodology

The previous chapter focused on existing Web application design models and development methodologies. In particular, the testing phase of the software development cycle was found to be time- and resource-intensive, and some of the most difficult and lengthiest phases of the whole cycle. The hardware development field has been one step ahead, with a theoretical framework and tools that support the simulation of system designs, for validation and verification purposes. This research argues that an identical approach can be adopted with similar benefits during the development of Web applications. Several questions stem from the proposed hypothesis, and will need to be answered in order to test it. These questions are:

RESEARCH QUESTION 1 – *What aspects of a Web application should be simulated?*

RESEARCH QUESTION 2 – *What simulation model captures those aspects?*

RESEARCH QUESTION 3 – *Is it possible to simulate the model to evaluate its functional requirements?*

RESEARCH QUESTION 4 – *How do simulation and implementation of a Web application design for functional requirements evaluation compare?*

These research questions address all the issues that are being argued in the hypothesis. The first research question investigates what aspects of a Web application should be modelled so that its simulation provides sufficient information to evaluate its functional requirements. Answering this question will contribute to define the relevant information a Web application design model must provide for a meaningful simulation for functional evaluation purposes. Furthermore, the answer will allow us to identify the relevant aspects of Web applications to be modelled and separate them from the non-essential, therefore preventing the simulation model to become too complex to be simulated.

The second research question goes a step further into defining a simulation model of a Web application. After enumerating the aspects of Web applications to be modelled, this question addresses the issue of whether or not a suitable model based on these aspects can be defined. This is a especially important question since its outcome will dictate the capabilities of the simulated model to provide enough information for the functional requirements evaluation.

The third research question, which logically follow the last two, directly addresses the issue of the feasibility of the simulation of the defined model. The hypothesis clearly states that simulation of Web application design models for functional requirements evaluation is possible and, by answering this research question, the assessment of that statement can be done.

The answer of the fourth and last research question will provide information on how simulation and implementation for functional requirements evaluation compare. Since it is argued that both methods provide similar functional information, answering this question will allow to prove or disprove the validity of the hypothesis.

3.1 Research Strategy

To answer the research questions and to assess the validity of the hypothesis, a research strategy must first be defined. This will guide the research procedures, data gathering and analysis, and hopefully will allow meaningful conclusions to be drawn in a systematic way. Research methodology can be defined as:

DEFINITION 6 *Research Methodology – "A recommended collection of philosophies, phases, procedures, rules, techniques, tools, documentation, management and training for developers of information systems"* (Maddison 1983).

To describe a Web application from a functional perspective, this research will investigate what are considered the basic building blocks of the existing design models. Moreover, simulation will use these components to emulate the behaviour of a possible implementation, by identifying their functional characteristics. Furthermore, a mechanism that bridges the gap between design and its (simulated) implementation will be defined.

To define a model of Web applications that allows simulation, an investigation of their structural organisation, behavioural characteristics, and internal components has to be conducted. This needs to be done from a Web development and simulation perspective. Definition of the simulation model from a Web development perspective will be performed by examining what the involved key design concerns are and how existing design models address them. The definition of the simulation model will be done by following the phases and processes usually found in a simulation study. The methodology proposed by (Balci 1987), which consists of a ten-phase life cycle, will be particularly useful throughout the simulation study not only to successfully achieve a suitable simulation model, but also to enforce the validity of the simulation model results. The ten phases of Balci's (1987) life cycle are: (1)Communicated Problem; (2)Formulated Problem; (3)Proposed Solution

Technique; (4)System and Objectives Definition; (5)Conceptual Model; (6)Communicative Model; (7)Programmed Model; (8)Experimental Model; (9)Simulation Results; (10)Integrated Decision Support. The first four phases have already been completely or partially addressed in Chapter 2 – Literature Review, and some will be further refined in the following Chapters. The remaining six phases will be addressed in the next three Chapters, with especially emphasis put on the definition of the Conceptual Model, Experimental Model, and presentation of the Simulation Results.

To answer the third research question, the issue of the feasibility of the simulation process has to be assessed. Arguing that simulation of Web application designs for functional evaluation is feasible requires the definition of a formal model with a well-defined purpose and contents, able to mimic their behaviour, and reducing their complexity to manageable levels while retaining enough elements to serve as a faithful and meaningful representation. Thus, an initial study of the basic building blocks of Web application design models that possesses functional meaning will be performed, which will contribute to the definition of a simulation model. Furthermore, the modelling of these building blocks will determine the simulation's capabilities and limitations. Investigating what these are will determine the scope of the simulation and its applicability to the functional requirements evaluation of Web application designs, since one source of information from the simulation results will be the attribute values of the modelled building blocks. It does not suffice, however, to exhaustively define a description language and associated simulation model when proof of its practicability is required; it requires demonstration that the simulation process, based on the developed theory, can be implemented and used for the intended purposes. This can be asserted if a computer application can be produced from the defined simulation model.

The underlying concept of the present work is that it is possible to describe a Web application in a way that supports simulation for functional evaluation purposes directly from the design. This would eliminate, or at least reduce, the need for partial implementations. The functional requirements of a Web application design capture the intended behaviour of a system. The hypothesis claims that simulation and implementation for functional requirements evaluation can produce similar information, namely design errors identification and better operational understanding of the final product. To support this claim, the research has to demonstrate that, for identical design models, simulation and implementation can similarly support simulation of its functionalities. Hence, a comparison of the results of the two techniques is fundamental; and, in order to better isolate the subject from external and unidentified factors, a controlled *experiment* method was selected.

Experimental research deals with the "cause and effect" phenomenon (Walliman 2005). What is being investigated in this type of research is causal relationships, and how *independent variables* affect the *dependent variables.* In other words, manipulation of the independent variables will cause an observed *effect* on the dependent variables. The term *treatment* refers to adding or removing stimuli to or from a group, in order to evaluate the effect. There are several ways to manipulate the independent variables:

- *Presence or absence technique* – a treatment is given to one group, while another group is left untreated
- *Amount technique* – distinct groups receive different levels of the same treatment
- *Type technique* – distinct groups receive different treatments.

Furthermore, experimental research classifies experiments into three main classes: *pre-experimental*, *true experimental*, and *quasi-experimental*. Differences among them are mainly in the manner the samples are obtained, which may lead dangerously to dissimilarities, and in the degree of control of the variables. Pre-experimental designs make assumptions about the cause-effect relationship, despite lacking complete control over the variables. True experiments enforce that the groups subjected to the treatment are identical which, if not observed, is called a quasi-experimental design.

The strongest aspect of experimental research when compared with other research methods, is the researcher's control over the research setting, which enables the isolation of the treatment and rules out confounding effects; this allows the elimination of alternative explanations and leads to a strong internal validity (Wade and Tingling 2005). Jarvenpaa (1988) argues that laboratory studies are as valid as many other methodologies such as case studies and action research. Furthermore, Dennis and Valacich (2001) state that, since "all research methods are imperfect", it is simply not valid to claim that surveys or field studies are better suited for research in Information Systems than experimental research.

Evidence of experimental research in the software engineering field is abundant. Do et al. (2004) have conducted a survey of recent articles (from 1994 to 2003) in four major venues often recognized as pre-iminent in software engineering research: IEEE Transactions on Software Engineering, ACM Transactions on Software Engineering and Methodology, the ACM SIGSOFT International Symposium on Software Testing and Analysis, and the ACM/IEEE International Conference on Software Engineering. They found that, of those that address the software testing topic, nealry half are based on an empirical approach. Additionally, Perry et al. (2000) argue that empirical studies are the key to evaluation of new methods and tools from a cost-benefit perspective. Basili (1996) goes further, when stating that "software engineering is a laboratory science", involving an "experimental component to test or disprove theories, (and) to explore new domains". Brilliant and

Knight (1999) argue that "there is a significant need" for experimental research for hypothesis testing and demonstration of technology in software engineering, and Shull et al. (2001) claim that there is an increasing trend in using this methodology, which they consider to be useful for testing methods for the software development process.

What is being investigated is each technique's contribution for the evaluation of the functional requirements. By conducting a controlled experiment on both simulation and implementation, the relevant results for functional evaluation can be compared, since isolation of the treatments and avoidance of confounding effects can be achieved.

3.2 Research Design

The objectives of the research design are associated with each of the Research Questions (RQs). These objectives are:

- Identify the important functional constituents of a Web application – RQ1
- Develop a description language that supports the evaluation of the Web applications' functionalities – RQ1
- Clearly define how the modelled functional constituents interact with each other – RQ2
- Develop a suitable simulation model for Web applications – RQ2
- Implement a tool based on the simulation model and description language – RQ3
- Investigate and identify the capabilities and limitations of the simulation – RQ3

- Experiment with the tool to evaluate its capabilities – RQ3
- Compare the experiment's results with the "traditional" implementation for functional evaluation – RQ4
- Allow conclusions to be drawn regarding the similarities and differences in the two treatments – RQ4.

What is being investigated is the comparison between two methods of evaluating the functional requirements of a Web application design – the "traditional method" (implementation), and the proposed one (simulation). Therefore, the experiment needs to compare both methods' capabilities and limitations, and evaluate the hypothesis claim of both being able to provide similar information. By "similar" it is meant that verification of a functional requirement is possible by both treatments, based on the analysis of the results. The independent variable here is the testing method, whereas the dependent variable is the obtained data that contributes to the evaluation of the functional requirements. In order to obtain meaningful results from the experiment, the two methods must be used with identical Web application designs. In the experimental research field, this is called a *true experiment*, which is defined as those in which tests are conducted on identical groups (Walliman 2005, p. 119).

The *implementation treatment* will use the "traditional" method of implementing the design for the purpose of evaluating the functional requirements. Therefore, the treatment results will be a sequence of Web pages with just enough elements to properly evaluate their functionalities. Furthermore, snapshots of the application database content will be used to evaluate the requirements. To restrict the research to a comparison of the information provided by each treatment, it will be assumed that the implementation code is a direct translation of the Web application design and thus error-free in that sense. This means that no errors will be introduced in

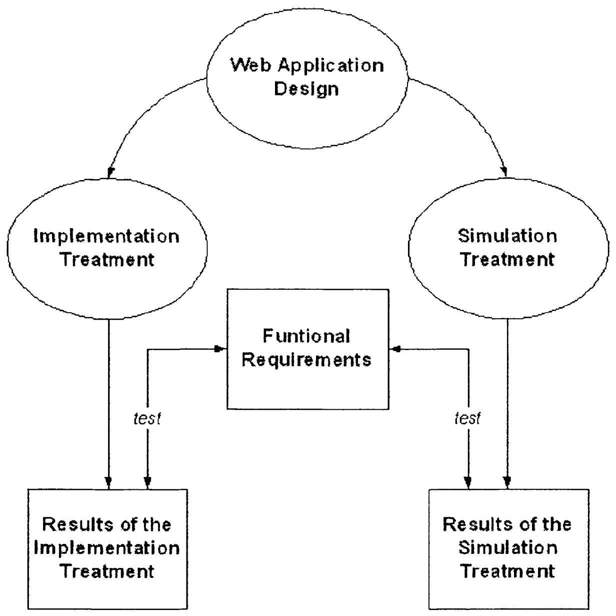


Figure 3.1: The Experiment

the implementation code by misinterpretation of the design. This assumption can be made with a high degree of certainty if the design model in which the Web application is expressed is formal in its definition and has a well-defined code mapping rules. However, since it will be implemented from a specific Web application design, it may contain conceptual design errors; this means that, although the code is correct and corresponds to what the design model expresses, the design may not properly model the goals of the functional requirements. These deliberately introduced design errors or fault seeding, will contribute to assess both treatments' suitability for functional evaluation. On the other hand, the *simulation treatment* will use the developed framework for the evaluation of the same set of functional requirements and on the same Web application design. This ensures that a valid comparison of both treatments can be conducted, and the suitability of each treatment be asserted (see Figure 3.1).

Prevented from experimenting with all possible designs, a decision was made to adopt a strategy of selecting a Web application that contains a broad spectrum of functional elements. This allows the research to focus on the evaluation of the functionalities of the design, rather than on aesthetic or informational issues. The selected experiment will be conducted on an e-commerce Web application design which models a music store. This class of applications presents the most functional-diverse characteristics, since it involves database accesses, personalised roles and access, and addresses security concerns. This type of application is typical on the Internet, and usually they display a similar functional design. Examples of these Web-based stores are found in Amazon©(Amazon 2005), Virgin©(Virgin 2005), and Sony©(Sony 2005). Although the purpose of the chosen sample is not important, the different functional elements within the application are. If a comprehensive set of these elements is achieved, generalisation to other designs can be asserted with a higher degree of confidence, therefore contributing to a high degree of the external validity of the experiment.

One other aspect that should be addressed is the adopted design model for the experiment. Prevented from experimenting with all the available design models, one had to be selected on the basis of its formalism, usage within the Web developing community, and suitability for describing Web applications. Of the existing design languages, WebML has been increasingly gaining momentum within the academic and industrial community. It is a model that is formally defined, especially well designed to tackle data-intensive applications, and with a sound theoretical background and tools (WebRatio 2005)(WebML 2005). However, generalisation of the claims of the hypothesis to other design models has to be made with caution. Indeed, over-generalising is a potential pitfall to avoid when using experimental research. This issue will be dealt with in a following section, when addressing the threats to validity within the experimental research scope (Section 3.4).

The two treatments, run on the same design, will provide information on their capabilities and limitations to evaluate functional requirements – this is what experimental research calls *type technique*. For this purpose, a set of functional requirements will be defined and evaluated. The selection of the set of requirements will be made to cover a wide range of functional evaluation and, naturally, based on the specifics of the selected Web application design. Definition of the functional requirements concept can be, sometimes, unclear; however, in this research functional requirements are considered as an artifact that captures the intended behaviour of a system. Although they may sometimes be expressed in a narrative format, there are more formal approaches to deal with them. For instance, UML, and in particular its *use cases diagram*, provide a semi-formal structure to capture functional requirements (Cockburn 2001). Use cases, as used to describe the functional requirements of a system, can be defined as:

DEFINITION 7 *Use cases: "capture who (actor) does what (interaction) with the system, for what purpose (goal), without dealing with system internals. A complete set of use cases specifies all the different ways to use the system, and therefore defines all behaviour required of the system, bounding the scope of the system"* (Malan and Bredemeyer 2001).

Within the Web engineering realm, an *actor* is a class of users or roles they can play. These actors *interact* with the system via a well-defined set of actions. Examples of these are starting a new session, entering data, following a link, and triggering a script. Finally, the *goals* of an actor are very much dependent on the services the system provides, examples of which are buying a service or item, and accessing certain information. The selected functional requirements to be evaluated have to be a representative set of the different interactions an actor can have with the Web application.

3.3 Data-gathering and Analysis

The nature of the hypothesis and associated research questions is such that a quantitative measurement of the involved factors can be, at least, non-conclusive and, at the other extreme, misleading. This is because what is being investigated is inherently non-quantifiable, and what is being sought is an explanation of the phenomenon and its implications. Stating that the developed framework will allow simulation of 'X' Web application designs, and the evaluation of a 'Y' number of functional requirements carries almost no real information, or, at least, none that can be put to a useful purpose. The results of that research would only indicate the success rate of the simulation treatment, but would give little information on the characteristics of the treatment itself. On the other hand, investigating what type of design models, which features and limitations can be simulated, what type of functional requirements can be evaluated, and how implementation and simulation compare, is far more interesting and useful to a Web engineer. As Higgs (1998) stated, if seeking to understand why a certain effect has been accomplished, then a "more descriptive and exploratory mode of research (that is, qualitative research) is preferable".

As Gittins (1999) notes, quoting Holloway's (1997) book *Basic Concepts for Qualitative Research*, Qualitative Research may be defined as:

DEFINITION 8 *Qualitative Research – "A form of social inquiry that focuses on the way people interact and make sense of their experiences and the world in which they live" (Gittins 1999).*

Historically, qualitative research was first developed and used in the social sciences, where social and cultural phenomena were studied. Notwithstanding the differences between social sciences and engineering, several articles have been devoted to the benefits and uses of this research method in the software engineering field, as found in the work of Fishwick and Zeigler (1992), Wixon (1995), Kelly and Shepard (2002), Myers (2003), Marcos (2005), and Zannier and Maurer (2005).

The results of the experiment will produce two sets of data: one that comes from the simulation of the selected scenario, and the other from its implementation. This data represents the Web application at specific points in time and corresponds to a particular state of its internal functional constituents. To this purpose, an *Empirical Observation* of the results of the experiment is crucial. An *empirical study* can be defined as a test to compare what we believe with what we observe (Perry et al. 2000, p. 347), and the collection of *empirical data* or, within the qualitative research context, *empirical material*, is characterised by the collection of a fairly large amount of data to derive a conclusion.

The data to be collected concerns the groups of design elements that provide and contribute to evaluate the functional requirements of Web applications. These are related to how the application reacts to a stimulus, what its behaviour is in the presence of a user, and what the allowed interactions are between users and application. Since it is a comparison between the implementation and simulation treatments, the data collected should be that traditionally used to assess prototypes. This data is related to the model of Web applications used for testing purposes. For instance, Liu et al. (2000) consider three main objects of a Web application, namely: *client pages*, *server pages*, and *components*. Ricca and Tonella (2001) propose a model of a generic Web application structure in which the central entity is the *Web page*. These can be *static* or *dynamic*, depending on the way content is generated, and navigation is provided by the association *link*, which connects Web pages. Lucca et al. (2002) refine and further detail this model, proposing a taxonomy of the elements to test in *pages*, *active elements* (scripts and applets), *links* between components, and diverse *data* elements such as cookies, variables and databases. Other researchers have a slightly different view; for example, Deng et al.'s

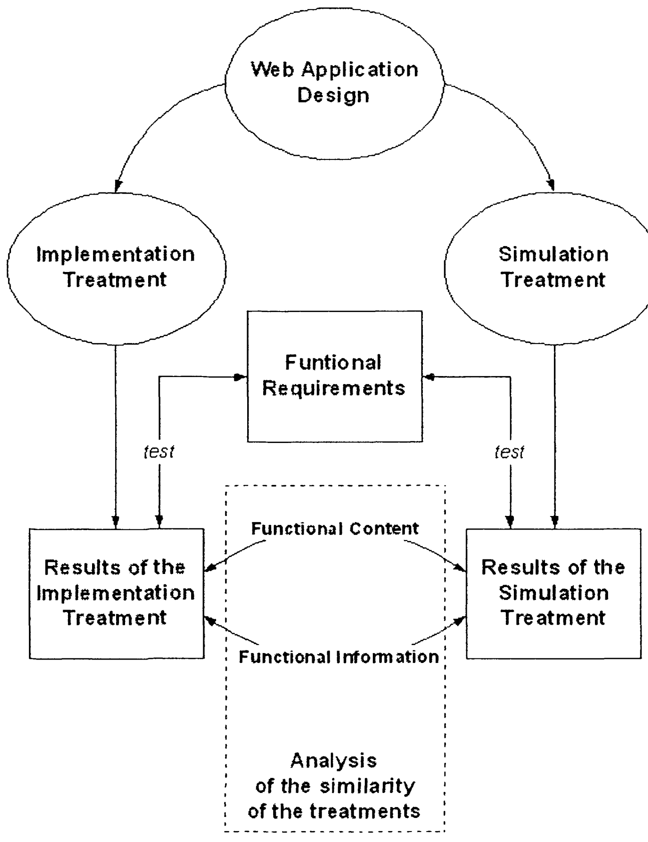


Figure 3.2: The Experiment – Data Analysis

(2004) model considers three logic layers of Web database applications, namely: *Database Management System*, the *application logic* or *scripts*, and the *Web browser* or *interface*. But whatever Web application model is adopted for testing purposes, its elements fall into one of four main groups: *pages*, *links*, *scripts*, and *data*, which constitute the necessary elements to implement functionalities in Web applications. During the course of the experiment, both structure and content of each of these elements will be gathered. The pages being displayed, the links being traversed, the scripts being executed, and the content of the application's data, all contribute to define, from a functional perspective, a Web application state at a specific point in time.

Individually considered, the two sets of data originating from the two treatments do not provide all the information needed to evaluate the hypothesis; however, in conjunction with an analytical comparison of the information they contain, the degree of similarity of the two methods can be appraised. We propose two measurements for the evaluation of the data results – *Functional content* and *Functional information*. They will provide two distinct and complementary analyse of the gathered data (see Figure 3.2). These two measures can be defined as:

DEFINITION 9 *Functional Content – the observable factors a treatment provides and their structure and value.*

DEFINITION 10 *Functional Information – the level of contribution of the observable factors for the evaluation of functional requirements.*

Functional Content will qualitatively evaluate each treatment result from a page, link, script, and data perspective. This means that from each of these perspectives, an analysis of what a developer can observe from the results will be conducted. Furthermore, the structure and value of each of these observable factors will serve to compare the content of each treatment. One important aspect is the identification of the active (executing or displayed) elements of each factor at any given time during the treatment. This means that, during each treatment, the identification of the set of displayed pages, followed links, executing scripts, and accessed data will be registered. Additionally, each factor contains specific information which is relevant for functional evaluation. For instance, pages can contain other relevant functional elements such as links, scripts, and data. Links, on the other hand, carry valuable information in their parameters. Furthermore, script processing often have an influence on other elements of the design, and the result of the script execution greatly determines the behaviour of the Web application. Finally, the content of the data elements will be registered during each treatment, since it is often used by scripts for their workflow and databases are usually the preferred method employed by Web developers to store user and application-related information.

Functional Information is a less concrete concept to define and evaluate than the *Functional Content*. It is related to the amount of information possible to be gathered from the treatment, which directly or indirectly contributes to the evaluation of a functional requirement. This information can be as diverse as the format in which the results of the treatments are presented, or the degree of difficulty in administering the treatment. There will be no attempt to quantify this measure, but rather to make a qualitative critical analysis of the factors observed from the two treatments. The *Functional Information* measure is further divided in three parameters: *Difficulty in administering the treatment*, *Format of the treatment results*, and *Adequacy of the results for functional requirements evaluation*. The first parameter is related to the amount of effort needed by Web developers in executing the treatment, namely: the necessary coding, expertise, and all the additional work involved in the preparation and conduction of the treatment. The second parameter of this measure, the *Format of the treatment results*, evaluates, from a Web developer perspective, how the treatment results are displayed. The third and last parameter, the *Adequacy of the results for functional requirements evaluation*, assesses the level of suitability of the treatment results for the evaluation of the proposed functional requirements. Table 3.1 summarises the aspects of the experiment's results which will be the object of analysis.

Functional Content	
Pages	Displayed set Page (functional) content
Links	Followed set Parameters content
Scripts	Executed set Outcome of their execution and its influence on other functional elements
Data	Accessed set Content
Functional Information	
Difficulty in administering the treatment Format of the treatment results Adequacy of the results for functional requirements evaluation	

Table 3.1: Qualitative Data Analysis

3.4 Threats to Validity

The gathered data from the experiment should reflect the influence of the controlled variables and allow generalisation of results. There are several threats to the validity of a research study, namely:

- Construct validity
- Internal validity
- External validity

Construct validity is concerned with the relationship between the measures taken and the goals of the study. Furthermore, it involves generalising from the study to the concept and theory underpinning it.

DEFINITION 11 *Construct validity – "the degree to which inferences can legitimately be made from the operationalisations in your study to the theoretical constructs on which those operationalisations were based"* (Trochim and Land 2004).

Threats to construct validity can be enumerated as:

- Inadequate Preoperational Explication of Constructs – poorly defined constructs may pose a threat
- Mono-operation Bias – single treatments may lead to bias of the results
- Mono-method Bias – single measurements type may lead to insufficient analysis
- Interaction of Different Treatments – other (external) treatments may be contributing to the observed results
- Interaction of Testing and Treatment – is the testing part of the treatment?
- Restricted Generalisability Across Constructs – a treatment may be causing other undesired effects
- Confounding Constructs and Levels of Constructs – the label of the study does not completely describe what was implemented.

Internal validity is concerned with the causal relationship of an experiment, in particular between the dependent and independent variables. In other words, internal validity means that evidence exists that what was carried out in the study caused what was actually observed, and that no external variables, known or unknown, had any influence on the results.

Definition 12 *Internal validity – "The quality of data gained from true experimental design should genuinely reflect the influence of the controlled variables"* (Walliman 2005, p. 294).

In Campbell and Stanley's (1966) classic book on experimental research, identification of the factors that can pose a threat to the internal validity is made:

- History – results affected by external events between the pre- and post-test observations
- Maturation – changes due to the normal passage of time
- Testing – contamination of the subject of the experiment due to extensive pre-testing
- Instrumentation – changes in the measurement methods during the experiment
- Statistical regression – extreme results due to subject retesting
- Selection – bias can occur if the subjects of the experiment are not functionally equivalent
- Experimental mortality – dropout of the subjects of the experiment leads to biased results of the remaining sample
- Selection interaction – bias of the results, due to the selection method interaction with other threats.

Finally, *external validity* can be defined as:

Definition 13 *External validity – "is the quality of an experimental design such that the results can be generalized from the original sample to another sample and then by extension to the population from which the sample originated" (Salkind 2003).*

Cohen and Manion (1994) enumerates the factors that may affect external validity:

- Vague identification of independent variables – impossibility replicating the experiment
- Faulty sampling – selection of the available population does not represent the entire population
- Hawthorne effect – unusual subject's reaction due to knowledge of being observed
- Inadequate operationalisation of dependent variables – incorrect generalisation of the results to a broader scope than that of the experiment
- Sensitisation to experimental conditions – manipulation of the results by the subject
- Extraneous factors – unnoticed effects on the results

3.4.1 Enforcing the Validity of the Experiment

Trochim and Land (2004) argue that, in order to minimise the threats to validity, one or a combination of several methods may be employed:

- By argument – arguing that the threat in question does not apply to the specific experiment
- By measurement or observation – ruling out a threat by demonstrating minimal influence in the cause-effect relationship
- By design – adding treatment or control groups
- By analysis – using, for instance, statistical analysis

- By preventive action – by knowing beforehand about the threat, it can potentially be minimised or even ruled out.

Construct validity is especially concerned with the relationship between theory and observations of the results. This means that evidence exists that the measurements taken from the two treatments of the experiment are the adequate ones to evaluate the functional requirements, thus allowing a valid comparison of information to be made. A valid approach to this problem is to identify which factors are present in functional requirements specifications. If one of these factors can be observed and either qualitatively or quantitatively measured during the course of an experiment, then chances are that it should be part of the selected set of measurements; as Conallen argues, a requirement must be "objectively" verifiable for it to belong to the requirement specification (Conallen 2000, p.115). The set of relevant factors for the experiment has been constructed by identifying, in the literature, which common elements are used to specify functional requirements. Sommerville (2004, p. 155) argues that the Unified Modelling Language is a *de facto* standard for object-oriented modelling, and that "UML use-cases and use-case-based elicitation is increasingly used for requirements elicitation". Although some researchers argue that UML is not the ultimate answer to model requirements (Glinz 2000, Fuentes et al. 2003, Bell 2005), it is still considered by many as one of the best modeling notations to semi-formally specify functional requirements (Conallen 1999, Baresi et al. 2001) and, as such, is one good source for the investigation of the relevant factors. When translated to the Web development realm, identification of these factors assures, with a high degree of certainty, that what is being measured is in fact contributing to the evaluation of the functional requirements. Consequently, a comparison of the benefits of each treatment is made feasible, validating the relationship between the measurements and the underpinning theory.

Experimental research has strong internal validity since the researcher fully controls the experiment (Wade and Tingling 2005). To contribute to this validity, the treatments will be performed on exactly the same design, ruling out differences between subjects as a possible explanation for the degree of information provided by each treatment. It could be argued that implementation and coding skills would pose a threat to the internal validity of the experiment, since it can strongly influence the implementation treatment results; however, if the design model selected provides a formal definition of the mapping rules for translating design elements into runnable code, then this threat is mostly ruled out since intervention of the researcher in the coding will be kept to an almost 'mechanical' task, where neither background nor skills play an important part. The design model selected – WebML – has these rules well-defined and code synthesis capabilities is one of its most interesting and distinguishable features, therefore implementation skills do not significantly pose a threat to the internal validity. One other factor to be considered as threat is the testing skills involved in the evaluation of the functional requirements of both treatments. This threat has been minimised by carefully and clearly describe the functional requirements to be evaluated and the test cases used during their evaluation. Lastly, since the subjects are not influenced by a temporal dimension, history and maturation threats simply do not apply.

Ensuring the external validity in an experimental research is often difficult to completely achieve. This is because generalisation of the results from an experiment to a wider population of subjects is, if not impossible, extremely hard to prove. However, by carefully planning the experiment, some generalisations from the results are possible with a reasonable degree of confidence, although the context in which these are made has to be defined. In the present research, the two major threats to external validity are faulty sampling – the subject of the experiment does not represent the entire population – and inadequate operationalisation of dependent variables – wrongly over-generalisation of the experiment results . This is due to the experiment being conducted on a particular case of Web application design, and attempts to generalise from the results to the whole spectrum of applications may be unreasonable. There is certainly no attempt in this research to extrapolate to every conceivable design and design model, which would be unachievable due to the nature of the problem. However, by planning the experiment in a way that a broad range of functional elements are present, it will be possible to assess whether that generalisation is a realistic one. Furthermore, claiming that evaluation of any set of functional requirements using the simulation treatment produces the same information as the implementation one, is also too far-fetched, especially because there are as many functional requirements as designs, and evaluating them all is simply not possible. Nevertheless, by considering a good representation of the universe of possible functional requirements, a high degree of confidence in generalising the results may be attained. These eventual generalisations, their scope and constraints, will be presented when discussing the research results.

3.5 Summary of the Experiment

Table 3.2 summarises the research design by enumeration and characterisation of the research methodology, the experiment treatments performed, and the data collected and subsequent analysis.

Research	
Methodology	Experimental Research
Type	True experiment
Treatment technique	Type technique
Treatments	
Type of treatments	Two treatments: - Implementation treatment - Simulation treatment
Groups	Two identical Web application designs
Data	
Data gathered	Empirical Observation Functional-relevant information from pages, links, scripts, and data
Data analysis	Qualitative approach Critical analysis of the *Functional Content* and *Functional Information*

Table 3.2: Summary of the Research Design.

Chapter 4

The Web-design Simulation Model

This chapter presents the definition of a framework for the simulation of Web application design models, constituting the foundations of the research. This will address the research questions of what aspects should be simulated and what simulation model captures those aspects.

4.1 The Simulation Study

Simulation is an analysis tool that supports decision-making (Robinson 2003). Shannon (1998) argues that it is one of the most powerful analysis tools for the study, analysis and evaluation of complex systems in different scenarios. It has been commonly used in science as a means to observe and better understand systems operation. Examples of its field of usage range from manufacturing (McLean and Shao 2003) to airspace traffic simulation (Shortle et al. 2003). This helps engineers to assess systems and experiment with new designs and scenarios, by providing the means to ask *what-if* questions. It is defined as a process whereby a given model behaves or operates as a real-world system when a specific set of inputs is present (*IEEE standard glossary of software engineering terminology – 610.12-1990* 1990).

There are two main modes in which a simulation can operate, related to how time advances during a simulation run: *continuous-event* and *discrete-event* simulation (Banks and Carson 1986). The clearly noticeable aspect that distinguishes them from each other is in the manner in which they affect the system state variables. A state variable defines a specific characteristic of a system, and the set of all these variables defines the state of the system. Both continuous and discrete modes are driven by events, but the discrete-event type changes the state variables at those discrete points of occurrence of events, whereas in continuous-event simulation time advances regularly and the system is constantly checked for changes in its state.

One of the most important issues in the design of a simulation model is the definition of the phases and processes a simulation study goes through. Balci (1987) proposes a ten-phase life cycle of a simulation study (see Figure 4.1). These phases are represented in the figure by oval symbols connected by dashed arrow lines representing processes; the solid arrows represent the credibility assessment stages. Balci further argues that the life cycle is iterative in nature and that it should not be interpreted as strictly sequential.

The first of the processes of the simulation study as proposed by Balci is the *Formulation of the Problem*, which consists of the definition of the problem in a clear and organized manner. In this research, the problem formulation has been done in Chapter 2 – Literature Review, when describing the problem of verifying the functional requirements of Web application designs. The *Investigation of Solution Techniques* is the process in which the solution that leads to highest expected benefits/cost ratio is selected. The adoption of a simulation approach to solve the problem was substantiated by investigating the use of simulation in other engineering fields, namely the hardware design field. The *System Investigation Process* examines the interdependencies and organization characteristics of the system. Investigation of how existing Web application design models manage design complexity by their decomposition into smaller subsystems was done in Chapter 2 and has contributed to the system investigation process. Further investigation of these issues will be done during the present Chapter in order to complement the study.

The *Model Formulation Process* consists of the definition of the objectives, content, stimuli, outputs, assumptions and simplifications of the conceptual model. At the end of this process, a complete definition of the conceptual model is achieved. The result of the *Model Representation Process* is the "Communicative Model", which is the representation of the Conceptual Model in suitable form to be communicated to humans; examples of some of these forms are graphs, flowcharts, structured English, pseudo-code, and entity-cycle diagrams. All issues involved in the Model Formulation and Model Representation Processes will be addressed thoroughly in the present Chapter and Chapter 5 – The Web-design Description Language.

The *Programming Process* consists of the translation of the Communicative Model into a *Programmed Model* (Computerised Model in Sargent's (2003) nomenclature). The programmed model is an executable representation of the communicative model in a programming language. This process is addressed in Chapter 6, when presenting the Web-design Simulation Tool. Finally, the *Design of the Experiment*, *Experimentation*, *Redefinition*, and *Presentation of the Simulation Results Processes* are all addressed in Chapter 7 – The Experiment, where the definition of an experiment, the use of the programmed model, and the interpretation of the results are done.

Sargent (2003) argues that the *conceptual model* is a representation of the *problem entity* – the system to be modelled – in a mathematical, logical, or verbal manner which leads to its later implementation in a computer environment – the *computerised model* or *simulator* (see Figure 4.2). Robinson (2003) expresses the

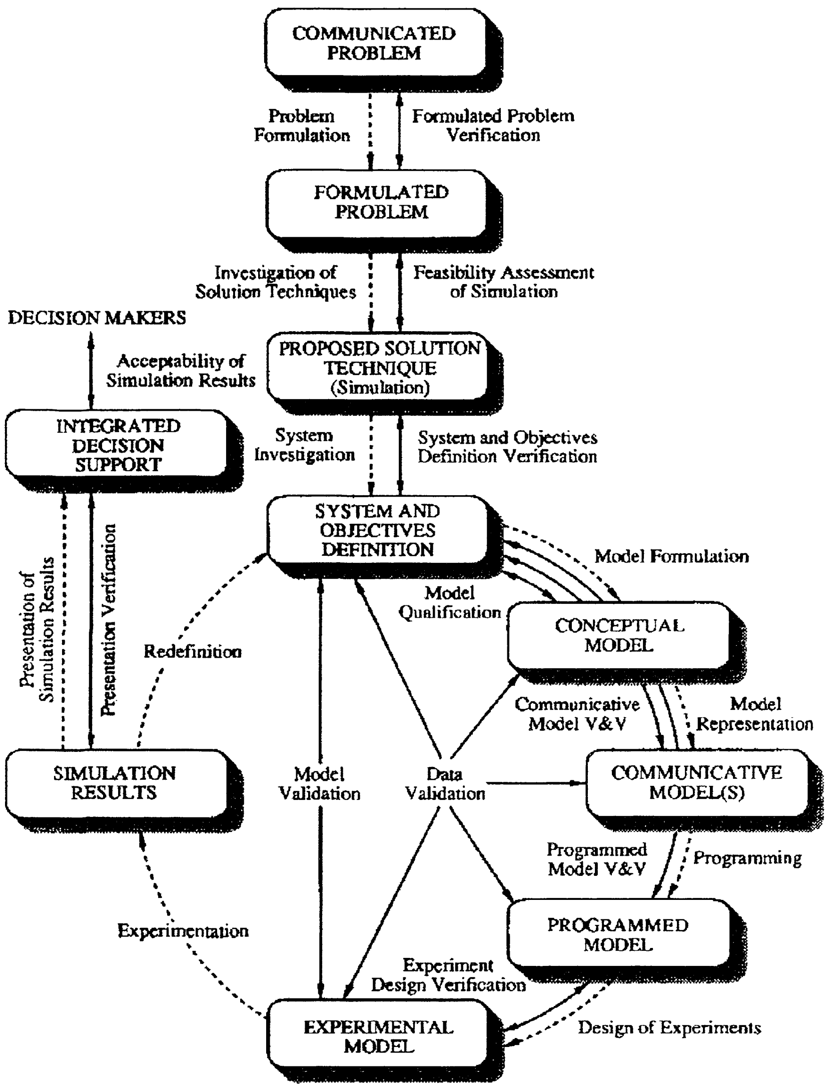

COMMUNICATED PROBLEM
Problem Formulation
Formulated Problem Verification
FORMULATED PROBLEM
Investigation of Solution Techniques
Feasibility Assessment of Simulation
DECISION MAKERS
Acceptability of Simulation Results
PROPOSED SOLUTION TECHNIQUE (Simulation)
INTEGRATED DECISION SUPPORT
System Investigation
System and Objectives Definition Verification
SYSTEM AND OBJECTIVES DEFINITION
Model Formulation
Model Qualification
CONCEPTUAL MODEL
Presentation of Simulation Results
Presentation Verification
Redefinition
Communicative Model V&V
Model Representation
SIMULATION RESULTS
Model Validation
Data Validation
COMMUNICATIVE MODEL(S)
Programmed Model V&V
Programming
PROGRAMMED MODEL
Experimentation
Experiment Design Verification
EXPERIMENTAL MODEL
Design of Experiments

Figure 4.1: The Life Cycle of a Simulation Study (Balci 1987)

definition of a conceptual model as a description of the simulation model entailing its *objectives*, *content*, *inputs*, *outputs*, *assumptions* and *simplifications*, without committing to a specific software language or computer platform. The *objectives* state the purpose of modelling the system – what its aims are and for what reason the model is being designed. Another important aspect of the conceptual model is its *contents*. This relates to the components that are represented by the model, their interfaces and interconnections, which mimic the behaviour of their real-world counterparts. The *inputs* of the conceptual model refer to all the signals that may be fed into the simulator – the *stimuli* – which will lead to a specific system state and simulation result. On the other hand, the simulation results or *outputs* are the data resulting from a simulation run. The *assumptions* are related to certain aspects of the real system being modelled that may not be fully understood (representing uncertainties), or empirical truths that may be incorporated into the conceptual model. Finally, some *simplification* may be made, without which the model would become extremely complex if not impossible to implement and manage.

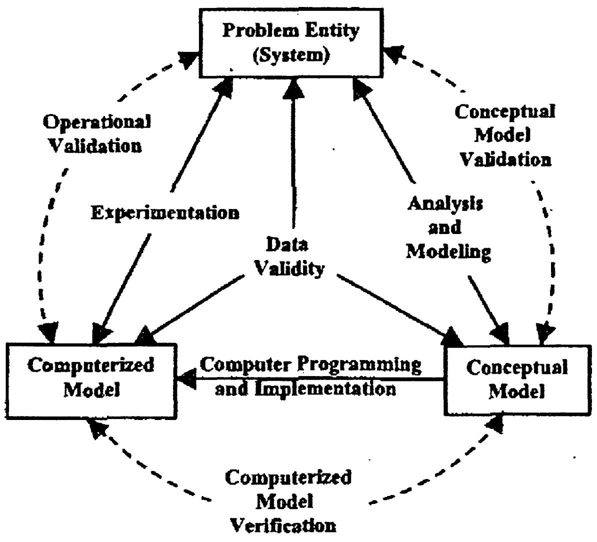


Figure 4.2: The Simulation Modelling Process (Sargent 2003)

The following sections address the definition of the Conceptual Model of the proposed Web-design Simulation Model. Definition of the objectives, content, stimuli, outputs, assumptions and simplifications of the conceptual model are presented. These address the *Model Formulation* and *Model Representation Processes* of the simulation study life cycle as proposed by Balci (1987).

4.2 Objectives of the Simulation Model

The main purpose of developing the Web-design Simulation Model is to enhance the testing phase of the Web application development cycle. By tightening the relationship between the design and testing phases, prototyping for verification purposes is bypassed (see Figure 2.6). Furthermore, by being able to observe how the proposed system responds to a determined stimulus, verification of the functional requirements specification is made possible.

The simulation model of a system reflects and mimics the structure and behaviour of a concept or real-world system. The main drive for this modelling process is the questions the simulation is trying to answer. In the present research what is being asked is how Web applications designs, described using some design model, would operate if they were implemented, and if this implementation would verify the intended functional requirements of the project. These questions raise the issue of modelling the implementation of a Web application from a functional point of view. Different researchers have distinct perspectives on this issue, mostly because the link between design models and implementation is usually loose, due to the advocated separation of concerns of the two respective phases. However, we propose to answer this by the analysis of what the key design concerns of the existing design models are, the key functional elements of an implementation, and the key testing concerns when performing functional evaluation.

4.3 The Content of the Simulation Model

Existing design models tackle Web application design by addressing its different concerns separately. One first division is usually made between *interface* and *functionality*. The Dexter Model accomplishes this by using the *presentation specifications* which map the components of the *Storage Layer* into their graphical representation in the *Run-time Layer* (Halasz and Schwartz 1994). On the other hand, WebML makes the distinction between the modelling of the front-end interface (the *Hypertext Model*) and the more operational units (contained in the *Content Management Model*) (Ceri, Fraternali, Bongio, Brambilla, Comai and Matera 2002). Furthermore, Baresi et al. (2001), in their W2000 framework, propose the *Visibility Design* to be separated from the *Functional Design* and *Navigation Design* tasks. But whatever the name it is given, researchers seem to unanimously agree that the user interface should be a modelling task by itself. Furthermore, the distinction between *functionality* and *data* is even clearer. Design models usually make this distinction very clearly, and some even propose that data modelling should be considered as one of the first tasks to undertake. WebML addresses it in its *Data Model*, Dexter when defining the *Storage Model*, and in HDM by the use of *entities* based on the *Entity-Relationship Model* (Chen 1976). Finally, the concept of *Navigation* is always present in design models since it is the fulcrum of hypertext systems. This is usually modelled by a *link* association between the model entities or components, and can be as "simple" as the HTML *hyperlink*, or entailing more complex types and browsing semantics. However, design models usually consider navigation as the "glue" between the other components of the model. In conclusion, the design models approach to tackle Web application modelling is based on a separation of concerns between interface, navigation, functionality, and data. These key design concerns will be used as the basis of the proposed *Web-design Simulation Model (WSM)*, by considering a multi-layered model approach of Web applications which separates presentation, navigation, functionality, and content.

Functional testing approaches of implementations are often carried out by considering Web applications as composed of key component types. Tonella and Ricca's (2004) two-layer testing model consists of *pages*, navigation *links*, and *control flow objects* which model the execution of statements and data access. Furthermore, Liu et al. (2000) define their Web Application Test Model (WATM) as composed of three type of objects: *client pages*, *server pages*, and *components.* A similar approach is adopted by Kung et al. (2000*b*) for the *Navigation Behaviour testing*, *State Behaviour testing*, and *Structural testing.* Their Object Relation Diagram (ORD) represents the Web application entities and their relationships, which consist of *client pages*, *server pages*, and *software components*, and the relationships types amongst them which can be *navigation*, *request*, *response*, and *redirect.* These testing approaches to functional verification are an indication of what developers consider to be a suitable Web application testing model, namely pages, links, and executable components. Therefore, the WSM will acknowledge this by considering these components as a representative set of the functional relevant entities.

One final aspect to investigate is the implementation process of Web application design models. After the design phase of development, the resulting design model has to be "realised" into an executable Web application. For this purpose, there are already automatic tools to map design models into code. One such notable example is the WebRatio tool that supports the automatic synthesis of WebML design models (WebRatio 2005). But whatever the used method is, automatic or interpretative, the design models still have to be mapped into HTML code, executable code, and database infrastructure. Since the proposed simulation model will be used to verify functional requirements, it is important to understand what this concept is, so that the "simulated implementation" reflects only the relevant concepts. Functional requirements can be captured using *UML Use Cases*, which can be defined as:

Use cases – capture who (actor) does what (interaction) with the system, for what purpose (goal), without dealing with system internals. A complete set of use cases specifies all the different ways to use the system, and therefore defines all behaviour required of the system, bounding the scope of the system (Malan and Bredemeyer 2001).

As can be seen from this definition, functional requirements do not provide information about the aesthetics of the Web application interface, nor of accessibility or usability issues. What they convey is the intended system reaction to a set of actions taken by a user; for this reason, the internal objects of the WSM must not only provide user interaction, but also mimic the behaviour of an implementation.

4.3.1 The Four-layer Model Definition

The WSM was designed adopting a common multi-layered approach that addresses each key design concern separately. Each layer is responsible for the correct management of a subset of concerns and entities. This design provides a high degree of flexibility in tackling the design complexities by dividing them into manageable subsets. As already noted, the research literature body seems to corroborate this concept by adopting layered models whenever addressing Web application design issues. All the existing major works on the topic consider the existence of several and often orthogonal layers that serve as a separation of concerns of the problem in hand.

The proposed WSM adopts a layered model assuming separation of concerns as a credible, researched and proven method of complexity management of Web application design. These layers were identified as the *Presentation*, *Navigation*, *Functional*, and *Content* layers (see Figure 4.3). Each layer addresses a key concern in a Web application, namely: the visible aspect by the Presentation layer, the paths

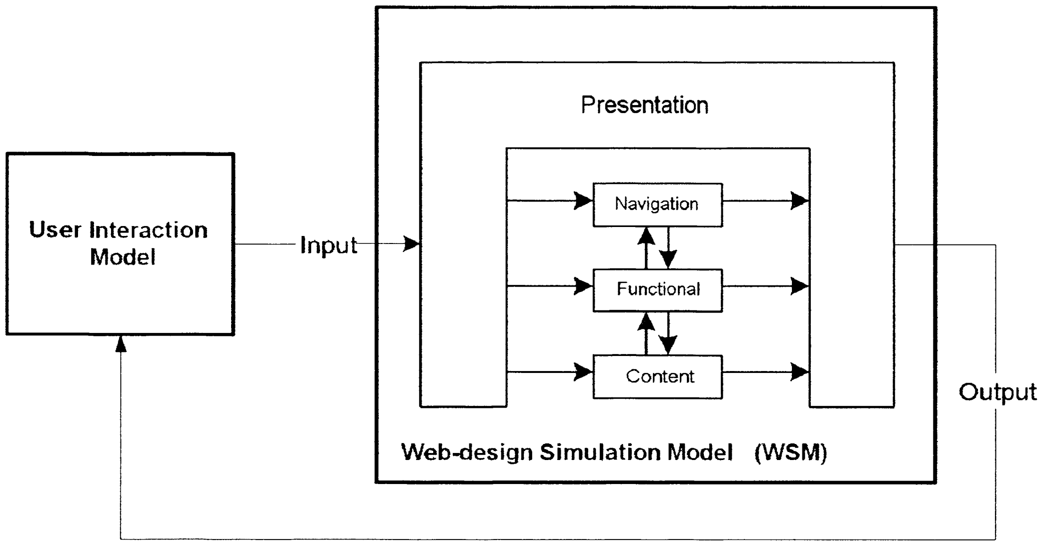


Figure 4.3: The Web-design Simulation Model (WSM)

traversed by the Navigation layer, script execution by the Functional layer, and data access by the Content layer. Each of these layers is responsible for managing a set of *entities* and *objects*. The concept of *entity* has been borrowed form the Object-Oriented technology field and is defined as templates from which *objects* are created (Wegner 1990). These entities represent the structure and emulate the behaviour of the design model components that are relevant to the testing and implementation phases, bridging the gap between design models and implementation components. During a simulation run several objects, which are runtime instantiations of entities, will simultaneously exist. This allows multiple objects to be simulated based on the same entity. One example of this mechanism is an entity representing a specific Web page which, during simulation, will originate several objects corresponding to several instantiations of that same Web page. In the following sections, the word *entity* will be used when referring to a template, and *object* when talking about its runtime instantiation (refer to Figure 4.4).

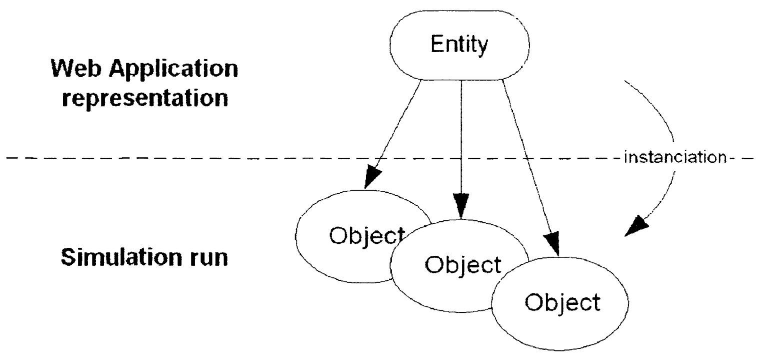


Figure 4.4: WSM Entities and Objects

The Presentation Layer

From a developer's perspective there are several key aspects of a prototype that should be available for analysis purposes. One such aspect is the inevitable existence of a graphical user interface, through which the user interacts with the system. This interface constitutes the sole means of user interaction with the Web application, and it is only by the use of this interface that the Web application displays to the user the consequence of her/his actions. In the WSM this was taken into account by considering a *Presentation layer* that lies between the user and the remaining layers of the model.

The Presentation layer represents the interface that we all have become accustomed to in the form of a Web browser, and implements its two main functionalities: handling user actions and inputs, and the rendering of components. For that purpose, this layer is populated with *Presentation entities* that are capable of handling these two types of services, namely: the *Window* and *Page* entities, which model the Web browser window and Web page. The Window is the basic container and at least one Window object must be present during simulation to hold any other objects, namely Page objects. On the other hand, the Page serves as a canvas on which other objects will be rendered.

To be able to simulate the design in a scenario as close to the real-world as possible, the Window entity will provide support to one important design feature of the WSM: the ability to simulate a multi-session environment. By this is meant that multiple, distinct, and simultaneous sessions may be open at any given point in time during a simulation run. This feature allows emulation of multiple simultaneously opened windows, similar to the scenario of a user opening several Web browser windows. In this case, each Window object has to manage its private objects and session separately. This allows multiple, distinct, and simultaneous accesses to the same objects without the danger of compromising their content integrity. Moreover, this mechanism enables several objects instantiated from the same entity to hold different content depending on the session (Window) in which they are contained.

User interaction entails both the actions a user may take and the data a user may enter in the available and appropriate input objects; these interactions will be handled by the Page object. Possible actions entail the selection of a rendered navigational object, corresponding to a probable system state change, or the selection of a functional object leading to the execution of a script object. One important design choice of the Presentation layer to note is the assumption that this layer is not responsible for the semantic interpretation of the user interactions. This means that, although this layer is the receiver of all user actions, it makes no attempt to interpret them. However, it is aware of the semantic and lexicon of the actions, making it possible to act as a gateway for the remaining layers. Although not being knowledgeable about the implications of selecting a rendered navigational object, the Presentation layer is capable of identifying it as a Navigation layer responsibility, resulting in the redirection of that action to the Navigation layer. Similarly, the Presentation layer may liaise with the Functional layer to execute a specific script object, or inform the Content layer of actions upon data objects.

The Navigation Layer

It is on this layer that user actions initiated on the Presentation layer involving navigational objects acquire a specific meaning and trigger a determined set of procedures. This layer is populated by *Link entities* which are the communication channel that connect and glue all the other entities together. It is the primary means of navigation throughout the Web application structure, and can also trigger procedural entities such as scripts. Navigation actions usually correspond to the selection of an available hyperlink rendered by the Presentation layer, which can either be HTML anchors on Web pages or buttons on Web forms. The first one connects two Web resources and serves as a unidirectional communication channel for parameter forwarding; the second is used to implement a data transfer mechanism.

As already noted, user actions performed on the Presentation layer are not interpreted there but delivered to the appropriate layer which, in the case of selecting navigation objects, is the Navigation layer. Here they are analysed and processed, and the corresponding set of processes is initiated. As with the Presentation layer, the Navigation layer can act as a gateway for the other layers. One important and common example of this role happens when a navigation object that triggers a script is selected. In this case, the Navigation layer acknowledges it as being part navigation, in the sense that it will possibly imply a change of position[1] within the Web application structure, and part a responsibility of the Functional layer to which the action is further forwarded.

This layer entails all the mechanisms necessary for the correct processing of all events that have the potential to change the current user position in the Web application structure. These changes may be initiated by the user as described above, or may have their origin on one of the remaining layers. A script that causes a navigational change, although being processed on the Functional layer, will rely

[1]'Change of position' in this context means that a new set of Web pages will be displayed to the user; 'position' is the current Web page a user is on.

upon the Navigation layer to handle that request. In this case there is no user intervention whatsoever, but internal processing and information flow amongst the layers of the WSM.

One other important function that this layer implements is the correct handling of the information flow amongst objects connected by a Link. These object are sometimes connected by special type of links that implement the fundamental function of information transport. The rationale for adding this role to the Navigation layer responsibilities is that this particular case of object linkage shares several proprieties with the "normal" hyperlink functionalities. They both connect two resources, have a well-defined source and destination points, and both unidirectionally convey information. This is not an unusual design option and examples are abundant in the research literature. For example, WebML makes use of *Transport* and *Automatic* links for this purpose, while Conallen's (2000) WAE-UML allows the extension of the object relationship semantics by using *stereotypes*, *tagged values* and *constraints*.

The Functional Layer

Whereas the Navigation layer provides mechanisms for traversing the Web application structure, the Functional layer provides support to the dynamic aspect of Web applications. This involves managing executable objects – *Script* objects – which are instantiations of the Script entities. Due to the programmatic aspect of these entities, obviously implying a myriad of possible implementations, they are the most complex entities within the WSM. If implemented, these objects would either run on the server or be executed on the client side. This distinction, although being sometimes disregarded and undervalued by some research on the grounds of it being mainly an implementation issue, may acquire an important significance for simulation purposes. One major aspect is the set of accessible objects from these scripts. For server-side Script objects, access to databases and Web application server variables is straightforward; on the other hand, user data input only becomes available when it is submitted to the server. Conversely, client-side Script objects have privileged access to user data input before it is even sent to the server, but constrained access to the information on the server. The Functional layer makes this distinction but, if the design model does not, it assumes that objects on this layer have unrestricted access to all the existing simulated objects.

As already mentioned, this layer implements the dynamic aspect of Web applications by providing mechanisms for the management of programmatic objects. These mechanisms include the processing of scripts and the handling of their input and output variables, which can be probed for analysis purposes. This possibility of accessing and probing object interface signals constitutes one major advantage of simulation. Objects on this layer provide a black box type of simulation, making the interface variables available by explicitly defining them. This topic will be further described in Chapter 5 where the definition of each layer entities is presented. Simulation from this layer perspective concerns the correct management of the programmatic objects that implement the dynamic aspect of Web applications. It is responsible for the triggering, execution and termination of these objects, thus handling their simulation life cycle.

The Content Layer

This last layer is dedicated to the data management of Web applications. It is on this layer that requests for accessing Data entities are handled. Data entities are the basic information content holders; databases, files, Web page variables, cookies, and Web form fields are all modelled by these entities. A fact worth noting is that the Content layer does not manage the variables of Script objects, which are directly handled by the Functional layer.

One fundamental aspect that this layer is responsible for is the management of Data objects that may be owned by different users in a multi-user environment simulation. The point being that Data objects may be shared amongst all simulated users, as is the case of a central database, or may belong to a specific user and thus private. This data-sharing mechanism emulates the real-world scenario. Also, this layer makes a distinction between the application and session scope of Data objects, being responsible, in conjunction with the Window, for their management in a multi-session environment.

The Content layer does not address the internal structure of Web applications databases. This aspect has been addressed and thoroughly researched and, as such, the Content layer does not attempt to propose or serve as yet another Data Model. In fact, similarly to other design models, the WSM assumes that if a database exists, then its design has already been addressed and implemented. What the WSM does provide is the means to access the database content via the Content layer and its Data objects.

4.3.2 Hierarchy of the WSM Entities

An important aspect of all WSM entities is their hierarchical relationship (see Figure 4.5). The Window entity is the basic container of all the remaining entities, and it can contain either Page or server-side Script entities. Similarly, the Page entity may also consist of several others, such as Link, Data, Script, and even inner Page entities, which is the case when Web frames are being modelled. Link and Data entities, on the other hand, are end nodes of the hierarchy and cannot be composed of any others. Script entities may be contained either by Page or Window entities. The former situation arises when there are client-side scripts to be modelled which need a Web page to exist. On the other hand, a server-side Script does not require a Web page. However, a server-side Script must be contained by a Window entity

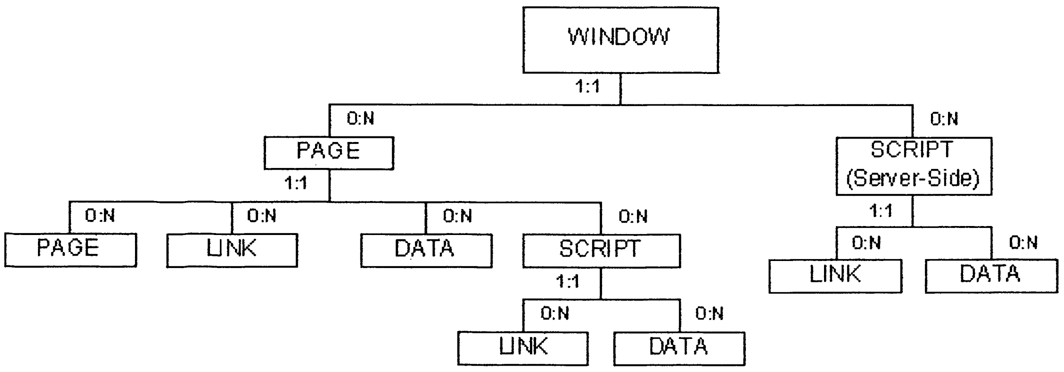


Figure 4.5: Hierarchical Topology

to be executed within the right session context, since the Window entity models a user session. In this way, a Script can access other entities and variables belonging to the same user session, which are all grouped by one Window entity.

Derived from these basic entities by instantiation, a collection of several objects populate the WSM during a simulation run. For example, distinct Page objects, resulting from instantiation of Page entities, simulate different active Web pages. Similarly, Window, Link, Script, and Data objects are also dynamically created during the simulation process and destroyed and eliminated when no longer needed. For example, at least one Page object will be created when a Web page is displayed to the user, and discarded when the user navigates outside that page scope; furthermore, a Window object must also exist to hold the Page. Link objects are dynamically created based on the Web application design. Script objects can be created by other Script, Link, and Page objects and will cease to exist after they have completed their workflow. One situation when objects once created remain in existence throughout the entire simulation is when a Data object is a *shared* resource. Examples of this situation occur when the object models a "cookie" or a database; in this case, the object will exist during the whole session or application duration because it must hold its value and be constantly available for other simulation objects to access.

4.3.3 The User Interaction Model

Users are viewed by the WSM as external entities that interact with Web applications, and the ones that constitute its main source of stimuli. The design option of not including the *User Interaction Model* in the WSM was based on the premise that, although they are two complementary aspects of the normal Web application workflow, they should be separately modelled in order to not become entangled. Separation of concerns played a major role in this decision, as well as the belief that future work will consistently improve and refine the user interaction model to a degree when it becomes more than just pure interaction, embracing users' objectives, motivations, and decision-making mechanisms. Such is the complexity of this topic that it is out of the scope of the present thesis and it deserves a separate research of its own.

The basic user interaction functionalities are modelled by the User Interaction Model and its *User* entities. The WSM defines the interface through which the User entity can interact with the Web application via the Presentation layer. The rationale for developing the User entity is to enable and emulate a multi-user environment simulation. By allowing multiple User objects to interact with the WSM, the real-world scenario of concurrent access can be simulated. This has a major impact simulation-wise, since the range of possible simulated scenarios becomes not only wider but closer to what in fact happens in a real implemented Web application. For example, a possible scenario can be the simulation of concurrent accesses to Web Forums, E-Commerce, or even Web auctions; in this case the possible impacts on the central database due to concurrent accesses can be a major source of interesting results for the functional evaluation of the designs.

To emulate a real-world multi-user scenario, the User object can refer to several Window objects (see Figure 4.6). Distinct User objects will have distinct sets of Window objects independent from each other. This perfectly emulates the real-world scenario of having several and distinct users each with their own set of Web browser windows, simultaneously accessing the same Web application.

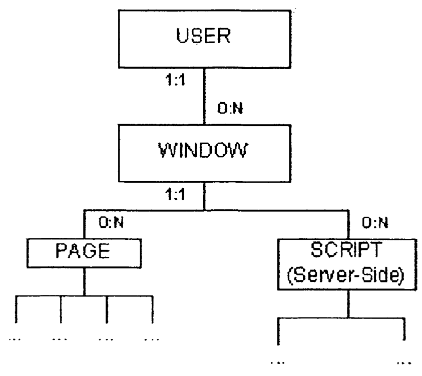


Figure 4.6: The User Entity

4.4 The Simulation Stimuli

From a WSM point of view, a Web application has two distinct states – either *Idle* or *Processing*. The idle state happens when the Web application is waiting for events to process and execute. In contrast, a Web application is in the processing state when receiving and processing one of those events. This state changing is driven, although not restricted to, by the user interaction with the Web application. Events are the force behind these states alteration. They have the potential to change the Web application state and will always imply that the Web application will remain active during the time needed for their processing. This perspective on the Web application mechanisms suggests a Web application as a state machine, bouncing back and forth from state to state, which is determined by the succession of events received.

Stimuli or *events* are what drive the simulation and make it progress through time. An event can be defined as:

DEFINITION 14 *Event – (1) An occurrence that causes a change of state in a simulation. (2) The instant in time at which a change in some variable occurs. (*IEEE standard glossary of modeling and simulation terminology – 610.3-1989 *1989)*

There are two types of stimuli: the endogenous or internal, and the exogenous or external (Banks 2000). The endogenous events have their source in the system and are triggered by the system objects, whereas the exogenous events have their origin in external objects. An event is an occurrence between two objects, and has an origin, a recipient, and a message. In the WSM, the events source and destination are either objects or one of the four layers. The information contained in the event message consists of the event identifier and a set of parameters according to the event syntax. The syntax of the WSM events is presented in Appendix B.

The only exogenous object of the WSM is the User. Users may trigger several events, although as mentioned before, they all interact with the Web application through the Presentation layer. These user events are named *User Interaction Stimuli.* User events have their origin in the actions that users perform on the Presentation objects and are classified by the WSM as belonging to one of the following classes: *(1)browser events*, *(2)interface object events*, and *(3)data input events.* Since interaction with the Presentation layer is the only channel for users to access the Web application, users have two classes of objects with which to interact: the Window and the Page. There are two types of *browser events*: opening and closing a Window. The opening of a new browser window – the *openWindow* event – contains the identifier of the Window and URL to be displayed, as well as identification of the User to whom this Window belongs. Similarly, the *closeWindow* event also conveys the identifier of the Window that is to be terminated by the Presentation layer.

The user actions that are performed on Page objects trigger *interface object* *events*. These interface objects are mainly navigational objects such as the ones modelling hyperlinks and buttons, and they usually lead to an alteration of the set of the active Page objects. Hyperlinks are unidirectional connections between two objects such as two Page objects. Actions performed on this type of objects will trigger an *actionLink* event. Buttons, such as the "submit" button on a Web form, are a means to transport information from a Page object to a Script object. Actions on these objects will issue an *actionButton* event.

Users also have the option of entering data into the appropriated data fields which are the rendered aspect of the Data objects – these will trigger *data input events*. These data fields found on Web forms are a special case of Data. They are rendered on the Presentation layer in the commonly found HTML formats of "text fields", "combo boxes", "check boxes", "radio buttons", and "lists". Normal operation procedures consist of selecting or entering data in these fields and submitting the form for processing. The selecting and entering data actions are represented by the *setData* event. Parameters of these events are the identifier of the Data object and its new value (or values if it is a multiple parameter field), which will be processed by the Content layer. The complementary event, which is not usually issued by a user but by another layer or object, is the *getData*, which returns the content of a Data object.

The possible endogenous events are those which the objects may issue either to one of the layers or directly to another existing object. One such event is the *readObject*; when issued it attempts to access an existing object. This is normally used by a Window or Page object when loading information for rendering. Also, a Script object may issue this event when including a Data object as one of its components.

Other endogenous events commonly found are those that specifically deal with Web page construction. One of these is the *displayPage* which will attempt to render a Page object. It can be triggered by a Navigation or Functional object, and will be processed by the Presentation layer. A special case of this event which emulates the dynamic construction of a Web page is addressed by the *buildPage* event. In this case, a Script object is responsible for the on-the-fly building of a Page object. It is a cumulative process where several Script objects may concurrently contribute for the construction of a Page object by adding objects to it. Finally, another internal event is the access of a Script variable content, either to set its value – the *setScriptVar* event – or to retrieve it – the *getScriptVar*. These events are usually used during the verification of the functional requirements to evaluate the behaviour of a specific Script by inspecting its internal variables.

4.5 Layer Interface Definition

Whenever an event is issued it must be processed by the simulation model. Our model has been designed to enforce separation of concerns by defining an orthogonal layered model and, therefore, distinct layers have their own management responsibilities and capabilities. However, the WSM implements a mechanism of interlayer procedure invocation, through which layers can request services from other layers. This enables services to be made readily available to all layers and objects when a necessary procedure is required but not implemented by the issuing layer.

An often-required service consists of the display of a Page object. This is implemented by the Presentation layer, which is in charge of supporting operations that deal with the visible aspect of Web applications. Other layers may issue a request to the Presentation layer for the rendering of a Page object. One such situation arises when a Link is selected by the user, and the Navigation layer, which processes these type of events, requires a new Page to be presented (see Figure 4.7). This requires the Presentation layer to construct a new Page object and read all its internal components; this is performed by successive calls from the Presentation layer to the Navigation, Functional, and Content layers.

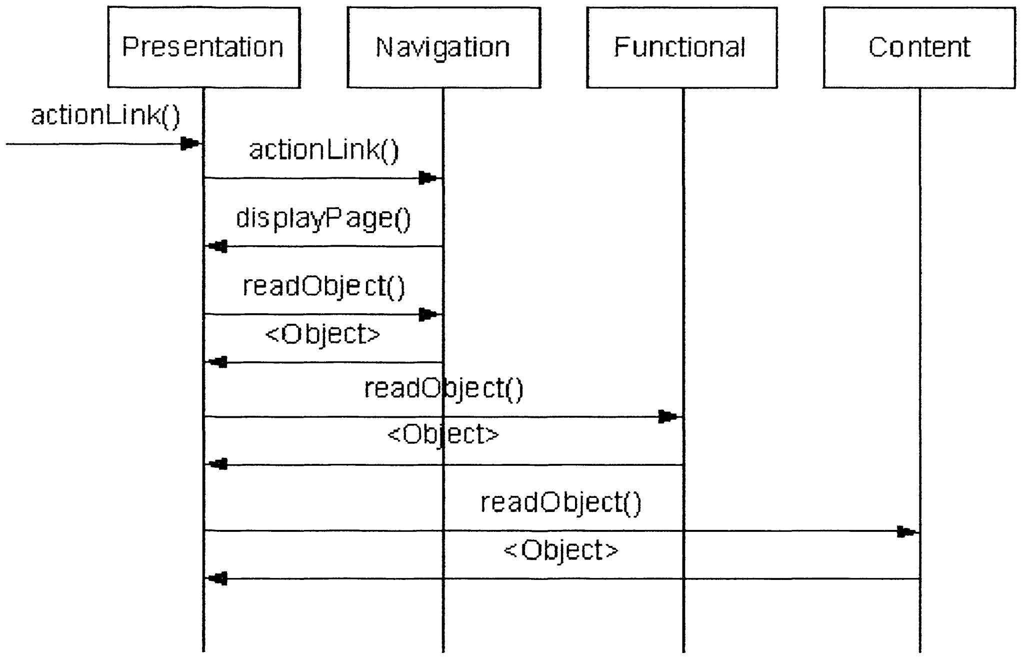


Figure 4.7: How an actionLink Event is Processed

One other scenario occurs when a server-side Script object dynamically constructs a Page object. In this case, a Page entity is instantiated and the object receives from the Functional layer its constituents, which will be part of the final Page object (see Figure 4.8).

As already noted, it is a responsibility of the Navigation layer to manage the events that give rise to a change of the user position within the Web application. Hyperlinks and buttons are the rendered aspect of Link objects, and actions upon them are processed by this layer. From the Presentation layer, the Navigation layer receives the *actionLink* and *actionButton* events; the former is processed by the Navigation layer leading to a reply back to the Presentation layer containing the new set of Page objects to be rendered. The latter event is further forwarded to the Functional layer which will trigger a Script object (see Figure 4.9). There is no

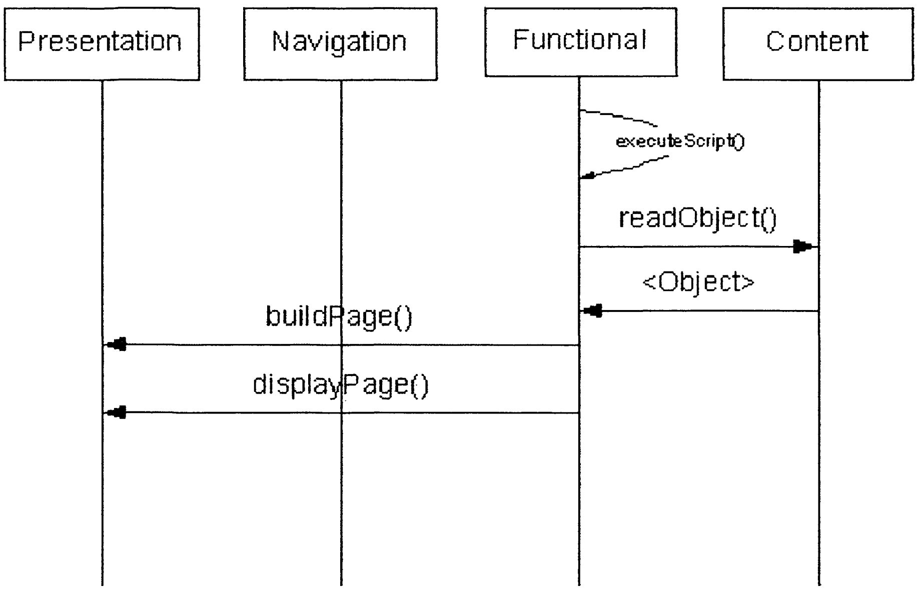


Figure 4.8: Dynamically Constructing a Page object

communication between the Navigation and Content layers, which do not require access to each other's support services.

The Functional layer provides the service for executing a Script object. This service receives the identification of a functional object and supports the *start*, *execution*, and *termination* phases of the object. One such example is the exogenous *actionButton* event. Another scenario occurs when a client-side Script is contained by a Page object; in this case, when simulating the loading stage of the Page, an execute command of the Script is issued by the Presentation layer directly to the Functional layer.

All the Web application information is handled by the Content layer, which manages files, variables, and databases. An important service that this layer provides is the accessing of that information by the other layers. For this purpose, the Content layer implements the operations of setting and retrieving content from the existing Data objects (see figures 4.10 and 4.11). For example, the Presentation

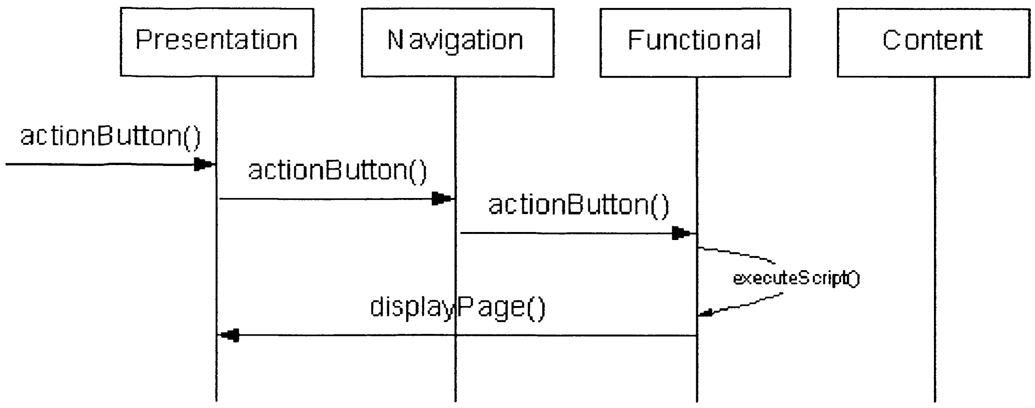


Figure 4.9: The Event Procedure of Selecting a Button

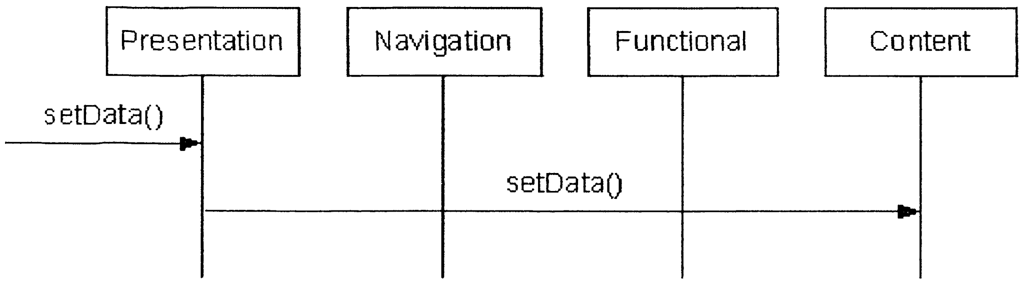


Figure 4.10: The setData Content Layer Service

layer may request a specific data object to be included on the rendered Web page. Furthermore, Data objects also model and manage input fields of Web forms; therefore, user actions on the rendered data objects will issue a *setData* command to the Content layer which alters the content of the respective Data object. Also Functional objects often require access to the content of Data objects for their workflows; this is provided by the *getData* service. Once again no communication is necessary between the Navigation and Content layers.

Figure 4.12 summarises the services provided by each WSM layer.

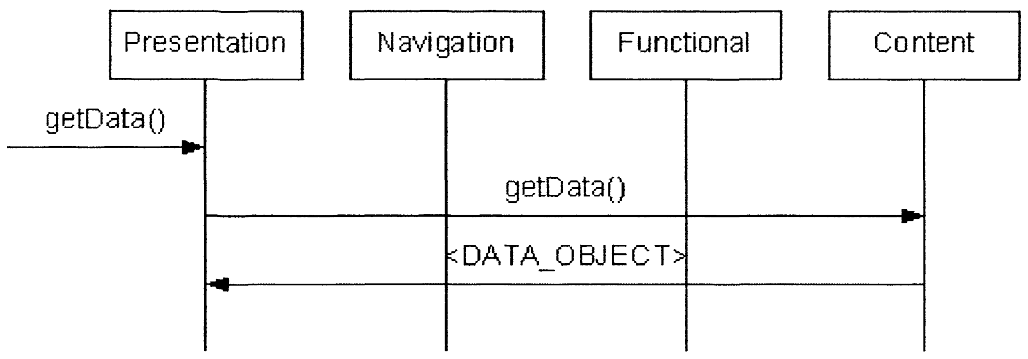


Figure 4.11: The getData Service Provided by the Content Layer

4.6 The Simulation Output

The results of the simulation will consist of a set of objects existing at a point in time during a simulation run. These can be User, Window, Page, Link, Script, and Data objects, which completely define the state of the system from a functional perspective. These entities represent a simulated implementation of the design, and emulate both its structure and behaviour in a multi-session/multi-user simulated environment.

The output of the simulation will allow inspection of the set of active pages, the links traversed, the scripts executed, and the data content. By analysing the stimuli that led to these results and by comparing them with the expected results, an evaluation of the functional requirements can be made.

4.7 Assumptions and Simplifications

In order to manage the simulation model effectively and to keep complexity within reasonable levels, assumptions and simplifications were made. We assumed that a Web application can be seen as a finite state machine which can be in one of two states: either Idle or Processing. This, however, has empirical evidence support and does not have an impact on the layers of the model, but only on the mechanism of

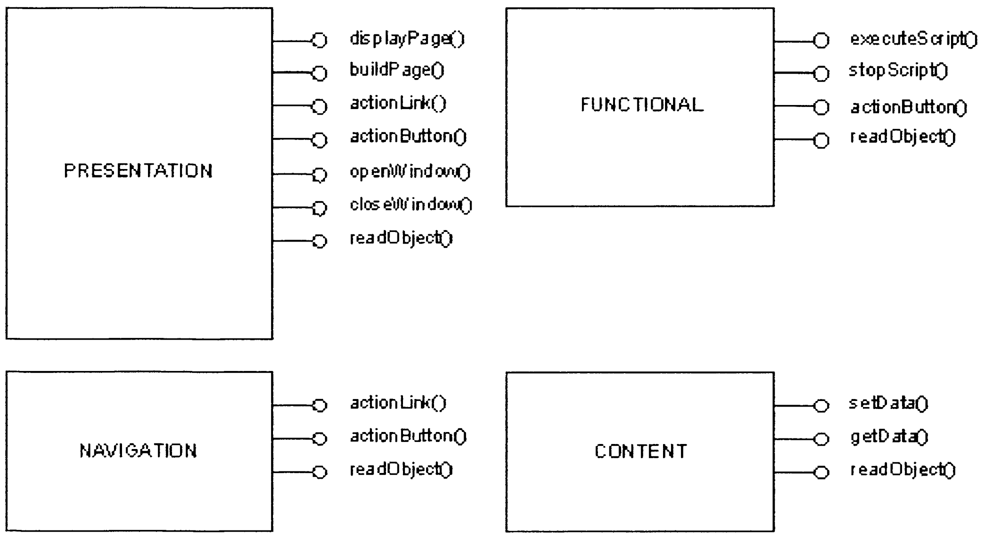


Figure 4.12: The Services Provided by Each Layer

the simulation itself.

The simplifications made pertain mostly to the graphical rendering of the Web pages. There is no intention of assessing the graphical representation, be it its quality, accessibility, or usability, since it does not add any value to the functional requirement evaluation of Web applications. Therefore, user interface assessment will not be object of discussion in this thesis. However, the interface implements several aspects such as navigation and data input capabilities, which are relevant to functional evaluation and that must be modelled. This simplification is corroborated by the main Web development models, whose graphic representation concerns are kept to a minimum or even non-existent.

4.8 Methodology of the Simulation

The simulation approach that is being considered is one of a discrete-event type performed on a *Web Application Design (WAD)*. These designs may be defined using one of many *Web Application Design Models (WADMs)*. This raises the question of how to develop a simulation framework that is not tied up to a specific WADM. Moreover, the designs will have to be translated and represented by WSM entities, which have to be formally defined in order to be simulated. This can be solved by the use of a middleware *Web-design Description Language (WDL)*. Existing Web applications are first translated into WSM using the middleware WDL, resulting in a *WDL Model* which will then be simulated (Figure 4.13). One advantage of this approach is the possible simulation of Web application designs using heterogeneous Web application design models.

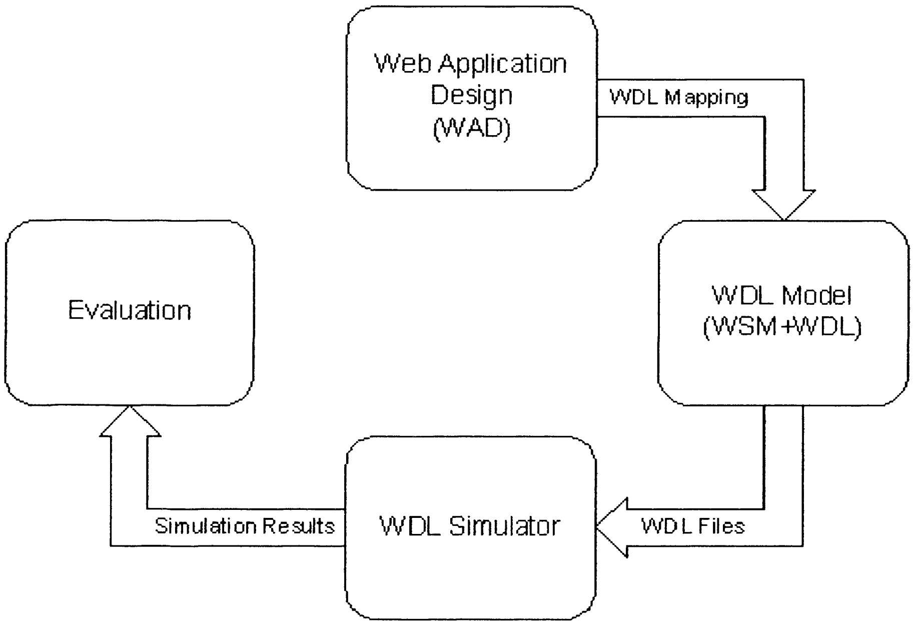


Figure 4.13: Methodology of the Simulation Procedure

WDL is a formally defined description language that is able to represent the most important functional aspects of the existing design models. It borrows concepts from hardware description languages, in particular from VHDL (*IEEE Standard VHDL Language Reference Manual - 1076* 2002), and is presented in Chapter 5.

The mapping procedure between a Web application design and the WSM entities consists of a representation of the former using WDL. Depending on the source of the WAD, either using WebML, Web Application Extension to UML, or some other design model, the appropriated translation procedure is selected. This consists of a set of mapping rules and patterns that automatically generates the target WDL Model description from the source WAD. The WDL Model description generated will contain all the WSM entities needed for an accurate representation of the Web application design from a functional requirements evaluation perspective.

This raises the issue that some existing design models may not be able to capture sufficient information to provide a meaningful simulation. Undoubtedly, if a developer chooses a specific WADM for Web applications design then it is because it can offer some support for the implementation phase of their functionalities. However, a valid question is whether a chosen WADM provides enough information needed by WDL. This will be answered in the next chapter by identifying the minimum information a Web application design model must provide in order to enable the mapping process, simulation, and evaluation of functional requirements.

4.9 Verification and Validation of the Proposed Web-design Simulation Model

Verification and validation of the Web-design Simulation Model assures that the results obtained by the simulation model are, in fact, credible. Simulation model verification assesses if we are building the *model right*, whereas simulation model validation assesses if it is the *right model*. Balci (1990) argues that, since a model is an abstraction of reality, *absolute* accuracy of a model is something that is usually difficult to attain. There are, however, some measures that allow us to assess the credibility, quality, validity, and verity of the model when used for the defined study objectives. Some of these measures – such as verification of the formulated problem, and the feasibility assessment of the simulation – are related to the definition of the problem, and have already been addressed. Formulated problem verification addresses the issue of if the formulated problem contains the actual problem, and if it is sufficiently structured to allow to allow derivation of a solution. This has been substantiated throughout Chapter 2 – Literature Review – when stating the problem, describing the background theory, and formulating the hypothesis. Feasibility assessment of simulation addresses issues such as whether it is possible to solve the problem with simulation, and if the benefits justify the estimated costs of devising a simulation model. This has been addressed throughout this Chapter, by clearly identify the basic building blocks of Web application design models that possesses functional meaning, and by definition of a conceptual model with a well-defined purpose and contents.

Other measures to assess the validity and verity of the simulation model exist, such as programmed model Validation and Verification, and model validation. These pertain to the computerised (or programmed) model, and will be addressed when the translation of the conceptual model into a computerised model is presented (Chapter 6).

4.9.1 Summary

The definition of the Web-design Simulation Model (WSM) has been presented. The adopted approach led to a multi-layered model, which implements a distributed management responsibilities among the layers. The four layers adopted are Presentation, Navigation, Functional, and Content. Each of these addresses, manages, and handles a key design concern. Definition of the layers' interface, simulation stimuli, and results was also presented.

Furthermore, the proposed Simulation Model consists of several entities that bridge the gap between design models and a simulated implementation. These entities are the Window, Page, Link, Script, and Data, and they will populate the layers of the WSM. Additionally, one other entity – the User – addresses a real-world scenario by enabling a multi-user simulation environment. The next chapter will present a more in-depth definition of these entities.

Chapter 5

The Web-design Description Language

The Web-design Description Language (WDL) has been developed not as another description language, but to bridge the gap between Web Application Design Models (WADMs) and the Web-design Simulation Model (WSM), by formally describe the WSM entities. It is not a description language for Web application design *per se* that is aimed at, but one that makes simulation achievable. This means that it must be formal in its syntax, comprehensive in its lexicon, and programmatic in its semantic. This chapter presents the WDL definition and the relationship between other design models and WDL.

5.1 A Formal Description Language for Web Simulation

A *formal language* is characterised by the prior knowledge of its rules before its use (*IEEE standard glossary of software engineering terminology – 610.12-1990* 1990, p. 34). It consists of a set of finite-length words written using some finite alphabet, of which the usage rules are known. A *description language* is a term used to refer to a class of languages used for the conceptual design of systems. They are usually platform-independent so as not to commit to a specific implementation, abstract enough to cover all the concepts involved, and allowing the right level of detailed description of its entities to serve for design purposes. When developing WDL, an inter-disciplinary approach was taken by borrowing concepts from the hardware development field, where description languages are a much researched topic. In fact, Hardware Description Languages (HDLs) have been around for many years, VHDL (VHSIC Hardware Description Language) and Verilog being two of the most successful examples. VHDL was developed to document the behaviour of Application-Specific Integrated Circuits (ASICs) that were introduced in equipment supplied to the US Department of Defense. It has been widely used since the early 1980s and became the IEEE standard 1076 in 1987, with the last revision dating from 2002 (*IEEE Standard VHDL Language Reference Manual - 1076* 2002). As stated in the standard, VHDL is a formal notation which "supports the development, verification, synthesis, and testing of hardware designs". Verilog has also been extensively used and became an IEEE standard 1364 in 1995 (IEEE 2001). The main differences between these two standards can be found in their data types, design reusability, libraries, large design management capabilities, and suitability for abstract description (Smith 1996, Maginot 1992). VHDL's capability for high-level abstraction description, both behavioural and functional, user-defined data types, and its support for large designs management and reusability, make it a better candidate to be transposed to the Web application development field than Verilog.

The concept of describing a hardware system's behaviour is, undoubtedly, a very attractive perspective – complex diagrams and blueprints of systems are no longer needed to define the hardware being built and its simulation is a straightforward process. Structural and behavioural description of a Web Application Design

(WAD) will provide similar benefits. WDL is, at this stage of development, not as ambitious as VHDL in the sense that support for synthesis is not yet provided. Furthermore, while VHDL is a language that developers use extensively to design their hardware, WDL is a middleware language that is mostly the output of an automatic mapping process. Therefore, only in special cases will Web developers program in WDL. These special cases happen when the mapping process is not fully automated, or when Web developers want to model a specific component behaviour. However, the structural and behavioural description of a WAD will allow its simulation and evaluation of the functional requirements. And that is exactly what WDL aims at – to support the development and verification of Web application designs.

5.2 The Entities within

This section presents the WDL formal definition of the six entities used during a simulation run – User, Window, Page, Link, Script, and Data. The adopted format of the definition of the entities is very similar to VHDL where a separate description for their *structural* and *behavioural* aspects is made by the **entity** and **architecture** constructs (see Figures A.1 and A.2 in Appendix A). The structure of each entity is firstly described, which allows an overview of all the *attributes* (*ports* in VHDL) accessible for content inspection and modification. This structure is further divided by WDL into two main groups – *Model* and *Runtime*. The former relates to those attributes which are gathered directly from the WAD and remain constant throughout the entire simulation. Their values define the general characteristics of the entity and are directly obtained by the mapping procedure which translates a WAD into WDL. The *Runtime* attributes values are changeable and they are related to the Web application state at a specific time during the simulation. These values change as stimuli are processed, and their progress during a simulation run constitutes the main source of results.

The behavioural description of each entity is defined in the **architecture** block, where the functions implemented by the entity are defined. This definition is one of the main reasons that makes WDL a simulation-enable description language. By individually defining each entity behaviour, each stimulus will be processed by the appropriate function. Therefore, each entity behaves differently, depending on its attributes and functions workflow. The adopted format is also similar to the VHDL standard, however, the block of each function is written using Java$^{\text{TM}}$-like statements, the rationale being that it is platform-independent and extensively used by developers. Also, the representation of the possible states and events of each object is done by the use of UML Finite State Machine diagrams (Object Management Group 2005). This allows a graphical representation of the 'life cycle' of the objects. The following sections will describe each of the entities' attributes and functions; their formal definition is presented in Appendix A.

5.2.1 The Page

As already mentioned, the Page entity resides on the Presentation Layer and emulates a Web page being displayed to the user. As such, the Page entity is modelled based on the Web page structure and functionality, and a selection of the considered important set of aspects of the Web page is made. This selection of the attributes is directly related to the aspects that should be simulated to allow the evaluation of the functional requirements of the WAD. Therefore, all aspects that do not contribute to this objective should be carefully evaluated, and discarded if they do not significantly contribute or support the simulation.

A Web page serves several purposes: to display information to the user, to collect information from the user, and to render navigational, functional, and data elements which allows the user to interact with the Web application. Simulation for the evaluation of the functional requirements is not concerned with the aesthetic issues of the HTML rendering, since they do not interfere with the operational aspects. This means that, for instance, issues such as colours, fonts, text, graphics, and placement on the Web page do not originate any WDL structural attributes.

Usually, functional requirements state which set of Web pages should be displayed to the user to implement some required functionality. Hence, the *state* of each Web page of the Web application – 'ACTIVE' if displayed, 'INACTIVE' otherwise – is a basic attribute that must be modelled. The functional requirements also refer to the Web page's main objective – to display information or to collect user input data. This means that the purpose of a Web page must also be captured. Although not usually covered by the functional requirements specification, some attributes have been added to provide developers with more information about possible implementations. Such is the case of how a Web page is constructed, which can be a static HTML file, a mix of HTML and embedded client-side scripts, or resulting from the execution of a server-side script. These "supplementary" attributes are often modelled by the existent WADMs, and although not absolutely necessary for the functional requirements assessment, they contribute by adding information about the dynamics of the design. Table 5.1 summarises what aspects of the Page entity are modelled by WDL, and which ones are not being considered for its definition.

Simulated	**Not simulated**
Uniquely identification	Look and feel
State – active or inactive	Usability issues
Its main purpose: displaying or gathering information	Accessibility issues
Building method: static or dynamic	Performance issues
Inner structure definition	
Identification of the set of pages to which this entity belongs	

Table 5.1: Simulation of the Page Entity.

Table 5.2 shows the structural and behavioural definition of the Page. Its attributes are categorised as *Model* and *Runtime*, depending on how their content is gathered and if they are constant or variable throughout the simulation run. The first attribute – *pageID* – is the identification of the entity. This attribute, existing in all WSM entities, merely provides the means to uniquely identify an entity. Recall that each defined entity will serve as a template from which, during runtime, several objects may be instantiated. Hence, identification of each template is necessary, as is the identification of the equivalent runtime object – the *pageOID*. By using these two attributes, simulation of distinct instantiations of the same template may be performed. Next the *pageStructure* classifies the entity's main purpose: to display information, to collect information from the user, or to serve as a container of other Page entities. The *pageBuild* simply determines how the Page is to be constructed – by loading a file, by a mixture of HTML and client-side scripts, or built by a server-side script. Since the Page entity is one of the WSM entities that may contain other entities, the *pageComponentsID* holds their identification in an Array. Finally, the entity may be further classified in terms of *pageContext*, which allows Web pages to be grouped together in terms of their functionality, and it is often referred to as areas or views by other Web application design models.

Entity	
Model attributes	**Runtime attributes**
pageID	pageOID
pageStructure	state
pageBuild	windowOID
pageContext	userOID
pageComponentsID	pageComponentsOID
Architecture	
displayPage()	
deactivatePage()	
readObject()	

Table 5.2: The Page Entity Definition.

The *Runtime* attributes of the entity are not gathered from the design model but set and altered during the simulation process. For instance, the value of the *state* attribute can determine whether a Web page is being displayed to the user or not. The remaining attributes hold the runtime values of the Window in which this

Page is contained, and to which User the Page belongs. Finally, an array holds the runtime Object IDentifiers (OIDs) of the Page components which are defined in the corresponding *Model* array.

Each entity implements several functions which emulate their behaviour. They are called during runtime by the respective layers to which they belong. The Page object implements three functions: the *displayPage()*, the *deactivatePage()*, and the *readObject()*. The first function, which may be issued by any of the WSM layers, simply activates the object and its internal components. This makes the Page object visible to the User. The *deactivatePage()* function makes this Page invisible by altering the object's *state* attribute to 'INACTIVE', and the *readObject()* reads every Page component found in the *pageComponentsOID* array. A representation of the possible states and events of the Page object using UML Finite State Machine (FSM) notation can be seen in Figure 5.1. The 'Presentation.Create' and 'Presentation.Delete' events represent the creation and destruction of the Page object by the Presentation layer.

Attribute	**Enables simulation of...**
pageID	several and distinct Page entities
pageStructure	the three main classes of pages: plain HTML pages (PAGE), forms (FORM), and frames (FRAMESET)
pageBuild	plain HTML pages (HTML), client-side scripts embedded in pages (DYNAMIC), and server-side scripts building pages (ASP)
pageComponentsID	hierarchical structure of the Page entity
pageContext	different areas of the Web application, by grouping pages in sets
pageOID	different instantiations of the same entity
state	the status of the entity at a given time
windowOID	multi-session environment
userOID	multi-user environment
pageComponentsOID	the runtime objects of the hierarchical structure of the Page entity

Table 5.3: Page Entity Simulation Features.

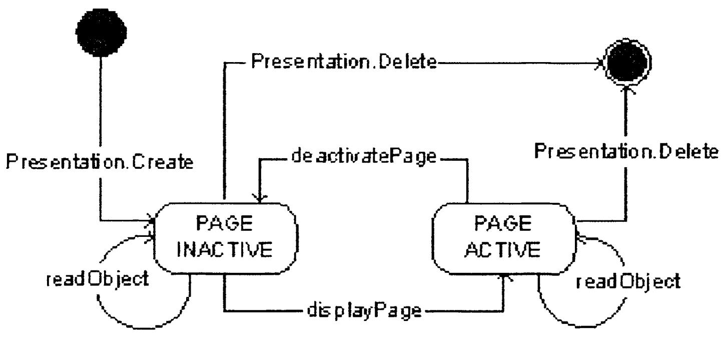


Figure 5.1: Finite State Machine Representation of the Page Object

5.2.2 The Link

The Link entity is found on the Navigation layer of the WSM. This entity models the connection between two Web resources, with a well-defined source, destination, direction, and information conveyed in the form of parameters. The main reason for implementing an entity with similar objectives to the HTML hyperlink is to allow navigation and transfer of information amongst the entities. Navigation is implemented by connecting two Page entities with a Link; similarly, transfer of information is achieved by connecting a Page to a Script, or two Script entities together. Table 5.4 lists the aspects that may be observed during the simulation. Its *state* attribute indicates whether the Link has been selected – the 'ACTIVE' state – or is available – the 'INACTIVE' state. When a Link object is created, it remains 'INACTIVE' until selected, thus indicating that it is available for interaction but not yet to be traversed. When a Link object does not belong to any Page or Script currently in use, it is discarded by the Navigation layer. Multiple ends, either source or target, are not considered by WDL. The alternative for this is to simply consider several Link entities which cover all the different possibilities. Image maps and sections within the same Web page are also not simulated; the first one tends to be an aesthetic option, whereas the last one is a usability issue, thus not contributing to functional evaluation. Finally, the only allowed event on a Link entity is the common "onclick" which triggers the connection; other events such as the "onfocus" and "onblur" defined in the HTML standard are not simulated, the rationale being that these actions are often used to enhance the graphical appearance of the interface rather than adding functionality and, furthermore, design models do not usually go into such detail.

The definition of the Link entity can be seen in Table 5.5. The *linkType* defines what class of Link this entity belongs to – connecting two Page entities ('LINK'), a Page and a Script ('SUBMIT'), or two Scripts ('TRANSPORT'). The two ends of the connection are defined by the source and target identifications attributes. Another important attribute is the *linkTargetWindowOptions*, which allows the target to be rendered in the same or in a new Window; also, when dealing with frames, it can be used to refer to the parent or topmost Page. The runtime attributes hold the OIDs of the entity, its object source and target identification, and the Window and User to which this Link belongs. Finally, the *linkParametersValues* attribute holds the runtime parameters content.

Simulated	**Not simulated**
Identification	Multiple targets
State – active (selected) or inactive (available)	Image maps
Source and Target identification	Sections within the Web page
Link and Submit types	Events other than onClick
Connecting Page entities	
Transfer of information between Page and Script entities	
Transport of information between Script entities	

Table 5.4: Simulation of the Link Entity.

The entity architecture definition shows the behavioural functions *actionLink* and *actionButton*. The former is called when activating a Link of the 'LINK' type, whereas the latter is executed when a 'SUBMIT' or 'TRANSPORT' Link is selected. Note that the Link state becomes temporarily active while deactivating the source and activating the target entities. Additionally, the *actionButton* updates the Link parameters with the latest values and executes the target Script. The *deactivateLink* simply changes the state to 'INACTIVE', and it is used at the end of the other two functions. Representation of the possible states and events of the Link object can be seen in Figure 5.2.

Entity	
Model attributes	**Runtime attributes**
linkID	linkOID
linkType	state
linkSourceID	linkSourceOID
linkTargetID	linkTargetOID
linkParametersNames	linkParametersValues
linkTargetWindowOptions	windowOID
	userOID
Architecture	
actionLink()	
actionButton()	
deactivateLink()	

Table 5.5: The Link Entity definition.

5.2.3 The Script

Scripts allow a Web application to process data, significantly contributing to its dynamic. Undoubtedly, scripts are one of the most important aspects when evaluating functional requirements. Errors in their workflows account for the main source of design problems and they are the most difficult errors to identify and correct. Simulation of its execution provides valuable data on the behaviour of a WAD. For a proper assessment, several aspects of a script must be modelled: the input parameters, the workflow, and the output. The most important aspects to be simulated from a functional evaluation perspective are listed in Table 5.6. The Script entity will be tested as a black-box type of component, where only the declared input and output signals are accessible for probing and testing.

One limitation of the simulation of the Script entity is the multi-thread

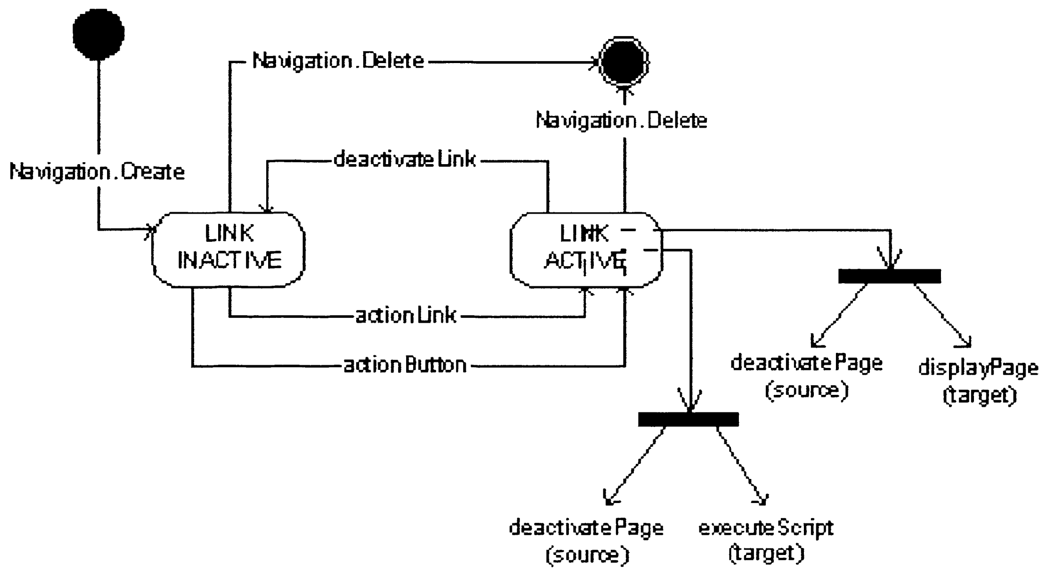


Figure 5.2: Finite State Machine Representation of the Link Object

Simulated	Not simulated
Uniquely identification	Multi-thread processing
State – active (executing) or inactive (available)	External access to data sources
Workflow – templates and Java^{TM}-based scripts	Programming languages other than Java^{TM}
Input parameters	Probing of internal variables
Output parameters	Performance issues

Table 5.6: Simulation of the Script Entity.

processing, which is more of an implementation than a functional issue. Another constraint is the access to sources external to the Web application, being them other scripts or data. Also, internal variables values cannot be observed – all the desired variables must be declared as input, output, or both, if their content is interesting for simulation analysis.

In the Script entity structural definition, the location where the script is executed is also considered to be one of its *Model* attributes (see Table 5.7). Although this is an implementation issue, it serves the purpose of indicating to the designer possible conflicting issues when a Script entity tries to access data that is not local.

Entity	
Model attributes	**Runtime attributes**
scriptID	scriptOID
scriptSide	state
scriptVariablesNames	scriptVariablesValues
scriptVariablesTypes	scriptComponentsOID
scriptComponentsID	callerOID
	windowOID
	userOID
Architecture	
executeScript()	
runScriptbehaviour()	
stopScript()	
readObject()	

Table 5.7: The Script Entity Definition.

The remainder of the model attributes are related to the definition of the variables that will be available for probing. Their identification and type are kept into the respective arrays. Their type – 'IN', 'OUT', and 'INOUT' – determines the direction of the signals from the Script perspective. However, their values are part of the runtime attributes, since they will change during simulation.

The *Runtime* attributes are similar to those already described in the Page and Link entities. The *state* attribute indicates if the Script is being executed ('ACTIVE') or available for execution ('INACTIVE'). The availability status happens when the Functional layer has already created the Script object but it is not yet being executed. This case occurs in three scenarios: the Script is a component of an active Page object; the target attribute of a Link object refers to the Script; or the Script is part of the components of an active Window object. When executing, the Script state attribute changes its value to 'ACTIVE', after which, if not longer referred to or being a component of any existing object, the Functional layer simply discards it. One especial attribute is the *callerOID*; this uniquely identifies the object which has triggered the execution of the Script object. Its value can be the OID of a Window, a Page, or another Script object. The first case occurs when dealing with a server-side script which is the case when the Script is a component of a Window object. The second happens when a Page loads and, therefore, it executes all its client-side Script components. And, finally, a Script object can execute another Script by issuing the special function *executeScript*. The *callerOID* attribute is necessary to return any results of the execution back to the caller object. Note, however, that in this attribute no Link object OID is allowed since they are unidirectional entities and cannot possibly be callers and, simultaneously, receivers of information.

The architecture of the Script defines the behaviour of the entity when a stimulus is received by the object. The first one is the *executeScript* which can be triggered by a Page, Link, Window, or another Script object. The Script input variables are updated with the current values (which are the parameters' content of the incoming Link), and then the *runScriptbehaviour* function is called. It is in this function that the workflow of the Script is defined. At the end of its execution, the output variables are updated, becoming available for probing, analysis, or outgoing Link objects. The Finite State Machine representation of the Script object can be seen in Figure 5.3.

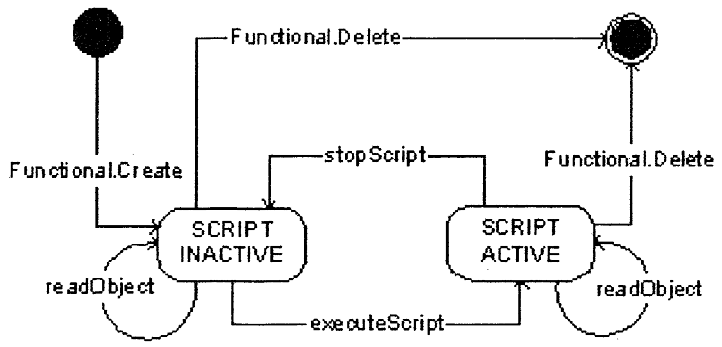


Figure 5.3: Finite State Machine Representation of the Script Object

5.2.4 The Data

If a script is what makes a Web application dynamic, the data is what makes it diverse. Data is intensively used by scripts in their workflows, and can take many forms, depending on its source. Firstly, there is the data stored in files, which is mainly used by Web pages to render information. This data is often found in the form of binary files, such as images, or just plain text. Then, there is the data that is contained in databases on the server-side, which is used by scripts to store or retrieve information. There is also that type of data that is closely related to the client-side, such as the information entered by the user on Web forms. And finally, data can take the form of application or session variables on the server-side, and persistent or session "cookies" stored on the client-side.

Data is important for the functional assessment of a Web application, due to several factors. One is that scripts often use data elements within their workflows, and decisions are made based on their content; for this reason, simulation of scripts without data becomes seriously limited. Also, functional evaluation requires that some variables or database content be observed in order to verify that the Web application is performing as expected.

Data objects populate the Content layer and are constantly accessed by the Presentation and Functional layers objects. There are, however, no interactions whatsoever between Data and Link objects and, consequently, between the Content and Navigation layers, the reason being that Link objects already have their own mechanism for transporting data in the parameters attribute. When submitting information from a Web form to the server, the Page object uses the Data objects corresponding to the form fields to set the Link parameters attribute, thus no direct communication between Link and Data objects is necessary. The Data entity is not, however, a data model of a Web application. It does not attempt to be a detailed description of the structure of the data involved, but to be an abstract representation of those elements. Web application design models usually address the data model issue in their framework; however, it is not WDL's intention to be another design language, but a description language that enables simulation. For this reason, WDL only models what is absolutely necessary to support simulation of the Data entity, assuming that the data structure and contents have already been addressed by the design model itself.

Table 5.8 summarises what will be observed when simulating the Data entity. Files, database access, form fields, and variables are all simulated. The drawback of using such a level of abstraction is that the structure of databases is not simulated. This, however, does not interfere with the simulation, since it can act as a proxy for the database engine. Furthermore, some of the capabilities demonstrated by database engines, such as rollback and transactions, are not supported by the Data entity.

Simulated	**Not simulated**
Unique identification	Structural database definition
State – active (being accessed) or inactive (available)	External sources
Text or binary files	File content
Database access – read, write, and update	Database specific functionalities (rollback and transactions)
Web form fields	Field default values
Application variables	Complex variables such as structures
Session variables	Script variables

Table 5.8: Simulation of the Data Entity.

The definition of the Data entity (Table 5.9) shows some interesting attributes, such as the enumeration of the allowed types using the *dataType* attribute which is further refined in the *dataSubType*, and the multiplicity and persistence defining the scope of the entity. The *dataMultiplicity* differentiates between shared data amongst all users, and multiple copies of the same Data object. A Data entity is 'SHARED' when only one instantiated object is allowed; this is the case of a central database or stored file which are shared by all users. On the other hand, it is considered 'MULTIPLE' when several objects, instantiated from the same Data entity, are allowed; this is the case of the Web form fields or cookies, for which, for each user, an object will be created. The *persistence* of a Data object is also defined here. If the Data object exists even after users disconnect from the Web application, then it is named 'APPLICATION'; this is usually used when the Web application must keep a record throughout its entire operational life time. If, on the other hand, the object's life terminates when a window (session) is closed, then it is classified as 'SESSION'. The different combinations of these two attributes allow definition of the scope and duration of the Data object, as well as the manner simulation will handle it.

Entity	
Model attributes	**Runtime attributes**
dataID dataType dataSubType dataSource dataMultiplicity dataPersistence	dataOID state dataValue windowOID userOID
Architecture	

setData(value [, SQLStatement])
getData([SQLStatement])
deactivateData( )	

Table 5.9: The Data Entity Definition.

The functions *setData* and *getData* are implemented by this entity. If the Data object is a database, the functions take an additional parameter – the *SQL-Statement* – which allows record values to be retrieved, inserted, updated, and deleted. These two functions are often called by the workflow of a Script object to access a database. In this case, the Data object's main task is to serve as a proxy to the database engine. On the other hand, if the Data object is not of a 'DATABASE' type, these two functions will act on the *dataValue* runtime attribute, altering or returning its value. The *deactivateData* function simply changes this object's state to 'INACTIVE'. The Finite State Machine diagram of the Data object can be seen in Figure 5.4.

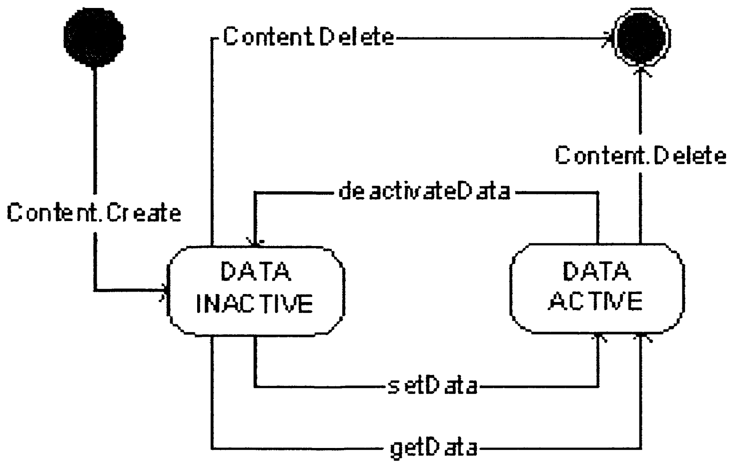


Figure 5.4: Finite State Machine Representation of the Data Object

5.2.5 The Window

The next two entities – Window and User – are very particular to the WSM and do not, usually, have a counterpart in existent design languages. Why, then, model a window or a user? The rationale for this is that functional assessment of Web applications cannot be properly performed if the environment where the application will be executed is not also modelled as closely as possible. Since the WSM does not attempt to evaluate performance, which would have to take into account not only the underlying operating system but also the hardware, there are only two relevant aspects of a real-world scenario to model: multiple sessions and multiple users. Emulating a multiple-session environment means that mechanisms that allow a user to simultaneously access the same Web application from different browser windows must be provided. Simulating a multi-user environment implies that concurrent access of a same Web application from distinct users must be handled. The first scenario is dealt with through the Window entity; the second one with the User entity.

Opening a browser window and typing the desired URL address is the normal mode of operation when accessing a Web application. What WDL intends to emulate is the opening, operation, and closing of browser windows. The Window entity serves as the main container of Page and Script entities. The rationale for allowing Script entities to be directly contained by a Window is that server-side scripts must be executed within a certain context. By modelling the browser window, WDL enables multiple sessions of the same user to simultaneously access the Web application, while keeping separate Page, Link, Script, and Data objects for each window. Note that a Window object is owned by one and just one user. Data objects belonging to a Window will be shared or private, and will be active while the window remains open or persist even after closing it, depending on its *dataMultiplicity* and *dataPersistence* attributes.

Simulated	**Not simulated**
Unique identification	Operating system's functions
State – active (opened) or inactive (closed)	Supported HTML subset
URL addressing pointing to either a Page or Script	Accessibility issues
Own set of Page, Link, Data and Script objects	Plug-ins and third-party modules
Private and shared Data objects	History related functions

Table 5.10: Simulation of the Window Entity.

The Window entity emulates a Web browser window and keeps session contexts separate from each other. What is not simulated are the several functions that uses the operating system's services to perform specific tasks (see Table 5.10). These can be of the file management type, such as the save, load, edit, and print functions, or accessibility-related manipulations, such as font size or encoding, or even content access and window pop-up management. Also, how the Web browser renders Web pages, which set of HTML it supports, and back and forth history functions are all aspects not modelled by this entity.

Entity	
Model attributes	**Runtime attributes**
	windowOID state userOID windowComponentsOID URL
Architecture	

openWindow()
closeWindow()	

Table 5.11: The Window Entity Definition.

Table 5.11 presents the definition of the Window entity. Note that there are no *Model* attributes, the reason being that a Window entity is not obtained by the mapping of a design model into WDL; rather it is a runtime entity that only has existence when performing a simulation. It has no attributes of which an instantiated Window object will make a copy; it is purely a runtime object made out of runtime attributes. Of these attributes, the *windowComponentsOID* holds the identifiers of the current objects contained by the Window, which is constantly changing and being updated. One other important attribute is the *URL*, which informs the Simulator of the object (and its descendants) to be loaded on the Window. The Finite State Machine diagram of the Window object can be seen in Figure 5.5.

5.2.6 The User

A user of a Web application is modelled by the User Interaction Model (UIM) and its User entity. To enable simulation of this external entity, WDL provides support to the entity's basic *create*, *discard*, and *userInteraction* functions, and models the absolutely necessary runtime attributes. The *userInteraction* function models the basic user interaction, and allows any of the exogenous stimuli defined in Appendix

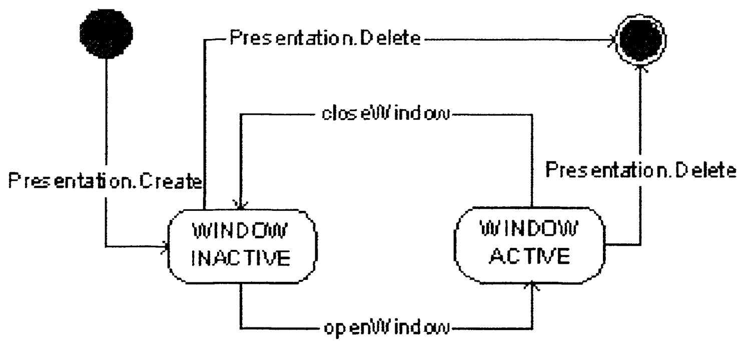


Figure 5.5: Finite State Machine Representation of the Window Object

B to be issued from this function. Further refinements of user behaviour may be accomplished by changes in the UIM itself. This means that complex user behaviour such as reacting to past and present states of the Web application, and goal-driven behaviour, can be done without changing the Web-design Simulation Model. By making a clear distinction between UIM and WSM, simulation of Web application users is enabled and a scalable architecture of the models is achieved. Modelling complex user behaviour, however, is not being considered in the present research due to the complexity of the topic which deserves a new research of its own. This issue is further addressed in Section 8.3, when considering some avenues for future work.

The Used entity is especially designed to promote multi-user simulation of Web application design models. Its main purpose is to aggregate all the Window entities belonging to the same User, and to keep an historic record of all the issued events by the simulated user. These features allow simulation of distinct and concurrent users accessing the Web application (see Table 5.12). From a functional evaluation perspective of the Web application design, this constitutes a major enhancement, because requirements often directly or indirectly state how the application shall behave in a multi-user scenario. For instance, how security is enforced and how data will be accessed in a multi-user scenario, are concerns that developers often have to deal with. Ignoring this fact will often lead to major security, functionality, and data-management problems.

Simulated	**Not simulated**
Unique identification	User profiles
State – active or inactive	User behaviour
Multi-user environment	
User interaction with the Web application	
Events history	

Table 5.12: Simulation of the User Entity.

This entity has only runtime attributes, since there are no structural model attributes mapped from a Web application design (see Table 5.13). Its definition comprises the runtime identification, state, components, and history arrays. The components can only be Window objects, of which it can contain one or more. Lastly, the history arrays keep all the events and corresponding slots (which is a measure of the elapsed "time" of the simulation) issued by a User object. The representation of the Finite State Machine diagram of the User object can be seen in Figure 5.6.

Entity	
Model attributes	**Runtime attributes**
	userOID state userComponentsOID historyEvents historySlots
Architecture	

createUser()
discardUser()
userInteraction()	

Table 5.13: The User Entity Definition.

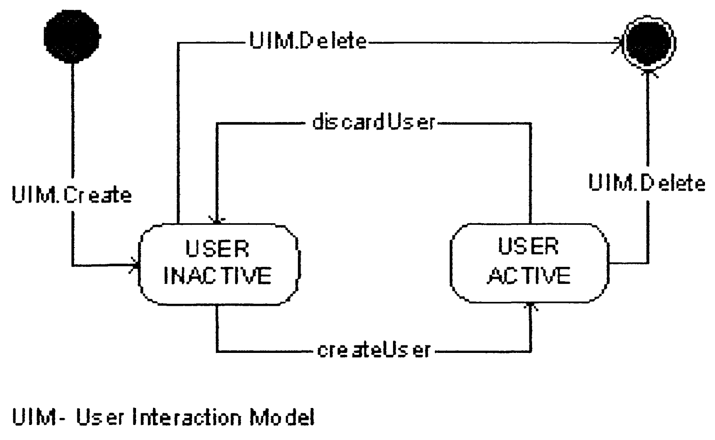


Figure 5.6: Finite State Machine Representation of the User Object

5.3 Mapping of Existing Web Design Models

Before simulating a Web application Design, there is a need to *map* the design model into WSM entities described using WDL. This mapping process is performed by the *WDL Parser*, which replaces the WAD elements by the WDL counterparts, setting the *Model* attributes of each one of them to reflect the characteristics and particularities of the original design. At the end of this process, the WDL design shall reflect the structure of the WAD in a WDL format, and will be ready for simulation.

To speed up the mapping and subsequent simulation, this mapping process must be as automatic as possible, keeping developers' intervention to a minimum. However, each WADM has to be considered as a separate case, having its own set of mapping rules to follow. A very important prerequisite is that the design model must be formal in its syntax, semantic, and lexicon. If there is any margin for dubious interpretation of the design model elements, then the fully automatic mapping process becomes seriously compromised and developers' intervention will definitely be necessary.

The process can be conceptually categorised into three main steps: (1)the parsing and main mapping of the elements; (2)extraction of the model attributes; and, finally, (3)the verification of the WDL model file correctness. The first step translates the design elements and patterns into WSM entities described using WDL. The parser identifies special patterns of the WADM elements and performs the mapping by searching in a library of templates. The second step attempts to set as many WDL attributes as possible. It extracts intricate attributes from the WAD and sets the appropriate WDL attributes with the right content. Finally, the third step looks for eventual entity duplicacies originated from the first step, and verifies the correctness of the whole WDL file. Such duplications occur when Data entities, which should be shared, have been mapped from distinct WAD elements several times. The parsing process is performed by an XSL Transformation (XSLT) of the design files if written in XML, or by a Perl© script otherwise.

5.3.1 The WebML Case

The present research work is based on using WebML as the WADM, defining the set of rules needed for an accurate mapping process. WebML is composed of distinct models, namely: *data*, *hypertext*, and *content management*. As already noted, the data model will not be covered by the WDL. The WebML hypertext model defines the *units*, *pages*, *links*, *global parameters*, and *hypertext organisation* of the Web application's front-end interface. The content management model deals with operations on data such as *creating*, *modifying*, *deleting*, and *manipulation* of the Web application's database content, and also defines general-purpose *operations* which may invoke external programs. The mapping process consists of the translation of the WebML elements into WSM entities described using WDL, and the extraction of their *Model* attributes. Note, however, that there is no mapping of the *Runtime* attributes, since these are purely simulation-related. The third step – elimination of duplicates and verification of correctness – does not involve the WAD description files, therefore does not depend on a specific WADM but on the resultant WDL file.

From a WDL perspective, WADM elements can belong to one of six possible patterns: Presentation, Navigation, Functional, and three Content Patterns. The Content Patterns are subdivided into three categories to reflect the manner in which each of them interacts with the WSM Data entity. The Database Publishing Pattern accesses the central database for retrieving information to be rendered on a page; the Database Management Pattern modifies the database content; and the Data Management Pattern directly handles the dataValue attribute of Data entities, either by retrieving, or setting their content.

Depending on the WebML design element, the mapping procedure can create a different number of WSM entities. Some have a one-to-one mapping rule; however, there are special cases where one WebML element will map into more than one WSM entity or, conversely, more than one WebML element will be aggregated into one WSM entity. WebML elements are grouped into patterns. These patterns define the way WebML elements are related to the WSM layers and entities (see Table 5.14).

WebML	**WDL**		
Element	**Entities**	**Pattern**	
page, site view, and area	Page and Link	*Presentation Patterns*	
link, OK-link, and KO-link	Link	*Navigation Patterns*	
Generic operation, and sendmail units	Script (Data,Link)	*Functional Patterns*	
Data, multidata, index, multi-choice index, hierarchical index, scroller units	Data and Script	*Database Publishing Pattern*	*Content Patterns*
create, delete, modify, connect, disconnect, login, logout units	Data, Script and Link	*Database Management Pattern*	
global parameter and entry, set, and get units	Data	*Data Management Pattern*	

Table 5.14: The WebML Elements and the WDL Patterns.

The Presentation Pattern

Perhaps the most logical starting point for defining the templates and rules is the WebML page element.[1] This is the principal container of the remaining WebML elements, and it has an almost one-to-one correspondence with the WSM Page entity. The template used with this element is represented in Table 5.15, and the rules are shown in Table 5.16. Note that only the structure of the Page entity is shown here; however, if the page is a "landmark", which means that it is reachable from all other pages within the same site view, the parser will construct as many Link entities as needed, and the target field will be set with this page's identifier. WebML "AND" sub-pages translate into a frameset, with as many frames as the number of sub-pages. WebML "OR" sub-pages, on the other hand, are mapped into separate Page and Link entities, with each sub-page translated into a (Page,Link) pair. The WebML site view and area elements are used to set the pageContext attribute of the entity; this is done during the second step of the mapping procedure, by inspection of which area and siteView to which the page belongs. However, if the "home" attribute is set, the pageContext will contain an additional 'INDEX' tag to indicate the starting point of the application. This, however, does not mean that the simulation will always start from this page, since it can start from any page the developer chooses to. Note that the 'DYNAMIC' value of the *pageBuild* attribute is not used with WebML, since all script modules are considered server-side.

The Navigation Pattern

Without any doubt, links are the most ubiquitous of all the hypertext entities. The WebML link element models not only the hypertext link but also communication between other elements. Its properties are mapped into the WDL Link attributes

[1]Note: In the following templates and rules, the WebML syntax defined in (Ceri, Fraternali, Bongio, Brambilla, Comai and Matera 2002, p.526-532) is followed. When referring to WSM and WDL, the words 'entity' and 'attribute" are used.

WebML	**WDL**

<PageDef>::=
  Page <PageName> [home] [landmark]
  "(" [units <ContentUnitName>
  {"," <ContentUnitName>}";"]
  [and-pages <PageName>
  {"," <PageName>}";"]
  [or-pages <PageName>
  {"," <PageName>}";"]
<ContentUnitName>::=<DataUnitName>|
<MultidataUnitName>|<IndexUnitName>
<ScrollerUnitName>|<EntryUnitName>
	entity Page **is**
	  pageID: '<PageName>';
	  pageStructure: ('PAGE' | 'FORM'
	      | 'FRAMESET');
	  pageBuild: ('HTML' | 'ASP');
	  pageComponentsID: {<ContentUnitName>};
	  pageContext: **String**;
	end entity Page;

Table 5.15: Template Which Maps a WebML Page into WDL.

Attribute	**Rules**
pageID	<PageName> property of the WebML *page*
pageStructure	'FORM' if contains a WebML *entry* or a *multi-choice Index unit* 'FRAMESET' if there are any *nested pages* 'PAGE' otherwise
pageBuild	'ASP' if contains any WDL sever-side Script 'HTML' otherwise ('DYNAMIC' is not used with WebML)
pageComponentsID	array of the WebML *page* components {<ContentUnitName>}
pageContext	<SiteViewName>.<AreaName> to which this *page* belongs <SiteViewName>.<AreaName>.'INDEX' if the WebML *page* has the *homepage* property set

Table 5.16: Setting the WDL Page Attributes.

following a template (Table 5.17) and set of rules (Table 5.19).

The Link entity attribute setting is similar to the Page case, in which the parser searches for the right values of the Link attributes in the WebML design model file (see Table 5.19). The *linkType* attribute defines which classes of entities it is connected with. If it is of a 'SUBMIT' type, then parameters are being passed from a Web form to a Script; if both ends of the Link are Script entities, then it is of the 'TRANSPORT' type; the remaining cases set this Link attribute to 'LINK'. The source and target attributes are set simply by looking at both ends of the WebML link element and mapping them accordingly. The link parameters are added to the

WebML	WDL

<LinkDef>::= link <LinkName> [automatic] [transport] "("from <LinkSource> to <LinkDest> [";" parameters <ParamDef> {","<ParamDef>}] [";" type (automatic|manual)] [";" newWindow ":" (True|False)]	**entity** link **is**
	linkID: '<LinkName>';
	linkType: ('LINK' | 'SUBMIT' | 'TRANSPORT');
	linkSource: '<LinkSource>';
	linkTargetID: '<LinkDest>';
	linkParametersNames: {<ParamDef>};
	linkTargetWindowOptions: ('BLANK' | 'SELF');
	end entity Link;

Table 5.17: Template for the WebML Link.

WebML	WDL

link se2re
(from Search to Results
parameters Artist.FirstName
parameters Artist.LastName)
	entitylink **is**
	linkID: 'se2re';
	linkType: 'LINK';
	linkSource: 'Search';
	linkTargetID: 'Results';
	linkParametersNames: {'Artist.FirstName',
	'Artist.LastName' };
	linkTargetWindowOptions: 'SELF';
	end entity Link;

Table 5.18: Example of Mapping a WebML Link Element.

linkParamNames array of identifiers. Finally, the attribute that defines where the target entity is going to be rendered is set. There are two options for this attribute when dealing with WebML – a new window will be opened, or the target will be loaded into the same window in which the caller element resides.

The Functional Patterns

These patterns are only found in two of the WebML elements – the sendmail and the generic operation units. The former is a call to a specific operating system's service of which, from a WDL perspective, the workflow is not required to evaluate the functional aspects of the design. Therefore, the mapping of such elements is a Script entity with an empty workflow. On the other hand, the generic operation unit allows designers to define a module which is executed outside the WebML context

Attribute	Rules
linkID	WebML <LinkName>
linkType	'SUBMIT' if the source is a Page entity of the 'FORM' structure 'TRANSPORT' if source and target are Script entities 'LINK' otherwise (e.g. connecting two Page entities)
linkSourceID	WebML <LinkSource>
linkTargetID	WebML <LinkDest>
linkParamNames	WebML <ParamDef>
linkTargetWindowOptions	'BLANK' if WebML <newWindow> is 'True' 'SELF' otherwise

Table 5.19: Setting the WDL Link Attributes.

(Ceri, Fraternali, Bongio, Brambilla, Comai and Matera 2002, p. 163). WDL goes a step further with this unit – it allows designers to define its workflow, the rationale being that this is the only unit which does not have a fixed workflow template and maybe there are some functional aspects that the developer would like to see simulated. Table 5.20 shows the template for the generic operation unit.

WebML	WDL

<OpUnitDef>::=
external <OpUnitName>
["("parameters
[<OpParamName> := <ParamName>
{","<OpParamName> :=
<ParamName>}]")"]
	entity Script **is**
	/ Model */*
	scriptID: <OpUnitName>;
	scriptSide: 'SERVER';
	scriptVariablesNames:{<OpUnitName> [custom-defined]};
	scriptVariablesType: {'IN' | 'OUT' | 'INOUT'};
	scriptComponentsID: {[custom-defined]};
	end entity Script;
	
	architecture behaviour **of** Script **is**
	begin
	runScriptbehaviour: **process is**
	begin
	...
	end process runScriptbehaviour;
	end architecture behaviour;

Table 5.20: Functional Patterns.

The Content Patterns

Since WebML is especially suited for data-intensive Web applications design, it constitutes no surprise that most of its elements are dedicated to some sort of data manipulation. Of the three identified *Content Patterns*, two of them – the *Database Publishing* and the *Database Management Patterns* – access the central database. From a WDL's perspective, access to a database requires at least two entities: a Data entity which acts as a proxy to the database engine, and a Script which specifies the SQL statement to be executed. Different types of database accesses will determine different workflows of the Script entity, depending on the specific template applied. Table 5.21 defines the Data entity required by these two specific patterns. The multiplicity and persistence attributes indicate that this is a shared resource and that the object will not be discarded when ending a user session.

WDL

entityData **is**
    dataID : 'database';
    dataType : 'DATABASE';
    dataSubType : 'SQL';
    dataSource : 'webml_database';
    dataMultiplicity : 'SHARED';
    dataPersistence : 'APPLICATION';
end entity;
 	
entityScript **is**
    ...
    scriptComponentsID : {'database'};
end entity;

Table 5.21: The Required Pair of Data and Script Entities Needed to Access a Central Database.

The *Database Publishing Pattern* consists of accessing the central database, retrieving the required data, and dynamically constructing a page. There is no

Attribute	**Rules**
scriptID	WebML <UnitName>
scriptSide	always 'SERVER'
scriptVariablesNames	all the parameters of incoming and outgoing links
scriptVariablesTypes	'IN' for the incoming link parameters 'OUT' for the outgoing link parameters
scriptComponentsID	at least the 'database' Data entity

Table 5.22: Rules for the Script Entity when Mapping a *Database Publishing* and *Database Management Patterns*.

modification whatsoever of the data contained in the database; rather, the data is gathered and rendered on a page. The manner in which the data is retrieved depends on the element and, consequently, a slightly different SQL statement will be included in the Script workflow. Furthermore, since the data will be rendered on a Page entity, whose identification depends on which WebML page the element belongs to, its content will be dynamically constructed from the Script workflow. Of the six elements of the Database Publishing Pattern, two of them – the Data and Multidata units – simply add information to the rendered Page, whereas the remaining four – index, hierarchical index, multi-choice index, and scroller units – also dynamically construct the necessary Link objects from the Script workflow. This difference can be noticed on the workflow since the special function *buildPage* takes either the Link object or a string as parameter – the former constructs a Link whereas the latter adds information to the Page. Note, however, that these Link objects will be instantiations of an already defined Link entity connecting the Script with another entity. Therefore, there will be no new Link entities created, but rather instantiations of a Link entity with the parameters' values set to specific values.

The templates used to map the Database Publishing Patterns are very similar to each other. Differences between the WebML data and multidata are only in the way data is selected from the database. The data unit retrieves one record from a table, whereas the multidata selects and orders multiple records. Therefore, the multidata unit can be considered as a repetition of the data unit (Ceri, Fraternali, Bongio, Brambilla, Comai and Matera 2002, p. 82). Table 5.23 displays the template used with a data unit [2] where the specific Script workflow is represented. First it reads the Data object which models the shared database object and is contained in the Script components array. Then it constructs the SQL statement with the appropriate WebML properties included, which is then used to get the results from the database Data object. This workflow is the same for each WebML data unit present in the design models, only altering some attributes such as tables and records to be accessed. The setting of these attributes are carried out during the second pass of the parser, by extracting the necessary values from the WebML elements.

WebML	**WDL**
<DataUnitDef>::=	**entity**Script **is**
DataUnit <DataUnitName>	scriptID : ‘<DataUnitName>’;
“(” source <EntityName>	scriptSide : ‘SERVER’;
[“;” selector <SelectorDef>	scriptVariablesNames : {<SelectorDef>};
{“,”<SelectorDef> }]	scriptVariablesType : {‘IN’};
[“;” attributes <AttrName>	scriptComponentsID : {‘database’};
{“,” <AttrName> }] “)”	**end entity**;
<DataUnitName>::= <Name>	
	architecture behaviour **of** Script **is**
	begin
	runScriptbehaviour: **process is**
	begin
	readObject(‘database’);
	String sqlStatement =
	“SELECT <AttrName> {“,” <AttrName> }
	FROM <EntityName>
	WHERE <SelectorDef>
	{“,”<SelectorDef> }”;
	ResultSet rs = database.getData(sqlStatement);
	buildPage(pageID, rs);
	end process runScriptbehaviour;
	end architecture behaviour;

Table 5.23: Template Which Maps a WebML Data Unit into WDL.

[2]Since all the *Database Publishing* and *Database Management Patterns* originate the same Data entity, which models the database, the next tables will omit it.

WebML	WDL
DataUnit ShortArtist	**entity**Script **is**
(source Artist;	scriptID : 'ShortArtist';
selector FirstName="Jim";	scriptSide : 'SERVER';
attributes FirstName,LastName)	scriptVariablesNames : {'FirstName'};
	scriptVariablesType : {'IN'};
	scriptComponentsID : {'database'};
	end entity;
	architecture behaviour **of** Script **is**
	begin
	runScriptbehaviour: **process is**
	begin
	readObject('database');
	String sqlStatement =
	"SELECT FirstName,LastName
	FROM Artist
	WHERE FirstName='Jim' ";
	ResultSet rs = database.getData(sqlStatement);
	buildPage(pageID, rs);
	end process runScriptbehaviour;
	end architecture behaviour;

Table 5.24: Example of Mapping a WebML Data Unit.

The remaining four WebML elements belonging to the *Database Publishing Pattern* – index, hierarchical index, multi-choice, and scroller – dynamically construct Link objects besides the Data and Script entities. There are, however, slight variations among them. The multi-choice forces the Page on which it is contained to be of the 'FORM' type, and also dynamically creates Data objects for each of the records retrieved. These Data objects emulate the checkboxes the user can choose from. The scroller unit provides a manner to scroll through the records of a database table. This unit is a special case of a multidata unit implementation, since it scrolls through all the records of a table and displays them one at a time. WDL does not considers look and feel aspects to be important for the functional evaluation and, for this reason, this unit is implemented as displaying the same information a multidata would. This does not interfere with the functional assessment since all the information is simultaneously presented, instead of one record at a time. However, to implement the same functionalities the scroller unit possesses, a Link entity is created to emulate the click on one of the scroll cursors which, in reality, takes the user to the same Page. The index unit dynamically constructs a Link object for each record in a table, which can be seen in Table 5.25 where the *buildPage* function takes the Link entity to be dynamically included in the resulting Page.

WebML	**WDL**
<IndexUnitDef>::=	**entity**Script **is**
IndexUnit <IndexUnitName>	scriptID : '<IndexUnitName>';
"(" source <EntityName>	scriptSide : 'SERVER';
[";" selector <SelectorDef>	scriptVariablesNames : {[from-incoming-links]};
{","<SelectorDef> }]	scriptVariablesType : {'IN'};
[";" attributes <AttrName>	scriptComponentsID : {'database'};
{"," <AttrName> }] ")"	**end entity**;
[";" orderby <OrderByDef>	**architecture** behaviour **of** Script **is**
{"," <OrderByDef> }] ")"	**begin**
<IndexUnitName>::= <Name>	runScriptbehaviour: **process is**
	begin
	readObject('database');
	String sqlStatement =
	"SELECT { <AttrName> }
	FROM <EntityName>
	WHERE
	{scriptVariablesNames = <SelectorDef>}
	ORDER BY {<OrderByDef>}";
	ResultSet rs = database.getData(sqlStatement);
	while(rs.Next()) {
	builPage(pageID,
	Link(rs.getObject('<AttrName>')));
	};
	end process runScriptbehaviour;
	end architecture behaviour;

Table 5.25: Template Which Maps a WebML Index Unit into WDL.

WebML units belonging to the Database Management Pattern access the central database with a different purpose – to create, modify, or delete information. This, however, does not imply a significant difference of the templates, since the

Data entity modelling the database still exists and the Script workflows, instead of querying the database, execute a 'INSERT INTO', 'UPDATE', or 'DELETE' SQL statements. Its parameters are, once again, collected from the WebML elements. The only significant difference is that these units usually follow a WebML 'OK' link if the operation was successful, and a 'KO' link otherwise. This is accomplished by issuing the *actionLink()* function from within the Script workflow, which triggers an event that the layers of the WSM receive and react to accordingly. Table 5.26 shows the WebML create unit template, where the SQL statement can be seen, and the *try/catch* block which determines which Link to follow – the 'OK' or the 'KO' Link – depending on the outcome of *setData()* function.

WebML	**WDL**

<CreateUnitDef>::=
CreateUnit <CreateUnitName>
"(" source <EntityName>
[";" <Assignment>
{","<Assignment> }] ")"
<Assignment>::=
<AttrName> ":=" <ParamName>
	entity Script **is**
	    scriptID : '<CreateUnitName>';
	    scriptSide : 'SERVER';
	    scriptVariablesNames : {input link parameters};
	    scriptVariablesType : {'IN'};
	    scriptComponentsID : {'database'};
	end entity;
	
	architecture behaviour **of** Script **is**
	begin
	    runScriptbehaviour: **process is**
	    begin
	        readObject('database');
	        String sqlStatement =
	        "INSERT INTO <EntityName>
	        (getParametersNames())
	        VALUES(getParametersValues());
	        try{ database.setData(sqlStatement);
	            actionLink('OK-link'); }
	        catch{ actionLink('KO-link'); }
	        buildPage(pageID, rs);
	    end process runScriptbehaviour;
	end architecture behaviour;

Table 5.26: Template Which Maps a WebML Create Unit into WDL.

Finally, there are the *Data Management Patterns*. These patterns entail the WebML entry, get, set, and global parameter elements. The purpose of these elements is not to access the central database, but to create, inspect or set data elements. When a WebML entry unit is found, only Data entities are produced and, since this element is purely data-based, no Script workflow is necessary to perform data manipulation. Therefore, depending on the number of fields the entry unit has, several Data entities, one for each field, are created. Their identifiers are constructed based on the name of the element and the field name (Entry unit.<FieldName>), to achieve a unique ID. These entities are always associated with a Web form, therefore their type is set to "FORMVAR" and subtype as "FIELD" by default, and will be part of the components array of a Page entity. Table 5.27 shows an example of mapping an entry unit. In this example, the entry unit has two parameters – FirstName and LastName –, and this originates the construction of two Data entities. Furthermore, their *multiplicity* and *persistence* values ensure that there will be as many instantiated objects as there are users and active sessions.

WebML	WDL

EntryUnit ArtistEntry
(FirstName Text, modifiable;
LastName Text, modifiable)
	entityData **is**
	dataID : 'ArtistEntry.FistName';
	dataType : 'FORMVAR';
	dataSubType : 'FIELD';
	dataSource : null;
	dataMultiplicity : 'MULTIPLE';
	dataPersistence : 'SESSION';
	end entity;
	
	entity Data **is**
	dataID : 'ArtistEntry.LastName';
	dataType : 'FORMVAR';
	dataSubType : 'FIELD';
	dataSource : null;
	dataMultiplicity : 'MULTIPLE';
	dataPersistence : 'SESSION';
	end entity;

Table 5.27: Example of a WebML Entry Mapping.

The global parameter is mapped into a pure Data entity. These entities are created with a specific identifier, type, and value, based on the WebML element description. Table 5.28 shows the mapping template, the only significant aspect being that, if the default value of the global parameter is expressed, the *Runtime* dataValue attribute is also set.

WebML	**WDL**

<GlobalParamDef>::=
globalParameter
<GlobalParamName>
"(" ((type <Type> [";"
initialValue <value>]) |
(type OID ";" entity <EntityName>))
")"
<GlobalParamName>::=<Name>
	entityData **is**
	    */*Model*/*
	    dataID: <GlobalParamName>;
	    dataType: ('VARIABLE'[default]
	        | 'COOKIE');
	    dataSubType: null;
	    dataSource: null;
	    dataMultiplicity: ('MULTIPLE'[default]
	        | 'SHARED');
	    dataPersistence: ('SESSION'[default]
	        | 'APPLICATION');
	    */*Runtime*/*
	    dataValue: <value>;
	end entity;

Table 5.28: Template for WebML Global Parameter.

The get and set units' templates are even simpler than the preceding ones – they merely access the *dataValue* attribute of a Data object, either retrieving or setting it. As already mentioned, no entities are created from these two elements, but the workflow of the entity that calls them will contain an additional instruction line. The get unit retrieves a data value, therefore the line to be added will be at the start of the caller's workflow to provide its value to the rest of the workflow. On the other hand, when adding the extra line, the set unit will place it at the end of the caller's workflow; this ensures that the data value contains the latest content (see Table 5.29).

A final word on the two first mapping steps: a WebML artifact is used to enforce a set of operations being either successfully executed or, in the case that

WebML	WDL
<GetUnitDef>::= getUnit <GetUnitName> "(" parameter <GlobalParamName> ")"	value = <GlobalParamName>.getData();
<SetUnitDef>::= setUnit <SetUnitName> "(" parameter <GlobalParamName>":" <ParamName> ")"	<GlobalParamName>.setData(<ParamName>);

Table 5.29: Template of a WebML Get and Set Unit.

one of them fails, a *roll-back* of the sequence being executed. This artifact, named *transaction*, implements a database concept that prevents some problems, usually encountered in a user concurrent access environment, to occur. Since this is a database-related issue and one that does not necessarily have to be dealt with by the Web application itself, WDL does not support it at this time. Furthermore, most of the existing database engines available already support the roll-back feature, and implementing it with WDL would be redundant. A possible solution to implement the transaction feature is to set, during the simulation, the present Web application state with the same content of a previous state. In other words, the runtime attributes of *some selected* objects would be set with the same values of a past state. This is a plausible solution since the content of each object during a simulation run is kept in a database.

Verification of Correctness of the Resultant WDL File

This last parsing step is necessary to ensure that no duplicates exist, and that all WSM entities generated conform with the WDL formal definition. Duplicate entities occur, for instance, when multiple shared Data entities are mapped. This is essentially the case of a shared database, mapped by multiple Content Patterns. This step enforces the redundant duplicates being eliminated. Also, verification checks that the identifier of each entity is unique. This means that each WSM entity must have its own and unique identifier tag, and also each source and target, specified by an entity's attribute, exist within the WSM set of entities. Since the parsing of the WebML files is automatic, assumption of its syntax and semantics correctness is made.

5.3.2 Mapping of Other Web Application Design Models

The previous sections dealt with the mapping of WebML designs into WDL. The set of rules was described with the WebML case in mind. These rules inspect the WebML elements properties and patterns, and set the corresponding WDL entities' attributes and construct their workflows. If, however, the design models are not WebML-based, a new set of rules may have to be applied. The more WDL model attributes can be set, the more information can be obtained by the simulation results. There are, however, a set of requirements a design model must possess in order to enable a meaningful simulation. If not met, either it will be impossible to simulate a WAD or it becomes seriously compromised.

Table 5.30 summarises what the WDL parser must be able to extract from a design in order to be able to simulate it. Note that only the Model attributes are listed; the Runtime attributes are specific to the simulation run and are not used when mapping a WAD. There are a number of "Required" attributes without which simulation is simply not possible. However, others exist that are either optional or can be deduced from their interconnections and internal components. If an optional attribute cannot be retrieved from a WAD then the simulation is still possible, although certain details related to the attribute will no longer be possible to be observed. If it can be deduced from the other components, then the WDL parser is required to apply the right rules and templates for its extraction.

The present research does not pretend to claim that is possible to simulate every conceivable design model. It would be imprudent to generalise the WebML case to other design models. However, there is a set of features that may indicate if a design model may or may not be a suitable candidate. As mentioned before, the

Requirement	Observations and Implications if not met
	Page
pageID	Required!
pageStructure	Can be deducted from links and scripts attributes
pageBuild	Can be deducted from the internal components types
pageComponentsID	Required!
pageContext	Optional
	Link
linkID	Required!
linkType	Can be deducted from the type of the source and target components
linkSourceID	Required!
linkTargetID	Required!
linkParamatersNames	If not met it still allows limited navigation, but no contextual links or Web forms
linkTargetWindowOptions	Optional
	Script
scriptID	Required!
scriptSide	Optional
scriptVariablesNames	If not met, it does not allow probing of the variables
scriptVariablesType	Optional
scriptComponentsID	Required!
	Data
dataID	Required!
dataType	Required!
dataSubType	Optional
dataSource	Required!
dataMultiplicity	If not met, multi-session simulation not possible
dataPersistence	If not met, multi-session simulation not possible

Table 5.30: Requirements a WADM Must Meet to Enable a Meaningful Simulation.

first and foremost requisite is for the design model to be formal in its syntax. This is an essential feature if an automatic mapping procedure is to be attained. Without formalism, design models can still be simulated but at the cost of developers having to write and verify the WDL code themselves. If the design model is semi-formal, meaning that some design elements follow a certain formalism, then some parts may still be automatically mapped and the remaining hand-coded; this may allow simulation with an acceptable coding effort, however, the trade-off between coding effort and simulation benefits has to be assessed carefully.

A good indication that the design model can be simulated happens when it is template-based, meaning that there is a finite and well-determined set of graphical design elements. If there is a finite set of graphical design elements, then the lexicon of the design model is finite (or, at least, its graphical lexicon). This does not automatically ensure that a mapping to WDL is possible, but is a good indication that a connection may be established between the existing design elements and a WDL description. One final and decisive attribute is the existence of an automatic procedure to map the design model into executable code. If there is a tool that automatically constructs the HTML and program code from the design model, then it is very likely that the design model can be simulated. This is a strong indication that the design model is both formal and template-based, and that a connection between design elements and executable code is possible. If the design elements can be mapped into executable code, then the probability of being able to map the design elements into WDL and subsequent simulation is very high.

Table 5.31 summarises the set of features likely to enable simulation of other design models. As can be seen, WebML does meet all the requirements. However, every design model is a different case and evaluating their simulation capabilities has to be conducted on a case-to-case basis.

Feature
Formalism
Template-based of the design elements
Existence of an Automatic Code Construct tool

Table 5.31: Features a WADM Should Possess to Enable Simulation.

5.3.3 Summary

The definition of the entities which make up the WSM have been presented. WDL is a formally defined description language, used by the WSM for the representation of Web application designs in a suitable format for simulation purposes. WDL does not, however, pretend to be another description language for Web application design but one that is especially tailored to enable and support meaningful simulation of Web application design models.

The basic entities of WSM formally described using WDL were introduced by enumeration and explanation of their intrinsic attributes and behaviour. Furthermore, it has been shown how the existent Web application design models, focusing in particular the WebML case, can be mapped to WDL. The set of rules for mapping other design models can be similarly developed, thus contributing to a common meta-language enabling simulation of different WADMs. However, each WADM is a different case and analysis of the information each can provide has to be individually assessed. For the elucidation of this issue, a set of minimum requirements that the WADM must meet was presented.

Chapter 6

The Web-design Simulation Tool

The proposed four-layer Web-design Simulation Model (WSM) and the Web-design Description Language (WDL) led to the development of the Web-design Simulation Tool. This tool takes advantage of the multi-layer structure of the WSM and the formal characteristics of the WDL to enable simulation. It supports model browsing, graphical representation of the simulation, structure and content object inspection, and automatic functional requirements evaluation.

6.1 Design and Implementation

The Web-design Simulation Tool (or *Simulator*) is designed to implement all the requirements and features of the WSM and to facilitate easy analysis of the simulation results. Table 6.1 lists the main requirements for the the Web-design Simulation Tool.

The non-functional requirements were designed to facilitate the use of the Web-design Simulation Tool by the Web developers community, and the main de-

Non-Functional Requirements
The Simulator shall be implemented using an Object-Oriented programming language; The Simulator shall be, to the extent possible, platform independent;
Functional Requirements
The Simulator shall implement the Web-design Simulation Model; The Simulator shall implement the WSM entities described in WDL format; The Simulator shall display the results in a convenient graphical manner in order to maximise the results analysis;

Table 6.1: The Web-design Simulation Tool Requirements.

cisions were the adoption of the Java^{TM}programming language and the use of the MySQL database engine, which are extensively used and platform-independent. The functional requirements enforced the use of the developed framework, and a careful design of the Simulator's graphical interface.

The Simulator has been developed based on a Model-View-Controller philosophy – the MVC paradigm (Goldberg and Robson 1983) – which advocates a separation of concerns, namely: events, model, and rendering. Each of the Model, View, and Controller objects manage the three different areas involved in the Simulator, namely the input of events for which the Controller is responsible, the processing of these events and the reaction of the model to them managed by the Model object, and the rendering of the state of the model achieved by the View object. Therefore, changes in the implementation of the View object do not have an impact on the remaining objects, making it possible to experiment with different renderings without compromising the validity and integrity of the underlying objects. To make the GUI more interactive and interesting, the rendered objects are all in the form of dynamic graphical elements (commonly known as buttons), which allows developers to interact with them to inspect or trigger a specific action. This, however, does not compromise the MVC paradigm since developer actions upon the objects are still handled by the Controller object. Figure 6.1 shows the Simulator's architecture based on the MVC paradigm. The different windows and modules of the architecture shown in the Figure will be described throughout this Chapter.

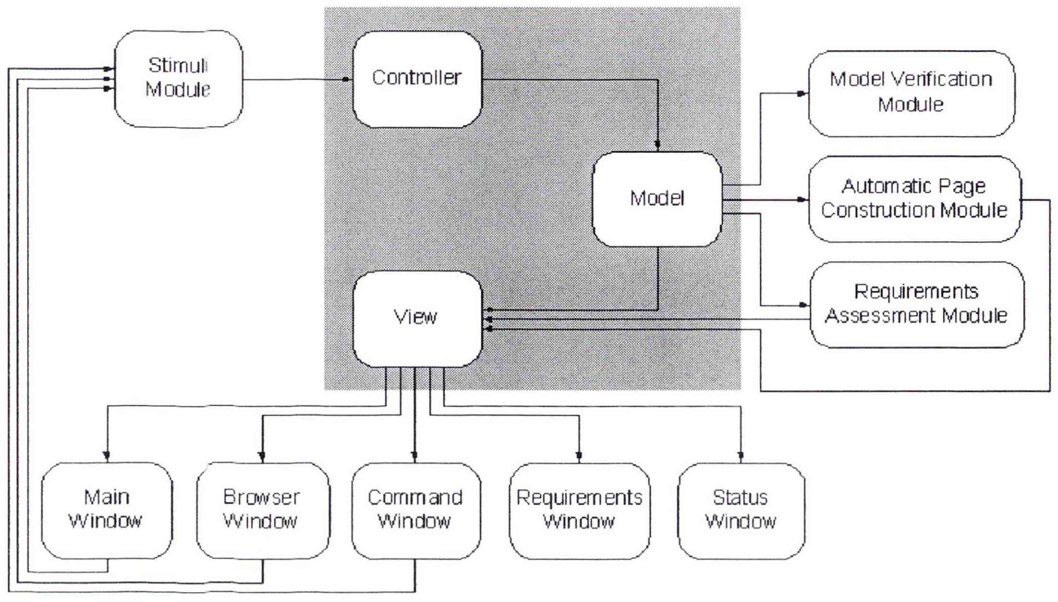


Figure 6.1: The Simulator's MVC Architecture

6.1.1 The Model and Runtime Arrays

A WDL model file consists of a number of WDL entities-architecture pairs, representing the WAD to be simulated. This representation is used as inputs by the Simulator to create arrays of Page, Link, Script, and Data entities. These arrays are named *Model Arrays* and will remain constant in structure and content during the whole simulation process. On the other hand, when the simulation process starts, the Simulator will create instantiations (objects) of these entities and keep them in the respective Page, Link, Script, and Data *Runtime Arrays*. This set of arrays, containing the *copies*, *entities instances*, or *objects*, represents the Web application state at a specific point in time. These Runtime Arrays are dynamic and their size and content change during the simulation. Each object that is an instantiation of a WSM entity receives a unique runtime identification tag (the *OID*) which serves the purpose of unambiguously distinguishing it from the pool of runtime objects. Therefore, different instantiations of the same entity can be unambiguously created, identified and selected. When any of the WSM layers no longer needs an object during a simulation run, the Simulator releases and discards it from the Runtime Array. However, the Model Array still holds the original entity and further instantiations of it are possible.

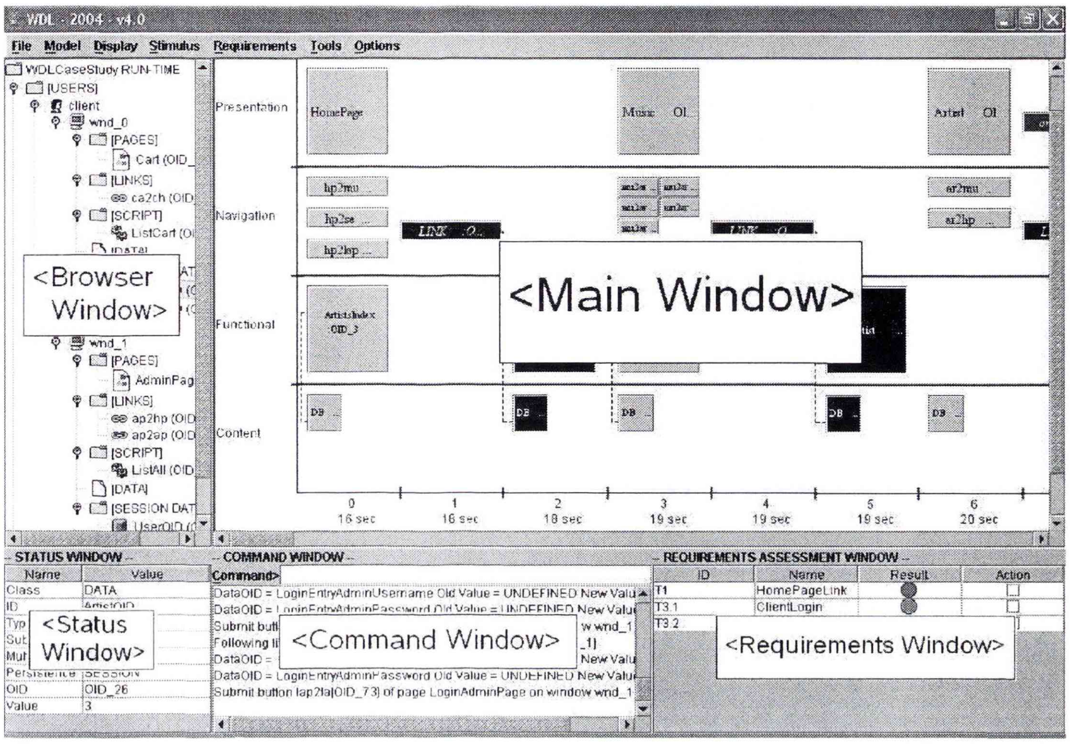
6.1.2 The Simulator's Databases

The Web-design Simulation Tool keeps a record of all the entities and their attributes instantiated during the simulation process. These records are saved on the *system database* at the end of each processed stimulus (see Figure C.1 in Appendix C). This allows the developer to inspect and analyse the state of a particular object at any point in time, and the rendition of past events.

One other database – the *model database* – contains all the data needed by the WAD for its scripts workflow. The Simulator assumes that this database has already been created, that it is structurally correct, and that contains the necessary test data for the intended simulation scenarios.

6.2 Interface

The interface of the Simulator was designed to provide meaningful graphical representation and easy interpretation of the results. It has a Main Window which displays the present state of the Web application. All past states can be scrolled back for inspection and analysis by the developer (Figure 6.2). Within the interface there are five windows, namely: the Command Window, the Browser Window, the Main Window, the Status Window, and the Requirements Window.



Name	Value
Class	DATA
ID	ArtistOID
Typ	
Sub	
Mul	
Persistence	SESSION
OID	OID_26
Value	3

```
Command>
DataOID = LoginEntryAdminUsername Old Value = UNDEFINED New Valu
DataOID = LoginEntryAdminPassword Old Value = UNDEFINED New Valu
Submit butt                                              w wnd_1
Following li                                             _1)
DataOID =                                                New Valu
DataOID = LoginEntryAdminPassword Old Value = UNDEFINED New Valu
Submit button lap2la(OID_73) of page LoginAdminPage on window wnd_1
```

ID	Name	Result	Action
T1	HomePageLink	●	☐
T3.1	ClientLogin	●	☐
T3.2			

Figure 6.2: The Web-design Simulation Tool

6.2.1 The Command Window

This area of the interface provides an input field for issuing commands and events to the Simulator. This is the primary and most basic means for interacting with the Web-design Simulation Tool. The command set ranges from user-like stimuli, such as opening windows, activate links, or entering data into fields (see Appendix B), to specific Simulation commands such as loading the WDL model, or start the simulation (see Appendix C). However, since it is command line based, it can become somewhat time-consuming, and other interface interaction options are provided to overcome this issue, namely through the Main Window.

6.2.2 The Browser Window

This area of the interface allows developers to inspect the WDL model in two distinct modes: *Model* and *Runtime*. The former is a graphical representation of the Simulator's Model Arrays, which depends on the WDL model, and represents the *entities* that the WDL model is composed of (an example is shown in Figure C.2 in Appendix C). The developer may browse or select one specific entity for inspection. This is a useful feature which provides the means to visually inspect the model in the search for design errors. However, the Simulation has automatic tools for this type of analysis, and browsing the model for identification of design errors should only be performed in situations in which there already are strong indications where the error might be located.

The Browser Window in its runtime mode allows the visual inspection of all instantiated objects, grouped by Window and User. During the simulation, objects are dynamically created and destroyed according to the behaviour and state of the Web application at any given point in time. These objects, contained in the RunTime Arrays, are conveniently displayed in a tree structure. This constitutes an excellent opportunity to observe how objects alter their values during simulation of a particular scenario (see example in Figure C.3 in Appendix C). Furthermore, in runtime mode the Browser Window allows the developer to select a specific Link to follow. This is an alternative to the command line input, and it is usually easier to select a specific Link belonging to a specific Window and User than through the Command Window.

6.2.3 The Main Window

It is in this window that the graphical representation of the Web application state during the simulation is displayed. In accordance with the WSM, the Main Window is divided into four areas, each corresponding to one of the four layers – Presentation, Navigation, Functional, and Content. This enables the simultaneous rendering of the distinct layers and objects for a better visual inspection of the Web application state. Both the Browser and Main Windows present the existing WSM objects at the current point in simulation, only differing in how that information is grouped and displayed. Therefore, these two areas constitute the primary source of information for the developer, giving a precise indication of the existing objects. The horizontal axis of the window is measured in "slots", each corresponding to a specific and discrete time of an occurrence. Simulation for functional evaluation is not concerned with performance, therefore the most important aspect of the slot is not as much the time of the event as the order in which they occur.

The Simulator distinguishes between two Web application states – *Processing* and *Idle*. They are related, respectively, to the receiving and processing of an event, and to the state a Web application would stay in if no further events occurred. The distinction between the graphical representation of these two states is made by the different background colours of the buttons representing the objects. In a Processing state objects are displayed with a dark background, whilst in an Idle state the colour is brighter. This permits a rapid visual identification of the two states by the developer (see Figures 6.3 and 6.4).

Several decisions were made concerning the representation of the diverse patterns of the state of the Web application that occur during the simulation. These patterns, although certainly important for an intuitive assessment of the results, are merely a graphical representation of a state. The patterns identified as needing special attention were:

- The rendering of the Processing and Idle states of the Web application;
- The rendering of the Page objects;
- The rendering of the Link objects;

- The rendering of the Client-Side Script objects;
- The rendering of the Server-Side Script objects;
- The rendering of the Data objects.

Page objects displayed on the Presentation layer of the Main Window correspond to the active Web pages a user would see on her/his Web browser if the WAD had been implemented. In the multiple active Page scenario the rendered button displays all the active Page identifiers (the PageID attribute). Once a Page becomes active as result of an event, it will remain in that state until further events. When this happens the runtime Page object will be discarded on the next Idle state if it no longer belongs to the set of active objects.

Link objects are a very common type of entity instantiations during a simulation run. These object will appear on the Navigation layer of the window. The rendering of these objects is achieved by displaying the active Link object during the Processing state, which indicates which Link has been followed, and the "available" ones for selection during the Idle state. These available Link objects correspond to the ones that, at that point in time, are contained on the active pages and can be triggered. During the Idle state, each Link object is rendered on the Navigation layer as a button with the label set to the LinkID attribute. When in the Processing state, the Link traversed is rendered as a button with the label set to the LinkID plus the runtime LinkOID attribute. This allows developers to quickly identify the sequence of events and the path followed.

Scripts are rendered on the Functional layer of the Main Window. During the Idle state, a Script object is displayed if it is "available" to be triggered; on the other hand, the Script objects displayed during the Processing state are the ones that are being executed. Furthermore, the Simulator makes a distinction between client and the server-side Scripts. The former has no existence without a Page object, thus

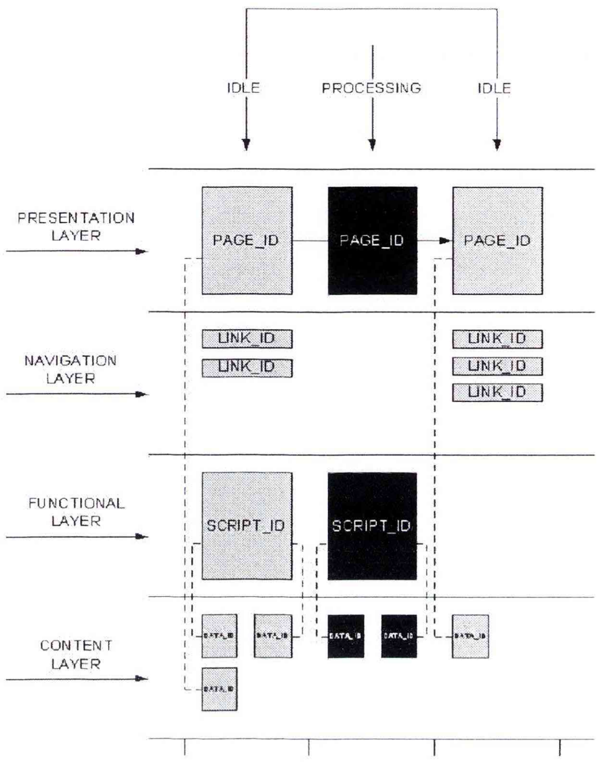


Figure 6.3: The Client-Side Script Pattern

when active the container object should also be presented (see example in Figure 6.3). On the other hand, a server-side Script does not require a Page object to be present for it to be processed; hence, the rendition of such a pattern is carried out by solely displaying the active Script object during the Processing state (refer to Figure 6.4).

Data objects are rendered on the Content layer of the Main Window by displaying the "available" objects during the Idle state, and the active ones during the Processing state. The available objects are those to which a direct reference is made by other objects, such as Page and Script objects. For instance, when a Web form page is displayed, the data fields which are all Data objects will appear as "available". On the other hand, Data objects are often needed by scripts to access

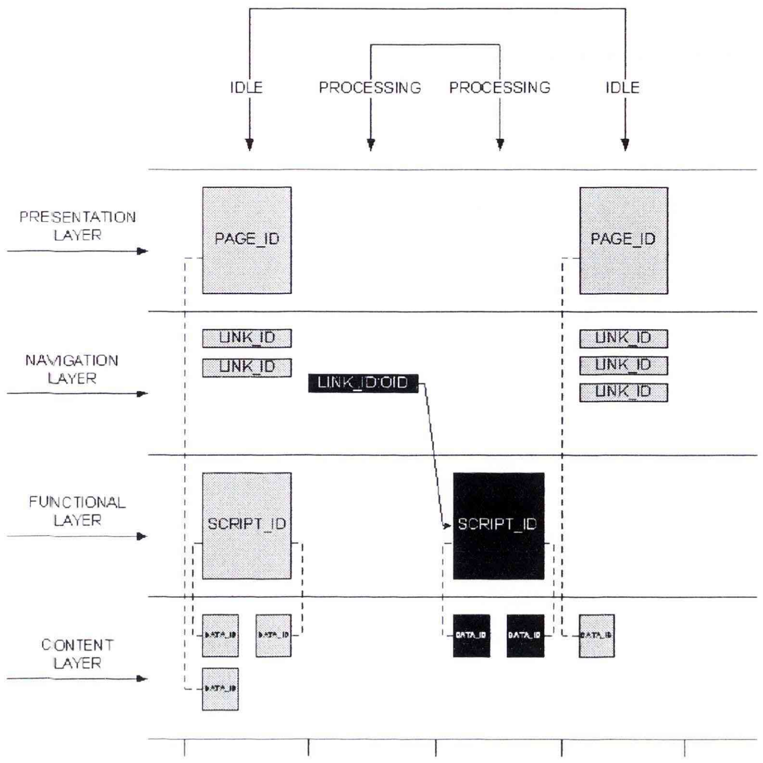


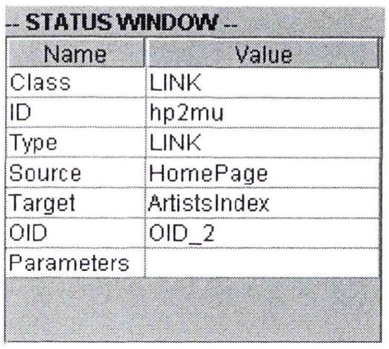
Figure 6.4: The Server-Side Script Pattern

databases, form fields, or other Web application variables; here, they are rendered as "available" during the Idle state, and "active" during the Processing state to indicate data access operations.

The Main Window allows developers to interact with the objects rendered on any of the four layers. For example, traversing a Link can be performed by clicking on the Link object, and inspecting a Script variables content is as simple as clicking on the Script object and observing the results on the Status Window (Section 6.2.4). Similarly, developers may inspecting a Data object content by simply clicking on the corresponding object button. Finally, if a Page object is clicked on, this will trigger the "Automatic Page Construction Module" described in Section 6.3.4, which will present a rendition of the Page based on its internal objects.

6.2.4 The Status Window

This is an important area of the interface that allows an in-depth inspection of the objects' internal attributes and values (see example in Figure 6.5). In order to do so, the developer has first to select a specific object either on the Browser or in the Main Window. If selecting the Browser Window in Model mode (a tree structure representation of the loaded WDL model), only the Model attributes of the object are shown; on the other hand, if it is in Runtime mode, all the Model and Runtime attribute values are shown, corresponding to the current state of the object. A more flexible approach consists of selecting an object from the Main Window, which displays the object state at a particular slot of the simulation. This is particularly useful when inspecting the attribute values of past states.



-- STATUS WINDOW --

Name	Value
Class	LINK
ID	hp2mu
Type	LINK
Source	HomePage
Target	ArtistsIndex
OID	OID_2
Parameters	

Figure 6.5: The Status Window of a Link Object

6.2.5 The Requirements Window

This window is used by the Requirements Assessment auxiliary module to display information about the functional requirements to be evaluated during simulation. The Requirements Assessment Module will be describe later in this Chapter, in Section 6.3.2, page 158. The Requirements Window displays all the requirements files loaded into the Simulator, their identification, their status, and the action the

Simulator should take once a requirement has been met – to suspend or continue the simulation. The status indicates if a requirement has been met, never met, or already met during the simulation run.

6.3 Auxiliary Modules

The Simulator has several built-in modules that automate important tasks such as stimuli construction, requirements assessment, verification of model integrity, and automatic page construction. These aid the developer in the testing of the design and further improve the Simulator's capabilities.

6.3.1 The Stimuli Module

Stimuli drive the simulation and the greater their diversity, the richer the results and analysis become. The Web-design Simulation Tool provides developers with a rich set of stimuli to interact with the simulated design. They belong to two main groups: the (1)*WSM Stimuli* which have a direct impact on the state and content of the Web application, and the (2)*Simulation Control Stimuli* which control the Simulator's mode of operation. The WSM Stimuli is further divided into *WSM Exogenous Stimuli* (or *User Interaction Stimuli*) and *WSM Endogenous Stimuli.* The User Interaction Stimuli are used by the User entity to emulate user interaction with the Web application – definition of these event can be found in Appendix B. The Simulation Control Stimuli only have an effect on the Simulator configuration such as, for instance, how to display the simulation results – these events are defined in Appendix C.

The stimuli module provides two input options – either by the Simulator's interface or by file loading. In the first mode the developer inputs the stimuli through the Command Window, the Browser Window, or the Main Window (see Figure 6.1). This is the normal procedure when the developer is simulating a Web application in a step-by-step mode and exploring specific scenarios. However, working in this mode can be rather tiresome and time-consuming when simulating large Web applications. The file loading option allows an automated mode of operation by previously constructing the events regarding a specific scenario to be simulated and analysed, and then loading them into the Simulator. This allows test cases, which enumerate a specific sequence of user interactions to achieve a particular goal, to be saved and reused with different designs. Table 6.2 shows an example of such a file; the events are either "SYSTEM" – belonging to the Simulation Control Stimuli – or "EVENT" – of the WSM Stimuli group. In the example, the first command sets a one-second interval between the triggering of two consecutive stimuli. Then, a file of requirements to evaluate is loaded, and the Browser Window is set in runtime mode. After this preliminary phase, the User actions are issued – a simulated User is created, a Window is opened and loaded with a Page, a Link is followed, a form field is set with a value, and a submit button is pressed. This simplicity of writing an event file contributes to making the Web-design Simulation Tool a very attractive tool for evaluation purposes.

```
SYSTEM(setInterStimuliDelay 1);
SYSTEM(loadRequirements T1);
SYSTEM(tree Runtime);

EVENT(createUser client);
EVENT(openWindow HomePage client);
EVENT(actionLink hp2mu HomePage wnd_0);
EVENT(setData Artist.FirstName Frank);
EVENT(actionButton se2re Search wnd_0);
```

Table 6.2: Example of an Event File.

6.3.2 The Requirements Assessment Module

Although visual inspection of the simulation displayed on the Main Window provides all the necessary data for functional evaluation, a tool has been developed to help developers assess functional requirements. The *Requirements Assessment Module* is composed of an editor and window where, respectively, the requirements are coded and the results are presented. This feature allows developers to let the tool verify the requirements by itself without solely relying on visual inspection for that purpose. Furthermore, complex assessments can be performed, as will be presented in the following paragraphs, consisting of past and present simulation states.

The core of the requirements assessment coding is based on the state of the WSM objects at a given point in time during the simulation. The developer defines a number of objects by setting their properties' values, constructs a boolean expression with them, and let the requirements assessment module to confirm its logical value throughout a simulation run. When, during the simulation, the expression is calculated as true, the module suspends the simulation and signals to the developer that the requirement has been met. The requirements code is composed of the common Java^{TM}commands, such as block constructions (for example, *while* blocks) and conditional keywords (for example, *if-else-clauses*). The only new commands to be learnt relate to the manner in which the object attributes are set or retrieved, and a limited set of very specific keywords for the assessment of the requirements. The specific command for setting or retrieving an object attribute value depends on the entity type. Tables 6.3 and 6.4 present only the *set* requirements commands; however, for each *set* command there is a *get* counterpart command. Similar to the events files, these requirements files can be written and saved for reuse with different designs. Moreover, the right combination of stimuli and requirements files allows a complete description of scenarios to be simulated and evaluated.

Furthermore, a boolean expression with the defined objects can be constructed using several logical functions that this module implements. These functions verify the existence or past occurrences of a defined object. The simplest of these is the *Exists()* function; by calling it with an object as its parameter, the

PAGE	LINK	SCRIPT	DATA
setId( )	setId( )	setId( )	setId( )
setOID( )	setOID( )	setOID( )	setOID( )
setState( )	setState( )	setState( )	setState( )
setWindowOID( )	setWindowOID( )	setWindowOID( )	setWindowOID( )
setUserOID( )	setUserOID( )	setUserOID( )	setUserOID( )
setPageStructure( )	setLinkType( )	setScriptType( )	setDataType( )
setPageBuild( )	setLinkSourceId( )	setScriptSide( )	setDataSubType( )
	setLinkSourceOID( )	setCallerId( )	setDataSource( )
	setLinkTargetId( )	setCallerOID( )	setDataStructure( )
	setLinkTargetOID( )	setVariableValue( )	setDataValue( )
	setParameters( )		setDataPersistence( )
			setDataMultiplicity( )

Table 6.3: Page, Link, Script, and Data *set* Requirements Commands.

WINDOW	USER
setId( )	setId( )
setOID( )	setOID( )
setState( )	setState( )
setUserOID( )	

Table 6.4: Window and User Requirements Commands.

module asserts if, at the present time, a *similar* runtime object is present in the runtime array. By *similar* it is meant that all the object's attributes are checked for parity with the corresponding attribute of a simulated object. The way the module assesses their similarity is by the boolean conjunction of all the checked attributes, disregarding for comparison all those that were not defined. Other functions are available for more complex logic expressions. These are presented in Table 6.5 and their return value is a boolean "true" or "false", depending on the past and current states of the simulated objects and their parameters. A "true" logical value indicates that the requirement has been met, whereas a "false" signifies that, at the present moment in the simulation, the required conditions have not yet been reached.

For the sake of clarification, a fairly simple example is presented in Table 6.6. Here a new Link object is defined with only two of its attributes set: the State and the TargetId. This leaves all the remaining attributes free from any constraints,

Function	**Result**
Exists( )	True if there is at least one similar object
DoesNotExist( )	True if there are no similar objects
OnlyOne( )	True if only one similar object exist
All( )	True if all objects of the same type are similar
Happened( )	True if a similar object exists or existed in the past
NeverHappened( )	True if a similar object never existed or happened in the past

Table 6.5: Functions for Evaluation of Requirements.

which the requirements assessment module interprets as true for *any* value. In this example, the requirement specifies that a Link must be active and that its target must be an object which has its identification field set to "HomePage". The last two lines of the code state that the requirements will be met (or "true") if a similar Link exists, and not met (or "false") otherwise.

```
Link l = new Link( );
l.setState( "ACTIVE" );
l.setTargetId( "HomePage" );
if(Exists(l)) return true;
else return false;
```

Table 6.6: A Simple Requirement Function.

6.3.3 The Model Verification Module

For checking purposes the Simulation has a built-in verification tool that allows an automatic assessment of the structure of the model. This reassures the developer that the model does not contain any design errors such as "orphan pages" – unreachable Page entities –, "cul-de-sac pages" – Page entities without outgoing links –, Link entities with unknown target or source entities, and "orphan" Script and Data entities – Script and Data entities which are not components of or linked to any other entity. The two levels of severity – "warning" and "error" – are presented to the developer and evaluation of the degree of inconsistencies can be readily made. This feature allows a basic assessment of both the design and WDL model, pinpointing potential sources of design problems.

This module implements a static model checking approach (Varro 2003). Static model checking is often a built-in feature of WADM commercial tools. For example, the WebRatio tool implements a static model checker in its development studio (WebRatio 2005). The WSM Simulator model verification module looks into the properties of the WSM entities to verify certain conditions. It does not require any simulation run to operate; instead, the module checks the model attributes of the WSM entities for specific conditions. It differs from simulation testing in that the module cannot verify the runtime attributes of the WSM objects since these require a simulation run to be set.

The model verification module can be used to check if the model is syntactically correct, such as, for example, if all pages are reachable. However, it cannot check attribute values that depend on the operation of the Web Application to be set, such as data submitted by a user or dynamically created objects. Table 6.7 lists what can and cannot be checked by operating this module. Note, however, that the shortcomings of the model verification module will be covered by the simulation technique implemented by the WSM Simulator. Therefore, the module should be considered as a complement to the WSM Simulator – the model verification module verifies the model on a static level, while the WSM Simulator verifies the model's dynamic aspect.

Model Verification Module	
Can Check	**Cannot Check**
Unreachable pages	Dynamically created pages
'Cul-de-sac' pages	Dynamically created links
Unknown link sources	Script output variables
Unknown link targets	Script input variables
'Orphan' scripts	User data input (form pages)
'Orphan' data	

Table 6.7: Features of the Model Verification Module

6.3.4 The Automatic Page Construction Module

Further verification of the design functionalities can be made by using this module. It consists of the rendering of a Page based on its internal objects (contained in the pageComponentsOID array). An XML file is constructed based on the Page and its objects at that time in the simulation. By applying a suitable XSLT transformation, the XML file can be mapped into a graphical representation of the Page object. Figures C.4 and C.5 in Appendix C display an example of such XML description and its XSLT mapping into an HTML page. This allows the developer to quickly have a snapshot of a conceptual rendition of a possible implementation, and to assess whether all the necessary elements are contained on the interface. This module is not concerned with the Page's aesthetic aspects, but on the conceptual objects rendered.

6.4 Verification and Validation of the Web-design Simulation Tool

Balci (1990) proposes several criteria for the verification and validation of the simulation study life cycle. One of them pertains to the verification and validation of the computarised model. This is related to software testing techniques used for the V&V of the computarised model. Whitner and Balci (1989) describe these techniques in their work, and propose a taxonomy of the techniques in six categories: Informal, Static, Dynamic, Symbolic, Constraint, and Formal. A complete list of these techniques are found in (Balci 1997), where the authors describe 77 techniques for conventional simulation models, and 38 for object-oriented simulation models. Testing the computarised model using all these techniques is out of the scope of the present work, however, informal and dynamic verification and validation techniques of the model have been made to ensure that it is an accurate representation of the proposed conceptual Web-design Simulation Model.

Model validation investigates if the simulation model behaves with sufficient accuracy, within the domain of applicability (Balci 1990). In other words, it assesses if the model is an accurate representation of the base (real-world) system (Banks 2000). Banks argues that the ideal way to validate the model is to compare both simulation output and base system output. This base system, in the present work, is some Web application implementation. Sargent (1992) proposes a classification of model validation techniques into subjective and objective (or subjectively and statistical as proposed in (Balci and Sargent 1984)). These techniques include event validation – pattern of events are used to compare model and base system; face validation – people knowledgeable about the system compare model and system behaviors; predictive validation –past system input data is used to predict output; and sensitivity analysis – the systematic changing the inputs and observing the model behaviour.

To assess the validity of the computarised model several test were carried out. These tests consisted in checking that all the WSM entities behaved as expected; that interactions among the entities were an accurate representation of a real application; and that the entities' content during simulation was correct. Tests were performed by comparing the simulation output (computarised model) with an implementation (base system) of a same Web application design. These Web applications consisted of typical small scale designs, such as: static pages interconnected by links; dynamic page construction; script execution; and data accesses. These tests included the already mentioned techniques – event, face, and predictive validations, and sensitivity analysis. Test results showed that the model behaves as expected when compared to the implemented base system, therefore indicating with a high degree of certainty the validity of the computarised model.

6.4.1 Summary

The Web-design Simulation Tool has been presented, highlighting its implementation decisions and features. How the Simulator implements the WSM and uses the entities described by WDL during the simulation process has been described. The graphical interface of the Web-design Simulation Tool was described in detail. Furthermore, additional built-in modules, in particular the stimuli and requirements assessment modules, were explained.

Chapter 7

The Experiment

To allow comparison of the information provided by both implementation and simulation methods, an experiment was conducted. The following sections present the selected Web application design, the results obtained from simulation and implementation treatments, and a discussion of the results.

7.1 The Web Application Design

The Web application design chosen for the experiment displays many features that are interesting for the research. Firstly, it is based on a sound theoretical Web Application Design Model approach – the WebML design model. Secondly, it contains several different functional design elements which contribute to evaluating the claim of WDL being able to model a wide range of distinct and common Web design options. And finally, it is based on examples from the book *Designing Data-Intensive Web Applications* by Ceri, Fraternali, Bongio, Brambilla, Comai and Matera (2002) which is considered by many as one of the best sources for the theory and practical applications of the WebML design model.

The adapted WebML design is a typical e-commerce application representing an online music store (see Figures D.2 and D.3 in Appendix D – "The Experiment Design") with two "site views" – "Client" and "Administration". These correspond to two main groups of users – the "client" and the "administrator" – who have two distinct roles: the "client" is a user who accesses the Web application with the objective of purchasing albums, while the "administrator" is the manager of the Web site. The "Client" site view is further divided into two "areas" – the "ClientArea" and the "ClientLogin". The "ClientArea" allows users to browse a database of artists and their albums, adding the selected ones to their individual shopping carts. Users are identified by the tuple *(username,password)* validated against the records contained in the database by the "Login" unit. A search of records is carried out either by selecting an artist from a list displayed on the "Music" page, or by entering the artist's name in the "Search" Web form. This search produces a short biographical record of the artist and her/his available albums, and the results are rendered either on the "Artist" or the "Results" pages. When satisfied with the search result, the client may proceed to add it to his shopping cart, whereby the "CreateItemCart" unit creates a new entry in the "cart" table of the database. However, to be eligible to do this, a client must first be logged in using the "LoginPage" form. The script "CheckLogin" checks that the global variable "CurrentUser" has already been set with a value different from the default 'UNDEFINED'. If it has not, the user is required to login. If the login is successfully carried out, the "client" is redirected to the default "HomePage"; if not the client is redirected to the "LoginError". However, if the "client" has already logged in, a new entry in the shopping cart is written in the database. At the end, the client is confronted with all the items contained in the shopping cart which are displayed on the "Checkout" page.

The "Administration" site view is accessed by login as an "administrator" on the "LoginPage"; this WebML unit is capable of distinguishing different user roles and redirect them to the appropriated default page. In the case of an "administrator", after a successful login, the user is redirected to the "Administration" site view, and placed on the "AdminPage". The main objective of this page is to display all clients and their respective shopping cart items contained in the database. The operation of the design is supported by a database containing the artists, albums, and users' login information (see Figure D.1 in Appendix D). As mentioned previously, the Web-design Simulation Tool assumes that both the database structure and its content have been created beforehand. This assumption is not far-fetched, since the supporting database is often one of the first tasks in the design phase. Furthermore, developers often evaluate the completeness and coherence of the database structure by using *test data*. This data is usually close to a real-world scenario and, for this reason, suitable for simulation purposes.

This design contains several interesting functional elements that, once mapped into WDL, will allow observation of the Web-design Simulation Tool's features and to assess its usefulness. By selecting a wide range of WebML units, commonly found in WebML designs, the capabilities of the WDL to conveniently and accurately represent the design can be asserted (Table 7.1). The remaining WebML units not used in the present design are those that either do not contribute to significantly adding knowledge, or some of their aspects have already been covered by other units, namely, the "Multi-choice unit" and the "Hierarchical index unit", which the "Index unit" already covers when listing and choosing a specific link out of many; the "Scroller unit" which is a special and merged case of a "Data unit" and "Index unit"; the "Delete unit" and the "Modify unit", which are only operational modifications of the "Create unit"; and the "Sendmail unit", which does not exactly fit the definition of being a basic Web application design unit but more a special case of a generic operation accessing an operating system service.

In order to evaluate and compare both treatments' results, a comprehensive

Hypertext Model	
	Pages, Links, Global parameters
Publishing of Information units	*Data units, Multidata units, Index units*
Acquisition of Information units	*Entry units*
Hypertext Organisation	*Home Pages, Site view, Areas*
Content Management Model	
Predefined Operations	*Object creation, Relationship creation*
Generic Operations	*Generic operation*
Access Control	*Login, Logout*

Table 7.1: WebML Units Present in the Design.

set of functional requirements is listed, describing the intended behaviour of the system. These requirements cover a wide range of functional evaluation and complexity, so that analysis of the results may be more easily generalised to a broader population. The proposed requirements are a representative set of the commonly found Web application functionalities, which include changes of the displayed elements, user roles, workflow execution, and data manipulation:

R1- At any time the user shall be able to return to the Home Page;

R2- Browsing database items

- **R2.a-** Clients shall be allowed to view a list containing all "artists" found in the database;
- **R2.b-** Clients shall be allowed to view a short biographical description of the "artist";
- **R2.c-** Clients shall be allowed to search for a specific "artist" name;

R3- The Web application shall make a distinction between the two allowed types of users –"clients" and "administrators";

- **R3.a-** Both types of users shall be properly identified by entering the "username" and "password" on input data fields of a login form page;
- **R3.b-** verification of the "username" and "password" shall be performed by a Script;
- **R3.c-** After a successful login, a global variable shall be set to identify the user for the duration of the Web session;

R4- Purchasing selected items

- **R4.a-** A "client" user shall be able to add items to the shopping cart;
- **R4.b-** Items added to the shopping cart shall result in the creation of a new record in the database;
- **R4.c-** Each record in the "cart" table of the database shall correspond to one item contained in the "client" shopping cart;

R5- Users of the "administrator" type shall be allowed to view the "clients" shopping cart content.

These requirements cover the four aspects of Web applications we are trying to evaluate – presentation, navigation, functional, and content. Each requirements addresses a specific functional aspect of the Web application. Requirement R1 focuses on the navigation aspect; R2 aims at assessing dynamic page construction; requirement R3 is especially concerned with scripts processing; and R4 and R5 with database access. These different areas will allow a balanced functional evaluation of all the involved aspects when simulating a Web application design.

7.2 The Simulation Treatment

Prior to the simulation, the WebML design is firstly mapped into a WDL format applying the rules described in Chapter 5. This is accomplished by a transformation of the WebML elements into WSM entities using WDL, which contain all the necessary information for the simulation process. Although WSM does not propose a specific graphic representation for its entities, for the sake of clarification, a possible representation is presented in Figure D.4 in Appendix D, where the entities and their interconnections are displayed. The definition of the Page, Link, Script, and Data entities resulting from the mapping process can also be found in the same Appendix.

7.2.1 The Test Cases

To execute the treatment, each requirement is verified by a *test case.* In the Web-design Simulation Tool context, each test case is completely defined by addressing four parameters: the (1)*initial state*, the (2)*sequence of actions*, a (3)*test method* and the (4)*expected results.* The *initial state* is defined by the starting page and variable values representing the starting point of a particular scenario. The *sequence of actions* consists of an ordered list of user interactions with the Web application, comprising a sequence of navigational events and content input actions. The *test method* specifies in which manner the requirement will be evaluated which, in the Simulator context, can either be *visual inspection* or *logic expression.* The former relies on the developer observing how the Web application evolves through time, using the Simulator interface to assist in the evaluation of the test case; the latter uses the Simulator's "Requirement Assessment Module" (Section 6.3.2) for the automatic evaluation of a boolean expression. The *expected results* lists the state and content of the relevant elements to be observed in order to assert the test case.

Test Case T1 – Verifying Requirement R1

Testing the requirement R1 is performed by writing the logic preposition which will evaluate the existence of an active Link object that contains identification of the "HomePage" as its targetID value (see Table 7.2). This logic expression evaluation will return "true" if the requirement is met – meaning that, at the present time in the simulation, there exists a link whose target is the "HomePage" – or "false" otherwise. The initial state is not an important aspect to evaluate this requirement, and the user may be placed at any one Page, however, to properly evaluate its validity, all Page objects must be visited at least once. However, if it fails once, the requirement is not met and the test may end. Undoubtedly, this requirement can be easily assessed by visual inspection of the simulation results, but the automatic assessment of the logic expression makes it much easier to test.

Test Case T1
Requirement: R1

Initial state: User at any Page
 	
Sequence of actions: Navigate through all Pages
 	
Test method: Logic expression
 	
Link l = new Link( );
l.setState("ACTIVE");
l.setTargetId("HomePage");
 	
if(Exists(l)) return true;
else return false;
 	
Expected results:
A Link back to the 'HomePage' exists on every Page

Table 7.2: The T1 Test Case.

Test Case T2 – Verifying Requirement R2

This consists of three sub-test cases, one for each requirement of the R2 group, as can be seen in Table 7.3. The expected results are to be observed in the Main Window of the Simulator at the end of the test case's sequence of actions. Test case T2.1 verifies requirement R2.a by checking that page "Music" contains a list of all artists and corresponding links. In order to evaluate test cases T2.2 and T2.3, a comparison between the data contained in the database and the results of the Simulator is needed. This means that the developer must observe that a short biography is presented on the constructed Page object (for test case T2.2), and that the correct artist is found (for test case T2.3).

Test Case T2.1
Requirement: R2.a
Initial state: User at Page "Music" *Sequence of actions*: N/A *Test method*: Visual inspection *Expected results*: 1. Inspect Link objects on Page "Music" 1.1. One for each "artist" entry 2. Inspect each Link parameters 2.1 Parameter "artist.key" must be set with "OID" from database

Test Case T2.2	**Test Case T2.3**
Requirement: R2.b	Requirement: R2.c
Initial state: User at Page "Music" *Sequence of actions*: 1. Select a Link to traverse *Test method*: Visual inspection *Expected results*: 1. Inspect the traversed Link parameters 2. The constructed "Artist" Page shall correctly display the artist corresponding to the Link traversed	*Initial state*: User at Page "Search" *Sequence of actions*: 1. Input artist name in "Artist.FirstName" form field 2. Submit form *Test method*: Visual inspection *Expected results*: 1. The constructed "Results" Page shall correctly display the searched artist

Table 7.3: The T2.1, T2.2 and T2.3 Test Cases.

Test Case T3 – Verifying Requirement R3

The next set of requirements concerns the login process of the two groups of users – "client" and "administrator". Since there are two user roles, the test case is divided in two parts, with only slight differences in the sequence of actions (see Table 7.4). The logic expressions try to evaluate whether two form fields exist, a Script being executed, and the "CurrentUser" Data object with a content different from the default "UNDEFINED". This Data object is set by the mapped WebML "Login" unit when a user is validated.

Test Case T4 – Verifying Requirement R4

Verification of the requirements R4 aspires to evaluate how the Web application behaves when a user is satisfied with the search and wants to conclude the purchase. A possible test case for their evaluation consists of simulating a user searching for an item and adding it to the shopping cart (see Table 7.5). Furthermore, requirements R4.b and R4.c address the create item procedure, modelled by the "CreateItemCart" Script, and the insertion of new data into the database. The initial conditions for the test case consist of having a user on the "Results" Page after a successful search of an "artist". The user will then perform a sequence of actions that attempts to add the searched item to her/his shopping cart. By visually inspecting the database content, in particular the "cart" table, the developer can evaluate whether the Web application is behaving according to what is expected.

Test Case T5 – Verifying Requirement R5

One possible test case for this requirement evaluation consists of placing a user on the "HomePage", navigating to the "LoginPage", and performing the login as an "administrator" (see Table 7.6). This will take him/her to the "Administration" site view and to its "AdminPage" default Page; there a list of the "clients" versus

Test Case T3	
Requirement: R3.a, 3.b, 3.c	
3.1 Clients	**3.2 Administrators**

Initial state:
User at “LoginPage”
 	
Sequence of actions:
1. Set “LoginEntryUsername” Data with valid “client” username
2. Set “LoginEntryPassword” Data with valid “client” password
3. Submit form
 	
Test method:
Logic expression
 	
Data d1 = new Data( );
d1.setState(“ACTIVE”);
d1.setId(“LoginEntryUsername”);
 	
Data d2 = new Data( );
d2.setState(“ACTIVE”);
d2.setId(“LoginEntryPassword”);
 	
Script s = new Script( );
s.setState(“ACTIVE”);
s.setId(“Login”);
 	
Data d3 = new Data( );
d3.setState(“ACTIVE”);
d3.setId(“CurrentUser”);
d3.setDataValue(“UNDEFINED”);
 	
if(Exists(d1) & Exists(d2)
& Exists(s) & !Exists(d3)) return true;
else return false;
 	
Expected results:
1. Two form field Data objects set with (username, password)
2. Script “Login” executing
3. “CurrentUser” set with database value
	Initial state:
	User at “LoginPage”
	
	Sequence of actions:
	1. Set “LoginEntryUsername” Data with valid “administrator” username
	2. Set “LoginEntryPassword” Data with valid “administrator” password
	3. Submit form
	
	Test method:
	Logic expression
	
	Data d1 = new Data( );
	d1.setState(“ACTIVE”);
	d1.setId(“LoginEntryUsername”);
	
	Data d2 = new Data( );
	d2.setState(“ACTIVE”);
	d2.setId(“LoginEntryPassword”);
	
	Script s = new Script( );
	s.setState(“ACTIVE”);
	s.setId(“Login”);
	
	Data d3 = new Data( );
	d3.setState(“ACTIVE”);
	d3.setId(“CurrentUser”);
	d3.setDataValue(“UNDEFINED”);
	
	if(Exists(d1) & Exists(d2)
	& Exists(s) & !Exists(d3)) return true;
	else return false;
	
	Expected results:
	1. Two form field Data objects set with (username, password)
	2. Script “Login” executing
	3. “CurrentUser” set with database value

Table 7.4: The T3.1 and T3.2 Test Cases.

Test Case T4
Requirement: R4.a, 4.b, 4.c
Initial state: User logged as "client" User at Page "Results" *Sequence of actions*: 1. Select Link "aacl" *Test method*: Visual inspection *Expected results*: 1. "Client" should be allowed to add item 2. Inspect database 2.1 Table "cart" should contain a new record

Table 7.5: The T4 Test Case.

shopping cart items is displayed.

7.2.2 The Simulation Results

Table 7.7 summarises the Test Cases to be evaluated during the simulation treatment. The results of the simulation will be evaluated from a *Functional Content* and *Functional Information* perspective, as described in Chapter 3 – Methodology. To perform the simulation, the assumption that the *model database* containing *test data* is made (see Section 6.1.2). This carefully selected data should represent what the Web application goes through on a typical execution. For our purposes, the data used for the test cases is presented in tables 7.8, 7.9, 7.10, 7.11. The only one that has no initial data is the "cart" table, which will be accessed and modified during simulation when adding items to the shopping cart.

Of the nine test cases, three can be automatically verified by the "Requirements Assessment Module" and the remaining six require a visual inspection of the Main Window of the Simulator. In order to be able to evaluate all the test cases

Test Case T5
Requirement: R5
Initial state: User at Page "HomePage" *Sequence of actions*: 1. Navigate to the "LoginPage" through the "Search" and "Results" pages; 2. Enter valid ("username","password") for the "administrator" type in the appropriated form fields (username='admin', password='a') *Test method*: Visual inspection *Expected results*: 1. User is logged as an "administrator" 2. Page "AdminPage" lists all "clients" and their shopping carts from database

Table 7.6: The T5 Test Case.

with one simulation run, a set of stimuli was carefully designed (see Table E.1 in Appendix E). These stimuli cover all the defined test cases by following a specific path and setting the right data values, and will attempt to evaluate the test cases in the following order: T1, T2.1, T2.2, T2.3, T3.1, T4, T3.2, and T5, the rationale for this order being that the first six test cases concerns a "client" user, and the last two an "administrator" user. Therefore, by selecting a specific path and input, the simulation can evaluate each different user role at a time. The simulation starts with the loading of the WDL Model and the three requirements files for test cases T1, T3.1, and T3.2, which will be used by the "Requirements Assessment Module". Also, two User objects are created – one "client" and one "administrator" –, which allow simultaneous simulation of the two user roles. The stimuli originate a sequence of pages and data entered by each class of user, which can be observed in Figures 7.1 and 7.2. These diagrams also display on which pages the Test Cases have been evaluated. The results of the simulation are shown in Appendix E, where the Main Window of the Simulator is displayed; in this Appendix the content of the tables

Test Case		Purpose	Requirements covered
T1		Test Navigation – Existence of a link to the "Home-Page"	R1
T2		Test Dynamic page construction – Browsing database items	
	T2.1		R2.a
	T2.2		R2.b
	T2.3		R2.c
T3		Test Script processing – User roles	
	T3.1	"Client" user	R3.a, R3.b, R3.c
	T3.2	"Administrator" user	R3.a, R3.b, R3.c
T4		Test Database access – Item purchase	R4.a, R4.b, R4.c
T5		Test Database access – Shopping cart inspection	R5

Table 7.7: Summary of the Test Cases.

Table: artist			
OID	*FirstName*	*LastName*	*Photo*
1	Celine	Dion	cd.jpg
2	Lenny	Kravitz	lk.jpg
3	Frank	Sinatra	fs.jpg
4	Jim	Morrison	jm.jpg

Table 7.8: The "artist" Table Content.

Table: album		
OID	*ArtistOID*	*Name*
5	1	DionAlbum
6	2	KravitzAlbum
7	3	SinatraAlbum
8	4	MorrisonAlbum

Table 7.9: The "album" Table Content.

Table: user					
OID	*Username*	*Password*	*Type*	*Logged*	*EMail*
1	admin	a	administrator	NO	admin@musicstore.au
2	pedro	p	client	NO	pedro@musicstore.au

Table 7.10: The "user" Table Content.

Table: cart		
OID	*UserOID*	*ItemOID*
null	null	null

Table 7.11: The "cart" Table Content.

"pages", "links", "scripts", and "data" of the Simulator database is also presented.

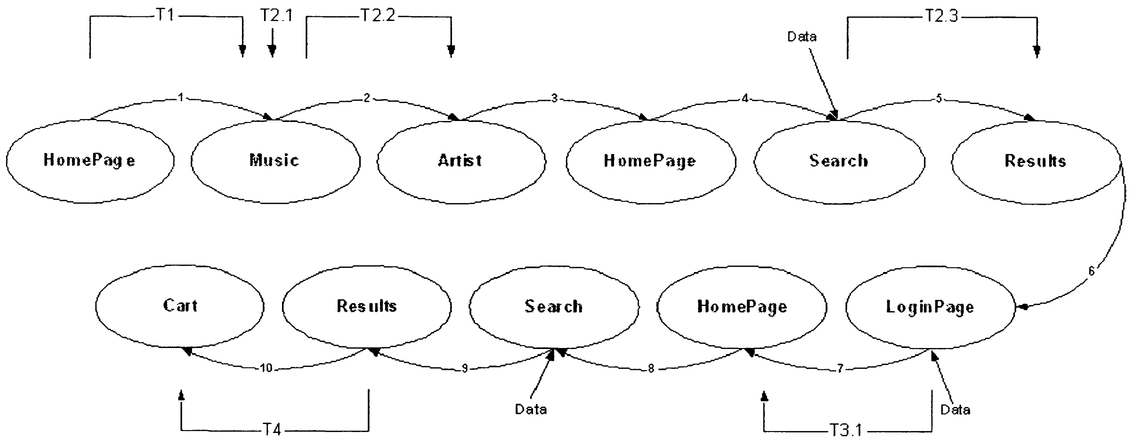


Figure 7.1: The "client" Simulation Path

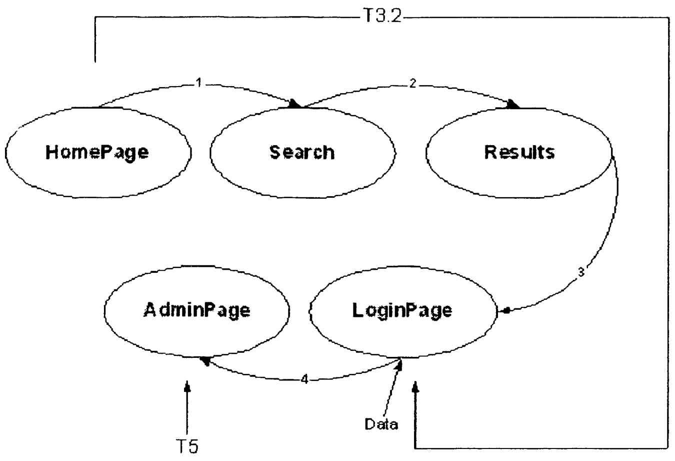


Figure 7.2: The "administrator" Simulation Path

Figure 7.3 shows the first seven slots of the results of the Simulator. Test case T1 can be promptly evaluated at slot 3 since there are no links whose target is the "HomePage". This can be seen in Table 7.12 which shows the existing Link objects at the "Music" page. Also, the constructed logic expression takes the "false" value at this point, and the "Requirements Assessment Module" signals the developer to this fact.

Table 7.12 shows the simulated Link objects at slot 3, which serves to evaluate test case T2.1. As it can be seen here, four Link objects were created with the parameter "artist.key" set to the artist identification contained in the database. It is by the comparison of the parameters values with the content of the "artist" table that test case T2.1 is verified.

	Object				
Slot	**Id**	**OID**	**Type**	**State**	**Content**
3	mu2ar	OID_9	Link	INACTIVE	artist.key=1
3	mu2ar	OID_10	Link	INACTIVE	artist.key=2
3	mu2ar	OID_11	Link	INACTIVE	artist.key=3
3	mu2ar	OID_12	Link	INACTIVE	artist.key=4

Table 7.12: Simulation Results for Test Case T1 and T2.1

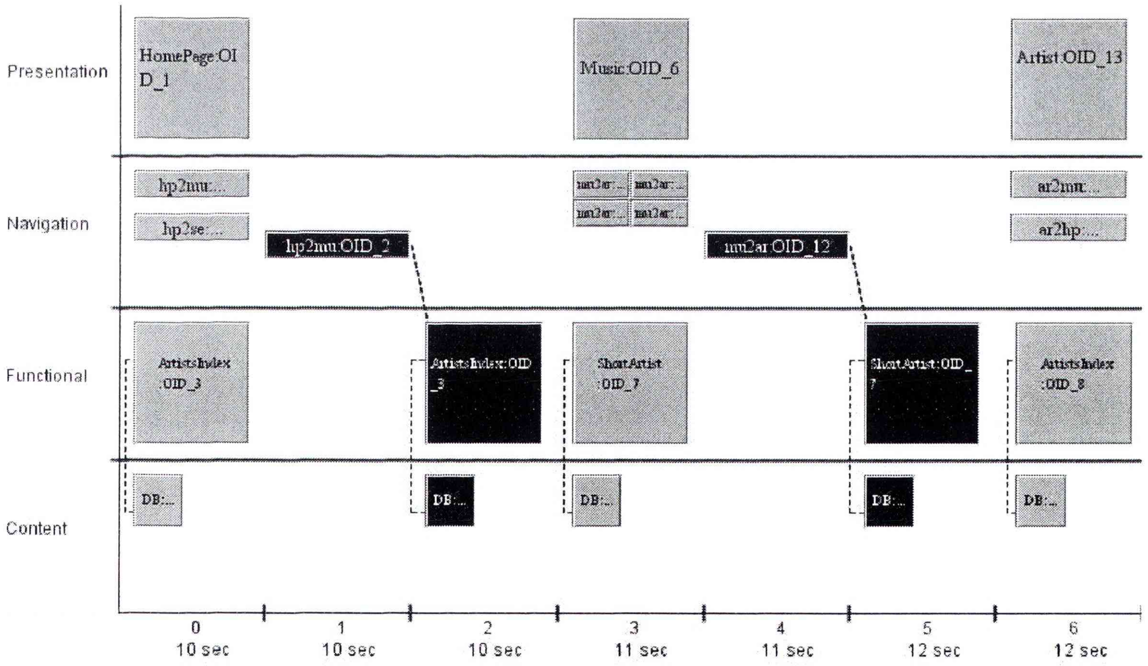
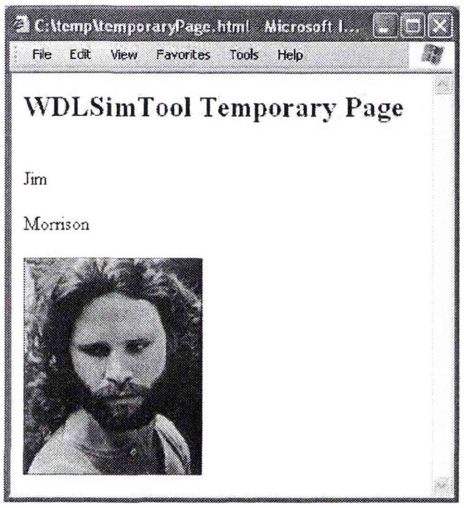


Figure 7.3: On the "Artist" Page (Slot 6)

Test case T2.2 is assessed from slot 3 to slot 6. A Link has been selected

(OID_12) and becomes active at slot 4, which triggers the "ShortArtist" Script. This, in turn, dynamically constructs the "Artist" Page, whose rendition can be observed by invoking the "Automatic Page Construction Module" feature of the Simulator (see Figure 7.4). As can be observed, the parameter of the selected Link (artist.key=4) leads to a Page which displays the correct artist biography. This is what was expected, therefore verifying the requirement.


C:\temp\temporaryPage.html Microsoft I...
File Edit View Favorites Tools Help

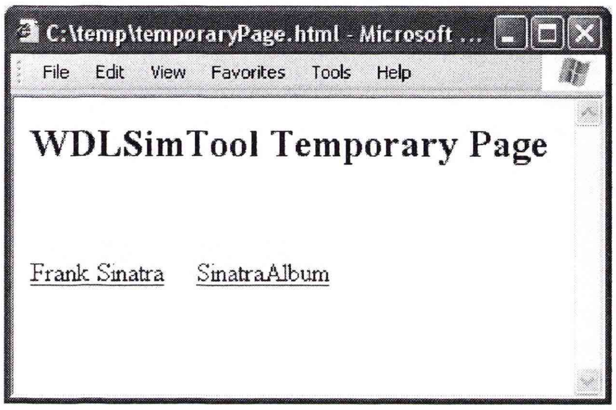
WDLSimTool Temporary Page

Jim

Morrison

Figure 7.4: The Rendered "Artist" Page (Slot 6)

We proceed to verify the test case T2.3 which assesses requirement R2.c and corresponds to slots 10 to 13 in the simulation. For this purpose, a value is entered into the form field modelled by the "Artist.FirstName" Data object, and the Page is submitted to the server which attempts to find an artist with this characteristic in the "artist" table of the database. This is performed by issuing the event *"setData Artist.FirstName Search wnd_0 Frank"*, which requests that the Data object with ID "Artist.FirstName" contained on the "Search" Page on window "wnd_0", be set with value "Frank", which is one of the valid values of the "artist" table. At the end of slot 13 the "Results" Page shall display the right content. To assert it, the form Page is submitted by the issuing the event *"actionButton se2re Search wnd_0"*, and the outcome of the automatic Page construction is inspected (Figure 7.5). By observing the constructed Page it is possible to verify test case T2.3, therefore concluding that the system meets requirement R2.c.


C:\temp\temporaryPage.html - Microsoft ...
File Edit View Favorites Tools Help

WDLSimTool Temporary Page

Frank Sinatra SinatraAlbum

Figure 7.5: The Automatically Constructed "Results" Page

	Object				
Slot	**Id**	**OID**	**Type**	**State**	**Content**
10	Search	OID_20	Page	ACTIVE	
11	Artist.FirstName	OID_21	Data	ACTIVE	Frank
11	se2re	OID_23	Link	ACTIVE	
12	Find	OID_24	Script	ACTIVE	
13	ArtistAlbums	OID_26	Script	ACTIVE	
14	Results	OID_29	Page	ACTIVE	

Table 7.13: Simulation Results for Test Case T2.3.

Test case T3.1 requires that the user be placed on the "LoginPage" and that two form fields be set with a valid (*username,password*) combination. To reach the "LoginPage", however, the user has to go through the "CheckLogin" script for the first time. This script checks if the user has already been identified within the application by inspecting the "CurrentUser" Data object content (the behaviour of this Script is presented in Appendix D). The "CheckLogin" Script has an output variable – "Success" – that will be set with the result of the Script execution. Table 7.14 shows the Script variable values between slots 14 to 16. At the end of the Script execution, the output variable "Success" is set with the value 'false', which means that the user has not yet been identified and that he/she has to login on the form page "LoginPage". However, the second time this Script is executed, between slots 26 to 28, the user has already logged in and the "CurrentUser" Data holds a valid value, therefore the Script execution results in setting the Script variable "Success" with the value 'true'.

	Object	
Slot	**Variable Name**	**Value**
14	Success	UNDEFINED
15	Success	UNDEFINED
16	Success	false
26	Success	UNDEFINED
27	Success	UNDEFINED
28	Success	true

Table 7.14: Script "CheckLogin" variables for Test Case T3.1

Going back to slot 17, when the user is redirected by the "CheckLogin" Script to the "LoginPage" form page, a valid (*username,password*) set of values is entered in the data input fields. After submitting the form, the "CurrentUser" Data shall contain the identification of the logged user. This can be observed on the simulation results at slots 17 to 20, which Table 7.15 shows.

Test case T4 checks whether a user can add items to her/his cart or not – slots 26 to 33. This requires an inspection of the "cart" database table, looking for new added items. The user is on the "Results" Page (slot 26), and tries to add a selected record. Since the user has already logged in the system, s/he is redirected to the "Cart" Page through the "CreateItemCart", "ConnectToCart", and "ConnectAlbum" Scripts which inserts a new entry to the "cart" table of the database. At the end of slot 33 the table "cart" contains one new record (Table 7.2.2) referring to the User "pedro" (UserOID = 2) having added the album with Id 3 (a "Frank Sinatra" album) to his cart, which verifies test case T4.

Slots 34 to 48 are related to test case T3.2. The new "administrator" user

Slot	Object				
	Id	**OID**	**Type**	**State**	**Content**
17	LoginPage	OID_36	Page	ACTIVE	
17	lp2lo	OID_39	Link	INACTIVE	
17	Login	OID_40	Script	INACTIVE	
17	DB	OID_4	Data	ACTIVE	
17	CurrentUser	OID_33	Data	ACTIVE	UNDEFINED
17	LoginEntryUsername	OID_37	Data	ACTIVE	UNDEFINED
17	LoginEntryPassword	OID_38	Data	ACTIVE	UNDEFINED
18-19	lp2lo	OID_39	Link	ACTIVE	
18-19	Login	OID_40	Script	ACTIVE	
18-19	LoginEntryUsername	OID_37	Data	ACTIVE	pedro
18-19	LoginEntryPassword	OID_38	Data	ACTIVE	p
18-19	CurrentUser	OID_33	Data	ACTIVE	2
20	HomePage	OID_41	Page	ACTIVE	
20	CurrentUser	OID_33	Data	ACTIVE	2

Table 7.15: Simulation Results for Test Case T3.1.

Slot	Object				
	Id	**OID**	**Type**	**State**	**Content**
26	Results	OID_53	Page	ACTIVE	
27	aa2cl	OID_55	Link	ACTIVE	
28	CheckLogin	OID_56	Script	ACTIVE	
29	CreateItemCart	OID_59	Script	ACTIVE	
30	ConnectToCart	OID_60	Script	ACTIVE	
31	ConnectAlbum	OID_61	Script	ACTIVE	
32	ItemAdded	OID_62	Script	ACTIVE	
33	Cart	OID_63	Page	ACTIVE	

Table 7.16: Simulation Results for Test Case T4.

Table: cart		
OID	*UserOID*	*ItemOID*
1	2	3

Table 7.17: The "cart" Table Content.

starts at the "HomePage" and navigates to the "LoginPage". Since the "Search" and "Results" pages have already been covered, Table 7.18 only displays the relevant results for the user login and access to the "Administration" area. These parts of the test case are represented between slots 43 to 48.

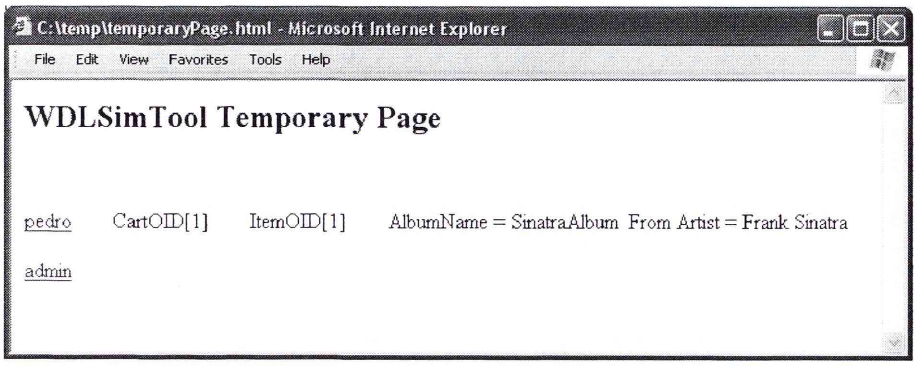
	Object				
Slot	**Id**	**OID**	**Type**	**State**	**Content**
43	LoginPage	OID_86	Page	ACTIVE	
43	lp2lo	OID_89	Link	INACTIVE	
43	Login	OID_90	Script	INACTIVE	
43	DB	OID_4	Data	ACTIVE	
43	CurrentUser	OID_83	Data	ACTIVE	UNDEFINED
43	LoginEntryUsername	OID_87	Data	ACTIVE	UNDEFINED
43	LoginEntryPassword	OID_88	Data	ACTIVE	UNDEFINED
44-48	lp2lo	OID_89	Link	ACTIVE	
44-48	Login	OID_90	Script	ACTIVE	
44-48	LoginEntryUsername	OID_87	Data	ACTIVE	admin
44-48	LoginEntryPassword	OID_88	Data	ACTIVE	a
44-48	CurrentUser	OID_83	Data	ACTIVE	1
44-48	AdminUsers	OID_91	Script	ACTIVE	
44-48	AdminCart	OID_96	Script	ACTIVE	
44-48	ListAll	OID_97	Script	ACTIVE	

Table 7.18: Simulation Results for Test Case T3.2.

The results of test case T5 are shown in Table 7.19, where it can be seen that after executing the Script "ListAll", the Page "AdminPage" becomes active. However, to fully evaluate this test case, the resultant page has to be inspected. This is accomplished by invoking the "Automatic Page Construction Module" which displays the correct contents of the "cart" table (see Figure 7.6); this verifies test case T5.

	Object				
Slot	**Id**	**OID**	**Type**	**State**	**Content**
48	ListAll	OID_97	Script	ACTIVE	
49	AdminPage	OID_92	Page	ACTIVE	

Table 7.19: Simulation Results for Test Case T5.


C:\temp\temporaryPage.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help

WDLSimTool Temporary Page

pedro CartOID[1] ItemOID[1] AlbumName = SinatraAlbum From Artist = Frank Sinatra

admin

Figure 7.6: The Rendered "AdminPage" Page

7.3 The Implementation Treatment

In order to validate the results provided by simulation, an implementation of the design was produced. For this purpose, only the elements contributing to the verification of the design functionalities were coded. However, to test the functional requirements, the interface has to provide the basic user-system interaction elements, encompassing data input and navigation elements. Furthermore, to provide the functionalities, all the scripts contained in the design had to be implemented. Although there can be several and different ways to implement the design, the WebML approach was adopted to map the design elements into an executable Web application (Ceri, Fraternali, Bongio, Brambilla, Comai and Matera 2002). To assess each functional requirements, the same *test cases* of the simulation treatment were used. However, the only method allowed is the visual inspection of the resulting Web pages and database content. The results are presented as an ordered list of initial state, actions taken, and obtained results, and, by comparing these with the expected results, the developer can verify each test case.

7.3.1 The Implementation Results

Test Case T1 – Verifying Requirement R1

Testing requirement R1 is an easy task since there are no return links back to the "HomePage" from most of the pages. Table 7.20 shows the test case T1 results. By following the link to the "Music" Web page, it is possible to conclude by observation that the requirement is not met (see Figure 7.7). However, if not using a tool to automatically analyse the Web application response, it is up to the developer to look for the required element, which can be a cumbersome technique if a large number of links exist.

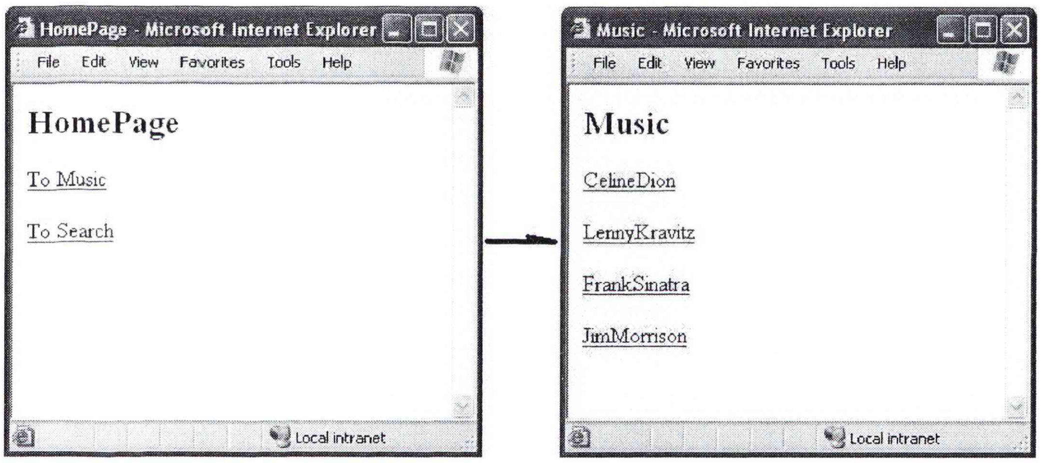

HomePage - Microsoft Internet Explorer
File Edit View Favorites Tools Help
HomePage
To Music
To Search
Local intranet
Music - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Music
CelineDion
LennyKravitz
FrankSinatra
JimMorrison
Local intranet

Figure 7.7: Testing Requirement R1

n.	Initial State	Action	Result
1	user at 'HomePage'	click on link 'hp2mu'	user at 'Music' page
Test case result:		*Failed*	

Table 7.20: Implementation Results for Test Case T1.

Test Case T2 – Verifying Requirement R2

Functional requirements R2 are related to the browsing through the artists list contained in the database. To test them, the developer has to navigate through the "Music", "Artist", "Search" and "Results" pages (see Tables 7.21, 7.22, 7.23). Once again, the results of the tests consisted of a set of Web pages and their content. By comparing these pages with the database, the developer can assess whether the underlying scripts are implementing the required functionalities. However, test case T2.1 further requires the inspection of the resulting HTML code of the links to assess whether the script produces the correct parameter values.

n.	**Initial State**	**Action**	**Result**
1	user at 'Music'		'Music' page with correct contextual links from database
Test case result:		*Verified*	

Table 7.21: Implementation Results for Test Case T2.1.

n.	**Initial State**	**Action**	**Result**
1	user at 'Music' page	Traverse one artist link	'Artist' page show correct artist biography
Test case result:		*Verified*	

Table 7.22: Implementation Results for Test Case T2.2.

Although the requirements can be asserted, the developer has no way to inspect the server global parameter *ArtistOID* defined in the design. The only alternative would be to include extra code in the implementation files to display these variables on a Web page. This is one of the disadvantages of implementation for functional assessment – testing is performed by evaluating the results from a user's perspective, making it difficult to assess what is happening on the server side.

n.	Initial State	Action	Result

1	user at 'Search' page	Enter data:
		FirstName='Celine'
		LastName='Dion'	

2	user at 'Search' page	submit form	'Results' page with correct artist
Test case result:		*Verified*	

Table 7.23: Implementation Results for Test Case T2.3.

Test Case T3 – Verifying Requirement R3

To evaluate test case T3, a similar approach is adopted. What is being tested is the application response to a user login action. The username and password are provided and, at the end of the script's execution, a global parameter (*CurrentUser*) and the database's *user* table are altered. For that purpose, the developer positions herself/himself on the "LoginPage" Web form, enters possible combinations of the (username,password) tuple, and verifies that the implementation operates as expected.

Once again, the developer has no direct means for observing the execution of the scripts, but only to visually assess their outcome in the form of the resulting Web pages. In particular, and similar to what has happened with T2, the developer cannot observe the content of the global parameter *CurrentUser* unless changes are made and introduced into the implementation code itself. Nevertheless, implementation can successfully verify requirements R3.a and 3.b; to completely verify R3.c, however, some code modification would be necessary to observe how the global parameter is affected.

n.	Initial State	Action	Result
1	user at 'LoginPage'	Enter form data 'LoginEntryUsername'='pedro' 'LoginEntryPassword'='p'	
2	user at 'LoginPage'	submit form	user back at 'HomePage'
Test case result:		*Verified*	

Table 7.24: Implementation Results for Test Case T3.1.

n.	Initial State	Action	Result
1	user at 'LoginPage'	Enter form data 'LoginEntryUsername'='admin' 'LoginEntryPassword'='a'	
2	user at 'LoginPage'	submit form	user at 'AdminPage'
Test case result:		*Verified*	

Table 7.25: Implementation Results for Test Case T3.2.

Test Case T4 – Verifying Requirement R4

Test case T4 attempts to evaluate the application's response to a user's purchase. It logically follows tests T3 – the client login – and T2.2 – searching for a specific artist. To test the requirement an inspection of the database was made, and found that it contained the new added item. However, unless the developer resorts to the design s/he cannot observe the executed scripts and path followed, but only the effects her/his actions produced. Therefore, scripts 'CheckLogin' and 'CreateItemCart' are only indirectly perceived. Furthermore, the developer cannot directly inspect the server's global parameters to assess its valid content.

n.	Initial State	Action	Result
1	user at 'Results'	traverse link 'aa2cl'	user at 'Cart' page Database with a new record
Test case result:		*Verified*	

Table 7.26: Implementation Results for Test Case T4.

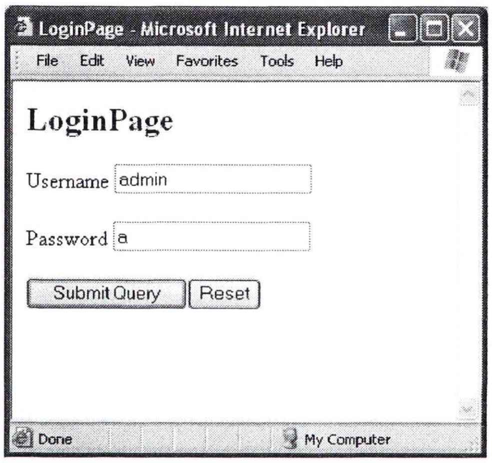
Test Case T5 – Verifying Requirement R5

The intended functionality of requirement R5 is to allow "administrators" to inspect the "client" shopping carts. Testing this implies that a user be logged as "admin", and proceed to the "AdminPage" where a list of the clients and their carts will be displayed. This logically follows test case T3.2 – the login of an "administrator".

The results showed a Web page with the right content extracted from the database, thus verifying the requirement.

n.	Initial State	Action	Result
1	user at 'AdminPage'		page correctly displays user's carts' content from database
Test case result:		*Verified*	

Table 7.27: Implementation Results for Test Case T5.


LoginPage - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Submit Query
Reset
Done
My Computer

LoginPage

Username admin

Password a

Figure 7.8: Testing Requirement R5

7.4 Verification and Validation of the Simulation Results

The simulation results described in this Chapter strongly indicate that the conceptual model and its translation into the computarised model are an accurate representation of the base system. This is supported by the results obtained by the implementation treatment, which when compared with the simulation treatment results display a high degree of correlation. This strongly suggests the verity and validity of the formulated problem, the proposed solution, the conceptual model definition, and its translation into the computarised model. It is, therefore, with a high degree of confidence that the assumption of the correctness of the results produced by simulation treatment is made.

7.5 Discussion of the Results

The results obtained illustrated the Web-design Simulation Model, Web-design Description Language, and Web-design Simulation Tool's capabilities for the functional requirements evaluation. The two used simulation methodologies – "visual inspection" and "logic expression" – allowed us to verify the suitability of the multi-layer graphic representation of the Main Window, the "Requirements Assessment Module", and the "Automatic Page Construction Module" for the functional evaluation of a Web application design. Furthermore, how the Simulator displays the WDL intrinsic entities' attributes, and how these can be used to construct logic expressions to verify functional requirements, was also shown. By using these attributes a variety of aspects of a Web application design may be simulated and tested. Moreover, by displaying the state and interactions amongst the objects contained in one of the four layers of the Web-design Simulation Model, the Simulator is capable of precisely indicating what pages are being presented to the user, what links have been traversed, what scripts are being executed, and the content of data objects. Furthermore, the history of the test can be checked at any time. Test case T1, a simple example of the use of the WDL object properties for logic evaluation, demonstrates how by setting only two attributes of a Link object and by using the "Exists" requirement assessment function, a global property of a Web application can be assessed. Test cases T3.1 and T3.2 further explored the capabilities of evaluating a set of requirements by using these logic functions; the possibility of combining different object classes in the same logic expression, allows a powerful and flexible testing method that uses the intrinsic WDL attributes for verification purposes.

The "visual inspection" method used by the remaining test cases made use of the multi-layer display of the Simulation Model. The four orthogonal perspectives of a simulation allow an in-depth evaluation of the state and dynamics of the design. This not only facilitates the testing process and provides better understanding of the Web application behaviour, but may also lead to an increase in the quality of the end product and potentially decreasing the software development cycle.

If simulation of a Web application design provides developers with a better understanding of its behaviour, the multi-session/multi-user Simulator feature takes it to the next level, as close as it gets to a real-world scenario. This is often extremely difficult to perform by simple observation of the design, since design models are not usually especially suited for and concerned with this type of scenario. Hence, it is left to the developer to envisage how the design will respond in such situations. Undoubtedly, as designs become more and more complex, it becomes almost impossible to correctly evaluate all the different scenarios. Moreover, such evaluation is error-prone and may lead to defective Web application designs that will probably only be noticed during or even after implementation. By using the Window and User entities, the Simulator is capable of managing shared and private resources, thus simulating both session and application objects. This is best observed when the "client" and "administrator" users are being simultaneously simulated. In this case, both users have their own private session and application Data objects, such as the *CurrentUser* and *ArtistOID*, and share a common database object. By allowing multi-user simulation, concurrent interaction with the Web application may be performed and eventual problems due to this type of access may be evaluated. These problems are often linked to data integrity, and in particular to its consistency, accuracy, and correctness. By simulating the design in a multi-session/multi-user environment, the application's behaviour in a real-world scenario may be better understood.

The implementation treatment was also able to evaluate the proposed set of functional requirements. The results of the test cases produced a sequence of Web pages which allowed the assessment of the intended functionalities by visually inspecting their content. Test case T1 required the visual inspection of the navigational elements contained on the Web page to assert it. Test case T2.1 further needed inspection of the HTML code to reach a conclusive assessment. On the other hand, tests T2.2 and T2.3 were easily assessed with this testing method; this is because the sequence of actions taken on the initial page and the expected results share a strong connection. In other words, when the data entered on a form or a link traversed on a page produces a very specific content on the resulting page, the implementation method performs well. In this case in particular, it is very easy to verify that the "Artist" page displays the right content due to the traversed link; similarly, it is a straightforward task to assess the content of the "Results" page due to the data entered on the "Search" form.

There were, however, some difficulties in assessing tests that intensively use scripts. This happened because there is no direct evidence of the running scripts, only their effects. Furthermore, global parameters are not observable unless changes in the implementation code are made. This was the case in tests T3.1 and T3.2, where the inspection of the server variable *CurrentUser* is not possible, and in T4 with *ArtistOID*. Test T5, on the other hand, showed that this is a suitable method for assessing scripts that result in observable content extracted from a database.

7.5.1 Comparison of Treatments

The two treatments were conducted on the same design to evaluate an identical set of functional requirements. By comparing the treatments from functional content and information perspectives, an evaluation of the benefits and shortcomings of each treatment can be made. Since both treatments agreed on the results of the verification of the requirements (see Table 7.28), the following comparison is more concerned with what each treatment can contribute to the functional testing, rather than the actual values of the results.

Requirements assessment results		
	Result	
Requirement	*Simulation*	*Implementation*
R1	**fail**	**fail**
R2		
R2.a	pass	pass
R2.b	pass	pass
R2.c	pass	pass
R3		
R3.a	pass	pass
R3.b	pass	pass
R3.c	pass	pass
R4		
R4.a	pass	pass
R4.b	pass	pass
R4.c	pass	pass
R5	pass	pass

Table 7.28: The Functional Requirements Assessment Results.

From a functional content perspective, both treatments provide similar observable factors, especially on the page level by indication of which Web pages are displayed to the user (see Table 7.29). In fact, the implementation treatment relies on these very pages to assess the functionalities of the Web application. On the other hand, the Web-design Simulator achieves it by presenting the active pages on the Main Window, and by rendering the Page objects and their internal components when using the "Automatic Page Construction Module". Since these internal simulated objects represent the structure and content of the Web application at any time during the simulation, an accurate representation of the Web page can be observed.

On a link level the treatments take a slightly different approach. Evaluation of the implementation results depends on the observation of the navigational elements on the pages to assert their existence and correct parameters values. There is no clear and direct evidence of these values, unless inspection of the HTML code itself is made. On the other hand, simulation results clearly identify which links were traversed and inspection of each link parameters is made possible by observing the runtime attributes of the Link objects on the Status Window of the Simulator.

However, if both treatments produce similar results, especially when dealing with pages and links, scripts and data do pose a different challenge. In the implementation treatment, there is no clear evidence which scripts were executed when interacting with the Web application. The only observable evidence is the effect of the scripts on the resulting Web pages and database content. Furthermore, inspection of the input and output script variables is only possible if the implementation code is changed in such a way that they are displayed along with the resulting Web page. This makes it harder for the developer to test the scripts, which undoubtedly are the most important functional element of a Web application. Simulation, on the other hand, presents which scripts were executed and the order in which they were triggered, making a clear connection between their workflows and internal interactions with the other Web application objects. This provides a better understanding of the behaviour of the Web application in faulty conditions.

Similarly to the script, the data the Web application uses for functional processing is somewhat hidden by the implementation treatment. Both treatments require direct inspection of the database content to assess some functionalities; however, simulation differs from implementation by allowing observation of the variables used by the Web application on the server side – such as global parameters – and on the client side – such as "cookies". This is an important contribution to functional evaluation, since scripts often use these in their workflows, and knowledge of their content is an essential factor to understand the Web application behaviour.

The functional information comparison aims to evaluate each treatment's difficulty of usage, suitability of the presentation of results toward testing, and level of

	Implementation	**Simulation**
Functional Content		
Pages	Both methods allow inspection of the active pages	
Links	Available links <u>*can*</u> be observed.	Available and Active links and their parameters <u>*can*</u> be observed.
Scripts	<u>*Cannot*</u> be observed executing. Outcome <u>*indirectly*</u> perceived.	Executing scripts and their variables <u>*can*</u> be observed.
Data	Global variables <u>*cannot*</u> be observed.	Global variables, form fields, and server variables <u>*can*</u> be observed.
Functional Information		
Difficulty in administering the treatment	(Partial) Coding necessary. Web server application needed for supporting Script processing.	Straightforward and automatic procedure. Coding is only necessary for very specific and custom-made (not provided by the design elements) Script procedures.
Format of the treatment results	A client's perspective. Evaluation is performed by observing the end-user graphical interface.	Both client and server perspectives. The whole test case can be observed on a timeline. Possibility to inspect both present and past states.
Adequacy of the results for functional requirements evaluation	Sufficient for functional evaluation.	High functional content and information, leading to a better understanding of the application's behaviour. Possibility to automate tests through the use of the Requirements Assessment Module.

Table 7.29: Comparison of the Implementation and Simulation Treatments.

contribution for the functional testing. Testing performed with the implementation method requires the developer to produce the necessary code. This, undoubtedly, leads to a slower testing process with the additional danger of introducing errors in the code which may jeopardise the validity of the tests. It can be argued that this additional coding effort is an advantage of the implementation method, since some of the code is produced and tested, which would contribute to shortening the implementation phase. However, using testing prototypes as a means to alleviate the workload of the implementation phase is against good software engineering practices (MacIntosh and Strigel 2000). Furthermore, Lowe et al. (1999) argue that early-stage prototyping is not always appropriate, and that it may lead to specification of requirements that although will not be met, take a considerable development effort. Implementation for functional testing purposes should not be confused with and included in the implementation phase. Furthermore, if the design does not meet all the functional requirements this effort would, at least partially, be lost. In the end, it all comes down to whether the design is or is not correct in the first place. On the other hand, since simulation is performed on the design itself, it has two major advantages: one is that it is an automatic and effortless process; secondly, no additional errors are introduced into the testing phase by faulty prototyping code. Simulation also makes it easier to observe how the Web application evolves throughout the test cases; it provides the means to present the results in a suitable and intuitive format, where the developer can go back and forth in time to inspect how the different elements reacted to the stimuli and to identify possible design problems. These interactions presented on a timeline provide the developer with a more global approach to testing, instead of merely observing snapshots of Web pages. In conclusion, the simulation treatment has the potential to provide better and further functional content and information than the implementation treatment does; furthermore, it can shed light on how the Web application will operate when implemented in a real-world multi-user/multi-session environment.

Chapter 8

Conclusion

In this chapter, the contributions of the research to the topic of functional requirements evaluation of Web applications designs are described, and a critical analysis of the findings is made. Furthermore, the hypothesis is evaluated based on the results analysis, and suggestions for future work are presented.

8.1 Summary and Critical Analysis

As the Web has evolved, so has the complexity of its applications, leaving Web developers facing an unprecedented level of integration of content and procedural processing to be tested (Pressman 2000, Lucca et al. 2002, Deng et al. 2004). Challenged to develop highly sophisticated Web applications in short periods of time, developers yearn for more efficient testing methods to improve the quality of their end-product. Testing accounts for a major part of the Software Development Cycle and Web developers are becoming overwhelmed by the test phase complexities. In fact, this is reflected by the increasing number of publications in the past few years dedicated to the Web testing topic (Kung et al. 2000*b*, Lucca et al. 2002, Elbaum et al. 2003, Nilawar 2003, Kung 2004, Xu et al. 2005, Bellettini et al. 2005, Elbaum et al. 2005).

Simulation for testing purposes has been successfully used in the hardware field for the last decade, and is still a very active field of research which is in constant evolution (Bailey et al. 2004). Its use there has allowed hardware engineers to implement increasingly complex functionalities and to experiment with alternative system designs. Freed from having to implement the systems to actually test them, engineers can simulate would-be systems in would-be scenarios with a high degree of certainty that, when implemented, they will operate as intended. Simulation of Web application design models aims to achieve similar results in the Web development field – to alleviate the test phase workload, and to provide developers with the means to experiment with would-be Web application designs and scenarios without the need to implement them.

This research has devised a way to model Web application designs in order to make possible their simulation for functional evaluation purposes. By developing a Simulation Model based on a layered structure that highlights the elements with functional significance, evaluation of functional requirements is made feasible. The proposed Simulation Model consists of four layers, namely: Presentation, Navigation, Functional, and Content. Each of these layers models a specific perspective of a Web application. Presentation is the one and only means a user has at her/his disposal to interact with the application, and it is also the only manner a user has to perceive the application's response to her/his actions. The Navigation layer models changes of the current user location in the Web application's structure, either due to user interaction with hyperlinks and buttons rendered on the Presentation layer, or to other internal elements. These changes model the mechanism by which it is possible to alter the set of displayed Web pages and their content, and the triggering of programmatic elements. However, if this layer allows navigation through the Web application, the Functional layer is what makes it dynamic. All the programmatic elements are found on the Functional layer, which models and supports their execution. Finally, the Content layer models all the data that the application uses, encompassing databases, files, global parameters, and form fields.

In addition to the Web-design Simulation Model, a Web-design Description Language (WDL) was proposed. This complements the Simulation Model and provides a formal definition of the structure and behaviour of the entities that populate each WSM layer. The description language borrows concepts from the VHDL language (*IEEE Standard VHDL Language Reference Manual - 1076* 2002), which is extensively used in the hardware field for hardware systems development. WDL acts as a middleware language into which existing Web application design models are mapped, enabling simulation of designs. Each entity described by WDL is modelled from structural and behavioural perspective. This makes simulation of an entity a straightforward task since each entity 'knows' how to process and respond to a stimulus; the individually defined behaviour formally describes how each stimulus will be processed, which is based on its workflow and the entity's structural content.

The four WSM entities identified as having functional significance are the Page, Link, Script, and Data, which are related to each of the four layers of the Simulation Model. The Page entity resides on the Presentation layer. It models the user interface in such a manner that makes interaction with the application possible. Only the interface elements that may contribute to a functional assessment are modelled; this encompasses the rendition of the hyperlinks, buttons, and form fields. In accordance with other design models approaches, there was no attempt to model the aesthetic aspects of a Web page which, although very important, for instance, for usability and accessibility evaluation, convey no significant functional meaning. The Link entity, which populates the Navigation layer, models the structure and behaviour of the hyperlinks and buttons, and also supports information flow to and from Script entities. In its simplest form, it models a hyperlink used by a user to change position within the application. It can also model a submit button which is responsible for transferring information from a Web form to the application, or can even provide a mechanism for the transport of information among Script entities. Script entities model the programmable units of a Web application and are found on the Functional layer; they are responsible for the processing of data and provide the richest functional information available in a Web application. For this reason, they are the most complex entities from a behavioural point of view. WDL allows behaviour to be individually defined for each Script, making it a flexible and scalable description language. Finally, the Data entities, which can be found on the Content layer, model the data present in a Web application and are used by Page and Script entities for rendition and data access and management, respectively. They can take the form of files, databases, variables, or Web form fields, covering the whole spectrum of information-handling. The emphasis of this type of entity is on the modelling of the elements that support functional workflow, rather than for purely aesthetic reasons. Some media types such as images, video, and audio, are not usually considered by design models on the grounds that they do not usually possess functional significance. Therefore, these media type will not usually be modelled by the Data entity. However, other types of media that may have a navigation or functional significance such as, for example, Adobe® (formerly Macromedia) Flash® presentations, can be modelled by a combination of Data and associated Link and/or Script entities which can model the specific media functionalities.

In addition to the four main entities just described, two other auxiliary entities were introduced: the Window and User entities. These are what WSM calls *runtime entities* – they do not originate from the design model mapping, but are part of the simulation process. The Window entity models the Web browser and allows a multi-session simulation; the User entity models a user and provides a multi-user simulation environment. Although not part of the design, these two entities types enrich the simulation process in ways that implementation for testing purposes is unable or makes it difficult to achieve. For example, the testing of a Web application behaviour when one user simultaneously accesses a Web application from two or more Web browser instances – multi-session testing – is difficult to perform and track with the implementation treatment due to the multitude of the Web application's variables that are being changed at the same time. By modelling the Window entity, WDL allows Web developers to easily inspect session variables and quickly detect design errors due to this type of access. Similarly, multi-user testing – the testing of a Web application behaviour when two or more users simultaneously access it – requires inspection of the variables content of the Web application, which prototype testing does not easily provide. On the other hand, the simulation treatment uses the User entity to separate each user application variables, thus providing inspection and analysis of the impact of his/her actions. The main reasons for the unsuitability of an implementation for multi-session and multi-user testing is the lack of control and level of difficulty to inspect the state of the supporting Web server application, which manages the Web application state for each session and user. Testing a prototype requires a Web server application executing the implementation and managing the implementation variables, and tests are evaluated based on the Web server responses; however, unless additional code is inserted into the Web application code itself for variable display and analysis, Web developers do not usually have the means to directly have a snapshot of the state of the Web application on the server-side. This usually leads to a partial view and analysis of the Web application behaviour, since it is performed from a client-side point of view. Simulation overcomes this problem by providing Web developers with both client and server-side points of view, and with no additional coding necessary.

To further provide support to the test phase, simulating a design should be as automatic and effortless as possible. This was accomplished by providing an automatic mapping process between design models and the WSM entities described using WDL, based on a library of templates and rules. Since WSM entities are highly configurable from a behavioural point of view, by defining the rules and workflow of each template a straightforward mapping process is achieved. This enables a design, written using some design model, to be automatically mapped into WDL and thus be ready to be simulated and tested. The advantages of this process are that developers do not need to directly deal with WDL which reduces the complexity and time required for the simulation procedure, and implementation for functional evaluation is no longer necessary. Examples of these templates were developed for WebML.

The use of controlled experimentation, although providing additional certainty of causality, has some unavoidable implications for external validity. Therefore, generalisation of this method to other design models has to be made cautiously. The design model must be formally defined and consisting of a finite lexicon. If these conditions are not met, then defining the templates for the automatic mapping is almost impossible to attain, due to the indeterministic nature of the design model. Without an automatic mapping process, the developer would have to manually code each entity which would, inevitably, lead to a lengthy and impractical procedure. On the other hand, if the design model has a formal syntax and semantics and is restricted to a finite number of design elements, then WDL is sufficiently flexible and scalable to model it. Furthermore, if a design model supports the means to automatically construct the code, then the probability of being able to simulate its designs is very high. The rationale being that automatic code construction is usually based on templates that map the design elements into executable code; this is similar to the mechanism used by the WDL parser. Therefore, if such a tool exists, there is a high level of confidence that simulation will in fact be possible. In conclusion, generalisation of the WDL capabilities to represent other design models would be imprudent to claim; however, due to the characteristics of WDL which separates structure and behaviour and allows internal workflows to be expanded, modified and tailored, it is this author's belief that there is a high probability of it being able to model other formally defined design models.

To test the suitability, feasibility, practicability and meaningfulness of the proposed WSM and developed WDL, an implementation of an application was conducted. This led to the development of the Web-design Simulation Tool (or Simulator) which was utilised to carry out an experiment. The Simulator, which started as a demonstration of a concept tool, ended up as a fairly sophisticated application with some very interesting and useful features. It provides simulation of WDL models; automatic testing of functional logic expressions; functional requirements editing; inspection of the structure and content of entities; stimuli processor and editor; and display of the simulation results from the four layers' perspective on a convenient timeline format.

An experiment was conducted to assess the developed framework for functional evaluation. It consisted of evaluating the functional requirements of a Web application design using two different treatments: an implementation method, and a simulation method. Furthermore, the experiment supported an attempted to compare both methods' benefits and disadvantages to reach a conclusion as to their suitability to tackle functional evaluation. The comparison was performed from two different perspective: *Functional Content* and *Functional Information.* The former aimed at assessing the results each treatment provides for the functional evaluation. A comparison of pages, links, scripts, and data was performed. These were evaluated regarding the observable active elements and their content. It was shown that implementation does present some shortcomings. In particular, scripts and data elements are only indirectly perceived by the Web developer when using implementation, preventing her/him from fully observing the application's behaviour, perceiving only its effects. On the other hand, both methods showed a similar Functional Content measurement of the resultant pages and traversed links. The Functional Information measurement aimed at a qualitative analysis of both treatments, regarding the degree of contribution each of them provides to functional evaluation. This perspective was further divided into three items: the difficulty of implementing the treatment, the suitability of the format of each treatment's results, and the level of adequacy of the results to evaluate functional requirements. Once again, simulation was found to be a better method than implementation. Simulation is directly performed on the design model itself via an automatic mapping process, without the need for prototyping. Moreover, it provides a better understanding of the behaviour of the designed system by allowing inspection from a client's and server's perspectives, whereas the implementation method mostly presents the system from the client's side. This is an extremely useful feature of the Simulator, since the application's functionalities are mostly and usually implemented on the server and not on the client side. Implementation can be regarded as a method through which only the *effects* of the functionalities are evaluated; the *causes* cannot be seen and may only be indirectly assessed. This one-sided observation for the evaluation of functionalities which are mostly implemented on the other end, leads to partial and incomplete evaluation. Simulation, on the other hand, supports both perspectives and groups them in a more holistic evaluation approach. The advantages of using the simulation treatment were found to exceed those provided by implementation and, although the latter provides sufficient information for evaluation purposes, simulation strongly improves the understanding and knowledge of the Web application's behaviour contributing to a more reliable and meaningful evaluation.

8.2 Conclusions about the Research Questions and Hypothesis

As stated in Chapter 3, this thesis set out to answer several research questions, namely:

1. What aspects of a Web application should be simulated?

2. What simulation model captures those aspects?

3. Is it possible to simulate the model to evaluate its functional requirements?

4. How do simulation and implementation of a Web application design for functional requirements evaluation compare?

The first two research questions have been answered by the definition of the Web-design Simulation Model (Chapter 4) and the Web-design Description Language (Chapter 5). The WSM captures what Web developers consider to be the key design concerns, as well as the key testing and implementation elements. This proved to be an excellent model for simulating Web applications from a functional evaluation perspective, as demonstrated during the planned experiment that followed. The definition of WDL answered the question of what aspects of designs need to be captured for the purpose of simulation. The structural definition of the WSM entities by WDL covers the necessary aspects that should be observable in order to perform a functional evaluation. Furthermore, the behavioural definition of each entity is what makes functional simulation possible. The combination of these two aspects proved to be a powerful method to describe the elements within a Web application, allowing their simulation.

The third research question has been answered by the implementation of a simulation tool (Chapter 6). This tool was developed based on the specifications of the WSM. By using the WSM entities, the Simulator is capable of providing meaningful results that assist in the evaluation of the design functionalities. Furthermore, several auxiliary modules were developed, which complemented the Simulator to enhance the functional evaluation process. Examples of theses are the "Requirements Assessment Module", the "Stimuli Module", and the "Automatic Page Construction Module". Based on the WSM and WDL framework they provide the means to automatically inspect the results, to support construction of stimuli files, and to observe rendered Web pages in the same manner an implementation would. This tool was later used in the experiment and found to be suitable for providing meaningful results, thus asserting the feasibility of the simulation based on the developed framework.

Lastly, the question of how implementation and simulation for functional requirements evaluation compare was answered by utilising both methods in an experiment (Chapter 7). The experimental results allowed a comparison of the advantages and shortcomings of the two methods. This was performed by comparing both treatments' results from both Functional Content and Functional Information perspective. This allowed the assessment of what each treatment provides and their suitability for functional evaluation. The results showed that simulation provides as much content and information on functionality as implementation does. In fact, simulation goes beyond implementation by allowing observation of the state of the Web application on both the client and server side. Furthermore, simulation provides the means to probe internal variables, which cannot be done when using implementation unless code is embedded. Finally, the format of the test results was found to be significantly better when using simulation; displaying the results on a timeline axis allows straightforward observation of present and past Web application states. Furthermore, representation of the results in a multi-layer format allows a rapid and easy identification of the active elements and their interactions. On the other hand, when using implementation, developers can only observe snapshots of the Web application state and not the whole process. These snapshots are taken from a client's perspective and inspection of the server's variables is usually not possible without modifying the code. Furthermore, the key functional elements and their interactions cannot be fully observed, but only their rendered effects. Therefore, although the implementation treatment was able to evaluate the proposed functional requirements, the comparison favoured simulation as a better evaluation method.

The hypothesis that this thesis proposed to prove is:

"It is possible to simulate Web applications based on their designs. Further, the simulation provides similar information with regard to evaluation of functional requirements as would an evaluation based on an actual implementation."

The developed framework, simulation tool, and experiment, have all contributed to test the hypothesis. It has been supported by experimentation that the hypothesis is true. In fact, simulation was found to provide better results than implementation does. However, caution must be exerted not to over-generalise the results to other designs and design models, as has already been noted. External validity has been addressed by the design of the experiment. This led to an experiment using common functional design elements, which extends the scope of simulation feasibility to a broader group of designs.

This work has made an important contribution to the Web development field, in particular to the evaluation of the functional requirements of Web application designs. The need for a testing framework and suitable methods has long been claimed by the Web developers' community. The maturity of the field has reached such a level that, as happened in the software engineering field, it is time to progress toward a more scientific approach to functional evaluation. By offering an automatic procedure, with meaningful and intuitive results, simulation has the potential to decrease the length of a Web project and raise the quality of the end-product.

This has occurred in other engineering fields, such as in the hardware and software development, and this thesis argues that it is now time for this to happen in the Web development field.

8.3 Suggestions for Future Work

The research conducted has proved the feasibility of simulation and its advantages in the functional evaluation of Web applications. The author would like to expand the research in several ways. Some suggestions would include the following:

Providing templates for the main design models would potentially lead to the adoption of the WSM and WDL by the Web developers' community. That would entice developers to adopt the WSM to evaluate functional requirements, which would have the potential to increase the quality of the end-product and result in faster development. Enumeration of the features a design model should possess in order to enable simulation has already been made in this thesis. Theses features allow a selection of the possible and potential candidates for a future research on simulation using other design models.

One feature that has not been addressed in this thesis is the automatic code construction. Hardware Description Languages, such as VHDL and Verilog, are used by engineers not only to support the design and testing phases, but also to assist them during the implementation phase. In the implementation phase, the textual description of the future system is used by synthesis tools to automatically generate the hardware and associated code. Tools such as ispLever® from Lattice Semiconductor Corporation (Lattice Semiconductor Corporation 2006) and Xilinx® ISE from Xilinx Inc. (Xilinx, Inc. 2006), are just some examples of combined HDL simulator and synthesis tools. In the Web development context, this feature would be extremely interesting, since an automatic code construction tool would assist Web developers during the implementation phase. Although not directly addressing aesthetic issues, WDL formally describes the functionalities of Web applications and a substantial part of the applications' code could be generated by such a tool. When considering WebML, however, the WebRatio® tool already possesses automatic code construction features; in this case, an integration of the WDL's simulation capabilities with the WebRatio automatic code construction feature could result in one of the most complete and comprehensive Web application developing tools currently available.

Another interesting line of future work could be the improvement of the WSM Simulator. This could lead to the automatic generation of oracle tests for the verification of specific properties of the system under observation. Based on which system properties the developer would like to be verified – for example liveness (existence), reachability, or safety – the WSM Simulator would automatically generate the necessary tests. Furthermore, a possible quantification of the level of confidence in the performed verification could be done, by estimating the degree of coverage of the observed system's properties that a given set of simulation stimuli can achieve. This level of confidence would be a very interesting indicator to a developer, assuring him/her that the performed simulation would adequetely address all the intended verification concerns.

One interesting area of future work would be to expand and improve the User Interaction Model to accommodate a more automatic evaluation procedure. If it included a more realistic user model, such as an automatic response to the Web application's state, very interesting tests could be performed. This could, for example, give support to Monte Carlo simulation and analysis. Additionally, if the user model is goal-driven such as being on a specific page or achieving a particular state of the Web application, that would allow tests to be automatically performed, with fascinating results to be observed. Furthermore, definition of sets of template goal-driven tests could be done based on the type of Web applications to be tested.

For example, Web-based stores follow a similar navigation and functional pattern from the start a user accesses them till the purchase of a specific product; therefore, a set of goal-driven tests that would include evaluation of the processes of 'logging in', 'product browsing and selection', and 'product purchasing', could be used to test many of this specific type of applications. The definition of these goal-driven tests are not as dependent on a particular Web design as 'traditional' tests are, but rather on the achievement of a specific Web application state, making it possible to use the same test templates for design evaluation of a specific type of Web application.

Another possible line of future work could be to further improve and upgrade the Web-design Simulation Tool and develop it into a product (commercial or open-source) available to the Web developers' community. It would be very interesting to observe whether developers would embrace it, and extremely rewarding if they did.

8.4 Final Conclusions

In summary, this research has developed and defined a framework for the simulation of Web application designs, with the purpose of verifying their functional requirements. The framework, which consists of the Web-design Simulation Model and associated Web-design Description Language, enables Web application design simulation by formally describing the functional elements within the designs. The simulation study followed the life cycle proposed by Balci (1987), contributing to a systematic approach to the formulated problem. Furthermore, validation and verification techniques of the simulation model were used throughout the research, contributing to the claim of the simulation model being able to accurately represent the behaviour of Web applications.

The developed Simulation Model allows evaluation of the functionalities from four key layers. These four layers are populated by the four main WSM entities which completely describe a Web application design from a functional perspective.

Additionally, two other entities – the Window and User – complement the framework and allow multi-session and multi-user simulation capabilities. The adopted research methodology consisted of an experiment to support the hypothesis claims of simulation being able to provide similar information as prototyping does for the evaluation of Web application functional requirements. For this purpose, the Web-design Simulation Tool was developed and its results and those produced by the implementation of a Web application design were compared. The results of this comparison were found to support the claims of the hypothesis, and even to favour simulation as a better suited evaluation method than prototyping.

This work has made a number of significant contributions to the Web application development field, in particular to the testing phase of the Software Development Cycle of such projects. First, it applies concepts traditionally used in the hardware development field, namely the simulation of hardware designs, to tackle the evaluation of the functional requirements of Web application designs. Simulation of Web application designs for functional requirements evaluation purposes had never been attempted before. This work has demonstrated the feasibility of simulation of Web application designs, and the suitability of its results for functional evaluation. Second, a study was done on the requirements a Web application design model must meet to enable meaningful simulation. This allows researchers to assess design models from a simulation-enable perspective, and to include these requirements in future design models definitions if simulation is a desired feature. Finally, a practical contribution was the development of the Web-design Simulation Tool. This tool, which can be readily available to the Web developers community, will surely assist Web developers in the evaluation of the functional requirements of WebML designs.

The benefits of using the Web-design Simulation Model, the Web-design Description Language, and the Web-design Simulation Tool include:

- Evaluation of the functional requirements of Web application designs without the need for prototyping;
- Earlier conceptual error detection, on a functional level, of Web application designs;
- Contribution to the shortening of the Software Development Cycle of Web applications, in particular of the testing phase;
- Enhancement of Web application testing by automatically evaluating pre-defined test cases;
- Enhancement of Web application testing by scenario-based simulation;
- Enhancement of Web application testing by supporting multi-session and multi-user simulation;
- Contribution to a high quality of the end product.

Finally, the framework developed in this research promises a more automatic and in-depth evaluation of complex Web application designs based on simulation techniques. This will allow Web developers to tackle the development of Web applications with suitable evaluation methods and tools, thus potentially contributing to enable Web developers to experiment and explore new and more complex Web application designs.

Appendix A

The WDL Syntax

```
<entity_declaration> ::=
        entity <identifier> is
                /* Model */
                <entity_model_attributes>
                /* Runtime */
                <entity_runtime_attributes>
        end;

<identifier> ::= <Name>
<entity_model_attributes> ::= <entity_attribute>
<entity_runtime_attributes> ::= <entity_attribute>
<entity_attribute> ::= <attribute_clause>
<attribute_clause> ::= <attribute_name> : <attribute_type>;
<attribute_name> ::= <Name>
<attribute_type> ::= 'String' | 'ID'
        | 'Integer' | 'Float' | 'Boolean'
        | <array_type> | 'URL'
<array_type> ::= "Array of" <attribute_type>
<Name> ::= <Letter> <Letter> | <Digit>
<Letter> ::= ['A'-'Z']|['a'-'z']
<Digit> ::= ['0'-'9']
```

Figure A.1: The **entity** Declaration Definition.

```
<architecture_body> ::=
            architecture <identifier> of <entity_name> is
            begin
                  {<function_name> : process is
                  begin
                        <architecture_statement_part>
                  end process <function_name>;}
            end [ architecture ] [ <identifier> ];

<architecture_statement_part> ::=  <programmatic_statement>
```

Figure A.2: The **architecture** Declaration Definition.

```
entity Page is
/* Model */
       pageID: ID;
       pageStructure: ( 'PAGE' | 'FORM' | 'FRAMESET' );
       pageBuild: ( 'HTML' | 'DYNAMIC' | 'ASP' );
       pageComponentsID: Array of ID;
       pageContext: String;

/* Runtime */
       pageOID: ID;
       state: ('ACTIVE' | 'INACTIVE');
       windowOID: String;
       userOID: String;
       pageComponentsOID: Array of ID;
end entity Page;
```

Figure A.3: The Page Entity Structure

```
architecture behaviour of Page is
begin
    displayPage: process is
    begin
        state = 'ACTIVE';
        readObject()
        ...
    end process displayPage;

    deactivatePage: process is
    begin
        state = 'INACTIVE';
        ...
    end process deactivatePage;

    readObject: process is
    begin
        for each componentOID in pageComponentsOID
            readObject(componentOID);
        end for each;
        ...
    end process readObject;
end architecture behaviour;
```

Figure A.4: The Page Architecture

```
entity Link is
/* Model */
        linkID: ID;
        linkType: ('LINK' | 'SUBMIT' | 'TRANSPORT' );
        linkSourceID: String;
        linkTargetID: String;
        linkParametersNames: String;
        linkTargetWindowOptions: ('SELF' | 'BLANK' | 'PARENT' | 'TOP');

/* Runtime */
        linkOID: ID;
        state: ('ACTIVE' | 'INACTIVE');
        linkSourceOID: String;
        linkTargetOID: String;
        linkParametersValues: String;
        windowOID: String;
        userOID: String;
end entity Link;
```

Figure A.5: The Link Entity Structure

```
architecture behaviour of Link is
begin
    actionLink: process is
    begin
        state = 'ACTIVE';
        deactivatePage( linkSourceOID );
        displayPage( linkTargetOID );
        ...
        deactivateLink()
    end process actionLink;

    actionButton: process is
    begin
        state = 'ACTIVE';
        updateParameters();
        deactivatePage( linkSourceOID );
        executeScript( linkTargetOID );
        ...
        deactivateLink()
    end process actionButton;

    deactivateLink: process is
    begin
        state = 'INACTIVE';
    end process deactivateLink;
end architecture behaviour;
```

Figure A.6: The Link Architecture

```
entity Script is
/* Model */
        scriptID: ID;
        scriptSide: ('CLIENT' | 'SERVER');
        scriptVariablesNames: Array of ID;
        scriptVariablesType: Array of ('IN' | 'OUT' | 'INOUT');
        scriptComponentsID: Array of ID;

/* Runtime */
        scriptOID: ID;
        state: ('ACTIVE' | 'INACTIVE');
        scriptVariablesValues: Array of String;
        scriptComponentsOID: Array of ID;
        callerOID: String;
        windowOID: String;
        userOID: String;
end entity Script;
```

Figure A.7: The Script Entity Structure

```
architecture behaviour of Script is
begin
    executeScript: process is
    begin
        state = 'ACTIVE';
        updateInputVariables();
        runScriptbehaviour();
        ...
        updateOutputVariables();
        stopScript();
    end process executeScript;

    runScriptbehaviour: process is
    begin
        <Template-based or Custom-made Java™script>
        ...
    end process runScriptbehaviour;

    stopScript: process is
    begin
        state == 'INACTIVE';
        ...
    end process runScriptbehaviour;

    readObject: process is
    begin
        for each componentID in scriptComponentsID
            scriptComponentObject = new Object(scriptComponentsID);
            readObject(scriptComponentObject);
        end for each;
        ...
    end process readObject;
end architecture behaviour;
```

Figure A.8: The Script Architecture

```
entity Data is
/* Model */
        dataID: ID;
        dataType: ('DATABASE' | 'FILE' | 'COOKIE' | 'VARIABLE' | 'FORMVAR' );
        dataSubType: (ACCESS | SQL ),
                     ('TEXT' | 'BIN'),
                     ('FIELD' | 'COMBOBOX' | 'LIST' | 'RADIOBUTTON' | 'CHECKBOX');
        dataSource: String;
        dataMultiplicity: ('SHARED' | 'MULTIPLE');
        dataPersistence: ('APPLICATION' | 'SESSION');

/* Runtime */
        dataOID: ID;
        state: ('ACTIVE' | 'INACTIVE');
        dataValue: String;
        windowOID: String;
        userOID: String;
end entity Data;
```

Figure A.9: The Data Entity Structure

```
architecture behaviour of Data is
begin
    setData( value, SQLStatement ): process is
    begin
        state = 'ACTIVE';
        if dataType== 'DATABASE'
                openDatabaseConnection();
                executeStatement( SQLStatement );
        else
                dataValue = value;
        end if
        ...
        deactivateData();
    end process setData;

    getData( SQLStatement ): process is
    begin
        state == 'ACTIVE';
        if dataType == 'DATABASE'
                openDatabaseConnection();
                return executeQuery( SQLStatement );
        else
                return dataValue;
        end if
        ...
        deactivateData();
    end process getData;

    deactivateData: process is
    begin
        state = 'INACTIVE';
    end process deactivateData;
end architecture behaviour;
```

Figure A.10: The Data Architecture

```
entity Window is
/* Model */

/* Runtime */
        windowOID: ID;
        state: ('ACTIVE' | 'INACTIVE');
        userOID: String;
        windowComponentsOID: Array of ID;
        URL: String;
end entity Window;
```

Figure A.11: The Window Entity Structure

```
architecture behaviour of Window is
begin
        openWindow: process is
        begin
                state = 'ACTIVE';
                ...
        end process openWindow;

        closeWindow: process is
        begin
                state = 'INACTIVE';
                ...
        end process closeWindow;
end architecture behaviour;
```

Figure A.12: The Window Architecture

```
entity User is
/* Model */

/* Runtime */
        userOID: ID;
        state: ('ACTIVE' | 'INACTIVE');
        userComponentsOID: Array of ID;
        historyEvents: Array of String;
        historySlots: Array of Integer;
end entity User;
```

Figure A.13: The User Entity Structure

```
architecture behaviour of User is
begin
        createUser: process is
        begin
                state = 'ACTIVE';
                ...
        end process createUser;

        discardUser: process is
        begin
                state = 'INACTIVE';
                ...
        end process discardUser;

        userInteraction: process is
        begin
                <Any User Interaction Stimuli defined in Appendix B>
                ...
        end process userInteraction;
end architecture behaviour;
```

Figure A.14: The User Architecture

Appendix B

Syntax of WSM Stimuli

B.1 The WSM Exogenous Stimuli

Browser Events:

openWindowEvent ::= openWindow <windowOID> <URL> <userOID>

closeWindowEvent ::= closeWindow <windowOID>

URL := <pageID> | <scriptID>

Interface Object Events:

actionLinkEvent ::= actionLink <linkOID>

actionLinkEvent ::= actionLink <linkID> <pageID> <windowOID>

actionButtonEvent ::= actionButton <linkOID>

actionButtonEvent ::= actionButton <linkID> <pageID> <windowOID>

Data Input Events:

setDataEvent ::= setData <dataOID> <New_Value>

setDataEvent ::= setData <dataID> <pageID> <windowOID> <New_Value>

getDataEvent ::= getData <dataOID>

getDataEvent ::= getData <dataID> <pageID> <windowOID>

Table B.1: User Interaction Stimuli

User Interaction Stimuli	
Stimulus	**Description**
openWindow	Opens a window for a User and loads an URL on it
closeWindow	Closes a window
actionLink	Emulates a user traversing a hyperlink
actionButton	Emulates a user submitting a form or acting on an action button
setData	Sets a Data object with a new value
getData	Gets a Data object value
createUser	Creates a new User object
discardUser	Discards a User object

B.2 The WSM Endogenous Stimuli

Endogenous Events:

createUserEvent ::= createUser <userOID>

discardUserEvent ::= discardUser <userOID>

readObjectEvent ::= readObject <ID>

displayPageEvent ::= displayPage <pageID>

buildPageEvent ::= buildPage <pageOID> <objectID>

getScriptVarEvent ::= getScriptVar <variableID> <scriptOID>

setScriptVarEvent ::= setScriptVar <variableID> <scriptOID> <newValue>

Appendix C

The Simulator

Table C.1: Simulator Control Stimuli

Simulator Control Stimuli	
Stimulus	**Description**
wdl <WDLModelFile>	Loads a new WDL Model file into the Simulator
setInterStimuliDelay <numberSeconds>	Sets the amount of seconds between two consecutive stimuli triggering
tree [model | runtime]	Sets the browser window in Model or Run-Time display mode
layer [Presentation | Navigation | Functional | Content]	Sets layer displayed on the Main window
loadStimuli <StimuliFile>	Loads a new stimuli file for simulation
startStimuli	(Re)starts the simulation of the stimuli file
suspendStimuli	Temporarily suspends all stimuli processing
resumeStimuli	Resumes a suspended simulation
stopStimuli	Stops all stimuli processing
loadRequirements <RequirementsFile>	Loads a new requirement file into the Simulator

timeSlot <timeSlot>	Sets the Main window start slot
resetSimTime	Resets the simulation time
clearDisplay	Clears the content of the Main window
toggleStepByStep	In mode "StepByStep", the Simulator stops after receiving and processing an event. This commands toggles this mode. (Default value: OFF)
model	Displays on the Message window all the loaded WDL Model variables
modelStructuralAnalysis	Performs an analysis of the structure of the loaded WDL Model
modelVerification	Performs a verification of the loaded WDL Model
quit	Exits the Web-design Simulation Tool

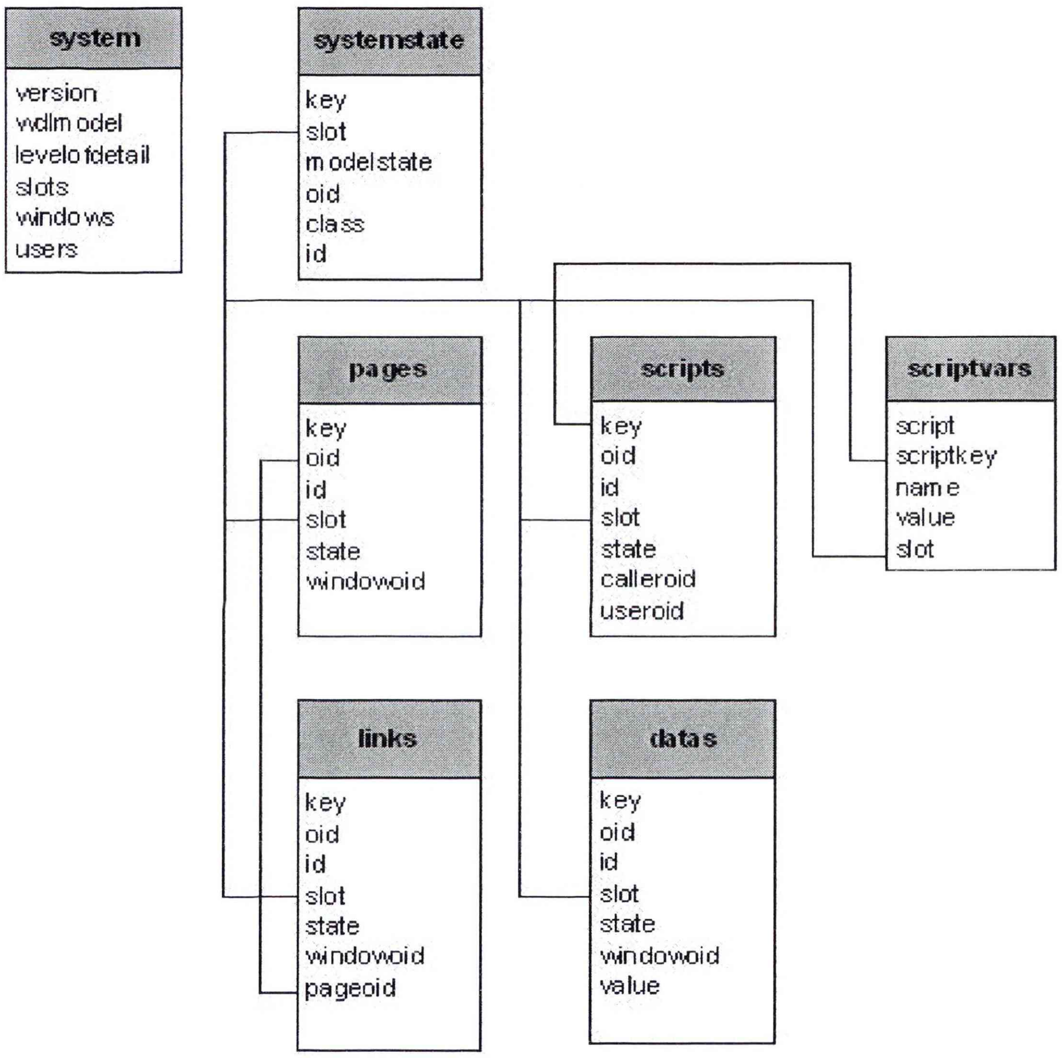
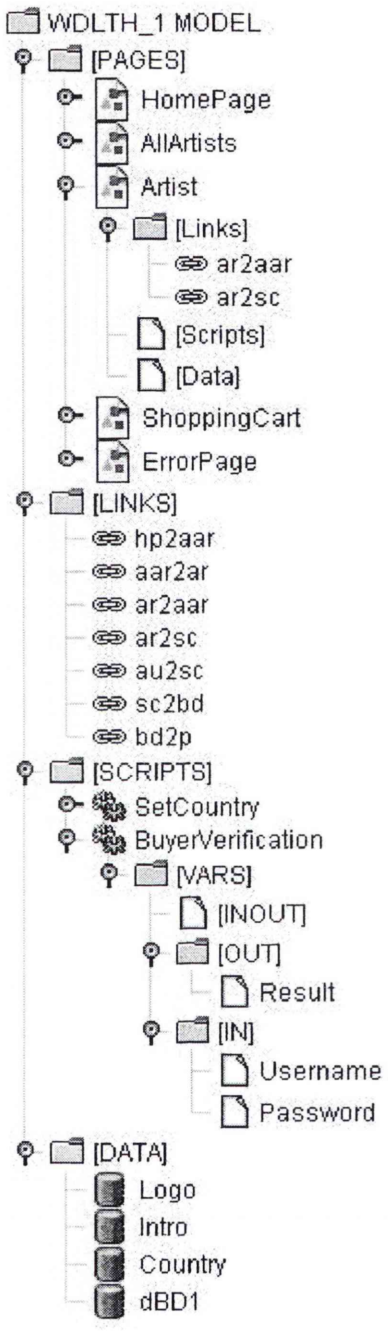


Figure C.1: The Simulator's Database



```
WDLTH_1 MODEL
  [PAGES]
    HomePage
    AllArtists
    Artist
      [Links]
        ar2aar
        ar2sc
      [Scripts]
      [Data]
    ShoppingCart
    ErrorPage
  [LINKS]
    hp2aar
    aar2ar
    ar2aar
    ar2sc
    au2sc
    sc2bd
    bd2p
  [SCRIPTS]
    SetCountry
    BuyerVerification
      [VARS]
        [INOUT]
        [OUT]
          Result
        [IN]
          Username
          Password
  [DATA]
    Logo
    Intro
    Country
    dBD1
```

Figure C.2: An Example of the Browser Window in Model Mode

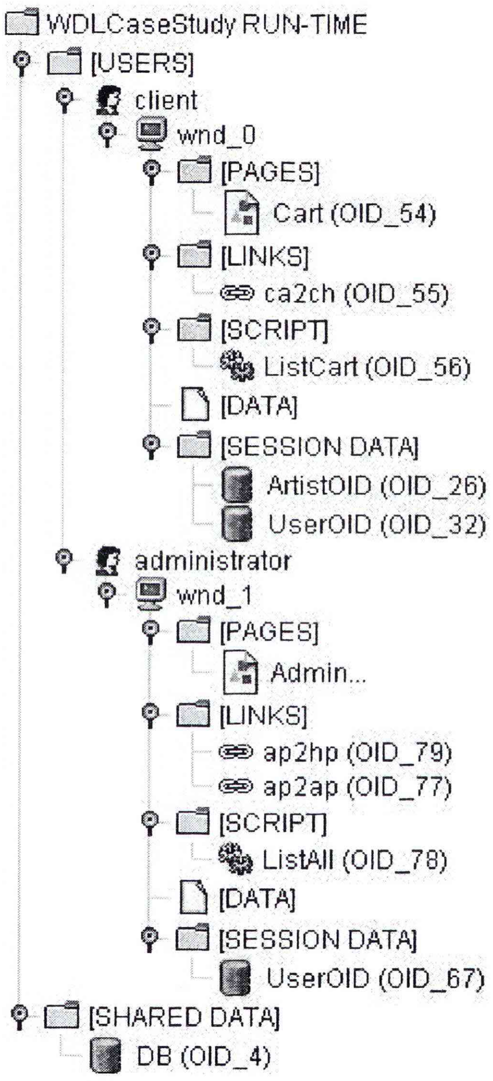
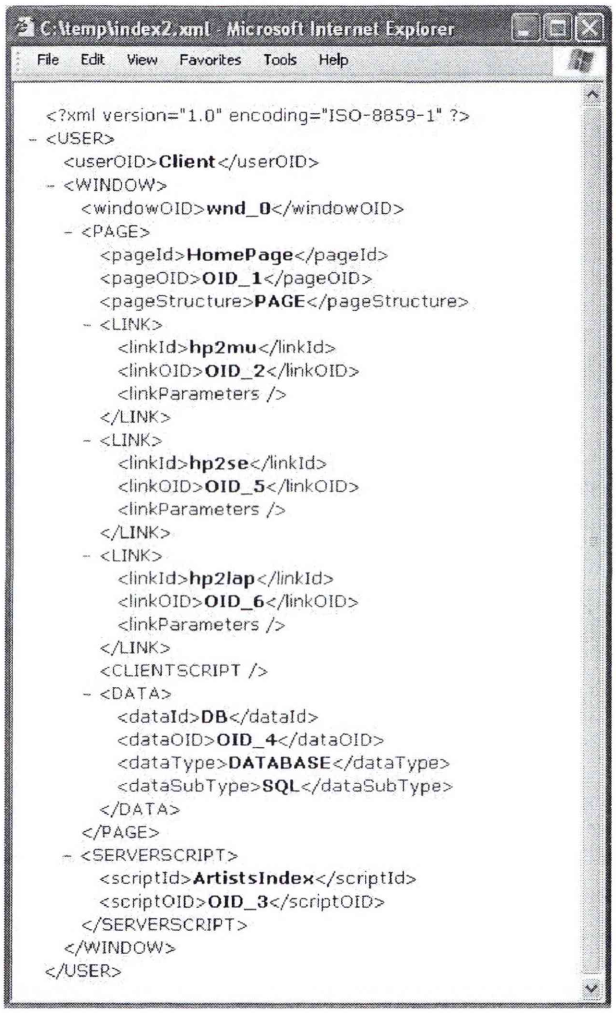
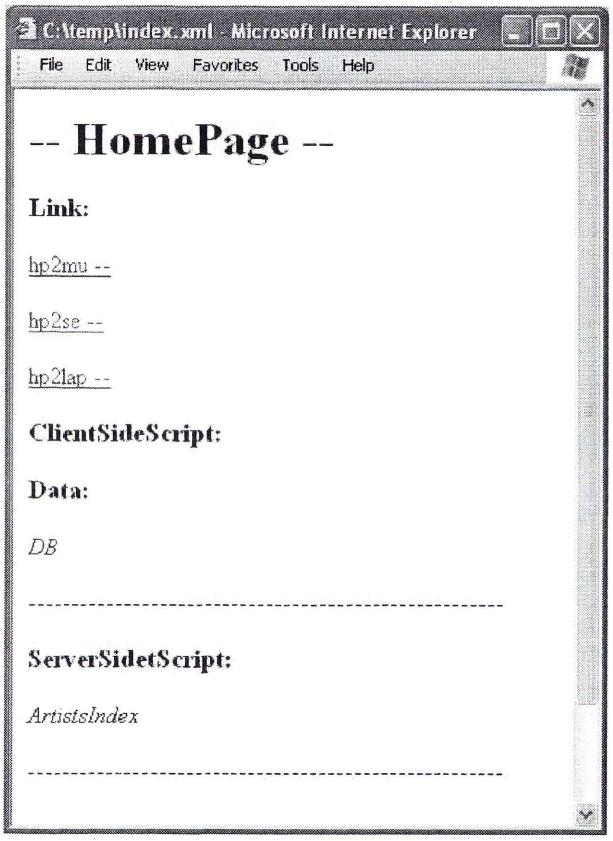


Figure C.3: An Example of the Browser Window in Runtime Mode


C:\temp\index2.xml - Microsoft Internet Explorer
File Edit View Favorites Tools Help

```
  <?xml version="1.0" encoding="ISO-8859-1" ?>
- <USER>
    <userOID>Client</userOID>
  - <WINDOW>
      <windowOID>wnd_0</windowOID>
    - <PAGE>
        <pageId>HomePage</pageId>
        <pageOID>OID_1</pageOID>
        <pageStructure>PAGE</pageStructure>
      - <LINK>
          <linkId>hp2mu</linkId>
          <linkOID>OID_2</linkOID>
          <linkParameters />
        </LINK>
      - <LINK>
          <linkId>hp2se</linkId>
          <linkOID>OID_5</linkOID>
          <linkParameters />
        </LINK>
      - <LINK>
          <linkId>hp2lap</linkId>
          <linkOID>OID_6</linkOID>
          <linkParameters />
        </LINK>
        <CLIENTSCRIPT />
      - <DATA>
          <dataId>DB</dataId>
          <dataOID>OID_4</dataOID>
          <dataType>DATABASE</dataType>
          <dataSubType>SQL</dataSubType>
        </DATA>
      </PAGE>
    - <SERVERSCRIPT>
        <scriptId>ArtistsIndex</scriptId>
        <scriptOID>OID_3</scriptOID>
      </SERVERSCRIPT>
    </WINDOW>
  </USER>
```

Figure C.4: An Example of an XML File of a Page Object


C:\temp\index.xml - Microsoft Internet Explorer
File Edit View Favorites Tools Help

-- HomePage --

Link:

hp2mu --

hp2se --

hp2lap --

ClientSideScript:

Data:

DB

---

ServerSidetScript:

ArtistsIndex

---

Figure C.5: An Example of the Rendition of an XML File, Done by the Automatic Page Construction Module

Appendix D

The Experiment Design

The WebML design used in the experiment is presented in this Appendix in Figures D.1, D.2 and D.3. The corresponding WDL mapping is displayed in Figure D.4.

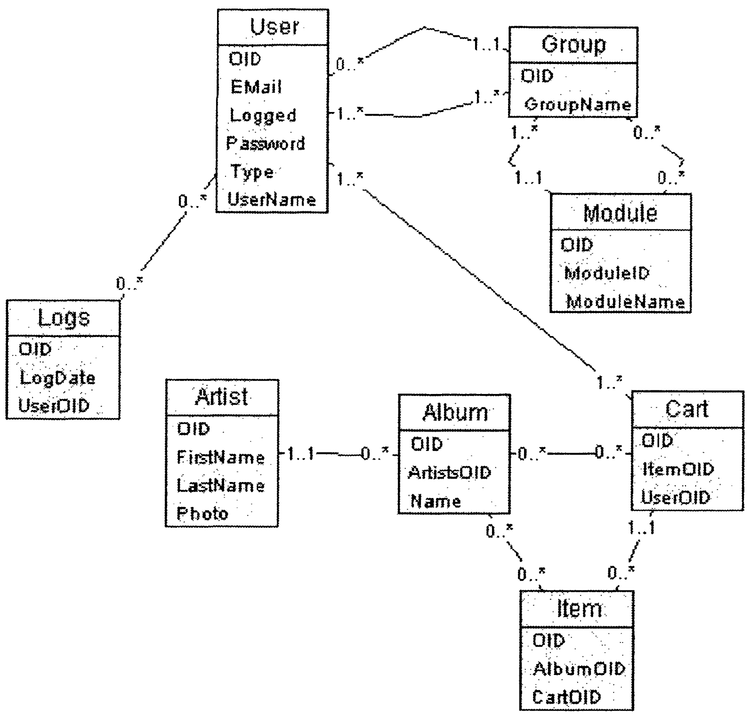


Figure D.1: The Design Model Database Structure

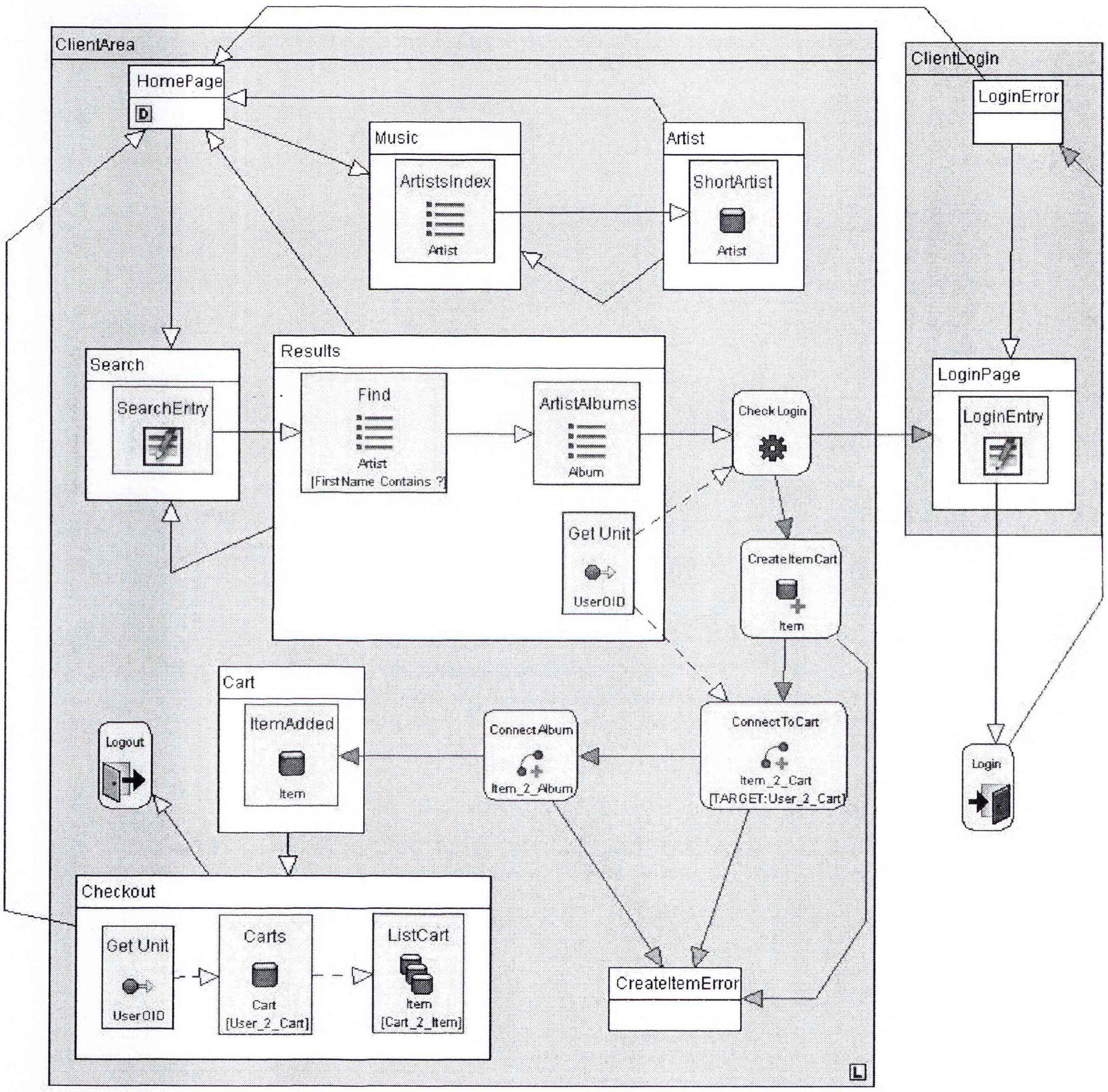


Figure D.2: The WebML Design – the "Client" Site View

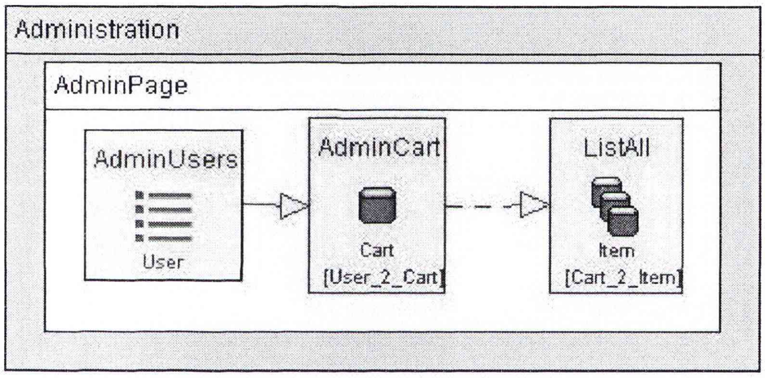


Figure D.3: The WebML Design – the "Administration" Site View

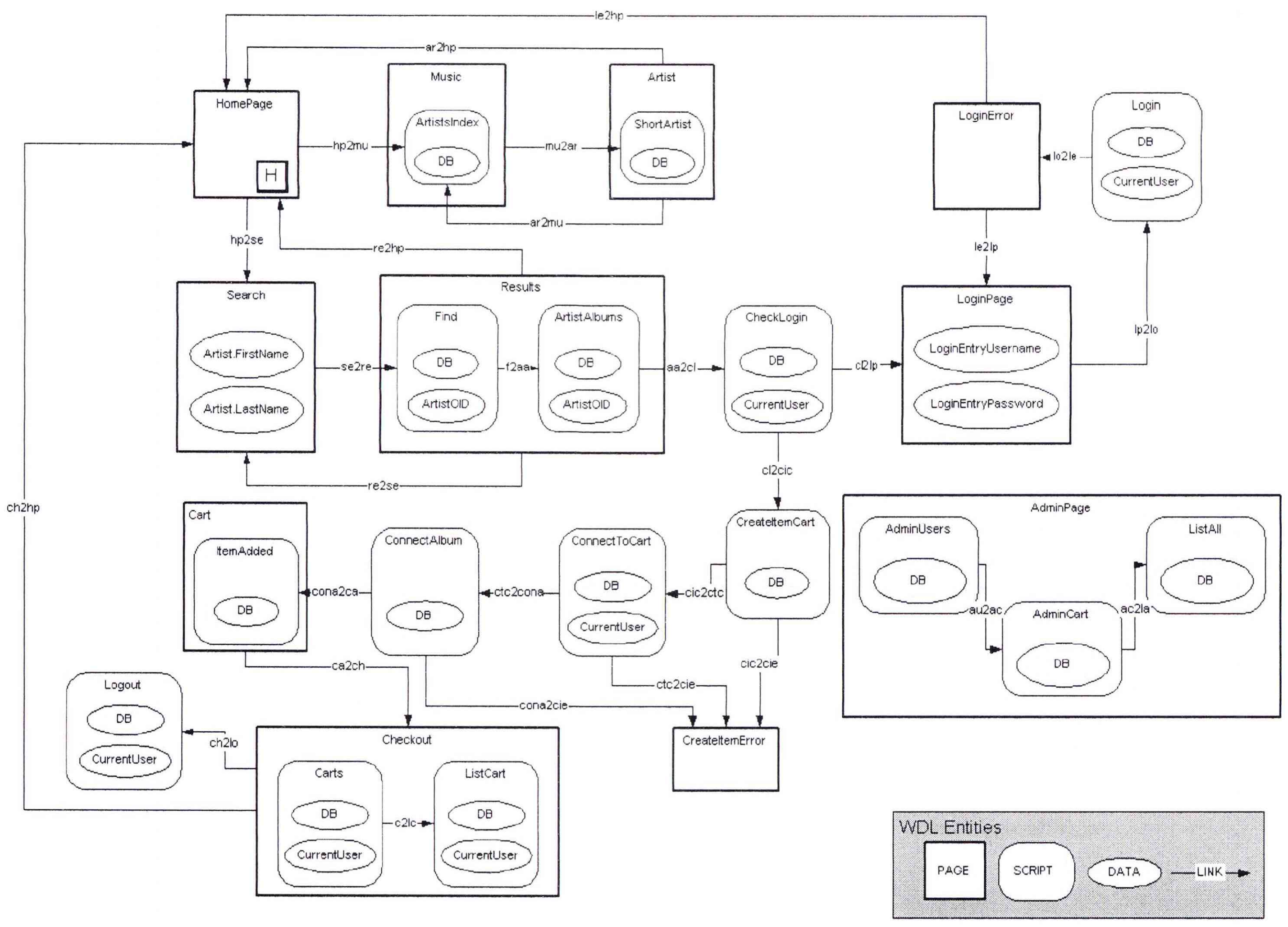


Figure D.4: The WDL Mapping of the WebML Design

```
entity Page is
/* Model */
        pageID: ‘HomePage’
        pageContext: ‘Client.ClientArea.INDEX’
        pageStructure: ‘PAGE’
        pageBuild: ‘HTML’
        pageComponentsID: {‘null’}
end entity Page;

entity Page is
/* Model */
        pageID: ‘Music’
        pageContext: ‘Client.ClientArea’
        pageStructure: ‘PAGE’
        pageBuild: ‘ASP’
        pageComponentsID: {‘ArtistsIndex’}
end entity Page;

entity Page is
/* Model */
        pageID: ‘Artist’
        pageContext: ‘Client.ClientArea’
        pageStructure: ‘PAGE’
        pageBuild: ‘ASP’
        pageComponentsID: {‘ShortArtist’}
end entity Page;

entity Page is
/* Model */
        pageID: ‘Search’
        pageContext: ‘Client.ClientArea’
        pageStructure: ‘FORM’
        pageBuild: ‘HTML’
        pageComponentsID: {‘Artist.FirstName’,
                           ‘Artist.LastName’}
end entity Page;

entity Page is
/* Model */
        pageID: ‘Results’
        pageContext: ‘Client.ClientArea’
        pageStructure: ‘PAGE’
        pageBuild: ‘ASP’
        pageComponentsID: {‘Find’}
        pageComponentsID: {‘ArtistAlbums’}
end entity Page;

entity Page is
/* Model */
        pageID: ‘LoginPage’
        pageContext: ‘Client.Login’
        pageStructure: ‘FORM’
        pageBuild: ‘HTML’
        pageComponentsID: {‘LoginEntryUsername’,
                           ’LoginEntryPassword’}
end entity Page;

entity Page is
/* Model */
        pageID: ‘LoginError’
        pageContext: ‘Client.Login’
        pageStructure: ‘PAGE’
        pageBuild: ‘HTML’
        pageComponentsID: {‘null’}
end entity Page;

entity Page is
/* Model */
        pageID: ‘CreateItemError’
        pageContext: ‘Client.ClientArea’
        pageStructure: ‘PAGE’
        pageBuild: ‘HTML’
        pageComponentsID: {‘null’}
end entity Page;

entity Page is
/* Model */
        pageID: ‘Cart’
        pageContext: ‘Client.ClientArea’
        pageStructure: ‘PAGE’
        pageBuild: ‘ASP’
        pageComponentsID: {‘ItemAdded’}
end entity Page;

entity Page is
/* Model */
        pageID: ‘Checkout’
        pageContext: ‘Client.ClientArea’
        pageStructure: ‘PAGE’
        pageBuild: ‘ASP’
        pageComponentsID: {‘Carts’}
        pageComponentsID: {‘ListCart’}
end entity Page;
```

```
entity Page is
/* Model */
        pageID: 'AdminPage'
        pageContext: 'Administrator.
                      Administration'
        pageStructure: 'PAGE'
        pageBuild: 'ASP'
        pageComponentsID: {'AdminUsers'}
        pageComponentsID: {'AdminCart'}
        pageComponentsID: {'ListAll'}
end entity Page;

entity Link is
/* Model */
        linkID: 'hp2mu'
        linkType: 'LINK'
        linkSourceID: 'HomePage'
        linkTargetID: 'ArtistsIndex'
        linkTargetWindowOptions: 'SELF'
        linkParametersNames: {'null'}
end entity Link;

entity Link is
/* Model */
        linkID: 'mu2ar'
        linkType: 'LINK'
        linkSourceID: 'Music'
        linkTargetID: 'ShortArtist'
        linkTargetWindowOptions: 'SELF'
        linkParametersNames: {'artist.key'}
end entity Link;

entity Link is
/* Model */
        linkID: 'ar2mu'
        linkType: 'LINK'
        linkSourceID: 'Artist'
        linkTargetID: 'ArtistIndex'
        linkTargetWindowOptions: 'SELF'
        linkParametersNames: {'null'}
end entity Link;

entity Link is
/* Model */
        linkID: 'ar2hp'
        linkType: 'LINK'
        linkSourceID: 'Artist'
        linkTargetID: 'HomePage'
        linkTargetWindowOptions: 'SELF'
        linkParametersNames: {'null'}
end entity Link;

entity Link is
/* Model */
        linkID: 'hp2se'
        linkType: 'LINK'
        linkSourceID: 'HomePage'
        linkTargetID: 'Search'
        linkTargetWindowOptions: 'SELF'
        linkParametersNames: {'null'}
end entity Link;

entity Link is
/* Model */
        linkID: 'se2re'
        linkType: 'SUBMIT'
        linkSourceID: 'Search'
        linkTargetID: 'Find'
        linkTargetWindowOptions: 'SELF'
        linkParametersNames: {'Artist.FirstName',
                              'Artist.LastName'}
end entity Link;

entity Link is
/* Model */
        linkID: 'f2aa'
        linkType: 'TRANSPORT'
        linkSourceID: 'Find'
        linkTargetID: 'ArtistAlbums'
        linkTargetWindowOptions: 'SELF'
        linkParametersNames: {'Artist.OID'}
end entity Link;
```

```
entity Link is
/* Model */
        linkID: 'aa2cl'
        linkType: 'LINK'
        linkSourceID: 'Results'
        linkTargetID: 'CheckLogin'
        linkTargetWindowOptions: 'SELF'
        linkParametersNames: {'Album.OID'}
end entity Link;

entity Link is
/* Model */
        linkID: 're2hp'
        linkType: 'LINK'
        linkSourceID: 'Results'
        linkTargetID: 'HomePage'
        linkTargetWindowOptions: 'SELF'
        linkParametersNames: {'null'}
end entity Link;

entity Link is
/* Model */
        linkID: 'lp2lo'
        linkType: 'SUBMIT'
        linkSourceID: 'LoginPage'
        linkTargetID: 'Login'
        linkTargetWindowOptions: 'SELF'
        linkParametersNames:
            {'LoginEntryUsername'
            'LoginEntryPassword'}
end entity Link;

entity Link is
/* Model */
        linkID: 'le2lp'
        linkType: 'LINK'
        linkSourceID: 'LoginError'
        linkTargetID: 'LoginPage'
        linkTargetWindowOptions: 'SELF'
        linkParametersNames: {'null'}
end entity Link;

entity Link is
/* Model */
        linkID: 'ctc2cie'
        linkType: 'LINK'
        linkSourceID: 'ConnectToCart'
        linkTargetID: 'CreateItemError'
        linkTargetWindowOptions: 'SELF'
        linkParametersNames: {'null'}
end entity Link;

entity Link is
/* Model */
        linkID: 'cona2cie'
        linkType: 'LINK'
        linkSourceID: 'ConnectAlbum'
        linkTargetID: 'CreateItemError'
        linkTargetWindowOptions: 'SELF'
        linkParametersNames: {'null'}
end entity Link;

entity Link is
/* Model */
        linkID: 'cic2cie'
        linkType: 'LINK'
        linkSourceID: 'CreateItemCart'
        linkTargetID: 'CreateItemError'
        linkTargetWindowOptions: 'SELF'
        linkParametersNames: {'null'}
end entity Link;

entity Link is
/* Model */
        linkID: 'ch2lo'
        linkType: 'LINK'
        linkSourceID: 'Checkout'
        linkTargetID: 'Logout'
        linkTargetWindowOptions: 'SELF'
        linkParametersNames: {'null'}
end entity Link;
```

```
entity Link is
/* Model */
        linkID: ‘ca2ch’
        linkType: ‘LINK’
        linkSourceID: ‘Cart’
        linkTargetID: ‘Checkout’
        linkTargetWindowOptions: ‘SELF’
        linkParametersNames: {‘null’}
end entity Link;

entity Link is
/* Model */
        linkID: ‘ch2hp’
        linkType: ‘LINK’
        linkSourceID: ‘Checkout’
        linkTargetID: ‘HomePage’
        linkTargetWindowOptions: ‘SELF’
        linkParametersNames: {‘null’}
end entity Link;

entity Link is
/* Model */
        linkID: ‘cl2cic’
        linkType: ‘TRANSPORT’
        linkSourceID: ‘CheckLogin’
        linkTargetID: ‘CreateItemCart’
        linkTargetWindowOptions: ‘SELF’
        linkParametersNames: {‘Album.OID’}
end entity Link;

entity Link is
/* Model */
        linkID: ‘cic2ctc’
        linkType: ‘TRANSPORT’
        linkSourceID: ‘CreateItemCart’
        linkTargetID: ‘ConnectToCart’
        linkTargetWindowOptions: ‘SELF’
        linkParametersNames: {‘null’}
end entity Link;

entity Link is
/* Model */
        linkID: ‘le2hp’
        linkType: ‘LINK’
        linkSourceID: ‘LoginError’
        linkTargetID: ‘HomePage’
        linkTargetWindowOptions: ‘SELF’
        linkParametersNames: {‘null’}
end entity Link;

entity Link is
/* Model */
        linkID: ‘lo2le’
        linkType: ‘LINK’
        linkSourceID: ‘Login’
        linkTargetID: ‘LoginError’
        linkTargetWindowOptions: ‘SELF’
        linkParametersNames: {‘null’}
end entity Link;

entity Link is
/* Model */
        linkID: ‘cl2lp’
        linkType: ‘LINK’
        linkSourceID: ‘CheckLogin’
        linkTargetID: ‘LoginPage’
        linkTargetWindowOptions: ‘SELF’
        linkParametersNames: {‘null’}
end entity Link;

entity Link is
/* Model */
        linkID: ‘au2ac’
        linkType: ‘TRANSPORT’
        linkSourceID: ‘AdminUsers’
        linkTargetID: ‘AdminCart’
        linkTargetWindowOptions: ‘SELF’
        linkParametersNames: {‘User.OID’}
end entity Link;
```

```
entity Link is
/* Model */
        linkID: ‘cona2ca’
        linkType: ‘TRANSPORT’
        linkSourceID: ‘ConnectAlbum’
        linkTargetID: ‘Cart’
        linkTargetWindowOptions: ‘SELF’
        linkParametersNames: {‘null’}
end entity Link;

entity Link is
/* Model */
        linkID: ‘ac2la’
        linkType: ‘TRANSPORT’
        linkSourceID: ‘AdminCart’
        linkTargetID: ‘ListAll’
        linkTargetWindowOptions: ‘SELF’
        linkParametersNames: {‘Cart.OID’}
end entity Link;

entity Link is
/* Model */
        linkID: ‘ctc2cona’
        linkType: ‘TRANSPORT’
        linkSourceID: ‘ConnectToCart’
        linkTargetID: ‘ConnectAlbum’
        linkTargetWindowOptions: ‘SELF’
        linkParametersNames: {‘null’}
end entity Link;
```

```
entity Script is
/* Model */
        scriptID: 'ArtistsIndex'
        scriptComponentsID: {'DB'}
        scriptSide: 'SERVER'
end entity Script;

entity Script is
/* Model */
        scriptID: 'ShortArtist'
        scriptVariablesNames: {'Artist.key'}
        scriptVariablesType: {'IN'}
        scriptComponentsID: {'DB'}
        scriptSide: 'SERVER'
end entity Script;

entity Script is
/* Model */
        scriptID: 'Find'
        scriptVariablesNames: {'Artist.FirstName','Artist.LastName'}
        scriptVariablesType: {'IN','IN'}
        scriptComponentsID: {'ArtistOID','DB'}
        scriptSide: 'SERVER'
end entity Script;

entity Script is
/* Model */
        scriptID: 'Logout'
        scriptComponentsID: {'CurrentUser','DB'}
        scriptSide: 'SERVER'
end entity Script;

entity Script is
/* Model */
        scriptID: 'ListAll'
        scriptComponentsID: {'DB'}
        scriptSide: 'SERVER'
end entity Script;

entity Script is
/* Model */
        scriptID: 'AdminUsers'
        scriptComponentsID: {'DB'}
        scriptSide: 'SERVER'
end entity Script;
```

```
entity Script is
/* Model */
        scriptID: ‘Login’
        scriptVariablesNames: {‘LoginEntryUsername’,‘LoginEntryPassword’}
        scriptVariablesType: {‘IN’,‘IN’}
        scriptComponentsID: {‘CurrentUser’,‘DB’}
        scriptSide: ‘SERVER’
end entity Script;

entity Script is
/* Model */
        scriptID: ‘CreateItemCart’
        scriptComponentsID: {‘CurrentUser’,‘DB’}
        scriptSide: ‘SERVER’
end entity Script;

entity Script is
/* Model */
        scriptID: ‘ItemAdded’
        scriptComponentsID: {‘DB’}
        scriptSide: ‘SERVER’
end entity Script;

entity Script is
/* Model */
        scriptID: ‘ListCart’
        scriptComponentsID: {‘CurrentUser’,‘DB’}
        scriptSide: ‘SERVER’
end entity Script;

entity Script is
/* Model */
        scriptID: ‘ConnectToCart’
        scriptComponentsID: {‘DB’}
        scriptSide: ‘SERVER’
end entity Script;

entity Script is
/* Model */
        scriptID: ‘ConnectAlbum’
        scriptComponentsID: {‘DB’}
        scriptSide: ‘SERVER’
end entity Script;
```

```
entity Script is
/* Model */
        scriptID: 'AdminCart'
        scriptComponentsID: {'DB'}
        scriptSide: 'SERVER'
end entity Script;

entity Script is
/* Model */
        scriptID: 'Carts'
        scriptComponentsID: {'CurrentUser','DB'}
        scriptSide: 'SERVER'
end entity Script;

entity Script is
/* Model */
        scriptID: 'CheckLogin'
        scriptVariablesNames: {'Success'}
        scriptVariablesType: {'OUT'}
        scriptComponentsID: {'CurrentUser','DB'}
        scriptSide: 'SERVER'
end entity Script;

architecture CheckLogin_behaviour of Script is
begin

        runScriptbehaviour: process is
        begin
                String user = getData('CurrentUser');
                if(user.equals('UNDEFINED'))
                        {
                                setScriptVar('Success','false');
                                displayPage('LoginPage');
                        }
                else
                        {
                                setScriptVar('Success','true');
                                executeScript('CreateItemCart');
                        }
        end process runScriptbehaviour;
end architecture CheckLogin_behaviour;
```

```
entity Data is
/* Model */
        dataID: 'ArtistOID'
        dataType: 'COOKIE'
        dataSubType: 'TEXT'
        dataMultiplicity: 'MULTIPLE'
        dataPersistence: 'APPLICATION'
end entity Data;

entity Data is
/* Model */
        dataID: 'CurrentUser'
        dataType: 'COOKIE'
        dataSubType: 'TEXT'
        dataMultiplicity: 'MULTIPLE'
        dataPersistence: 'APPLICATION'
end entity Data;

entity Data is
/* Model */
        dataID: 'Artist.FirstName'
        dataType: 'FORMVAR'
        dataSubType: 'FIELD'
        dataMultiplicity: 'MULTIPLE'
        dataPersistence: 'SESSION'
end entity Data;

entity Data is
/* Model */
        dataID: 'Artist.LastName'
        dataType: 'FORMVAR'
        dataSubType: 'FIELD'
        dataMultiplicity: 'MULTIPLE'
        dataPersistence: 'SESSION'
end entity Data;

entity Data is
/* Model */
        dataID: 'LoginEntryUsername'
        dataType: 'FORMVAR'
        dataSubType: 'FIELD'
        dataMultiplicity: 'MULTIPLE'
        dataPersistence: 'SESSION'
end entity Data;

entity Data is
/* Model */
        dataID: 'LoginEntryPassword'
        dataType: 'FORMVAR'
        dataSubType: 'FIELD'
        dataMultiplicity: 'MULTIPLE'
        dataPersistence: 'SESSION'
end entity Data;

entity Data is
/* Model */
        dataID: 'DB'
        dataType: 'DATABASE'
        dataSubType: 'SQL'
        dataMultiplicity: 'SHARED'
        dataPersistence: 'APPLICATION'
        dataSource: 'webmlatabase'
end entity Data;
```

Appendix E

The Experiment Simulation Results

The results of the simulation performed on the Web Application design are shown in the following figures. The complete sequence of events that produced these results are contained in table E.1.

Table E.1: The Experiment Stimuli

Slot	Simulator commands	Comments
-	***wdl*** *WDLExperiment*	Load the "Experiment" WDL Model
-	***loadRequirements*** *T1*	Load the "T1" requirements file
-	***loadRequirements*** *T3.1*	Load the "T3.1" requirements file
-	***loadRequirements*** *T3.2*	Load the "T3.2" requirements file
-	***createUser*** *client*	Create the User with ID "client"
-	***createUser*** *administrator*	Create the User with ID "administrator"
-	***tree*** *RT*	Tree browser in Runtime mode
0	***openWindow*** *HomePage client*	Creates a new Window for the User "client" and sets its URL with the "HomePage" object
1	***actionLink*** *hp2mu HomePage wnd_0*	Navigating to the "Music" Page

4	***actionLink*** *OID_12*	Following one of the dynamically constructed Link objects
7	***actionLink*** *ar2hp Artist wnd_0*	Placing the user back at the "HomePage"
9	***actionLink*** *hp2se HomePage wnd_0*	Navigating to the "Search" engine zone
10	***setData*** *Artist.FirstName Frank*	Setting the "Artist.FirstName" form field with value "Frank"
11	***actionButton*** *se2re Search wnd_0*	Submitting the data to the Web Application for processing
15	***actionLink*** *aa2cl Results wnd_0*	First attempt to add the item to the shopping cart
17	***setData*** *LoginEntryUsername pedro*	Since not yet logged in, the "client" is redirected to the "LoginPage" form Page; setting the username form field with value "pedro"...
17	***setData*** *LoginEntryPassword p*	...and the password form field with value "p"
18	***actionButton*** *lp2lo LoginPage wnd_0*	Submitting the login information
21	***actionLink*** *hp2se HomePage wnd_0*	Back at the "HomePage" due to the login procedure, navigating again to the "Search" Page
22	***setData*** *Artist.FirstName Frank*	Setting the "Artist.FirstName" form field with value "Frank"
23	***actionButton*** *se2re Search wnd_0*	Submitting the data
27	***actionLink*** *aa2cl Results wnd_0*	Second attempt to add the item to the shopping cart
34	***openWindow*** *HomePage administrator*	Opening a window for the "administrator" user
35	***actionLink*** *hp2se HomePage wnd_1*	Navigating to the "LoginPage" through the "Search"
37	***actionButton*** *se2re Search wnd_1*	Submitting the form
41	***actionLink*** *aa2cl Results wnd_1*	Administrator at the "LoginPage"
43	***setData*** *LoginEntryUsername admin*	Setting the username form field with a valid value

43	***setData*** *LoginEntryPassword a*	Setting the password form field with a valid value
44	***actionButton*** *lp2lo LoginPage wnd_1*	Submitting the login form

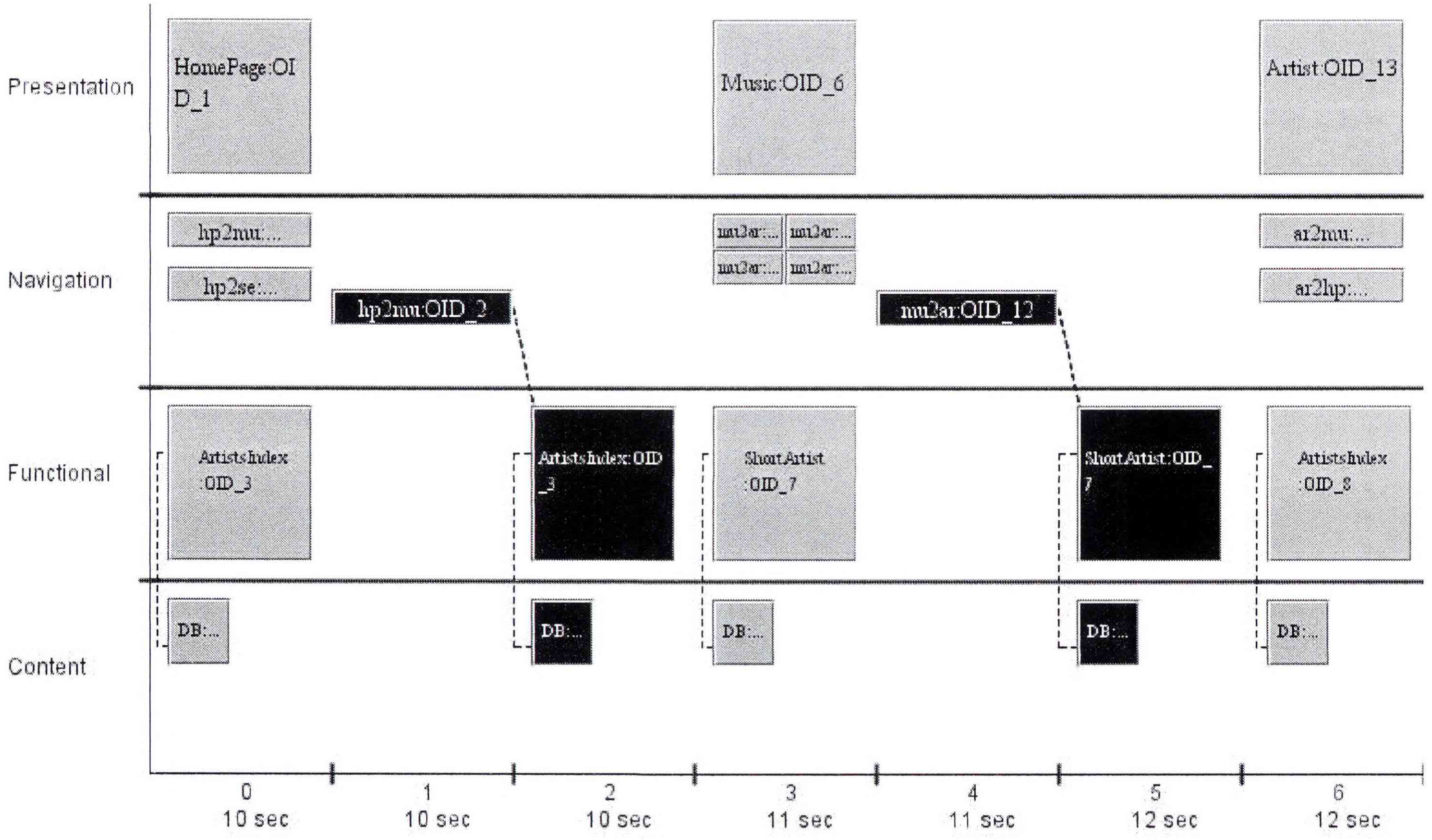


Figure E.1: Simulation Results – TimeSlots 0 to 6

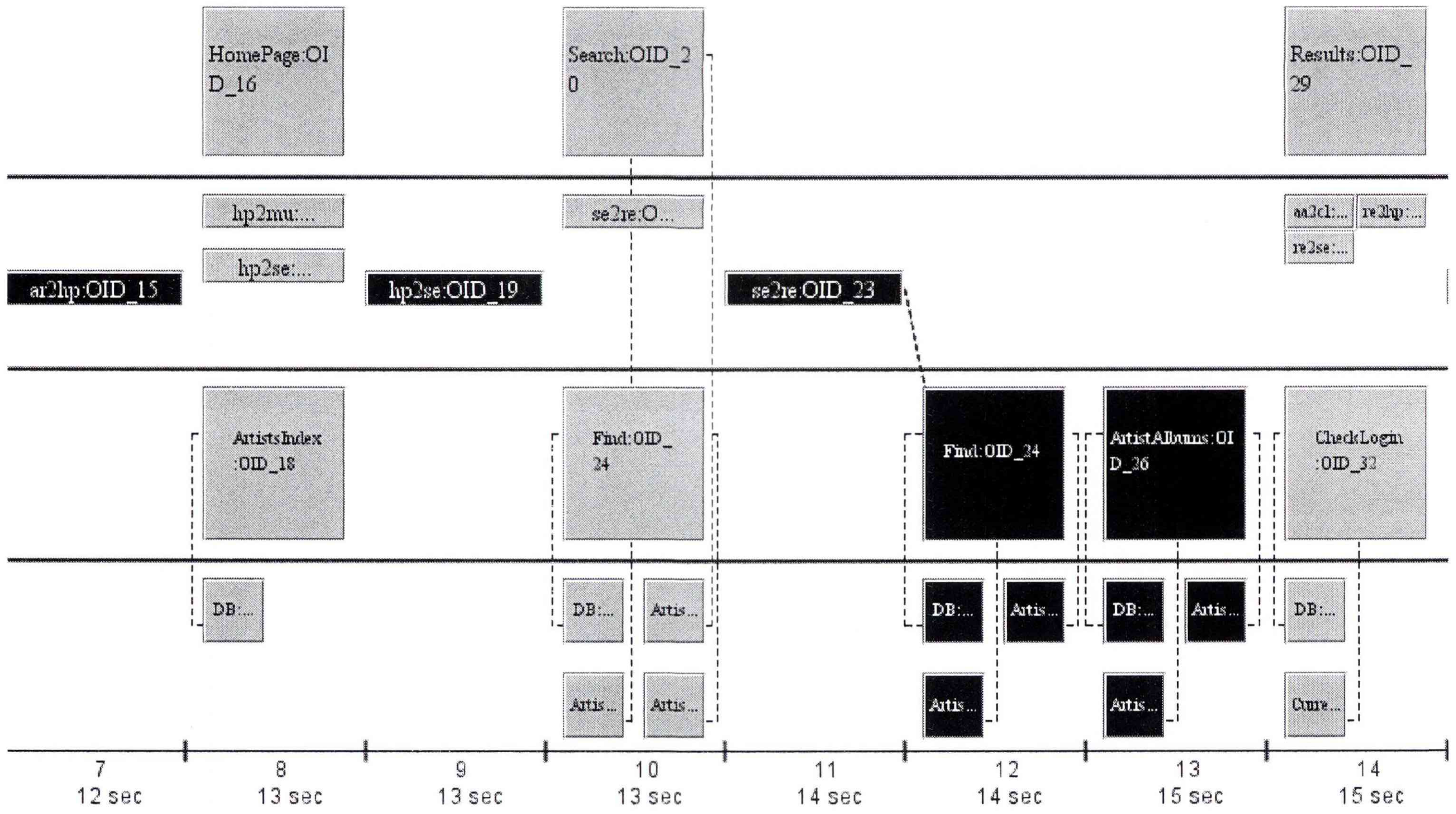


Figure E.2: Simulation Results – TimeSlots 7 to 14

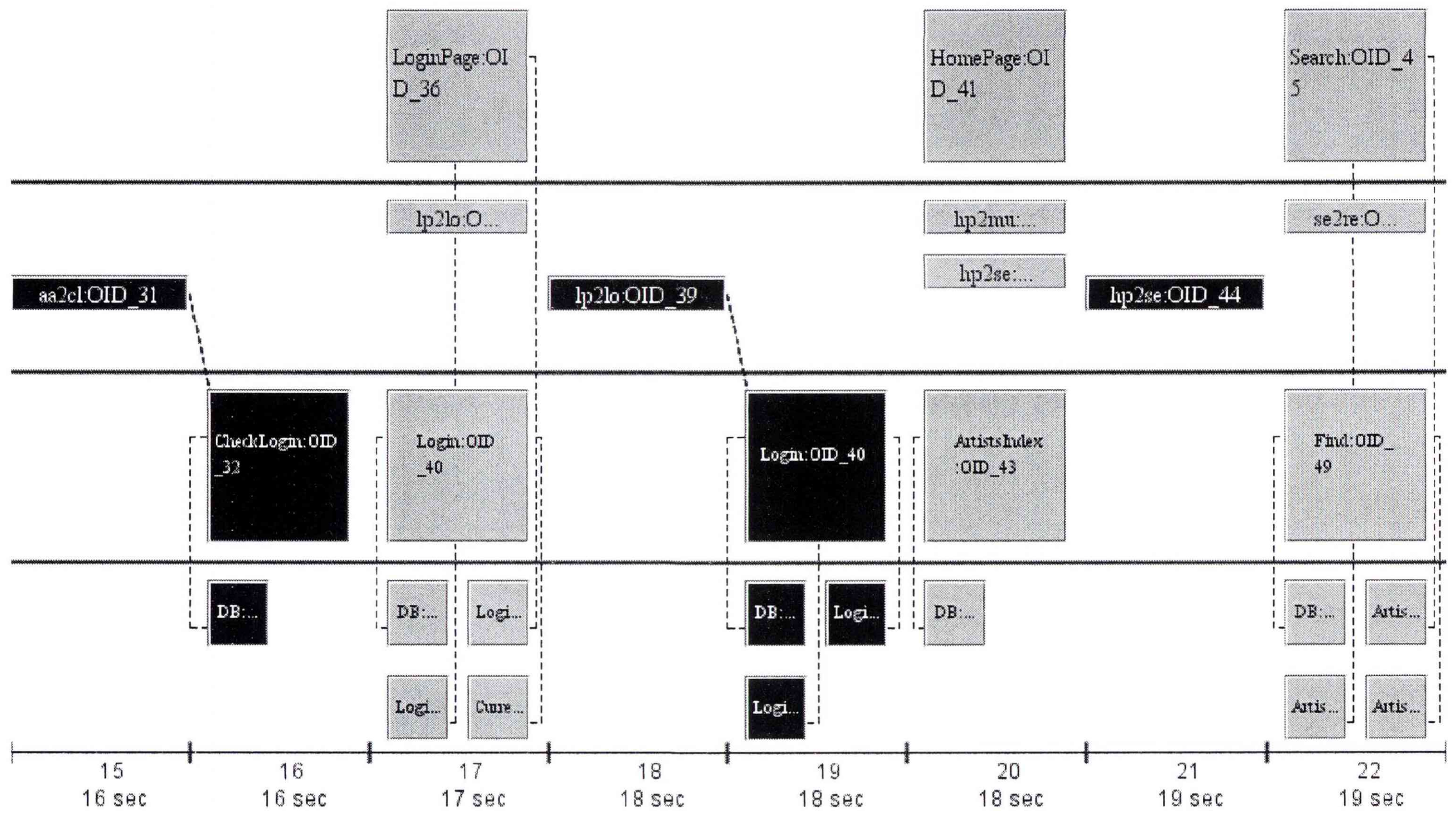


Figure E.3: Simulation Results – TimeSlots 15 to 22

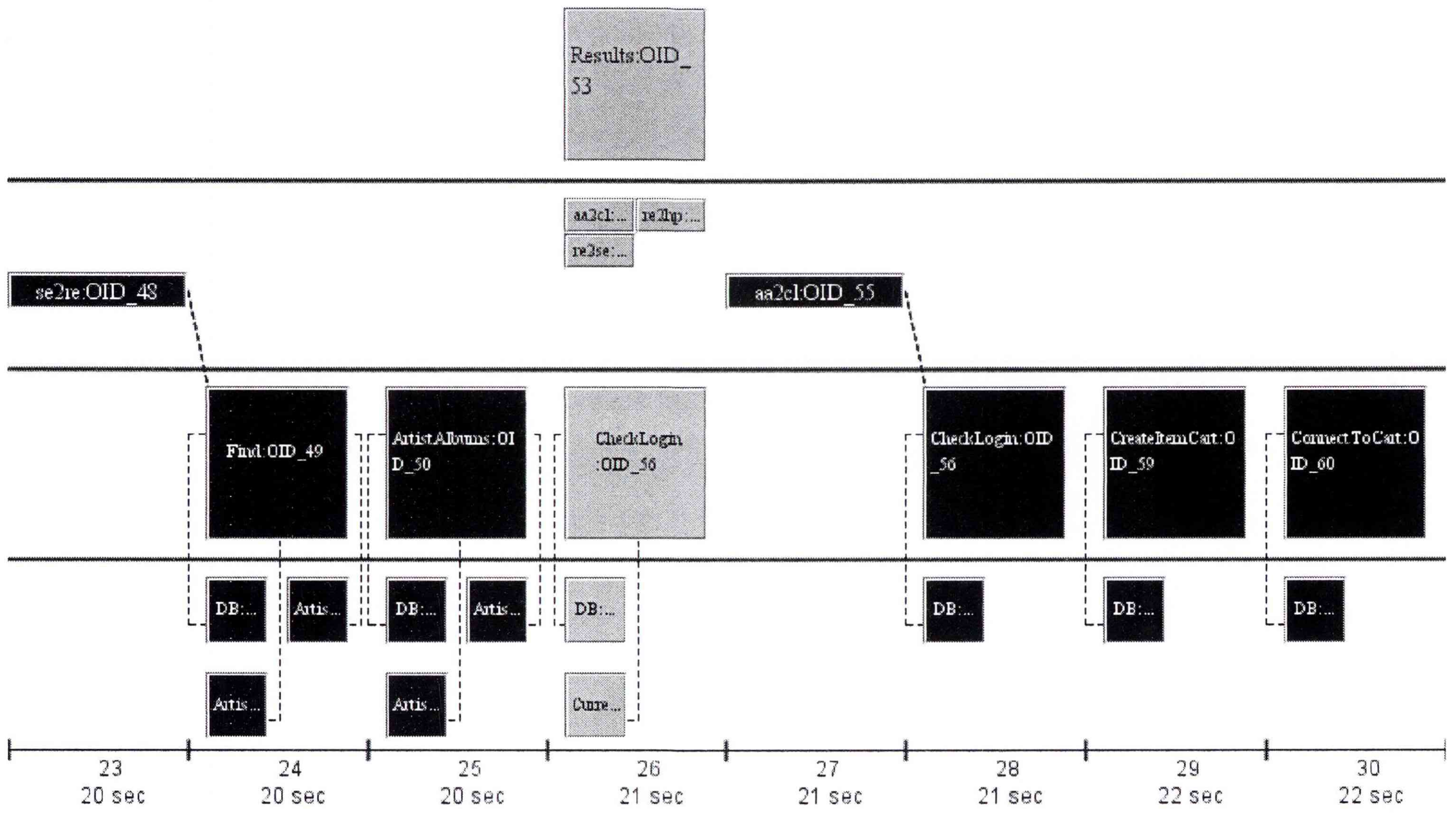


Figure E.4: Simulation Results – TimeSlots 23 to 30

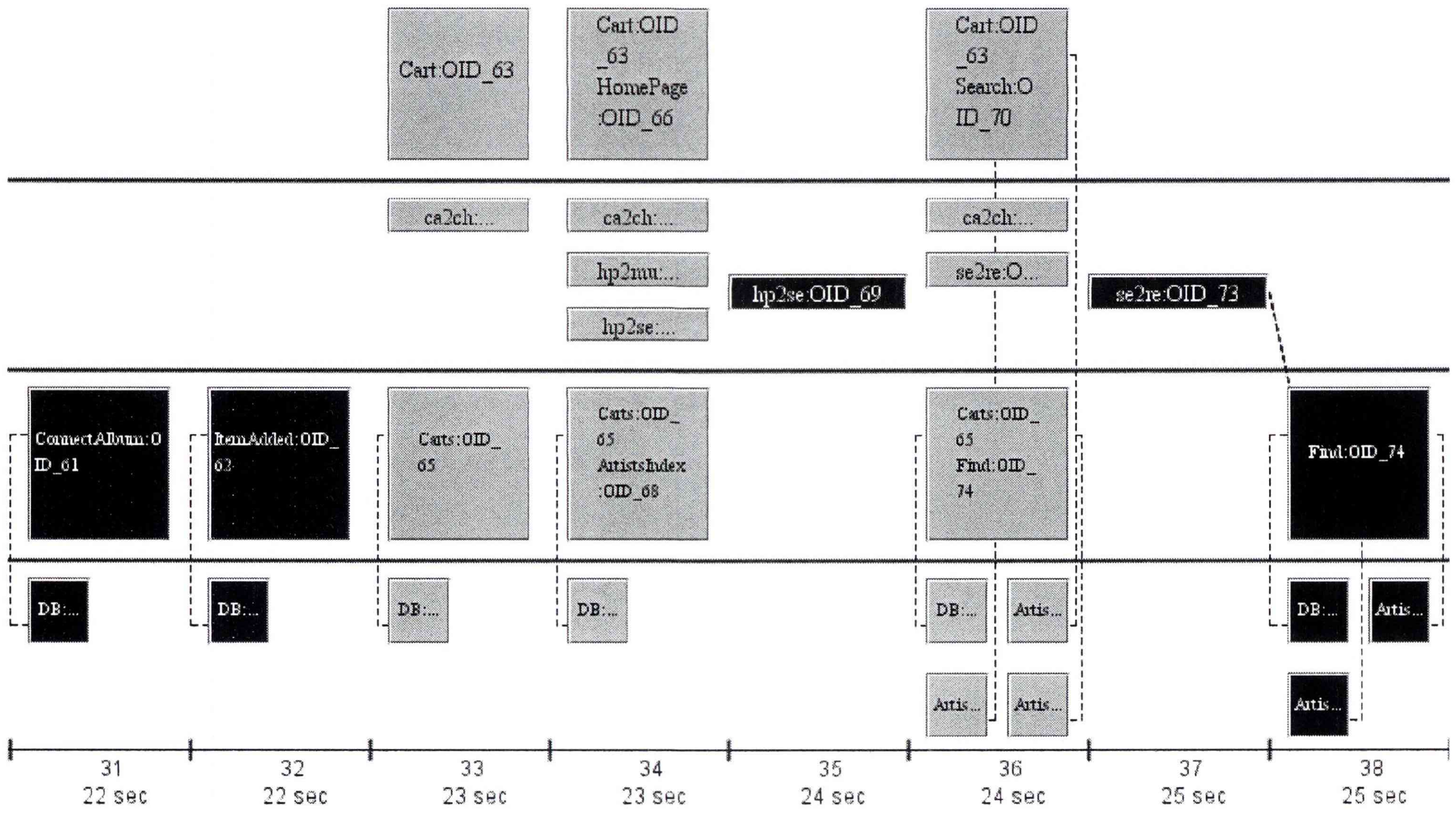


Figure E.5: Simulation Results – TimeSlots 31 to 38

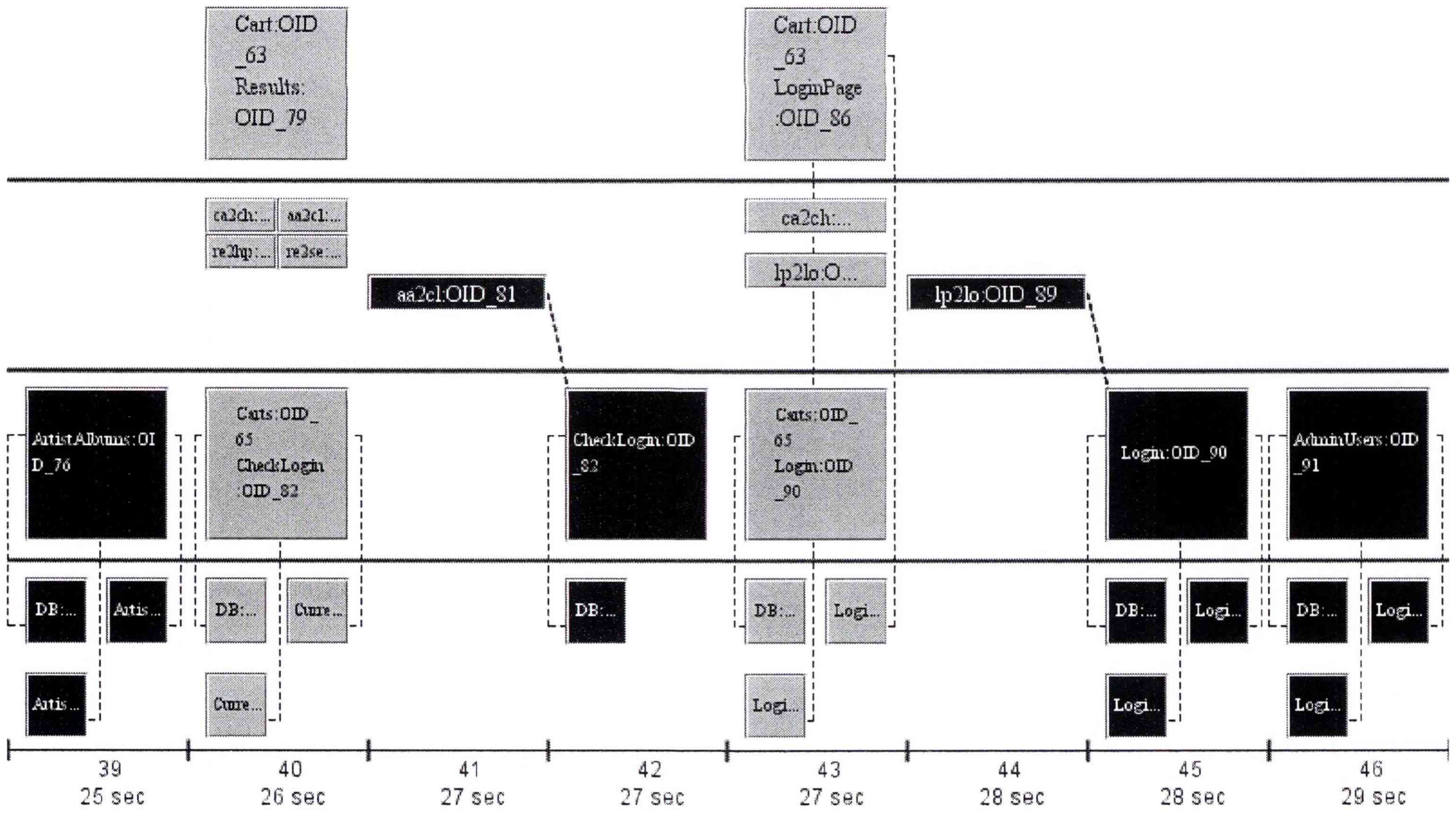


Figure E.6: Simulation Results – TimeSlots 39 to 46

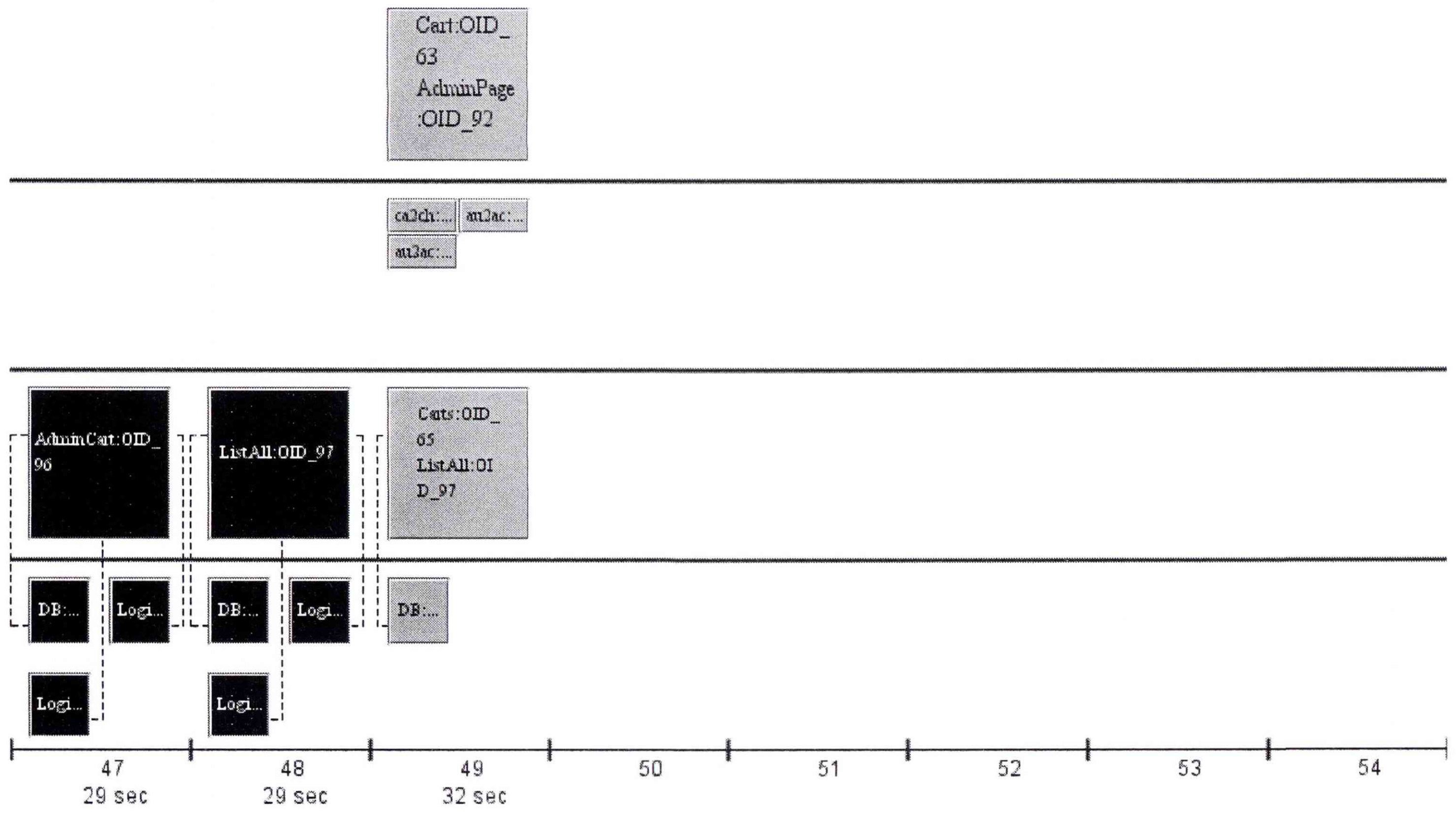


Figure E.7: Simulation Results – TimeSlots 47 to 54

Table E.2: The Contents of the Simulator "pages" Table

slot	id	oid	state	windowoid
0	HomePage	OID_1	ACTIVE	wnd_0
1	HomePage	OID_1	INACTIVE	wnd_0
2	HomePage	OID_1	INACTIVE	wnd_0
3	Music	OID_6	ACTIVE	wnd_0
4	Music	OID_6	INACTIVE	wnd_0
5	Music	OID_6	INACTIVE	wnd_0
6	Artist	OID_13	ACTIVE	wnd_0
7	Artist	OID_13	ACTIVE	wnd_0
8	HomePage	OID_16	ACTIVE	wnd_0
9	HomePage	OID_16	ACTIVE	wnd_0
10	Search	OID_20	ACTIVE	wnd_0
11	Search	OID_20	INACTIVE	wnd_0
12	Search	OID_20	INACTIVE	wnd_0
14	Results	OID_29	ACTIVE	wnd_0
15	Results	OID_29	INACTIVE	wnd_0
16	Results	OID_29	INACTIVE	wnd_0
17	LoginPage	OID_36	ACTIVE	wnd_0
18	LoginPage	OID_36	INACTIVE	wnd_0
19	LoginPage	OID_36	INACTIVE	wnd_0
20	HomePage	OID_41	ACTIVE	wnd_0
21	HomePage	OID_41	ACTIVE	wnd_0
22	Search	OID_45	ACTIVE	wnd_0
23	Search	OID_45	INACTIVE	wnd_0
24	Search	OID_45	INACTIVE	wnd_0
26	Results	OID_53	ACTIVE	wnd_0
27	Results	OID_53	INACTIVE	wnd_0
28	Results	OID_53	INACTIVE	wnd_0
33	Cart	OID_63	ACTIVE	wnd_0

34	Cart	OID_63	ACTIVE	wnd_0
34	HomePage	OID_66	ACTIVE	wnd_1
35	Cart	OID_63	ACTIVE	wnd_0
35	HomePage	OID_66	ACTIVE	wnd_1
36	Cart	OID_63	ACTIVE	wnd_0
36	Search	OID_70	ACTIVE	wnd_1
37	Cart	OID_63	ACTIVE	wnd_0
37	Search	OID_70	INACTIVE	wnd_1
38	Cart	OID_63	ACTIVE	wnd_0
38	Search	OID_70	INACTIVE	wnd_1
40	Cart	OID_63	ACTIVE	wnd_0
40	Results	OID_79	ACTIVE	wnd_1
41	Cart	OID_63	ACTIVE	wnd_0
41	Results	OID_79	INACTIVE	wnd_1
42	Cart	OID_63	ACTIVE	wnd_0
42	Results	OID_79	INACTIVE	wnd_1
43	Cart	OID_63	ACTIVE	wnd_0
43	LoginPage	OID_86	ACTIVE	wnd_1
44	Cart	OID_63	ACTIVE	wnd_0
44	LoginPage	OID_86	INACTIVE	wnd_1
45	Cart	OID_63	ACTIVE	wnd_0
45	LoginPage	OID_86	INACTIVE	wnd_1
49	Cart	OID_63	ACTIVE	wnd_0
49	AdminPage	OID_92	ACTIVE	wnd_1

Table E.3: The Contents of the "links" Table

slot	id	oid	state	parameters	pageoid	windowoid	useroid
0	hp2mu	OID_2	INACTIVE		OID_1	wnd_0	client
0	hp2se	OID_5	INACTIVE		OID_1	wnd_0	client
1	hp2mu	OID_2	ACTIVE		OID_1	wnd_0	client
1	hp2se	OID_5	INACTIVE		OID_1	wnd_0	client
2	hp2mu	OID_2	ACTIVE		OID_1	wnd_0	client
2	hp2se	OID_5	INACTIVE		OID_1	wnd_0	client
3	mu2ar	OID_9	INACTIVE	artist.key = 1;	OID_6	wnd_0	client
3	mu2ar	OID_10	INACTIVE	artist.key = 2;	OID_6	wnd_0	client
3	mu2ar	OID_11	INACTIVE	artist.key = 3;	OID_6	wnd_0	client
3	mu2ar	OID_12	INACTIVE	artist.key = 4;	OID_6	wnd_0	client
4	mu2ar	OID_9	INACTIVE	artist.key = 1;	OID_6	wnd_0	client
4	mu2ar	OID_10	INACTIVE	artist.key = 2;	OID_6	wnd_0	client
4	mu2ar	OID_11	INACTIVE	artist.key = 3;	OID_6	wnd_0	client
4	mu2ar	OID_12	ACTIVE	artist.key = 4;	OID_6	wnd_0	client
5	mu2ar	OID_9	INACTIVE	artist.key = 1;	OID_6	wnd_0	client
5	mu2ar	OID_10	INACTIVE	artist.key = 2;	OID_6	wnd_0	client
5	mu2ar	OID_11	INACTIVE	artist.key = 3;	OID_6	wnd_0	client
5	mu2ar	OID_12	ACTIVE	artist.key = 4;	OID_6	wnd_0	client
6	ar2mu	OID_14	INACTIVE		OID_13	wnd_0	client
6	ar2hp	OID_15	INACTIVE		OID_13	wnd_0	client
7	ar2mu	OID_14	INACTIVE		OID_13	wnd_0	client
7	ar2hp	OID_15	ACTIVE		OID_13	wnd_0	client
8	hp2mu	OID_17	INACTIVE		OID_16	wnd_0	client
8	hp2se	OID_19	INACTIVE		OID_16	wnd_0	client
9	hp2mu	OID_17	INACTIVE		OID_16	wnd_0	client
9	hp2se	OID_19	ACTIVE		OID_16	wnd_0	client
10	se2re	OID_23	INACTIVE		OID_20	wnd_0	client
11	se2re	OID_23	ACTIVE		OID_20	wnd_0	client

12	se2re	OID_23	ACTIVE		OID_20	wnd_0	client
14	f2aa	OID_30	INACTIVE	Artist.OID = 3;	OID_29	wnd_0	client
14	aa2cl	OID_31	INACTIVE	Album.OID = 3;	OID_29	wnd_0	client
14	re2hp	OID_34	INACTIVE		OID_29	wnd_0	client
14	re2se	OID_35	INACTIVE		OID_29	wnd_0	client
15	f2aa	OID_30	INACTIVE	Artist.OID = 3;	OID_29	wnd_0	client
15	aa2cl	OID_31	ACTIVE	Album.OID = 3;	OID_29	wnd_0	client
15	re2hp	OID_34	INACTIVE		OID_29	wnd_0	client
15	re2se	OID_35	INACTIVE		OID_29	wnd_0	client
16	f2aa	OID_30	INACTIVE	Artist.OID = 3;	OID_29	wnd_0	client
16	aa2cl	OID_31	ACTIVE	Album.OID = 3;	OID_29	wnd_0	client
16	re2hp	OID_34	INACTIVE		OID_29	wnd_0	client
16	re2se	OID_35	INACTIVE		OID_29	wnd_0	client
17	lp2lo	OID_39	INACTIVE		OID_36	wnd_0	client
18	lp2lo	OID_39	ACTIVE		OID_36	wnd_0	client
19	lp2lo	OID_39	ACTIVE		OID_36	wnd_0	client
20	hp2mu	OID_42	INACTIVE		OID_41	wnd_0	client
20	hp2se	OID_44	INACTIVE		OID_41	wnd_0	client
21	hp2mu	OID_42	INACTIVE		OID_41	wnd_0	client
21	hp2se	OID_44	ACTIVE		OID_41	wnd_0	client
22	se2re	OID_48	INACTIVE		OID_45	wnd_0	client
23	se2re	OID_48	ACTIVE		OID_45	wnd_0	client
24	se2re	OID_48	ACTIVE		OID_45	wnd_0	client
26	f2aa	OID_54	INACTIVE	Artist.OID = 3;	OID_53	wnd_0	client
26	aa2cl	OID_55	INACTIVE	Album.OID = 3;	OID_53	wnd_0	client
26	re2hp	OID_57	INACTIVE		OID_53	wnd_0	client
26	re2se	OID_58	INACTIVE		OID_53	wnd_0	client
27	f2aa	OID_54	INACTIVE	Artist.OID = 3;	OID_53	wnd_0	client
27	aa2cl	OID_55	ACTIVE	Album.OID = 3;	OID_53	wnd_0	client
27	re2hp	OID_57	INACTIVE		OID_53	wnd_0	client
27	re2se	OID_58	INACTIVE		OID_53	wnd_0	client

28	f2aa	OID_54	INACTIVE	Artist.OID = 3;	OID_53	wnd_0	client
28	aa2cl	OID_55	ACTIVE	Album.OID = 3;	OID_53	wnd_0	client
28	re2hp	OID_57	INACTIVE		OID_53	wnd_0	client
28	re2se	OID_58	INACTIVE		OID_53	wnd_0	client
33	ca2ch	OID_64	INACTIVE		OID_63	wnd_0	client
34	ca2ch	OID_64	INACTIVE		OID_63	wnd_0	client
34	hp2mu	OID_67	INACTIVE		OID_66	wnd_1	administrator
34	hp2se	OID_69	INACTIVE		OID_66	wnd_1	administrator
35	ca2ch	OID_64	INACTIVE		OID_63	wnd_0	client
35	hp2mu	OID_67	INACTIVE		OID_66	wnd_1	administrator
35	hp2se	OID_69	ACTIVE		OID_66	wnd_1	administrator
36	ca2ch	OID_64	INACTIVE		OID_63	wnd_0	client
36	se2re	OID_73	INACTIVE		OID_70	wnd_1	administrator
37	ca2ch	OID_64	INACTIVE		OID_63	wnd_0	client
37	se2re	OID_73	ACTIVE		OID_70	wnd_1	administrator
38	ca2ch	OID_64	INACTIVE		OID_63	wnd_0	client
38	se2re	OID_73	ACTIVE		OID_70	wnd_1	administrator
40	ca2ch	OID_64	INACTIVE		OID_63	wnd_0	client
40	f2aa	OID_80	INACTIVE	Artist.OID = 1;	OID_79	wnd_1	administrator
40	aa2cl	OID_81	INACTIVE	Album.OID = 1;	OID_79	wnd_1	administrator
40	re2hp	OID_84	INACTIVE		OID_79	wnd_1	administrator
40	re2se	OID_85	INACTIVE		OID_79	wnd_1	administrator
41	ca2ch	OID_64	INACTIVE		OID_63	wnd_0	client
41	f2aa	OID_80	INACTIVE	Artist.OID = 1;	OID_79	wnd_1	administrator
41	aa2cl	OID_81	ACTIVE	Album.OID = 1;	OID_79	wnd_1	administrator
41	re2hp	OID_84	INACTIVE		OID_79	wnd_1	administrator
41	re2se	OID_85	INACTIVE		OID_79	wnd_1	administrator
42	ca2ch	OID_64	INACTIVE		OID_63	wnd_0	client
42	f2aa	OID_80	INACTIVE	Artist.OID = 1;	OID_79	wnd_1	administrator
42	aa2cl	OID_81	ACTIVE	Album.OID = 1;	OID_79	wnd_1	administrator
42	re2hp	OID_84	INACTIVE		OID_79	wnd_1	administrator

42	re2se	OID_85	INACTIVE		OID_79	wnd_1	administrator
43	ca2ch	OID_64	INACTIVE		OID_63	wnd_0	client
43	lp2lo	OID_89	INACTIVE		OID_86	wnd_1	administrator
44	ca2ch	OID_64	INACTIVE		OID_63	wnd_0	client
44	lp2lo	OID_89	ACTIVE		OID_86	wnd_1	administrator
45	ca2ch	OID_64	INACTIVE		OID_63	wnd_0	client
45	lp2lo	OID_89	ACTIVE		OID_86	wnd_1	administrator
49	ca2ch	OID_64	INACTIVE		OID_63	wnd_0	client
49	au2ac	OID_94	INACTIVE	User.OID = 1;	OID_92	wnd_1	administrator
49	au2ac	OID_95	INACTIVE	User.OID = 2;	OID_92	wnd_1	administrator

Table E.4: The Contents of the "scripts" Table

slot	id	oid	state	calleroid	windowoid	useroid
0	ArtistsIndex	OID_3	INACTIVE	null	wnd_0	null
1	ArtistsIndex	OID_3	ACTIVE	OID_1	wnd_0	client
2	ArtistsIndex	OID_3	ACTIVE	OID_1	wnd_0	client
3	ShortArtist	OID_7	INACTIVE	null	wnd_0	null
4	ShortArtist	OID_7	ACTIVE	OID_6	wnd_0	client
5	ShortArtist	OID_7	ACTIVE	OID_6	wnd_0	client
6	ArtistsIndex	OID_8	INACTIVE	null	wnd_0	null
7	ArtistsIndex	OID_8	INACTIVE	null	wnd_0	null
8	ArtistsIndex	OID_18	INACTIVE	null	wnd_0	null
9	ArtistsIndex	OID_18	INACTIVE	null	wnd_0	null
10	Find	OID_24	INACTIVE	null	wnd_0	null
11	Find	OID_24	ACTIVE	OID_20	wnd_0	client
12	Find	OID_24	ACTIVE	OID_20	wnd_0	client
13	ArtistAlbums	OID_26	ACTIVE	wnd_0	wnd_0	client
14	CheckLogin	OID_32	INACTIVE	null	wnd_0	null
15	CheckLogin	OID_32	ACTIVE	OID_29	wnd_0	client
16	CheckLogin	OID_32	ACTIVE	OID_29	wnd_0	client
17	Login	OID_40	INACTIVE	null	wnd_0	null
18	Login	OID_40	ACTIVE	OID_36	wnd_0	client
19	Login	OID_40	ACTIVE	OID_36	wnd_0	client
20	ArtistsIndex	OID_43	INACTIVE	null	wnd_0	null
21	ArtistsIndex	OID_43	INACTIVE	null	wnd_0	null
22	Find	OID_49	INACTIVE	null	wnd_0	null
23	Find	OID_49	ACTIVE	OID_45	wnd_0	client
24	Find	OID_49	ACTIVE	OID_45	wnd_0	client
25	ArtistAlbums	OID_50	ACTIVE	wnd_0	wnd_0	client
26	CheckLogin	OID_56	INACTIVE	null	wnd_0	null
27	CheckLogin	OID_56	ACTIVE	OID_53	wnd_0	client

28	CheckLogin	OID_56	ACTIVE	OID_53	wnd_0	client
29	CreateItemCart	OID_59	ACTIVE	wnd_0	wnd_0	client
30	ConnectToCart	OID_60	ACTIVE	wnd_0	wnd_0	client
31	ConnectAlbum	OID_61	ACTIVE	wnd_0	wnd_0	client
32	ItemAdded	OID_62	ACTIVE	wnd_0	wnd_0	client
33	Carts	OID_65	INACTIVE	null	wnd_0	null
34	Carts	OID_65	INACTIVE	null	wnd_0	null
34	ArtistsIndex	OID_68	INACTIVE	null	wnd_1	null
35	Carts	OID_65	INACTIVE	null	wnd_0	null
35	ArtistsIndex	OID_68	INACTIVE	null	wnd_1	null
36	Carts	OID_65	INACTIVE	null	wnd_0	null
36	Find	OID_74	INACTIVE	null	wnd_1	null
37	Carts	OID_65	INACTIVE	null	wnd_0	null
37	Find	OID_74	ACTIVE	OID_70	wnd_1	administrator
38	Carts	OID_65	INACTIVE	null	wnd_0	null
38	Find	OID_74	ACTIVE	OID_70	wnd_1	administrator
39	ArtistAlbums	OID_76	ACTIVE	wnd_1	wnd_1	administrator
40	Carts	OID_65	INACTIVE	null	wnd_0	null
40	CheckLogin	OID_82	INACTIVE	null	wnd_1	null
41	Carts	OID_65	INACTIVE	null	wnd_0	null
41	CheckLogin	OID_82	ACTIVE	OID_79	wnd_1	administrator
42	Carts	OID_65	INACTIVE	null	wnd_0	null
42	CheckLogin	OID_82	ACTIVE	OID_79	wnd_1	administrator
43	Carts	OID_65	INACTIVE	null	wnd_0	null
43	Login	OID_90	INACTIVE	null	wnd_1	null
44	Carts	OID_65	INACTIVE	null	wnd_0	null
44	Login	OID_90	ACTIVE	OID_86	wnd_1	administrator
45	Carts	OID_65	INACTIVE	null	wnd_0	null
45	Login	OID_90	ACTIVE	OID_86	wnd_1	administrator
46	AdminUsers	OID_91	ACTIVE	wnd_1	wnd_1	administrator
47	AdminCart	OID_96	ACTIVE	wnd_1	wnd_1	administrator

48	ListAll	OID_97	ACTIVE	wnd_1	wnd_1	administrator
49	Carts	OID_65	INACTIVE	null	wnd_0	null
49	ListAll	OID_97	INACTIVE	null	wnd_1	null

Table E.5: The Contents of the “scriptvars” Table

slot	name	value
3	Artist.key	UNDEFINED
4	Artist.key	4
5	Artist.key	4
10	Artist.FirstName	Frank
10	Artist.LastName	UNDEFINED
10	ArtistOID	UNDEFINED
11	Artist.FirstName	Frank
11	Artist.LastName	UNDEFINED
11	ArtistOID	UNDEFINED
12	Artist.FirstName	Frank
12	Artist.LastName	UNDEFINED
12	ArtistOID	3
14	user	UNDEFINED
14	Success	UNDEFINED
15	user	UNDEFINED
15	Success	UNDEFINED
16	user	UNDEFINED
16	Success	false
17	LoginEntryUsername	pedro
17	LoginEntryPassword	p
18	LoginEntryUsername	pedro
18	LoginEntryPassword	p
19	LoginEntryUsername	pedro
19	LoginEntryPassword	p
22	Artist.FirstName	Frank
22	Artist.LastName	UNDEFINED
22	ArtistOID	3
23	Artist.FirstName	Frank

23	Artist.LastName	UNDEFINED
23	ArtistOID	3
24	Artist.FirstName	Frank
24	Artist.LastName	UNDEFINED
24	ArtistOID	3
26	user	UNDEFINED
26	Success	UNDEFINED
27	user	UNDEFINED
27	Success	UNDEFINED
28	user	pedro
28	Success	true
36	Artist.FirstName	UNDEFINED
36	Artist.LastName	UNDEFINED
36	ArtistOID	UNDEFINED
40	user	UNDEFINED
40	Success	UNDEFINED
41	user	UNDEFINED
41	Success	UNDEFINED
42	user	UNDEFINED
42	Success	false
43	LoginEntryUsername	admin
43	LoginEntryPassword	a
44	LoginEntryUsername	admin
44	LoginEntryPassword	a
45	LoginEntryUsername	admin
45	LoginEntryPassword	a

Table E.6: The Contents of the "datas" Table

slot	id	oid	state	windowoid	useroid	value
0	DB	OID_4	ACTIVE	SHARED	null	null
1	DB	OID_4	ACTIVE	SHARED	null	null
2	DB	OID_4	ACTIVE	SHARED	null	null
3	DB	OID_4	ACTIVE	SHARED	null	null
4	DB	OID_4	ACTIVE	SHARED	null	null
5	DB	OID_4	ACTIVE	SHARED	null	null
6	DB	OID_4	ACTIVE	SHARED	null	null
7	DB	OID_4	ACTIVE	SHARED	null	null
8	DB	OID_4	ACTIVE	SHARED	null	null
9	DB	OID_4	ACTIVE	SHARED	null	null
10	DB	OID_4	ACTIVE	SHARED	null	null
10	Artist.FirstName	OID_21	ACTIVE	wnd_0	client	UNDEFINED
10	Artist.LastName	OID_22	ACTIVE	wnd_0	client	UNDEFINED
10	ArtistOID	OID_25	ACTIVE	wnd_0	client	UNDEFINED
11	DB	OID_4	ACTIVE	SHARED	null	null
11	Artist.FirstName	OID_21	ACTIVE	wnd_0	client	Frank
11	Artist.LastName	OID_22	ACTIVE	wnd_0	client	UNDEFINED
11	ArtistOID	OID_25	ACTIVE	wnd_0	client	UNDEFINED
12	DB	OID_4	ACTIVE	SHARED	null	null
12	Artist.FirstName	OID_21	ACTIVE	wnd_0	client	Frank
12	Artist.LastName	OID_22	ACTIVE	wnd_0	client	UNDEFINED
12	ArtistOID	OID_25	ACTIVE	wnd_0	client	3
14	DB	OID_4	ACTIVE	SHARED	null	null
14	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
14	CurrentUser	OID_33	ACTIVE	wnd_0	client	UNDEFINED
15	DB	OID_4	ACTIVE	SHARED	null	null
15	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
15	CurrentUser	OID_33	ACTIVE	wnd_0	client	UNDEFINED

16	DB	OID_4	ACTIVE	SHARED	null	null
16	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
16	CurrentUser	OID_33	ACTIVE	wnd_0	client	UNDEFINED
17	DB	OID_4	ACTIVE	SHARED	null	null
17	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
17	CurrentUser	OID_33	INACTIVE	wnd_0	client	UNDEFINED
17	LoginEntryUsername	OID_37	ACTIVE	wnd_0	client	UNDEFINED
17	LoginEntryPassword	OID_38	ACTIVE	wnd_0	client	UNDEFINED
18	DB	OID_4	ACTIVE	SHARED	null	null
18	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
18	CurrentUser	OID_33	ACTIVE	wnd_0	client	2
18	LoginEntryUsername	OID_37	ACTIVE	wnd_0	client	pedro
18	LoginEntryPassword	OID_38	ACTIVE	wnd_0	client	p
19	DB	OID_4	ACTIVE	SHARED	null	null
19	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
19	CurrentUser	OID_33	ACTIVE	wnd_0	client	2
19	LoginEntryUsername	OID_37	ACTIVE	wnd_0	client	pedro
19	LoginEntryPassword	OID_38	ACTIVE	wnd_0	client	p
20	DB	OID_4	ACTIVE	SHARED	null	null
20	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
20	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
21	DB	OID_4	ACTIVE	SHARED	null	null
21	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
21	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
22	DB	OID_4	ACTIVE	SHARED	null	null
22	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
22	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
22	Artist.FirstName	OID_46	ACTIVE	wnd_0	client	UNDEFINED
22	Artist.LastName	OID_47	ACTIVE	wnd_0	client	UNDEFINED
23	DB	OID_4	ACTIVE	SHARED	null	null
23	ArtistOID	OID_25	ACTIVE	wnd_0	client	3

23	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
23	Artist.FirstName	OID_46	ACTIVE	wnd_0	client	Frank
23	Artist.LastName	OID_47	ACTIVE	wnd_0	client	UNDEFINED
24	DB	OID_4	ACTIVE	SHARED	null	null
24	ArtistOID	OID_25	ACTIVE	wnd_0	client	3
24	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
24	Artist.FirstName	OID_46	ACTIVE	wnd_0	client	Frank
24	Artist.LastName	OID_47	ACTIVE	wnd_0	client	UNDEFINED
26	DB	OID_4	ACTIVE	SHARED	null	null
26	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
26	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
27	DB	OID_4	ACTIVE	SHARED	null	null
27	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
27	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
28	DB	OID_4	ACTIVE	SHARED	null	null
28	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
28	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
33	DB	OID_4	ACTIVE	SHARED	null	null
33	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
33	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
34	DB	OID_4	ACTIVE	SHARED	null	null
34	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
34	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
35	DB	OID_4	ACTIVE	SHARED	null	null
35	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
35	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
36	DB	OID_4	ACTIVE	SHARED	null	null
36	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
36	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
36	Artist.FirstName	OID_71	ACTIVE	wnd_1	administrator	UNDEFINED
36	Artist.LastName	OID_72	ACTIVE	wnd_1	administrator	UNDEFINED

36	ArtistOID	OID_75	ACTIVE	wnd_1	administrator	UNDEFINED
37	DB	OID_4	ACTIVE	SHARED	null	null
37	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
37	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
37	Artist.FirstName	OID_71	ACTIVE	wnd_1	administrator	UNDEFINED
37	Artist.LastName	OID_72	ACTIVE	wnd_1	administrator	UNDEFINED
37	ArtistOID	OID_75	ACTIVE	wnd_1	administrator	UNDEFINED
38	DB	OID_4	ACTIVE	SHARED	null	null
38	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
38	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
38	Artist.FirstName	OID_71	ACTIVE	wnd_1	administrator	UNDEFINED
38	Artist.LastName	OID_72	ACTIVE	wnd_1	administrator	UNDEFINED
38	ArtistOID	OID_75	ACTIVE	wnd_1	administrator	UNDEFINED
40	DB	OID_4	ACTIVE	SHARED	null	null
40	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
40	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
40	ArtistOID	OID_75	INACTIVE	wnd_1	administrator	UNDEFINED
40	CurrentUser	OID_83	ACTIVE	wnd_1	administrator	UNDEFINED
41	DB	OID_4	ACTIVE	SHARED	null	null
41	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
41	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
41	ArtistOID	OID_75	INACTIVE	wnd_1	administrator	UNDEFINED
41	CurrentUser	OID_83	ACTIVE	wnd_1	administrator	UNDEFINED
42	DB	OID_4	ACTIVE	SHARED	null	null
42	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
42	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
42	ArtistOID	OID_75	INACTIVE	wnd_1	administrator	UNDEFINED
42	CurrentUser	OID_83	ACTIVE	wnd_1	administrator	UNDEFINED
43	DB	OID_4	ACTIVE	SHARED	null	null
43	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
43	CurrentUser	OID_33	INACTIVE	wnd_0	client	2

43	ArtistOID	OID_75	INACTIVE	wnd_1	administrator	UNDEFINED
43	CurrentUser	OID_83	INACTIVE	wnd_1	administrator	UNDEFINED
43	LoginEntryUsername	OID_87	ACTIVE	wnd_1	administrator	UNDEFINED
43	LoginEntryPassword	OID_88	ACTIVE	wnd_1	administrator	UNDEFINED
44	DB	OID_4	ACTIVE	SHARED	null	null
44	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
44	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
44	ArtistOID	OID_75	INACTIVE	wnd_1	administrator	UNDEFINED
44	CurrentUser	OID_83	ACTIVE	wnd_1	administrator	1
44	LoginEntryUsername	OID_87	ACTIVE	wnd_1	administrator	admin
44	LoginEntryPassword	OID_88	ACTIVE	wnd_1	administrator	a
45	DB	OID_4	ACTIVE	SHARED	null	null
45	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
45	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
45	ArtistOID	OID_75	INACTIVE	wnd_1	administrator	UNDEFINED
45	CurrentUser	OID_83	ACTIVE	wnd_1	administrator	1
45	LoginEntryUsername	OID_87	ACTIVE	wnd_1	administrator	admin
45	LoginEntryPassword	OID_88	ACTIVE	wnd_1	administrator	a
49	DB	OID_4	ACTIVE	SHARED	null	null
49	ArtistOID	OID_25	INACTIVE	wnd_0	client	3
49	CurrentUser	OID_33	INACTIVE	wnd_0	client	2
49	ArtistOID	OID_75	INACTIVE	wnd_1	administrator	UNDEFINED
49	CurrentUser	OID_83	INACTIVE	wnd_1	administrator	1

Bibliography

Adrion, W. R., Branstad, M. A. and Cherniavsky, J. C. (1982), 'Validation, Verification, and Testing of Computer Software', *ACM Comput. Surv.* **14**(2), 159–192.

Amazon (2005), 'Amazon', `http://www.amazon.com/`.

AppPerfect Co. (2005), 'AppPerfect', `http://www.appperfect.com/`.

Ash, L. (2003), *The Web testing companion: the insider's guide to efficient and effective tests*, Wiley Pub.

Bailey, S., Marschner, E., Bhasker, J., Lewis, J. and Ashenden, P. (2004), Improving design and verification productivity with VHDL-200x, *in* 'Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings', Vol. 3, pp. 332–333 Vol.3.

Balci, O. (1987), Guidelines for Successful Simulation Studies, Technical Report TR-85-2, VPI&SU, Blacksburg, VA.

Balci, O. (1990), Guidelines for successful simulation studies, *in* 'Simulation Conference, 1990. Proceedings., Winter', pp. 25–32.

Balci, O. (1994), Validation, verification, and testing techniques throughout the life cycle of a simulation study, *in* 'WSC '94: Proceedings of the 26th conference on

Winter simulation', Society for Computer Simulation International, San Diego, CA, USA, pp. 215–220.

Balci, O. (1995), Principles and techniques of simulation validation, verification, and testing, *in* 'WSC '95: Proceedings of the 27th conference on Winter simulation', ACM Press, pp. 147–154.

Balci, O. (1997), Verification validation and accreditation of simulation models, *in* 'WSC '97: Proceedings of the 29th conference on Winter simulation', ACM Press, New York, NY, USA, pp. 135–141.

Balci, O. and Sargent, R. (1984), 'A Bibliography on the Credibility Assessment and Validation of Simulation and Mathematical Models', *Simuletter* **15**(3), 15–27.

Banks, J. (1999), Introduction to simulation, *in* 'WSC '99: Proceedings of the 31st conference on Winter simulation', ACM Press, New York, NY, USA, pp. 7–13.

Banks, J. (2000), Introduction to Simulation, *in* 'WSC '00: Proceedings of the 32nd conference on Winter simulation', Society for Computer Simulation International, San Diego, CA, USA, pp. 9–16.

Banks, J. and Carson, J. S. (1986), Introduction to discrete-event simulation, *in* 'WSC '86: Proceedings of the 18th conference on Winter simulation', ACM Press, New York, NY, USA, pp. 17–23.

Baresi, L., Garzotto, F. and Paolini, P. (2000), From Web Sites to Web Applications: New Issues for Conceptual Modeling, *in* 'ER '00: Proceedings of the Workshops on Conceptual Modeling Approaches for E-Business and The World Wide Web and Conceptual Modeling', Springer-Verlag, London, UK, pp. 89–100.

Baresi, L., Garzotto, F. and Paolini, P. (2001), Extending UML for modeling Web applications, *in* 'Proceedings of the 34th Annual Hawaii International Conference on System Sciences', pp. 1285–1294.

Baresi, L., Garzotto, F., Paolini, P. and Valenti, S. (2000), HDM2000: The HDM Hypertext Design Model Revisited, Technical report, Politecnico di Milano.

Barry, C. and Lang, M. (2001), 'A survey of multimedia and Web development techniques and methodology usage', *Multimedia, IEEE* **8**(2), 52–60.

Basili, V. R. (1996), The role of experimentation in software engineering: past, current, and future, *in* 'ICSE '96: Proceedings of the 18th international conference on Software engineering', IEEE Computer Society, Washington, DC, USA, pp. 442–449.

Bell, A. E. (2005), 'UML fever: diagnosis and recovery', *Queue* **3**(2), 48–56.

Bellettini, C., Marchetto, A. and Trentini, A. (2005), TestUml: user-metrics driven Web applications testing, *in* 'SAC '05: Proceedings of the 2005 ACM symposium on Applied computing', ACM Press, New York, NY, USA, pp. 1694–1698.

Bichler, M. and Nusser, S. (1996), Modular design of complex Web-applications with W3DT, *in* 'Enabling Technologies: Infrastructure for Collaborative Enterprises, 1996. Proceedings of the 5th Workshop on', pp. 328–333.

BIGSF (2003), 'Government Web Application Integrity'. The Business Internet Group of San Francisco.

Boehm, B. (1986), 'A spiral model of software development and enhancement', *SIGSOFT Softw. Eng. Notes* **11**(4), 14–24.

Boehm, B. W. (1988), 'A spiral model of software development and enhancement', *Computer* **21**(5), 61–72.

Bolchini, D. and Randazzo, G. (2005), Capturing visions and goals to inform communication design, *in* 'SIGDOC '05: Proceedings of the 23rd annual international conference on Design of communication', ACM Press, New York, NY, USA, pp. 131–137.

Bongio, A., Ceri, S., Fraternali, P. and Maurino, A. (2001), 'Modeling data entry and operations in WebML', *Lecture Notes in Computer Science* **1997**.

Bosch, F., Ellis, J. R., Freeman, P., Johnson, L., McClure, C., Robinson, D., Scacchi, W., Scheff, B., Staa, A. and Tripp, L. L. (1982), 'Evaluation of software development life cycle: methodology implementation', *SIGSOFT Softw. Eng. Notes* **7**(1), 45–60.

Brilliant, S. S. and Knight, J. C. (1999), 'Empirical research in software engineering: a workshop', *SIGSOFT Softw. Eng. Notes* **24**(3), 44–52.

Bryhni, H., Klovning, E. and Kure, O. (2000), 'A comparison of load balancing techniques for scalable Web servers', *Network, IEEE* **14**(4), 58–64.

Cachero, C., Gomez, J. and Pastor, O. (2000), Object-Oriented Conceptual Modeling of Web Application Interfaces: the OO-HMethod Abstract Presentation Model, *in* 'EC-Web', pp. 206–215.

Campbell, D. T. and Stanley, J. C. (1966), *Experimental and quasi-experimental designs for research*, Rand McNally, Chicago.

Canoo (2005), 'WebTest', `http://webtest.canoo.com/`.

Cassandras, C. G. and Lafortune, S. (1999), *Introduction to discrete event systems*, Kluwer Academic, Boston; London.

Cazzola, W., Savigni, A., Sosio, A. and Tisato, F. (1998), A fresh look at programming-in-the-large, *in* 'Computer Software and Applications Conference, 1998. COMPSAC '98. Proceedings. The Twenty-Second Annual International', pp. 502–506.

Ceri, S., Fraternali, P. and Bongio, A. (2000), 'Web Modeling Language: a modeling language for designing Web sites', `http://webml.elet.polimi.it/webml/documents/www9.pdf`. Dipartamento di Elettronica e Informazione, Politecnico di Milano, WWW9 Conference, Amsterdam, May 2000.

Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S. and Matera, M. (2002), *Designing data-intensive Web applications*, Morgan Kaufmann; Elsevier Science, San Francisco, Calif. Oxford.

Ceri, S., Fraternali, P. and Matera, M. (2002), 'Conceptual Modeling of Data-Intensive Web Applications', *IEEE Internet Computing* **6**(4), 20–30.

Chen, P. P.-S. (1976), 'The entity-relationship model – toward a unified view of data', *ACM Trans. Database Syst.* **1**(1), 9–36.

Christodoulou, S. P., Styliaras, G. D. and Papatheodrou, T. S. (1998), Evaluation of hypermedia application development and management systems, *in* 'HYPERTEXT '98: Proceedings of the ninth ACM conference on Hypertext and hypermedia : links, objects, time and space—structure in hypermedia systems', ACM Press, New York, NY, USA, pp. 1–10.

Clear, T. (2003), 'The waterfall is dead..: long live the waterfall!!', *SIGCSE Bull.* **35**(4), 13–14.

Cockburn, A. (2001), *Writing effective use cases*, Addison-Wesley, Boston.

Coda, F., Ghezzi, C., Vigna, G. and Garzotto, F. (1998), Towards a Software Engineering Approach to Web Site Development, *in* 'IWSSD '98: Proceedings of the 9th International Workshop on Software Specification and Design', IEEE Computer Society, p. 8.

Cohen, L. and Manion, L. (1994), *Research methods in education*, 4th edn, Routhledge, London; New York.

Conallen, J. (1999), 'Modeling Web application architectures with UML', *Commun. ACM* **42**(10), 63–70.

Conallen, J. (2000), *Building Web applications with UML*, Addison-Wesley.

Costagliola, G., Ferrucci, F. and Francese, R. (2002), 'Web engineering: Models and methodologies for the design of hypermedia applications', *Handbook of Software Engineering and Knowledge Engineering, volume 2, Emerging Technologies, pages 181 - 199. World Scientific.* **2**, 181–199.

Crosby, P. B. (1979), *Quality is free: the art of making quality certain*, McGraw-Hill.

de Koch, N. P. (2001), Software Engineering for Adaptive Hypermedia Systems - Reference Model, Modeling Techniques and Development Process, PhD thesis, Fakultät für Mathematik und Informatik der Ludwig-Maximilians - Universität München.

Deng, Y., Frankl, P. and Wang, J. (2004), 'Testing Web database applications', *SIGSOFT Softw. Eng. Notes* **29**(5), 1–10.

Dennis, A. and Valacich, J. (2001), 'Conducting Research in Information Systems', *Commun. AIS* **7**(5), 1–41.

DeRemer, F. and Kron, H. (1975), Programming-in-the large versus programming-in-the-small, *in* 'Proceedings of the international conference on Reliable software', pp. 114–121.

Do, H., Rothermel, G. and Elbaum, S. (2004), Infrastructure support for controlled experimentation with testing and regression testing techniques, Technical Report 04-60-01, Oregon State University.

Eagan, M. E. (1986), 'Advances in software inspections', *IEEE Trans. Softw. Eng.* **12**(7), 744–751.

Elbaum, S., Karre, S. and Rothermel, G. (2003), Improving Web application testing with user session data, *in* 'ICSE '03: Proceedings of the 25th International Conference on Software Engineering', IEEE Computer Society, Washington, DC, USA, pp. 49–59.

Elbaum, S., Rothermel, G., Karre, S. and II, M. F. (2005), 'Leveraging user-session data to support Web application testing', *Software Engineering, IEEE Transactions on* **31**(3), 187–202.

Epner, M. (2000), 'Poor Project Management Number-One Problem of Outsourced E-Projects'. Cutter Consortium, Research Briefs.

Esprit (1990), 'HYTEA "Technical Annex", Esprit Project P5252', *"HYTEA"* .

Evanco, W. M. and Yang, J. (1992), A framework for the efficient petri net simulation of real-time systems, *in* 'ANSS '92: Proceedings of the 25th annual symposium on Simulation', IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 202–213.

Favre, J.-M. (1997), Understanding-in-the-large, *in* 'Program Comprehension, 1997. IWPC '97. Proceedings., Fifth Iternational Workshop on', pp. 29–38.

Feldmann, A., Caceres, R., Douglis, F., Glass, G. and Rabinovich, M. (1999), Performance of Web proxy caching in heterogeneous bandwidth environments, *in* 'INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE', Vol. 1, pp. 107–116 vol.1.

Finkelstein, A. and Kramer, J. (2000), Software engineering: a roadmap, *in* 'ICSE '00: Proceedings of the Conference on The Future of Software Engineering', ACM Press, pp. 3–22.

Fishwick, P. A. and Zeigler, B. P. (1992), 'A multimodel methodology for qualitative model engineering', *ACM Trans. Model. Comput. Simul.* **2**(1), 52–81.

Fuentes, J. M., Quintana, V., Llorens, J., Génova, G. and Prieto-Díaz, R. (2003), 'Errors in the UML metamodel?', *SIGSOFT Softw. Eng. Notes* **28**(6), 3–3.

Gamma, E. and Beck, K. (2005), 'jUnit', `http://www.junit.org/`.

Garzotto, F., Paolini, P. and Schwabe, D. (1991*a*), Authoring-in-the-large: software engineering techniques for hypertext application design, *in* 'Software Specification and Design, 1991., Proceedings of the Sixth International Workshop on', pp. 193–201.

Garzotto, F., Paolini, P. and Schwabe, D. (1991*b*), HDM – a model for the design of hypertext applications, *in* 'HYPERTEXT '91: Proceedings of the third annual ACM conference on Hypertext', ACM Press, pp. 313–328.

Garzotto, F., Paolini, P. and Schwabe, D. (1993), 'HDM – a model-based approach to hypertext application design', *ACM Trans. Inf. Syst.* **11**(1), 1–26.

Gelperin, D. and Hetzel, B. (1988), 'The growth of software testing', *Commun. ACM* **31**(6), 687–695.

Germán, D., Cowan, D. D. and Alencar, D. P. (1998), 'A formal specification language for hypermedia applications', *1st International Workshop on Hypermedia Development* .

Germán, D. M. (2000), 'HadeZ, a Framework for Specification and Verification of Hypermedia Applications', `http://www.turingmachine.org/~dmg/research/papers/dmg_hadez2000.pdf`. PhD Thesis, Computer Science Department, University of Waterloo, Ontario, Canada, 2000.

Ginige, A. (2002*a*), Web engineering: managing the complexity of Web systems development, *in* 'SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering', ACM Press, New York, NY, USA, pp. 721–729.

Ginige, A. (2002*b*), Web engineering: managing the complexity of Web systems development, *in* 'Proceedings of the 14th international conference on Software engineering and knowledge engineering', ACM Press, pp. 721–729.

Ginige, A. and Murugesan, S. (2001), 'Web engineering: an introduction', *Multimedia, IEEE* **8**(1), 14–18.

Gittins, R. G. (1999), 'Qualitative Research: An Investigation into methods and concepts in qualitative research', `http://www.sesi.informatics.bangor.ac.uk/english/home/research/technical-reports/sesi-020/sesi-020.htm`. Technical paper, School of Informatics University of Wales, Bangor.

Glass, R. L. (2003), 'A mugwump's-eye view of Web work', *Commun. ACM* **46**(8), 21–23.

Glinz, M. (2000), Problems and Deficiencies of UML as a Requirements Specification Language, *in* 'IWSSD '00: Proceedings of the 10th International Workshop on Software Specification and Design', IEEE Computer Society, Washington, DC, USA, p. 11.

Gold, R. (2004), 'HttpUnit', `http://httpunit.sourceforge.net/`.

Goldberg, A. and Robson, D. (1983), *Smalltalk-80: the language and its implementation*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Goldberg, C. (2005), 'WebInject', `http://www.webinject.org/`.

Gomez, J., Cachero, C. and Pastor, O. (2001), 'Conceptual modeling of device-independent Web applications', *Multimedia, IEEE* **8**(2), 26–39.

Guimaraes, L. R. and Vilela, P. R. S. (2005), Comparing software development models using CDM, *in* 'SIGITE '05: Proceedings of the 6th conference on

Information technology education', ACM Press, New York, NY, USA, pp. 339–347.

Halasz, F. and Schwartz, M. (1994), 'The Dexter hypertext reference model', *Communications of the ACM* **37**(2), 30–39.

Hansen, S. and Deshpande, Y. (1997), A Skills Hierarchy for Web Information System Development, *in* 'Proc. Australasian Web Conf.(AusWeb 97)', Southern Cross Univ. Press, Lismore, pp. 114–121.

Harrold, M. J. (2000), Testing: a roadmap, *in* 'ICSE '00: Proceedings of the Conference on The Future of Software Engineering', ACM Press, pp. 61–72.

Henry, D. J., Stiff, J. C. and Shirar, A. J. (2003), Assessing and improving testing of real-time software using simulation, *in* 'ANSS '03: Proceedings of the 36th annual symposium on Simulation', IEEE Computer Society, Washington, DC, USA, p. 266.

Hieatt, E. and Mee, R. (2002), 'Going faster: testing the Web application', *Software, IEEE* **19**(2), 60–65.

Higgs, J. (1998), *Writing qualitative research*, Hampden Press in conjunction with the Centre for Professional Education Advancement, Sydney.

Holck, J. (2003), 4 Perspectives on Web Information Systems, *in* 'HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 8', IEEE Computer Society, Washington, DC, USA, p. 265.2.

Holloway, I. (1997), *Basic Concepts for Qualitative Research*, Blackwell Science.

Hower, R. (2005), 'Software QA/Test Resource Center', `http://www.softwareqatest.com/qatweb1.html`.

IBM (2005), 'Rational Rose', `http://www.ibm.com`.

IEEE (2001), 'IEEE standard Verilog hardware description language', *IEEE Std 1364-2001* pp. 0–856.

IEEE guide to software requirements specifications (1984).

IEEE standard for software verification and validation – 1012-1998 (1998).

IEEE standard glossary of modeling and simulation terminology – 610.3-1989 (1989).

IEEE standard glossary of software engineering terminology – 610.12-1990 (1990).

IEEE Standard VHDL Language Reference Manual – 1076 (2002).

Ingalls, R. G. (2002), Introduction to simulation, *in* 'WSC '02: Proceedings of the 34th conference on Winter simulation', Winter Simulation Conference, pp. 7–16.

Isakowitz, T., Kamis, A. and Koufaris, M. (1998), 'The Extended RMM Methodology for Web Publishing', *Working Paper, Center for Research on Information Systems* **18**.

Isakowitz, T., Stohr, E. A. and Balasubramanian, P. (1995), 'RMM: a methodology for structured hypermedia design', *Commun. ACM* **38**(8), 34–44.

Iyengar, A., MacNair, E. and Nguyen, T. (1997), An analysis of Web server performance, *in* 'Global Telecommunications Conference, 1997. GLOBECOM '97., IEEE', Vol. 3, pp. 1943–1947 vol.3.

Jarvenpaa, S. L. (1988), 'The importance of laboratory experimentation in IS research (technical correspondence)', *Commun. ACM* **31**(12), 1502–1504.

Johnson, J. and Henderson, A. (2002), 'Conceptual models: begin by designing what to design', *interactions* **9**(1), 25–32.

Kappel, G., Michlmayr, E., Pröll, B., Reich, S. and Retschitzegger, W. (2004), Web Engineering - Old Wine in New Bottles?, *in* 'ICWE', pp. 6–12.

Kelly, D. and Shepard, T. (2002), Qualitative observations from software code inspection experiments, *in* 'CASCON '02: Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research', IBM Press, p. 5.

Kotonya, G. and Sommerville, I. (1998), *Requirements engineering: processes and techniques*, John Wiley & Sons Ltd.

Kung, D. (2004), An agent-based framework for testing Web applications, *in* 'Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International', Vol. 2, pp. 174–177 vol.2.

Kung, D. C., Liu, C.-H. and Hsia, P. (2000*a*), An object-oriented web test model for testing web applications, *in* 'COMPSAC '00: 24th International Computer Software and Applications Conference', IEEE Computer Society, Washington, DC, USA, pp. 537–542.

Kung, D., Liu, C.-H. and Hsia, P. (2000*b*), An Object-Oriented Web test model for testing Web applications, *in* 'Quality Software, 2000. Proceedings. First Asia-Pacific Conference on', pp. 111–120.

Laakso, S. and Laakso, K.-P. (2003), 'Ensuring quality of the user interface with the guide project model', `http://citeseer.ifi.unizh.ch/685872.html`.

Lang, M. (2002), Hypermedia systems development: Do we really need new methods?, *in* 'Informing Science + IT Education Conference', InformingScience.org, pp. 883–891.

Lang, M. and Fitzgerald, B. (2005), 'Hypermedia systems development practices: a survey', *Software, IEEE* **22**(2), 68–75.

Lange, D. (1993), Enhanced Relationships in Object-Oriented Database Modeling, *in* 'Proceedings of InfoScience'93'.

Lange, D. (1994), An Object-Oriented design method for hypermedia information systems, *in* 'Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences, 1994. Vol.III: Information Systems: Decision Support and Knowledge-Based Systems', pp. 366–375.

Laplante, P. A. and Neill, C. J. (2004), 'Opinion: The Demise of the Waterfall Model Is Imminent', *Queue* **1**(10), 10–15.

Lattice Semiconductor Corporation (2006), 'ispLever', `http://www.latticesemi.com//`.

Lee, H., Kim, J., Kim, Y. G. and Cho, S. H. (1999), 'A view-based hypermedia design methodology', *J. Database Manage.* **10**(2), 3–13.

Lee, H., Lee, C. and Yoo, C. (1999), 'A scenario-based object-oriented hypermedia design methodology', *Inf. Manage.* **36**(3), 121–138.

Liu, C.-H., Kung, D., Hsia, P. and Hsu, C.-T. (2000), Structural testing of Web applications, *in* 'Software Reliability Engineering, 2000. ISSRE 2000. Proceedings. 11th International Symposium on', pp. 84–96.

Lowe, D. B., Bucknell, A. J. and Webby, R. G. (1999), Improving hypermedia development: a reference model-based process assessment method, *in* 'HYPERTEXT '99: Proceedings of the tenth ACM Conference on Hypertext and hypermedia: returning to our diverse roots', ACM Press, pp. 139–146.

Lowe, D. and Hall, W. (1999), *Hypermedia & the Web: an engineering approach*, John Wiley & Sons Ltd.

Lucca, G. D., Fasolino, A., Faralli, F. and Carlini, U. D. (2002), Testing Web applications, *in* 'Software Maintenance, 2002. Proceedings. International Conference on', pp. 310–319.

MacIntosh, A. and Strigel, W. (2000), The Living Creature - Testing Web Applications, *in* 'Proceedings of the 13th International Internet & Software Quality Week Conferences', pp. 1–16.

Maddison, R. (1983), *Information Systems Methodologies*, Wiley Heyden, Chichester.

Maginot, S. (1992), Evaluation criteria of HDLs: VHDL compared to Verilog, UDL/I & M, *in* 'EURO-DAC '92: Proceedings of the conference on European design automation', IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 746–751.

Malan, R. and Bredemeyer, D. (2001), 'Functional Requirements and Use Cases', `http://citeseer.ist.psu.edu/493110.html`.

Marcos, E. (2005), 'Software engineering research versus software development', *SIGSOFT Softw. Eng. Notes* **30**(4), 1–7.

McLean, C. and Shao, G. (2003), Manufacturing case studies: generic case studies for manufacturing simulation applications, *in* 'WSC '03: Proceedings of the 35th conference on Winter simulation', Winter Simulation Conference, pp. 1217–1224.

Mellor, S. J. and Balcer, M. J. (2002), *Executable UML : a foundation for model-driven architecture*, Addison-Wesley, Boston ; San Francisco ; New York.

Moriguchi, S. (1996), *Software Excellence - A Total Quality Management Guide*, Productivity Press.

Murata, T. (1989), Petri nets: Properties, analysis and applications, *in* 'Proceedings of the IEEE', IEEE, pp. 541–580.

Murugesan, S., Deshpande, Y., Hansen, S. and Ginige, A. (1999), Web Engineering: A New Discipline for Development of Web-based Systems, *in* 'Proceedings of the First ICSE Workshop on Web Engineering, International Conference on Software Engineering'.

Murugesan, S., Deshpande, Y., Hansen, S. and Ginige, A. (2001), Web Engineering: A New Discipline for Development of Web-Based Systems, *in* 'Web Engineering', pp. 3–13.

Myers, M. D. (2003), 'Qualitative Research in Information Systems', `www.qual.auckland.ac.nz`. MIS Quarterly (21:2), June 1997, pp. 241-242. MISQ Discovery, archival version, June 1997.

Nanard, J. and Nanard, M. (1995), 'Hypertext design environments and the hypertext design process', *Commun. ACM* **38**(8), 49–56.

Nance, R. E. (1983), A tutorial view of simulation model development, *in* 'WSC '83: Proceedings of the 15th conference on Winter simulation', IEEE Press, Piscataway, NJ, USA, pp. 325–332.

Nilawar, M. (2003), *A UML-Based Approach for Testing Web Applications - MSc Thesis*, University of Nevada, Reno.

Nissanke, N. (1997), *Realtime systems*, Prentice Hall.

Object Management Group (2005), 'OMG Unified Modeling Language Specification', `http://www.uml.org/`.

OMG (2002), 'Omg unified modeling language specification (action semantics)', `http://www.omg.org/docs/ptc/02-01-09.pdf`.

OMG (2004), 'Unified Modeling Language Specification – Version 1.4.2', `http://www.omg.org/docs/formal/04-07-02.pdf`.

OMG (2005), 'OMG – Object Management Group', `http://www.omg.org/`.

Pasahow, E. J. (1973), Testing real-time systems with simulation, *in* 'ANSS '73: Proceedings of the 1st symposium on Simulation of computer systems', IEEE Press, Piscataway, NJ, USA, pp. 40–45.

Pastor, O., Insfran, E., Pelechano, V., Romero, J. and Merseguer, J. (1997), OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods, *in* 'CAiSE '97: Proceedings of the 9th International Conference on Advanced Information Systems Engineering', Springer-Verlag, London, UK, pp. 145–158.

Peixoto, P. (2005), Simulating Web Applications Design Models, *in* 'ICWE', pp. 627–629.

Peixoto, P., Fung, K. and Lowe, D. (2004*a*), A Framework for the Simulation of Web Applications, Technical Report UTS-Eng-TR-04-001, University of Technology, Sydney.

Peixoto, P., Fung, K. and Lowe, D. (2004*b*), A Framework for the Simulation of Web Applications, *in* 'ICWE', pp. 261–265.

Perrone, V., Bolchini, D. and Paolini, P. (2005), A stakeholders centered approach for conceptual modeling of communication-intensive applications, *in* 'SIGDOC '05: Proceedings of the 23rd annual international conference on Design of communication', ACM Press, New York, NY, USA, pp. 25–33.

Perry, D. E., Porter, A. A. and Votta, L. G. (2000), Empirical studies of software engineering: a roadmap, *in* 'ICSE '00: Proceedings of the Conference on The Future of Software Engineering', ACM Press, New York, NY, USA, pp. 345–355.

Pfleeger, S. L. (2001), *Software Engineering - Theory and Practice - 2nd Ed.*, Prentice-Hall.

Pressman, R. (1998), 'Can Internet-Based Applications Be Engineered?', *Software, IEEE* **15**(5), 104–110.

Pressman, R. S. (2000), 'What a Tangled Web We Weave', *IEEE SOFTWARE* **17**(1), 18–21.

Reasoning (2003), 'Automated Software Inspection – A New Approach to Increased Software Quality and Productivity', `http://www.reasoning.com/pdf/ASI.pdf`.

Reifer, D. (2000), 'Web development: estimating quick-to-market software', *Software, IEEE* **17**(6), 57–64.

Ricca, F. and Tonella, P. (2001), Analysis and testing of Web applications, *in* 'ICSE '01: Proceedings of the 23rd International Conference on Software Engineering', IEEE Computer Society, Washington, DC, USA, pp. 25–34.

Ricca, F. and Tonella, P. (2005), Web Testing: a Roadmap for the Empirical Research, *in* 'Web Site Evolution, 2005. (WSE 2005). Seventh IEEE International Symposium on', pp. 63–70.

Robinson, S. (2003), *Simulation: the practice of model development and use*, Wiley, Hoboken, NJ.

Rossi, G., Schwabe, D. and Guimaraes, R. (2001), Designing personalized Web applications, *in* 'WWW '01: Proceedings of the tenth international conference on World Wide Web', ACM Press, pp. 275–284.

Rosson, M., Ballin, J. and Rode, J. (2005), Who, What, and How: A Survey of Informal and Professional Web Developers, *in* 'Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on', pp. 199–206.

Rosu, D., Iyengar, A. and Dias, D. (2000), Hint-based acceleration of Web proxy caches, *in* 'Performance, Computing, and Communications Conference, 2000. IPCCC '00. Conference Proceeding of the IEEE International', pp. 30–37.

Royce, W. W. (1970), Managing the development of large software systems: concepts and techniques, *in* 'Proc. IEEE WESTCON', IEEE Computer Society Press, pp. 1–9.

Rumbaugh, J. (1987), Relations as semantic constructs in an Object-Oriented language, *in* 'Conference proceedings on Object-oriented programming systems, languages and applications', ACM Press, pp. 466–481.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1991), *Object-Oriented Modeling and Design*, Prentice-Hall, New York.

Salkind, N. J. (2003), *Exploring research - 5th Ed.*, Prentice Hall.

Sargent, R. (2003), Verification and validation of simulation models, *in* 'Simulation Conference, 2003. Proceedings of the 2003 Winter', Vol. 1, pp. 27–48 Vol.1.

Sargent, R. G. (1992), Validation and verification of simulation models, *in* 'WSC '92: Proceedings of the 24th conference on Winter simulation', ACM Press, New York, NY, USA, pp. 104–114.

Schwabe, D., de Almeida Pontes, R. and Moura, I. (1999), 'OOHDM-Web: an environment for implementation of hypermedia applications in the WWW', *SIGWEB Newsl.* **8**(2), 18–34.

Schwabe, D. and Rossi, G. (1995*a*), Building hypermedia applications as navigational views of information models, *in* 'System Sciences, 1995. Vol. III. Proceedings of the Twenty-Eighth Hawaii International Conference on', Vol. 3, pp. 231–240 vol.3.

Schwabe, D. and Rossi, G. (1995*b*), 'The Object-Oriented hypermedia design model', *Communications of the ACM* **38**(8), 45–46.

Schwabe, D., Rossi, G. and Barbosa, S. D. J. (1996), Systematic hypermedia application design with OOHDM, *in* 'HYPERTEXT '96: Proceedings of the the seventh ACM conference on Hypertext', ACM Press, pp. 116–128.

Selby, R. W., Basili, V. R. and Baker, F. T. (1987), 'Cleanroom software development: an empirical evaluation', *IEEE Trans. Softw. Eng.* **13**(9), 1027–1037.

Shannon, R. (1998), Introduction to the art and science of simulation, *in* 'Simulation Conference Proceedings, 1998. Winter', Vol. 1, pp. 7–14 vol.1.

Shannon, R. E. (1992), Introduction to simulation, *in* 'WSC '92: Proceedings of the 24th conference on Winter simulation', ACM Press, New York, NY, USA, pp. 65–73.

Shortle, J. F., Gross, D. and Mark, B. L. (2003), Simulation of large networks: efficient simulation of the national airspace system, *in* 'WSC '03: Proceedings of the 35th conference on Winter simulation', Winter Simulation Conference, pp. 441–448.

Shull, F., Carver, J. and Travassos, G. H. (2001), An empirical methodology for introducing software processes, *in* 'ESEC/FSE-9: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering', ACM Press, New York, NY, USA, pp. 288–296.

Smith, D. J. (1996), VHDL & Verilog compared & contrasted-plus modeled example written in VHDL, Verilog and C, *in* 'DAC '96: Proceedings of the 33rd annual conference on Design automation', ACM Press, New York, NY, USA, pp. 771–776.

Soft., J. (2005), 'JStudio', `http://www.jstudio.de/`.

Sommerville, I. (2004), *Software Engineering - 7th Ed.*, Addison-Wesley.

Sony (2005), 'Sony', `http://www.sonymusicstore.com/`.

Sunye, G., Pennaneac'h, F., Ho, W.-M., Guennec, A. L. and Jquel, J.-M. (2001), Using UML action semantics for executable modeling and beyond, *in* 'Conference on Advanced Information Systems Engineering', pp. 433–447.

Tonella, P. and Ricca, F. (2004), A 2-layer model for the white-box testing of Web applications, *in* 'Web Site Evolution, 2004. WSE 2004. Proceedings. Sixth IEEE International Workshop on', pp. 11–19.

Trochim, W. and Land, D. (2004), 'Designing Designs for Research', `http://trochim.human.cornell.edu/kb/index.htm`. The Research Methods Knowledge Base, 2nd Edition.

Varro, D. (2003), 'Towards automated formal verification of visual modeling languages by model checking', `http://citeseer.ist.psu.edu/article/varro03towards.html`.

Virgin (2005), 'Virgin', `https://www.virginmegastores.co.uk/`.

Wade, R. M. and Tingling, P. (2005), 'A new twist on an old method: a guide to the applicability and use of Web experiments in information systems research', *SIGMIS Database* **36**(3), 69–88.

Walliman, N. (2005), *Your research project: a step-by-step guide for the first-time researcher*, 2nd edn, SAGE Publications, London.

WebML (2005), 'WebML', `http://www.webml.org`. WebML.

WebRatio (2001), 'WebML User Guide - version 3.0', `http://www.webml.org`.

WebRatio (2005), 'WebRatio', `http://www.webratio.com`. WebRatio.

Wegner, P. (1990), 'Concepts and paradigms of Object-Oriented programming', *SIGPLAN OOPS Mess.* **1**(1), 7–87.

Wells, L., Christensen, S., Kristensen, L. and Mortensen, K. (2001), Simulation based performance analysis of Web servers, *in* 'Petri Nets and Performance Models, 2001. Proceedings. 9th International Workshop on', pp. 59–68.

Whitner, R. B. and Balci, O. (1989), Guidelines for selecting and using simulation model verification techniques, *in* 'WSC '89: Proceedings of the 21st conference on Winter simulation', ACM Press, New York, NY, USA, pp. 559–568.

Withers, B. D., Pritsker, A. A. B. and Withers, D. H. (1993), A structured definition of the modeling process, *in* 'WSC '93: Proceedings of the 25th conference on Winter simulation', ACM Press, New York, NY, USA, pp. 1109–1117.

Wixon, D. (1995), 'Qualitative research methods in design and development', *interactions* **2**(4), 19–26.

Xilinx, Inc. (2006), 'Xilinx', `http://www.xilinx.com//`.

Xu, L., Xu, B. and Jiang, J. (2005), 'Testing Web applications focusing on their specialties', *SIGSOFT Softw. Eng. Notes* **30**(1), 10.

Yalamanchili, S. (2001), *Introductory VHDL - From Simulation to Synthesis*, Prentice-Hall, Inc. - Xilinx Design Series.

Young, R. R. (2004), *The requirements engineering handbook*, Artech House, Boston, MA.

Zannier, C. and Maurer, F. (2005), A qualitative empirical evaluation of design decisions, *in* 'HSSE '05: Proceedings of the 2005 workshop on Human and social factors of software engineering', ACM Press, New York, NY, USA, pp. 1–7.

Zhu, H., Hall, P. A. V. and May, J. H. R. (1997), 'Software unit test coverage and adequacy', *ACM Comput. Surv.* **29**(4), 366–427.