UNIVERSITY OF SOUTHAMPTON

Formal Patterns for Web-based Systems Design

by

Abdolbaghi Rezazadeh

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

July 2007

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Abdolbaghi Rezazadeh

The ubiquitous and simple interface of Web browsers has opened the door for the development of a new class of distributed applications which they have been known as Web applications. As more and more systems become Web-enabled we become increasingly dependent on the Web applications. Therefore, reliability of such systems is a very crucial factor for successful operation of many modern organisations and institutes.

In the first part of this thesis we review how Web systems have evolved from simple static pages, in their early days, to their current situation as distributed applications with sophisticated functionalities. We also find out how the design methods have evolved to align with the rapid changes both in the new emerging technologies and growing functionalities. Although design approaches for Web applications have improved during the last decade we conclude that dependability should be given more consideration. In Chapter 2 we explain how this could be achieved through the application of formal methods. Therefore, we have provided an overview of dependability and formal methods in this chapter.

In the second part of this research we follow a practical approach to the formal modelling of Web Applications. Accordingly, in Chapter 3 we have developed a series of formal models for an integrated holiday booking system. Our main objectives are to gain some common knowledge of the domain and to identify some key areas and features with regard to our formal modelling approach. Formal modelling of large Web applications could be a very complex process. In Chapter 4 we have introduced the idea of formal patterns for specification and refinement to accelerate the modelling process and to help alleviate the burden of formal modelling.

In a further attempt to tackle the complexity of the formal modelling of Web applications, we have introduced the idea of specification partitioning in Chapter 5. Specification partitioning is closely related to the notion of composition. In this chapter we have extended some CSP-like composition techniques to build the system specification from subsystems or parts. The summary of our research, related findings and some suggestions for the future work are presented in Chapter 6.

Contents

List of Figures

Acknowledgements

My first and foremost thanks go to my supervisor Professor Michael Butler. This thesis would not have been possible without his inspiration, constructive criticism and experienced guidance. I thank Dr. Mike Poppleton for his comments and useful advice in some pars of this thesis. Many thanks to my friends and colleagues in the Dependable Systems and Software Engineering Group for many stimulating and pleasant discussions. Special thanks to Andrew Edmunds who took the time to read some part of the thesis and provide me with invaluable comments.

My most profound gratitude goes firstly to my parents and secondly to my family especially my wife. My parents deprived themselves in many situations to provide me with the means to make progress. My wife has been an endless source of inspiration and support for me to pursue my doctorate; to whom this thesis is dedicated.

Preface

This thesis investigates the formal modelling of Web-based applications using the B-Method. We begin with studying the current approaches and modelling practices for Web applications. Following this, based on our findings and general knowledge of formal modelling we apply the B formal method to Web applications modelling. The preliminary goal of this research is to find out:

- How the B-Method could be adapted for the modelling of distributed Web applications?
- If the current constructs of the B-Method are not adequate for the modelling of Web applications, how these could be extended based on the properties of such systems.

Background

Since its inception, the World Wide Web has revolutionised many aspects of our life and will continue to influence our society for years to come. Many technical and non-technical aspects of our life are changing everyday as we become more dependent on the Web. The Web browser's ubiquitous and simple interface has opened the door for the development of a new class of distributed applications which have been called Web-based or simply Web Applications.

In the early days of Web, these systems were fairly simple and were constructed form static HTML pages linked together. They were developed to provide easy access to information on servers that could be situated anywhere across the globe. With the introduction of new technologies such as Java developers suddenly realised the potential of the Web for developing large-scale distributed Web applications. Companies started developing Web applications with which they promised to deliver reliable services to their customers and provide outstanding value to their stakeholders. As a result of this the days of the Web as a medium for just documents sharing are over and the new era of the Web as a vehicle for e-commerce and other applications has begun.

In the early days of Web application development, too often, no design and modelling principles were applied to Web application development process. Most of developers were using ad hoc approaches which resulted in patchy and unreliable applications. System developer were justifying their reluctance to adopt concrete approaches for reasons such as speed of development, very rapid changes of under-laying technologies and the error-prone nature of early Web systems.

Eventually, when Web applications started to establish themselves as a widespread form of distributed systems, researchers recognised the need to apply well-structured and methodological approaches to the modelling and development process of Web applications. As Web applications evolved and their complexity increased, developers realised that their ad hoc approaches could not cope with the scope of the systems that should be developed.

Thesis Organisation

In the next chapter we review how Web applications and related modelling approaches were evolved. It provides a classification of Web systems and an overview of how modelling approaches succeeded to take into account different aspects of such systems.

Many different modelling and development approaches have emerged during the last few years. Some of these are well-established, but still more work is needed in this area. The reliability and dependability of Web applications has to be improved. In Chapter 2 a short overview of these concepts has been provided. In addition, we learn how these goals could be achieved through the application of formal methods to the development process.

Chapter 1 and 2 covers the domain study of this thesis. In Chapter 3 we introduce an example case study and by modelling it in B we try to gain a deeper understanding of formal modelling of Web-based applications. A set of interesting and challenging features have been identified and some formal solutions have been provided in this chapter. The models have been developed in Click-n-Prove environment which provided tool support for B-Method.

Based on the experiences that we have gained from the case study in the previous chapter, in Chapter 4 we have generalised some key features in the form of a few patterns. The basic assumption is that these general patterns should be applicable to a wide range of Web applications and it should promote re-usability and increases the productivity of system developers.

Specification partitioning and composition mechanisms are another promising aspect in the development of complex Web applications that are investigated in Chapter 5.

We argue that for a whole range of reasons, such as productivity, team-based development and the separation of concerns, the formal development approach needs to support specification partitioning. In addition, in this chapter we investigate how the proposed approach to specification partitioning could be extended to the high level architectural modelling. Composition mechanisms are an indispensable part of any specification partitioning approach for building a complete solution for complex systems. Different composition mechanisms for both specification and architectural level modelling have been introduced in this chapter.

Finally, Chapter 6 gives a conclusion, summarising the main results of this work and indicating several directions for future research.

Contribution of the thesis

In this thesis we have investigated the applicability of the B Formal Method for the modelling of Web applications. This in fact has two dimensions. In the first place our initial goal was to assess the usability and applicability of current methods and tools in developing this kind of distributed systems. Our second important goal was to find out how the current methods and tool support could be extended or improved in the light of case studies, which have been modelled. As we have taken a practical approach we initiated our work by a real case study. During formal modelling of the travel case study in Chapter 3 we have identified and modelled several key issues such as:

- Stepwise introduction of requirements based on the user view of the system
- Distributed database specification and refinement
- Complex data structure specification and their stepwise refinement
- Specification of communication links and a multiple-level refined model for them

The formal pattern for system modelling is a new concept, which we have introduced in Chapter 4. A formal pattern includes specification, refinement and their associated proofs. Developers can construct models by composing different patterns. In this chapter we have argued that a few patterns can represent a wide variety of requirements in the Web application domain.

In Chapter 5 we have developed the idea of specification partitioning and composition mechanisms for both specification and architectural level modelling. We have defined a number of different composition mechanisms that can simplify the formal modelling process of multi-layered web applications.

Chapter 1

An Overview of Web-Based Applications and Systems

1.1 Introduction

Distributed applications have gained a lot of attention in the past ten years. Among these applications, the most common and known ones are so called Web-based applications. Web-based applications have had enormous success during last few years. Today many applications are developed for the Web, in such different areas as banking and finance, e-commerce, education, government and entertainment. Legacy information and database systems are being migrated to Web environments, in order to deploy their functionality on the Web. Electronic commerce through the Internet is rapidly growing, cutting across national boundaries. Many people are affected by the Web. At the same time the complexity and sophistication of Web applications grows. Web applications are preferred over traditional applications for the following reasons:

- Web applications are more accessible: The HTTP protocol used in Web applications is a standard protocol that can travel across corporate firewalls. Thus, applications are accessible to many users ranging from home users to corporate users. In addition, a Web application does not require a specialized client. A Web browser, which nowadays comes packaged with almost all operating systems, is used as the client. Users do not need to install, configure or maintain client software. Also, the application is accessible on any platform as long as a Web browser exists for the platform.

- Web applications have lower maintenance and deployment costs: Since the browser is used as the client software for Web applications, there are no costs associated with development of the client's software. Maintaining the Web application requires only modifying the code that resides on the server. This reduces the cost of upgrade and deployment of Web applications compared to traditional client/server applications.

Now that many of us rely on Web-based systems and applications, they need to be reliable and perform well. To build these systems and applications, Web developers need a sound methodology, a disciplined and repeatable process, better development tools, and a set of good guidelines. The emerging field of Web engineering fulfils these needs. It uses scientific, engineering, and management principles and systematic approaches to successfully develop, deploy, and maintain high-quality Web systems and applications. It aims to establish a sound methodology for Web-based system development, minimize risks, and enhance Web site maintainability and quality.

1.2 What Are Web Applications?

A Web application is an application, designed to be executed in a Web-based environment. More precisely a Web application is a mix of programs that dynamically generate hyper-documents in response to some input from the user. A Web-based application is a distributed system that consists of a number of client and servers distributed over the Internet interacting with each other using the HTTP protocol. The Web clients most likely communicate with the Web server through a Web browser like Microsoft Internet Explorer or Netscape Navigator. An alternative is to use an application client where an HTTP agent acts as client on behalf of the user. The client agent can accept HTTP responses and is able to interact with the Web application by sending HTTP requests. The functions performed can range from relatively simple tasks like reading content or searching a local directory for a file or reference, to highly sophisticated applications that perform real-time sales and inventory management across multiple business partners. The technology behind Web applications has developed at a great speed. Web applications can range form fairly simple applications built and run on a single Web server (that may be connected to a simple database on the same host), to modern enterprise Web applications typically run on distributed application servers, (connecting to multiple data sources through complex business logic tiers). They can consist of tens or hundreds of servers each performing specific tasks or functions.

1.2.1 Taxonomy of Web Applications

Web applications are not limited to one type of application. They can range from simple static Web pages (such as a personal Web site, a home page) to sophisticated e-commerce applications (such as Amazon.com, eBay.com). Figure 1.1 shows the different categories of Web applications grouped according to their data and control complexity [62]:

1. Brochure: Brochure Web applications are the first generation of Web applications. They tend not to have much programming logic in them; rather they are composed of simple static Web pages. Their developers are referred to as content developers, as they are more concerned with the layout of graphics and text on a Web page and the content is very static and graphics intensive. Examples of Brochure applications include: the personal Web page of a person which simply contains their resume and personal information, and Web sites that contain technical documents (brochures) about a company's product. Simple editors or specialized HTML editors are used to develop Brochure sites. The number of pages is rather small as it is manually edited and maintained. These sites are more similar to desktop publishing than to traditional software systems. These sites are the easiest to visualize. They are not of interest to our visualization effort, because we are more concerned with the control and data flow across the different components; whereas these sites are rather static with no control or data flow.

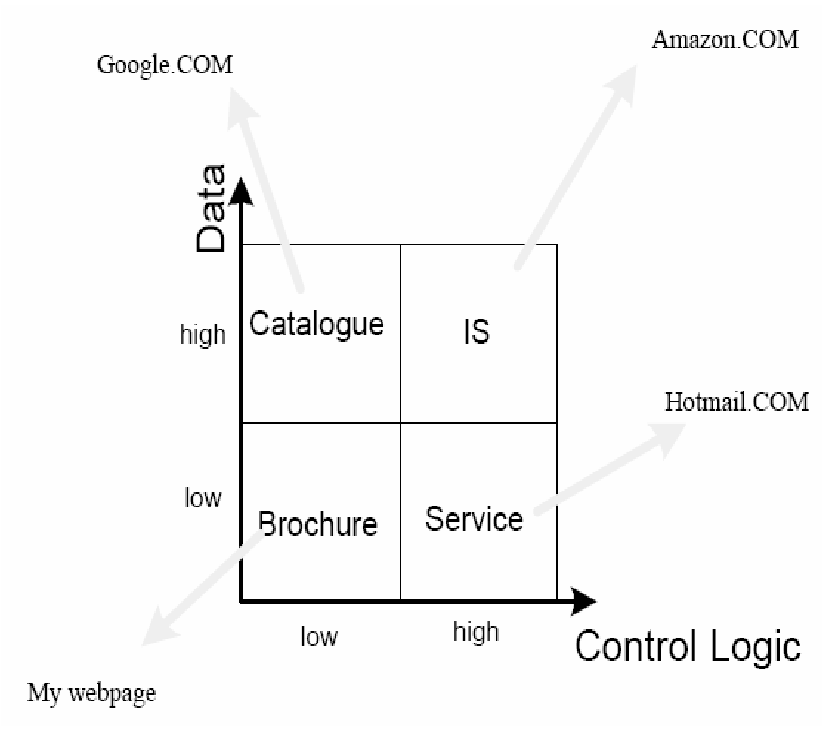


FIGURE 1.1: Taxonomy of Web Applications

2. Service oriented applications: These sites are dedicated to providing a service to Web users, such as free email service or online word-processing systems. In these applications, the layout of the data is a secondary concern. Instead, the developer is concerned with implementing the logic needed to provide the services online. For example, the developer of an online email service is more concerned with the different functional steps needed to store and retrieve email messages. The layout of the mail message displayed is of secondary Interest. During maintenance, the

developers need a good understanding of the control flow between the different components of the applications.

3. Data intensive applications: Theses are sites that provide an interface to browse and query large quantities of data, such as online library catalogues. The main emphasis in these applications is on the data, with minimal amount of logic or control involved. Large commercial examples of these applications are search engine sites such as Google.com, and online news sites such as CNN.com. A search engine provides an interface to query a large database that indexes Web pages. Data Intensive applications are closely tied to their database. A clear picture of the data flow is vital during maintenance.

4. Information system applications: These applications are a mix of Service Oriented and Data Intensive applications. An example of these sites is an online library system where you can; in addition to browsing books, borrow reserve and recall books. Most large electronic commerce sites are in this category such as Amazon.com. The developers of Information System applications are concerned with the data flow (for browsing and retrieving books) and control flow (for the different phases involved with ordering and shipping a book). We observe that developers need a good understanding of the data and control flow in their application as is needed in traditional applications. In addition, Web applications have more dependencies and interesting relations such as the navigation links between the different pages of the Web application. In [62] they pointed out that many Web application development tools are implementation oriented, they emphasize fast, one-time release with no continuity and process enforcement. This emphasis on implementation productivity with no concern on the maintenance and evolution of Web applications is attributed to the fast pace of their development and the immaturity of the Web applications domain.

1.3 Web Application Architecture Views

Web applications are distributed applications which are rely on different Web technologies as their infrastructure. They use Web browsers as their clients, HTTP protocol to communicate between clients and servers, and the HTML or XML based standards to express the content transmitted between servers and clients. They are complex systems and use many different technologies. Using a single architecture view is insufficient and not expressive enough to model such systems. Therefore a number of different views have been proposed by Kruchten [74]: Logical, Physical, and Development views. Each view captures specific design decisions and all views must be examined together to gain a good understanding of the whole application. Some additional views like the security architecture of the application could be taken into account for Web-based applications.

As Web applications are composed of many different components and they use the Internet, a public network, they are more vulnerable to attacks. The security of Web applications merits its own view.

Clear separation of concerns and modelling different aspects of the Web-based application is another reason for using multi-view models. This issue is widely regarded as a key attribute for obtaining high quality design by many Web design methodologies, such as HDM2000 [22], WebML [36] and OOHDM [121].

1.3.1 Logical View

Using logical layers in which different concerns and aspects are taken into account is recommended by the above methodologies. Layering is an engineering principle that helps to reduce the complexity of systems. The Logical view provides a high level abstraction of the system based on the domain of the problem. Usually diagrams are used to represent the different component in the system and the interactions between them. A Three-layer Architecture as depicted in Figure 1.2 is proposed by [30] for information systems and Web-based applications.

These layers are:

Presentation Layer: The Presentation Layer is responsible for presenting the data to the end user or system. The Web server serves up data and the Web browser renders it into a readable form, which the user can then interpret. It also allows the user to interact by sending back parameters, which the Web server can pass along to the application. This *Presentation Layer* includes Web servers like Apache and Microsoft's Internet Information Server and Web browsers like Internet Explorer and Netscape Navigator. It may also include application components that create the page layout.

Application Layer: The *Application Layer* is the *engine* of a Web application. It performs the business logic; processing user input, making decisions, obtaining more data and presenting data to the Presentation Layer to send back to the user. The Application Layer may based on technology like CGI's, J2EE, or .NET services deployed in products like IBM WebSphere and BEA WebLogic.

Data Layer: The Data Layer is used to store things needed by the application and acts as a repository for both temporary and permanent data. It is the bank vault of a Web application. Many modern systems now store data in XML format for interoperability with other system and sources.

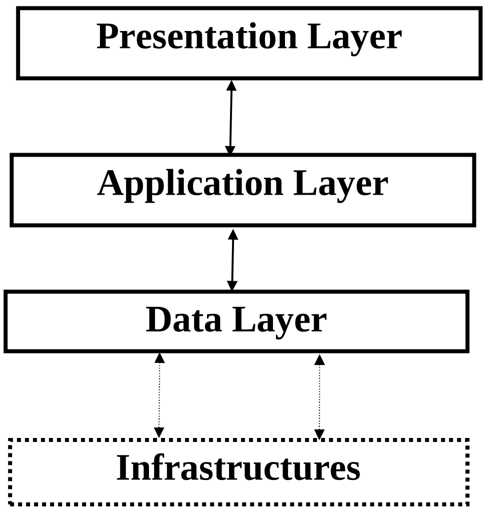


FIGURE 1.2: High Level Logical View of a Web Application

The three-tiered architecture provides a good separation of concerns for large Web applications, but some applications do not require a separation between the *Application Layer* and database tiers. For example, some Service oriented applications do not have a clear separation between the Database and the Business Logic tier, both tiers are combined together. In this situation, a 2-tiered architecture is used. For both architecture styles, an Infrastructure layer exists to provide support for the basic functionality needed by the different tiers, such as access to the local file system.

1.3.2 Physical View

The Physical view presents the mappings of the components in the Development view to the components in the environment. The environment of a Web application is composed of many components that are inter-linked together to implement its functionality. Web applications have a rich environment, which contains the following components:

- Web browsers (used by the clients)
- Web servers
- Web pages
- Application servers
- Databases
- Distributed objects such EJB and legacy systems.
- Multimedia Web objects such as Images, Videos, and etc.

The structure and interconnection between these parts is depicted in Figure 1.3.

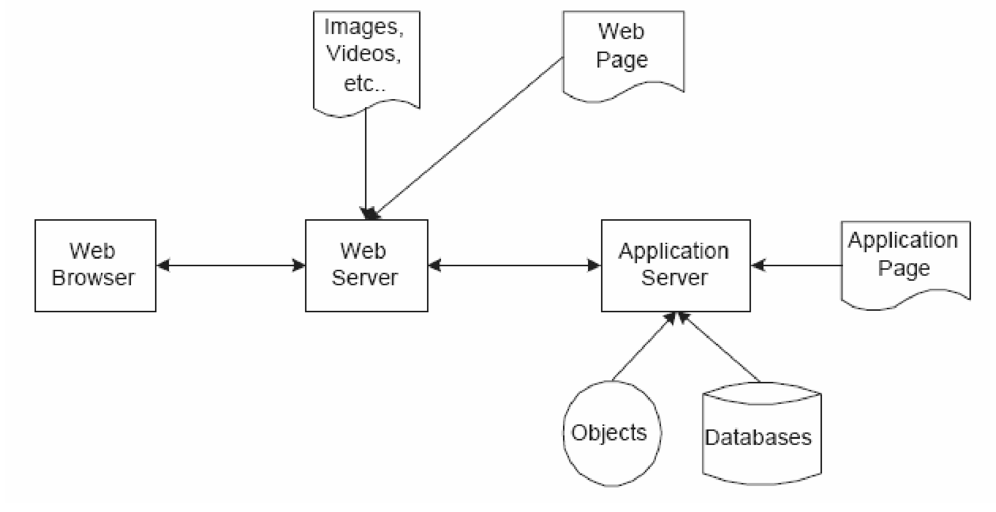


FIGURE 1.3: Physical View the Web Application

1.3.3 Development View

The Development view focuses on the mapping of the Logical view conceptual components to the actual implementation artifacts. It presents the actual software module organization in the development environment, such as the source code files, or the directory structure. Web applications are developed using many languages and technologies, compared to traditional applications, which are usually developed in one language. The Development view for Web applications must highlight the additional details such as:

- The Link structure of the application pages
- User's session management techniques
- Application page generation technology

1.4 Web Engineering

In 1998 the term Web engineering had been coined and efforts have been made to establish a new discipline to cover the life cycle of Web applications. Contrary to the general perception, Web engineering is not a clone of software engineering although both involve programming and software development [56]. While Web engineering adopts many software engineering principles, it incorporates new approaches and guidelines to meet the unique requirements of Web-based systems [56]. Web development is a mixture between print publishing and software development, between marketing and computing, between internal communications and external relations, and between art and technology [111]. Building complex Web applications calls for knowledge and expertise from many different disciplines such as: software engineering, hypermedia and hypertext engineering, human computer interaction, information engineering and user interface development [56].

Web Engineering is the application of systematic, disciplined and quantifiable approaches to the development, operation, and maintenance of Web-based systems and Web applications. Many of Web Engineering concerns are rooted in the fields of software engineering and distributed systems engineering. Research in the field focuses on applying and adapting classical software engineering techniques to the Web domain.

Traditional Web sites can be regarded as (universally distributed) hypermedia applications, and can be largely modelled using hypermedia modelling methods. Most approaches distinguish at least two main dimensions for hypermedia/Web site conceptual modelling: information modelling and navigation modelling. Information modelling describes the contents of the Web site. Navigation modelling describes its navigation capabilities, i.e., the paths that users can traverse, to explore the information universe. In the rest of this section we have presented a short overview of the most popular methods.

1.4.1 Web Modelling Language(WebML)

Ceri et al. presented the Web Modelling Language (WebML) [36]. WebML provides a high level conceptual description of a Web application. The language is geared towards Catalogue (data-driven) Web applications. It is composed of five models:

1. A structural model describes the data flow in the application.
2. A navigational model describes the topology of links between the different pages.
3. A compositional model describes the files and databases that are grouped together to represent some conceptual concept.
4. A presentation model describes the layout of each page and its graphical requirements.
5. A customization model describes the different groups of users of the software and their needs.

WebML is more suited for the high level specification of Web application than for modelling the actual implementation because it lacks the concepts needed to model control flow. For example, relations between the different source code objects and the call graph cannot be expressed using WebML's constructs and concepts.

1.4.2 Web Application Extension(WAE)

In [39], Conallen presents the Web Application Extension (WAE) for the Unified modelling language (UML) [129]. In WAE, each Web page is modelled as a UML component and each Web page has two aspects, the server-side aspect and the client-side one. The server-side aspect shows the Web page's interactions with the components that reside on the server. On the other hand the client-side aspect focuses on the page's interaction with the objects and applets that reside on the client's machine. This work is of great value for the maintainers of the application, if the initial developers of the application specified their application using these specification languages. The WAE defines a Web application as: "a Web system that allows its users to execute business logic with a Web browser". This modelling approach proceeds to point out the need to model Web applications due to their complexity. It is an important point that WAE is based on standard UML. In WAE Web pages are modelled as UML components. Every Web page is modelled using two different aspects:

1. Its server side aspect where it shows the page's interaction with other pages, the business logic objects, the databases and the server provided resources.

2. Its client side aspect where it shows the page's interaction with the browser built-in objects and Java applets.

1.4.3 UML-based Web Engineering(UWE)

In [63] Hennicker R. has proposed a UML-based Web Engineering (UWE) methodology for hypermedia design which is based on a UML profile for the hypermedia domain [24]. Starting with a use case analysis and a conceptual model of the application it first provides guidelines for modelling the navigation space. From the navigation space model we can derive, into the next step, a navigational structure model which shows how to navigate through the navigation space using access elements like indexes, guided tours, queries and menus. Finally, a presentation model is constructed that can be directly implemented by HTML frames. The different models of the design process are represented by using a hypermedia extension of UML. As it is clear, the only aspect of Web applications which this methodology covers is the presentation layer.

In [71] it has been shown how UWE can be supplemented by other views using the variety of UML diagram types and UML modelling elements. The proposed extensions focus on the dynamic aspects of the design of Web applications, such as task modelling and modelling of Web scenarios, graphical representation of the distribution of Web components and semi-automatic generation of Web applications based on design models.

1.4.4 Object-Oriented Hypermedia Design Method(OOHDM)

Hypermedia applications typically include complex information, and may allow sophisticated navigation behaviour. The Object-Oriented Hypermedia Design Method (OOHDM) [121] and [115] uses abstraction and composition mechanisms in an object oriented framework to, on one hand, allow a concise description of complex information items, and on the other hand, allow the specification of complex navigation patterns and interface transformations.

In OOHDM, a hypermedia application is built in a four-step process supporting an incremental or prototype process model. Each step focuses on a particular design concern, and an object-oriented model is built. Classification, aggregation and generalization/specialization are used throughout the process to enhance abstraction power and reuse opportunities.

According to OOHDM, the development of hypermedia applications occurs as a four activities process – Conceptual Design, Navigation Design, Abstract Interface Design, and Implementation – that is performed in a mix of iterative and incremental styles of development; in each step a model is built or enriched. The cornerstones of the OOHDM approach are:

1. The notion that navigation objects are views, in the database sense, of conceptual objects;
2. The use of appropriate abstractions to organize the navigation space, with the introduction of navigation contexts;
3. The separation of interface issues from navigation issues;
4. An explicit identification that there are design decisions that need only be made at implementation time.

1.4.5 Object-Oriented Hypermedia(OO-H)

OO-H (Object-Oriented Hypermedia) [58, 103, 57]looks at Web systems as unified software artifacts where structure, behaviour and presentation are all basic pieces that must be properly combined to get a correct final software product. The OO-H method is a generic model, based on the object-oriented paradigm that provides the designer with the semantics and notation necessary for the development of Web-based interfaces and its connection with previously existing application logic modules. OO-H defines a set of diagrams, techniques and tools that shape a sound approach to the modelling of Web interfaces. The OO-H proposal includes:

- Design process
- Pattern catalogue
- Navigationaccess diagram(NAD)
- Two-fold presentation layer(abstractpresentation diagram and composite layout diagram)

The extension to “traditional software” production environments is achieved by means of two complementary views: (1) the navigational access diagram (NAD) that defines a navigation view, and (2) the abstract presentation diagram (APD) and composite layout diagram (CLD) that gather the concepts related to abstract structure of the site and specific presentation details, respectively. The NAD diagram enriches the domain view provided by the UML use case and class diagrams with navigation and interaction features. Also, to define navigation and visualization constraints, OO-H uses the object constraint language [136], a subset of the standard UML that allows software developers to write constraints over object models augmenting the model precision. OO-H associates such constraints to the navigation model by means of filters defined upon links. On the other hand, the definition of abstract pages in the APD is based on a set of XML DTDs. Both the NAD and the APD capture the interface related design information with the aid of a set of patterns, defined in an interface pattern catalogue that is integrated in the OO-H proposal.

1.5 Modelling Requirements for Web Applications

Traditional Web sites can be regarded as (universally distributed) hypermedia applications and can largely be modelled using hypermedia modelling methods. Most approaches that we have presented in the previous section distinguish at least two main dimensions for hypermedia/Web site conceptual modelling: information modelling and navigation modelling. Information modelling describes the contents of the Web site. Navigation modelling describes its navigation capabilities, i.e., the paths that users can traverse to explore the information universe. Hereafter the union of information modelling and navigation modelling will be globally referred to as *hypermedia modelling* or *Web modelling*, interchangeably.

Traditional hypermedia modelling focuses on organizing the information structures and navigation paths. In many modelling approaches functional aspects are neglected or regarded as second class citizens.

A comprehensive study regarding gap analysis for a wide range of available modelling approaches has been made in [12]. The results have been presented in Figure 1.4. In this chart the level of support that each methodology provides for functional and information aspect has been assessed. The target zone indicates the required level of abstraction for both functional and information levels. The result of this study and similar study like [97] reveals that most of Web modelling approaches need to be improved in one or more of the following aspects:

- Ability to model functional and informational aspects with an integrated manner and the same level of emphasis
- Ability to model the system in different level of abstraction and refinement with support for architectural level and component based modelling
- Ability to model business domain concepts

In the following sections we try to summaries some recent attempts which have been made relating to these three pre-mentioned aspects.

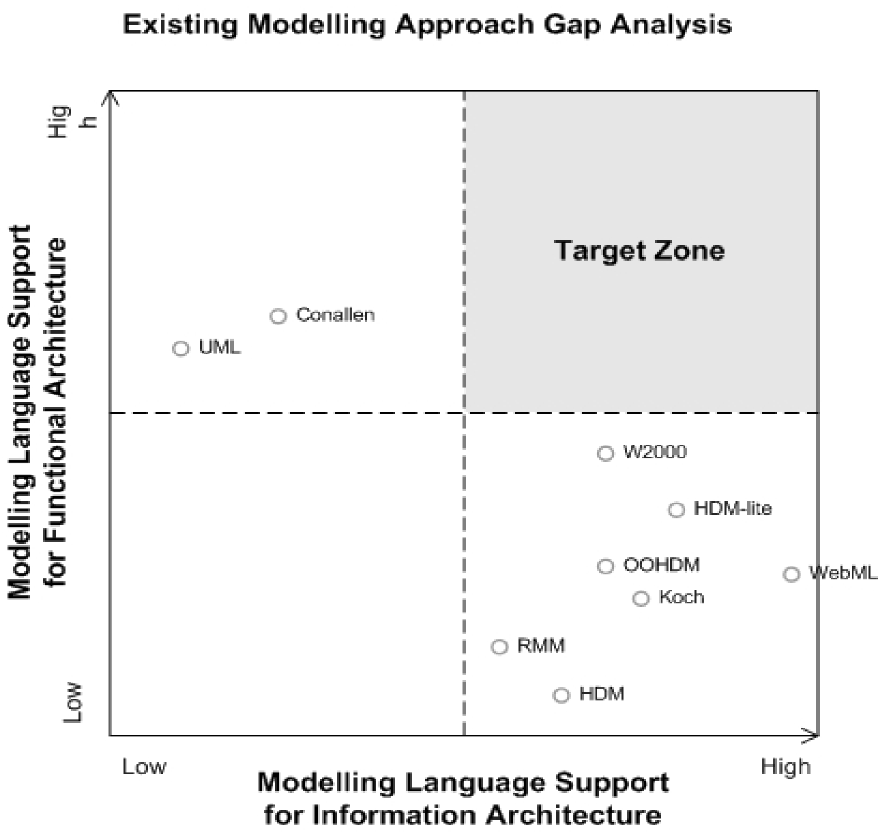


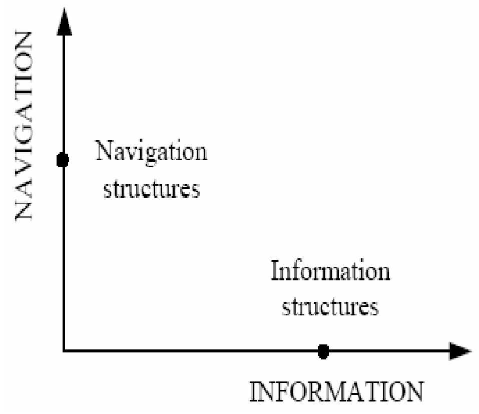
Figure 1.4: Existing Modelling Approach Gap Analysis

1.6 Modelling Support for Both Functional and Informational Aspects

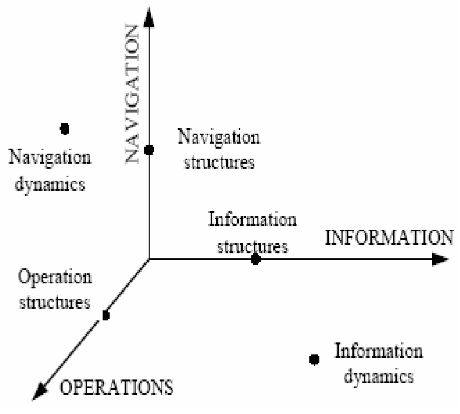
As we mentioned earlier current Web-based Systems are not just graphic design presentation and hypertext navigation systems any more; they are delivering a complex set of functional and transactional properties in e-business systems. Considering this fact indicates that Web Modelling languages (WMLs) should consider both functional and informational aspects as first class citizens. The problem of conceptual design of Web applications can be approached from two different perspectives:

- Web applications can be regarded as extensions to traditional information systems, complemented with navigation and complex information structures;
- Web applications can be regarded as extensions to traditional Web sites complemented with various kinds of application operations and conventional functionality of information systems.

Whatever the chosen approach, it is clear that Web applications introduce a new dimension into modelling space which must be modelled explicitly: non-navigational *operations*. These operations are not "read-only": They may modify individual contents as well as entire information/navigation structures of the application. As such, they add *dynamic* (i.e., evolution) properties to the two original dimensions (information and navigation) of conventional Web sites. In [21] a graphical representation of design space for Web sites and Web application is illustrated which we have reproduced in Figure 1.5.



(a) Web Site Modelling



(b) Web Applications Modelling

FIGURE 1.5: Design Spaces for Web Sites and Web Applications

Differently from traditional Web sites, hypermedia structures of Web applications are very often dynamic entities, in the sense that information and navigation objects evolve either along time, or by the (direct or indirect) effect of user operations. Therefore the integrity and cohesion of a Web system is largely dependent on a close and yet flexible interconnection between its information architecture and its functional architecture. The requirement is that WMLs need to support not only the modelling of both information architecture and functional architecture but, more importantly, the integration of them in a cohesive and consistent manner [12].

It is critical that these are linked for the Web system to successfully address business needs in an effective way [84]. Indeed, the integrity and cohesion of a Web system is largely dependent on a close and yet flexible interconnection between its information and functional modelling. The requirement is that WMLs need to support not only the modelling of both information and functional aspects but, more importantly, the integration of them in a cohesive and consistent manner. Several attempts including [35, 72, 131] have been made to integrate theses aspects.

For example in [35] the authors have presented an extension to the OO-H conceptual modelling approach for the specification of user-operation interaction (feeding of parameters, invocation of operations, either simple or compound, and view of operation results).

This approach increases the level of abstraction at which Web Applications have been traditionally developed and integrated. The main contributions of the above mentioned paper can be summarized as follows:

- An integration process that, departing from traditional software engineering techniques, extends the views provided by such approaches with a set of new complementary hypermedia views that include server interface definition.
- A set of interaction modes that define the way the user can introduce the values for the set of parameters involved in the service invocation.
- A set of modelling constructs that abstract the definition of One-Step and Multi-Step interfaces. At this moment efforts are being made towards the support of Compound Services that involve Internet Transactions. OO-H is supported by a CASE tool that, at this stage of development, already provides a model compiler for the automatic generation of interface prototypes. Intensive work is being performed on the OO-H Case tool to provide full support to the method.

1.7 Supporting Architectural Level and Component Based Modelling

A critical level of abstraction in the description of a complex system is its software architecture. At an architectural level one describes the principal system components and their pathways of interaction. Architectural descriptions are typically used to provide an intellectually tractable, birds-eye view of a system, and to permit design-time reasoning about system-level concerns, such as performance, reliability, portability, and conformance to external standards and architectural styles.

In practice most architectural descriptions are informal documents. They are usually centred on box-and-line diagrams, with explanatory prose. Visual conventions are idiosyncratic, and usually project-specific. As a result, architectural descriptions are only vaguely understood by developers, they cannot be analyzed for consistency or completeness, they are only hypothetically related to implementations, their properties cannot be enforced as a system evolves, and they cannot be supported by tools to help software architects with their tasks [52, 109, 125]. To improve the situation a number of people have suggested the use of more standardized and formal notations for architectural description. Viewed broadly, there are two main sources of such recommendations. One is from the software architecture research community, which has proposed a number of "architecture description languages" (ADLs). The other source is from the object-oriented community. A number of authors have examined ways to model architectures using object notations and specially UML. In the following sections short overviews of these two approaches are presented.

1.7.1 Architecture Description Languages(ADLs)

Architecture description languages specifically designed to represent software and system architectures are supported by some tools. Developers can use ADLs, such as Aesop [51], Adage [23], C2 [92], Darwin [89], Rapide [88], SADL [98], UniCon [124], MetaH [26], or Wright [13]. Although with a considerable overlap on the core, each ADL focuses on different aspects of architectural specification, such as modelling the dynamic behaviour of the architecture, or modelling different architectural styles. This diversity provides different approaches to solve specific families of problems. However, the interchange of information between different ADLs becomes a major drawback. Developing a single ADL providing all the features of the various ADLs would be a very complex endeavour. Instead, an ADL called Acme [53] emerged as a generic language which can be used as a common representation of architectural concepts in the interchange of information between specifications with different ADLs [20]. These languages have matured over the past years. Most come with tool sets that support many aspects of architectural design and analysis, such as graphical editing, code generation, run-time monitoring, anomaly detection, and performance analysis. Although ADLs allow for architecture in-depth analysis, their formality is not easily reconciled with day-to-day development concerns. New UML 2.0 features [102] provide a promising way to rectify this weakness and to bring architectural modeling to a larger community. In the later sections we discuss this subject in more detail.

While ADLs (and their associated tool sets) differ in many details, there has emerged a general consensus about the main ingredients of architectural description. Focusing on architectural structure, we take that core set of concepts as the starting point for this review. In this shared ontology there are six basic concepts: components, connectors, ports, interfaces, properties, systems, and styles [70, 52].

Components: *(parts)* represent the computational elements and data stores of a system. Intuitively, they correspond to the boxes in box-and-line descriptions of software architectures. Typical examples of components include clients, servers, filters, blackboards, and databases. Components may have multiple interfaces (which we will call *ports*), each interface defining a point of interaction between a component and its environment. A component may have several ports of the same type (e.g., a server may have several active http connections).

Connectors: *(relationships)* represent interactions among components. They provide the "glue" for architectural designs, and correspond to the lines in box-and-line descriptions. From a run-time perspective, connectors mediate the communication and coordination activities among components. Examples include simple forms of interaction, such as pipes, procedure call, and event broadcast. Connectors may also represent complex interactions, such as a client-server protocol or a SQL link between a database and an application. Connectors have interfaces that define the *roles* played by the participants in the interaction.

Ports: *(Interfaces)* a special type of architecture part that defines a set of interaction points between it and other parts. Well-defined interfaces ensure proper relationships between parts.

Properties: *(constraints)* represent additional information (beyond structure) about the parts of an architectural description. Although the properties that can be expressed by different ADLs vary considerably, typically they are used to represent anticipated or required extra-functional aspects of an architectural design. For example, some ADLs allow one to calculate system throughput and latency based on performance estimates of the constituent components and connectors. In general, it is desirable to be able to associate properties with any architectural element in a description (components, connectors, systems, and their interfaces). For example, an interface (port or role) may describe an interaction protocol.

Systems: represent graphs of components and connectors. In general, systems may be hierarchical: components and connectors may represent subsystems that have *internal* architectures. We will refer to these as *representations.* When a system or part of a system has a representation, it is also necessary to explain the mapping between the internal and external interfaces. We will refer to the elements of this mapping as *bindings.*

Types and Styles: represent families of related systems. An architectural *style* could be considered as an ***architectural configuration:*** (or topology). Topology refers to connected graphs of parts and relationships that describe an architectural structure. At its most basic level, an architectural modelling language must be able to specify an architectural structure consisting of parts, interfaces, relationships and constraints. For distributed systems and Web-based applications [52], it is desirable that the language also includes the ability to model the following:

Composition: Since a large system typically decomposed into subsystems, or may be used as a subsystem by another system, architectural models should support hierarchical composition.

Heterogeneity: Large, complex systems are often a mix of legacy and new subsystems, with components and connectors at various levels of abstraction and granularity. An architectural model needs to be able to connect these heterogeneous parts into **a** cohesive whole

System evolution: Architectural parts are subject to complex and changing requirements and protocols. Architectural models must be able to specify the correct and consistent refinement of parts from concept to implementation. In addition, they must be able to support traceability of requirements across abstraction levels.

Distribution and concurrency: Architectural models for distributed enterprise applications need to specify distribution strategies and concurrent processing requirements.

Non-functional requirements: Architectural models must also be able to specify non-functional requirements, such as reliability and performance constraints.

1.7.2 Using UML as Architecture Description Language

The OO approaches to software modelling are widely accepted in industry. In particular, the UML [129] has become *de facto* standard for Modelling OO systems. Using UML to describe software architectures can bring economies of scale, better tool support and inter-operability, as well as lower training costs. Despite the above mentioned advantages, using UML as an ADL has some shortcomings. For instance, all versions of UML before UML 2.0 are less expressive than ADLs when representing connections between components.

It is essential to appreciate that the UML is not specifically designed as an ADL. Therefore it is fair to say that all versions of UML before UML 2.0 suffer from some shortcoming for describing software architecture. Some of these issues have been resolved in UML 2.0. As in the time of writing this chapter the full specification of UML 2.0 had not yet been released, no substantial assessment on UML 2.0 was available. Consequently in the rest of this section we first review earlier versions of UML and then provide some recent researches related to UML 2.0.

UML unifies a number of object modelling notations in a common framework and is quickly becoming a standard object notation for industry. While a detailed description of UML is beyond the scope of this report, we summarize its principal constructs (known as model *elements*) that can be used to model software architectures:

Classes, Interfaces and Objects: Classes are the primary construct for describing the logical view of a system. Classes have properties in the form of *attributes*, provide abstract services in the form of *operations,* and can be logically related to one another using *associations.* Classes may expose their functionality through a set of supported *interfaces*, collections of related operations. Classes have instances called *objects*, which are used in models called *collaborations* to depict behaviour under particular scenarios.

Component and Component Instances: Components are used to describe the physical, deployable pieces of a system. Like classes, components in UML expose their functionality through interfaces. Components are typically related to each other using dependency relationships. The deployment of a system on a set of hardware is described by associating components with hardware *nodes.*

Packages: UML provides a grouping mechanism that is used to partition large UML models into manageable chunks called *packages.* UML also defines a type of grouping element called a *subsystem,* which is typically used to encapsulate the object models that define a coarse-grained module in a system.

Relationships: Model elements are related to one another with *associations* and *dependencies.* Dependency is the most generic relationship in UML, indicating that an element depends in some way on the definition of another element. Association is a richer relationship that describes an abstract relationship between classes and the roles the classes play in the relationship.

Stereotypes:To allow the extension of UML with domain-specific concepts, UML provides a mechanism for associating constraints with elements of a model, using a constraint language, *OCL* [129]. These constraints can be grouped and named using a construct called a *stereotype.* UML also includes a set of standard stereotypes.

The above constructs can be composed in various ways in a UML model and visualized in diagrams. Textual *annotations* may be associated with any of them. Frequently, these annotations are in the form of *tagged values:* arbitrary attribute-value pairs. UML also defines a set of models for describing the dynamic behaviour of a system, including collaboration diagrams that specify system behaviour using event-based *interaction scenarios*, descriptions based on *state machine*s, and *use cases.*

Several attempts to map ADLs to UML have been made in the past [93, 70, 43, 52]. One motivation for such attempts is to bring architectural modelling to a larger community, through the use of mainstream modelling notations. Another is to provide automatic refinement mechanisms for architectures. UML can be used as a bridge from architectural to design elements.

In [52] five strategies have been examined for encoding Architectural description in UML. For each of these the authors have considered a number of variations and strategies with respect to completeness, legibility, and semantic match. The following conclusions were drawn in this study.

First, *there is no single best way to encode ADLs in UML.* Each of the strategies has strengths and weaknesses, depending on how well they support the evaluation criteria. With respect to completeness and legibility there is a typically a trade-off: encodings that emphasize completeness (by providing a semantic home for all of the aspects of architectural design) tend to be verbose, while graphically appealing encodings tend to be incomplete. Hence, the best strategy will depend on what aspects of architectural design needed to be represented. In restricted situations (for example, if there is only one type of connector) it may be preferable to use an incomplete, but visually appealing, encoding.

Second, all of the encodings exhibit some form of semantic mismatch. UML's vocabulary of classes, objects, packages, components, associations, etc., while varied and rich, is ultimately designed to support an object-oriented view of software design. As such, UML does not provide a completely adequate foundation for architecture-based description of systems. In this paper, they illustrated a number of specific examples of mismatch, including the following:

- Neither the class, subsystem or UML's component concept is a perfect match to the ADL component concept;
- Unlike objects, architectural instances may need to define additional structure not defined by their types;
- the port concept has no good analogy in UML, since unlike interfaces, a port should be able to define both provided and required services, and a component might have multiple instantiations of a particular port type;
- There is no satisfactory way to fully describe a connector and its roles; and
- Although the ADL type and instance concepts are very similar to the class and object concepts of UML, neither class diagrams nor collaboration diagrams are wholly appropriate for describing architectural configurations.

Given these observations, one might well ask whether there are reasonable alternatives to the direct encoding of architecture in ADL. According to [52] there are two possible alternative paths:

1. Continue to use ADLs, but map to OO implementations. In this approach architecture description retains its own notations, but tools are provided to convert those descriptions to lower-level object notations in situations where the implementation is done in an object-oriented fashion.
2. Extend UML to include additional concepts for architectural modelling. This could be done by extending the UML meta-model, or by defining a profile for architectural design. Indeed, we can see the inclusion of architectural notions, such as ports, in proposals for real-time extensions to UML [123] and later in UML 2.0 [102].

The authors suggested the extension of their examination of mappings to non-structural aspects of software architecture, such as behaviour, performance, and reliability as future research. Furthermore, to make more progress in reconciling architecture description with UML it will be important to consider the problem at a more formal level.

All approaches were performed with UML 1.x, suffering from some problems including notation mismatch. In [59] an approach based on using the new UML 2.0 meta-model elements has been proposed. These new elements enhance the language's suitability for component-based design. The proposed mapping builds upon the added expressiveness of UML 2.0 for architectural concepts. The availability of components with ports typed by *provided* and *required* interfaces has proved to be a step forward in the exercise of bridging the gap between architectural and design information. This improves traceability between architectural description and its implementation, using the design as a middle layer between them. This traceability is relevant for keeping consistency between the architecture, design and implementation of a software system.

The proposed mapping focuses mainly on structural aspects and design constraints. Although it also points out to ways of dealing with the definition of system properties, including semantics and behavioural specification, further research is required to provide more specific guidance on these aspects.

1.7.3 The Need for Architectural Level Modelling in Web-based Applications

Modelling approaches for Web-based systems should be business-oriented to facilitate changes in customer and organization requirements and technology-neutral to insulate them from changes in technology. In addition, they must be able to address the issues of distribution and concurrency associated with distributed applications.

There is a critical need for architectural constructs, techniques and methods to manage the complexity of distributed Web-based applications and systems. The architectural level modelling can bridge between business modelling and functional and information modelling. The need for higher-level design abstraction in Web-based application could better understand in the light some key aspects of these systems. Here we have presented some of these aspects which have been summarised in [82]:

- **Short time frames for initial delivery.** Web development projects often have delivery schedules that are much shorter than for conventional IT projects. This is partly a consequence of the rapid pace of technological development and partly related to the rapid uptake of Web systems.
- **Increased importance of quality attributes.** Web systems represent an increase in mission-critical applications that are often directly accessible to external users and customers. Flaws in applications (be they usability, performance or robustness) are therefore much less able to be 'hidden' and hence much more problematic.
- **Open modularised architectures.** Although not unique to Web applications, it is still worth mentioning the emphasis that is typically placed on open and modularised architectures for Web systems. They are often constructed from multiple COTS (commercial off-the-shelf) components that are adapted and integrated together. Indeed, strong integration skills become much more critical in most Web projects.
- **Rapidly changing technologies.** The technology that underpins most Web systems is changing very rapidly. This has several consequences. The important one is that it increases the importance of creating flexible solutions that can be updated and migrated to new technologies with minimal effort. For example, the need for reusable data formats (such as XML) and technology neutral design is very much emphasised in Web System development.
- **Highly variable client.** It is extremely common for Web-based systems to interact with different type of clients on varieties of platforms. This typically means that interoperability and dealing with heterogeneous clients is an absolute property for these systems. Therefore the modelling approach should support high level and platform independent modelling of the system.

The evolution of Web-based applications and underlying complex middleware, demands high-level architecture modelling and modular design. This trend has been reflected is some recent works on Web-based application modelling like [85, 91, 73, 41, 139].

1.8 Business Domain Modelling

The growing importance of Web-based systems to organisations has become increasingly evident within the recent years. Internet and an increasingly complex set of Web standards, protocols and technologies provide sophisticated business solutions that merge Web-based front-ends with complex back-end software. The rapid and successful deployment of these systems is often critical to the business strategy of many organisations - particularly with respect to the way in which they interact with customers, clients, and/or business partners. In fact most Web applications actually form the channel between the organisation and its business partners or customers.

Many research and practical experiences suggest that there is a much stronger linkage between the business model and the technical architecture in Web systems in comparison to conventional software systems [106, 83, 61, 79, 84]. As a consequence, the quality of the technical architecture will largely depend on the developers' understanding of the current business model and the required changes to it, in the context of the impact from Web technologies. To facilitate and document this understanding, WMLs need to provide the ability to model business domain concepts e.g. business processes, business entities, workflows, business rules, together with the roles and responsibilities of users.

Business-related development and modelling artifacts are usually created and used by developers from both IT and business backgrounds. As a result, the modelling of business domain concepts needs to be designed with the consideration of target user types so that these model artifacts can be easily understood, communicated and modified within and across development teams and business units.

Once understood and documented, the business model needs to be effectively translated into a technical architecture so that the desired system functionality can be implemented and delivered. To support this requirement, WMLs need to provide the ability to identify the linkage between the business model and the technical architecture, and between the model elements in the business model and the model elements in the technical architecture. This interconnection needs to be represented at various abstraction levels.

The necessity to integrate business processes with WMLs is well recognised in many recent publications including [118, 75, 110]. Some of these analyses suggest extending available methods, meanwhile others introduce new methods. In [86, 130] an extension of the WebML has been presented to support business process modelling and its integration to information modelling..

An extension of the OOHDM which gives a behavioural definition to the core features of this method and propose new models to support business processes has been proposed in [117]. The authors claim that they derive application-specific model classes from predefined behavioural model classes that have operations with a well-defined semantics. The behavioural model classes collaborate with a Web Application virtual Machine (WAM). The WAM models basic Web-browser characteristics, i.e. HTTP-HTML characteristics. Thus, the semantics of an OOHDM Web application model is precisely defined in an executable way.

In [27]the authors propose UWA+, a framework which extends a methodology for modelling Web applications UWA (Ubiquitous Web Application)[134] with concepts of business process design. They claim that UWA+ is able to bridge the gap between the business process modellers and the Web system designers. The framework proposed in this paper aims to solve some of the problems haunting the relationship between business process modelling and Web systems. Having a unique and standard language to describe different aspects of business and systems is fundamental in order to create a common ground for discussing both business and the supporting systems. The emphasis of this framework is on providing the basis for creating such a common representation and simultaneously providing a way for addressing the traceability between the different views. The first view of the framework is about business goals and business processes. The second view shows the relation between the process and Web system. It allows the representation of how Web information systems support the business logic which is one of the main issues in today's organizations. Web system modelling is based on a user cantered approach, which is the cornerstone of today's Web system architecture.

In [131] a companion notation to an existing modelling language, WebML has been proposed. This companion model is referred to as the Web Information Exchange Diagram (WIED). A key point in this model is that the WIED approach is built around the notion of information flows at the level of understanding business processes. This enables the models to form a link between higher-level models (specifically, business models) and lower-level detailed design models.

The WIED approach still has some limitations, however. For example, while WIED provides linkages to some modelling approaches (e.g. a widely-adopted low-level information modelling approach such as WebML and a typical business modelling approach such as e3-value), it doesn't support linkages to standard modelling approaches such as UML. It has also yet to define clearly the relationship to functional modelling which can be appropriately represented by the UML model suite. The relationship between different levels of modelling in WIED has been depicted in Figure 1.6. In [87], the authors have demonstrated how the WIED model can be mapped into UML compliant notations and a new UML diagram. They have argued that this enables WIED to be compatible with UML and provide improved connections between WIED and other models that are typically constructed. Therefore, it will lead to a more standardized WIED and subsequently enhance the design process of Web-enabled systems. They believe that this will also assist developers and clients, who use UML-based notations in their system developments, in understanding the impact on business process and models which arise from

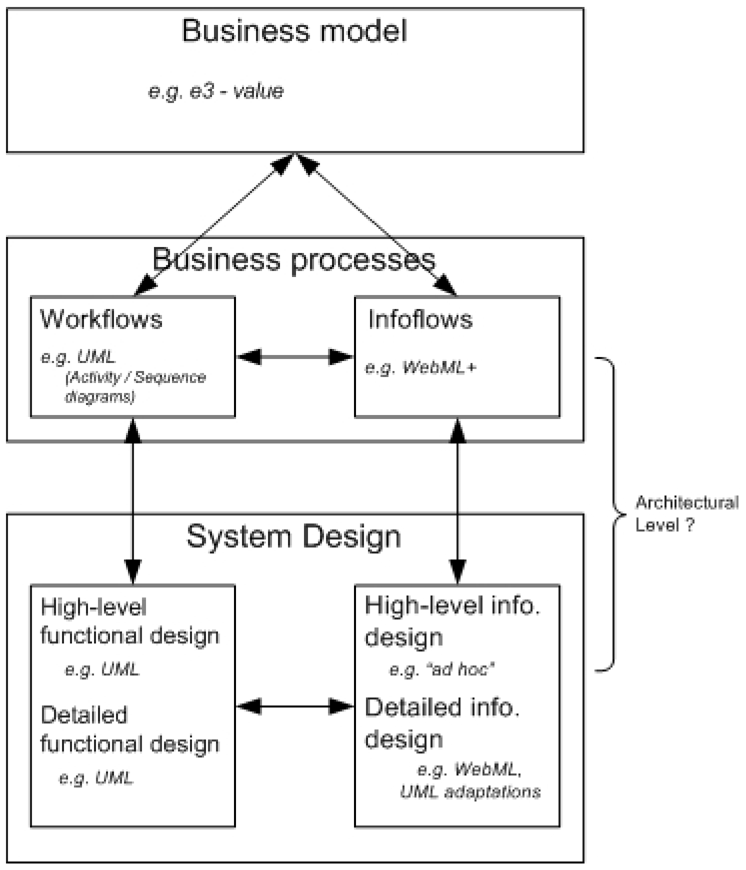


Figure 1.6: Relationships between modelling domains in WIED

changes that are made to the underlying information designs (as has been illustrated in our previous work with WebML-compliant WIED.

In this work, they have also briefly proposed guidelines to support the mapping process linking the WIED (UML-compliant) to other modelling approaches based on UML. This should be a good start for making the WIED a practical companion to existing widely-used UML models and potentially integrate with those models to create a better UML-based modelling suite for Web system developments.

1.9 Conclusions

In this chapter we reviewed, how simple static Web systems have been evolved to very sophisticated and complex Web applications. Although, as it was the case in other area of software engineering, design and development methods of Web applications in the early stages were very implementation oriented and ad-hoc in there approaches. As the need for building more reliable and scalable systems grew, a lot of attempts by researchers with different backgrounds have been made to introduce some methodical approaches for modelling Web applications. These modelling approaches either extend available general software modelling or introduce new methods.

It can be seen, in this chapter, that as the systems become more complex, higher level modelling methods like architectural and business level modelling become more essential. Tool support for the modelling approach is another key issue. Using proper uniform notation that can cover all necessary aspects of Web application modelling is another key property for a modelling approach. The simplicity and the level of compliance with standard notations like UML are other important aspects that play a great role to make a modelling approach more acceptable to a broader range of system developers.

Despite the fact that many Web applications are mission-critical systems and that they should provide a high level of dependability, almost all of the modelling approaches which we surveyed in this chapter lack any forms of verifiability. Verifiability is an essential approach to eliminate inconsistencies in the specification and design, and deploy a dependable system. In the next chapter we investigate how formal approaches could contribute toward more dependable Web systems.

Chapter 2

Formal Methods and Web-Based Applications

2.1 Introduction

Every day more and more organisations and businesses extend their services to a wider rang of users either by by developing new Web-based application or making legacy systems Web-enabled. Many of these system are mission-critical or business-critical systems; which implies any failure can cause high economical losses. For example online banking, financial transaction systems, online shopping, revenue and taxation systems are a few examples of business-critical systems. All these systems are being required to function at high levels of reliability and security. Formal methods have been advocated as a means of increasing the reliability of systems, especially those which are safety or business critical systems.

In this chapter we present a short overview of critical systems and required properties of such systems. Formal methods are one of the major common practices for developing critical systems. Therefore a brief introduction about formal methods will follow the first section. The B-Method is one the formal methods that we use it later to develop our formal models. A more detailed review on this method and its extensions is presented in section 3. Some of the extensions are inspired from other formal methods like CSP (Communicating Sequential Processes), which we will discuss this issue in the end of chapter.

2.2 An overview of critical systems

A critical system is a system where system failure can have severe human or economic consequence. According to [66] critical systems can be classified into three categories:

- *Safety-critical systems:* Failure results in loss of life, injury or damage to the environment; examples of such systems are Chemical plant protection systems, embedded control systems in airplanes and trains;
- *Mission-critical systems:* Failure results in failure of some goal-directed activity; example of such systems is: Spacecraft navigation systems;
- *Business-critical systems:* Failure results in high economic losses; example of such systems are: online banking, financial transaction systems, online shopping and credit cart systems;

Many Web-based applications belong to the third group of the above category. The most important property of such systems is the dependability of the systems. The dependability of a system reflects the users degree of trust in that system. It reflects the extent of the users confidence that it will operate as users expect and that it will not *fail* in normal use. Systems that are not dependable are either unreliable or insecure. Therefore they may be rejected by their users and consequently lead to financial losses. The four major dimensions of dependability are:

- Availability
- Reliability
- Security
- Safety

These aspects and their related properties have been summarised in Figure 2.1:

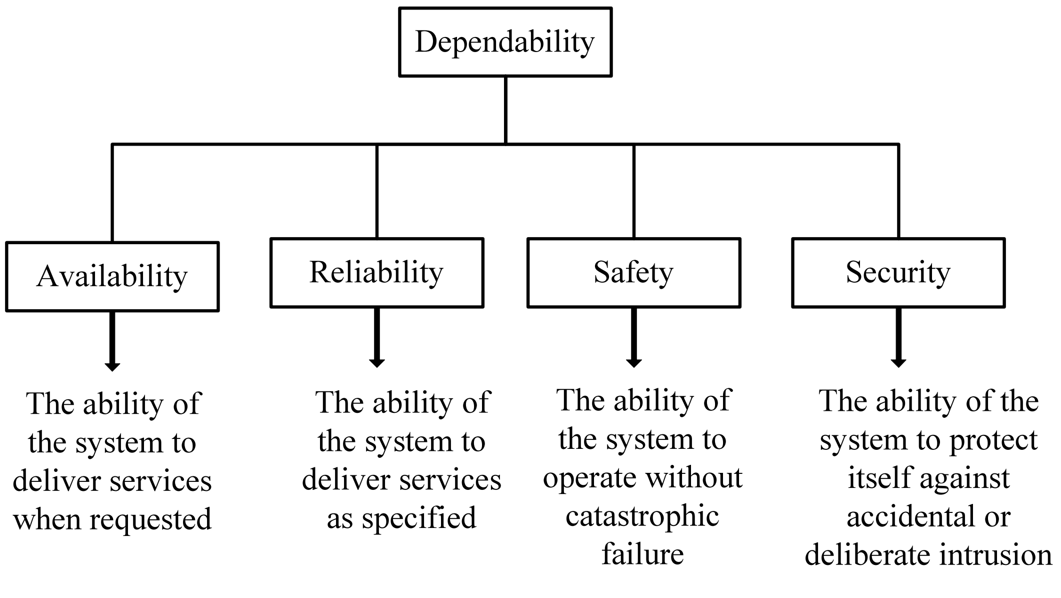

Dependability
Availability
Reliability
Safety
Security
The ability of the system to deliver services when requested
The ability of the system to deliver services as specified
The ability of the system to operate without catastrophic failure
The ability of the system to protect itself against accidental or deliberate intrusion

FIGURE 2.1: Main Dimensions of Dependability.

These are non-functional properties and they do not relate to any specific functionality of the system. Some or all of these properties are usually more important than detailed system functionality. It should be taken into consideration that the priority and the degree of importance of these aspects could vary between different types of critical systems. For example business systems are considered as security-critical system. This implies that the systems ability to protect itself from accidental or deliberate external attacks is becoming increasingly important. As systems are networked and external access to the system through the Internet is the main possibility to interact with them, security is an essential pre-requisite for availability, reliability and safety.

In today's technologically driven world the creation of high profile, mission-critical Web-based applications often requires the integration of a wide range of technologies and protocol standards. They need to deliver a high-performance, high-availability, scalable architecture, tightly supporting a wide range of Internet standards, databases and applications. Over the years, a variety of techniques have been developed to help us gain more insight into modelling and ultimately ensuring the quality of such systems. Based on the solid foundation of mathematics, formal methods have been found particularly usable for ensuring software quality and dependability. Formal methods now are on the verge of becoming best practice and/or required practice for developing safety-critical and mission-critical software systems. Although there are already established techniques and supporting tools for formal specification and development of software components, there is much scope for further development in areas like Web-based applications. As our aim is to apply formal method to the development process of Web-based applications, we present a short survey on different formal methods in the following sections.

2.3 An overview of Formal Methods

Formal methods used in developing computer systems are mathematically based techniques for describing system properties. Such formal methods provide frameworks within which people can specify, develop, and verify systems in a systematic, rather than ad hoc, manner [137].

A method is formal if it has a sound mathematical basis, typically given by a formal specification language. This basis provides the means of precisely defining notions like consistency and completeness and, more relevantly, specification, refinements,implementation, and correctness. It provides the means of proving that a specification is realizable, proving that a system has been implemented correctly, and proving properties of a system without necessarily running it to determine its behavior.

A formal method also addresses a number of pragmatic considerations: who uses it, what it is used for, when it is used, and how it is used. Most commonly, system designers use formal methods to specify a system's desired behavioral and structural properties. However, anyone involved in any stage of system development can make use of formal methods. They can be used in the initial statement of a customer's requirements, through system design, implementation, testing, debugging, maintenance, verification, and evaluation.

Formal methods are used to reveal ambiguity, incompleteness, and inconsistency in a system. When used early in the system development process, they can reveal design flaws that otherwise might be discovered only during costly testing and debugging phases. When used later, they can help determine the correctness of a system implementation and the equivalence of different implementations. For a method to be formal, it must have a well-defined mathematical basis.

One tangible product of applying a formal method is a formal specification. A specification serves as a contract, a valuable piece of documentation, and a means of communication among a client, a specifier, and an implementer. Because of their mathematical basis, formal specifications are more precise and usually more concise than informal ones. Since a formal method is a method and not just a computer program or language, it may or may not have tool support. If the syntax of a formal method's specification language is made explicit, providing standard syntax analysis tools for formal specifications would be appropriate. If the language's semantics are sufficiently restricted, varying degrees of semantic analysis can be performed with machine aids as well. Thus, formal specifications have the additional advantage over informal ones of being amenable to machine analysis and manipulation.

A formal method consists of three parts, namely *syntax*, *semantics* and *satisfactions*. The syntax part defines a set of symbols or notations and grammatical rules that define well-formed formulae. These rules characterise a language's *syntactic domain*. The syntax of a language shows how the symbols in the language are put together to form meaningful formulae. Neither the nature of the objects symbolised nor the meanings of the relationships between them are characterised by the syntax of a language. For example, the presentation of the propositional calculus is entirely syntactical.

Meanings, or interpretations of formulae, are specified by the semantics of a language. A set of objects, known as the language's *semantic domain*, can provide a model of a language. The semantics are given by exact rules which state what objects satisfy a specification. For example, Cartesian Geometry shows how theorems in Euclidean Geometry can be modeled by algebraic expressions.

The third part defines relations between syntax and semantics. It consist a set of precise relations and rules defining which objects satisfy each specification [137].

In [81] the authors suggested that a formal method should consist of some essential components: a semantic model, a specification language (notation), a verification system/refinement calculus, development guidelines and supporting tools:

1. The *semantic model* is a sound mathematical/logical structure within which all terms, formulas and rules used have a precise meaning. The semantic model should reflect the underlying computational model of the intended application.
2. *The specification language* is a set of notations which are used to describe the intended behaviour of the system. This language must have a proper semantics within the semantic model.
3. *Verification system/refinement calculi* are sound rules that allow the verification of properties and/or the refinement between specifications.
4. *Development Guidelines* are steps showing the use of the method.
5. *Supporting tools* may provide proof assistant, syntax and type checking, animator, and prototyping.

It should be emphasised that all formal methods do not necessarily consist all above elements. Thus in terms of the degree of rigorous mathematical support which they offer for system development they could be classified in four different groups or levels of formalism as following [99]:

- **Level 0:** No applied mathematics at all, but perhaps appeal to tabular or diagrammatic notations, pseudo code, and equations defining transfer functions, etc.
- **Level 1:** The use of concepts and notations from discrete mathematics, with proofs conducted in the traditional, informal style of mathematical discourse.
- **Level 2:** The use of formalised specification languages with mechanised support for syntax analysis, pretty-printing, and simple type checking.
- **Level 3:** The use of fully formal specification languages with comprehensive support environments including mechanised theorem proving and proof checking.

Proofs at levels 1 and 2 are conducted in the manner of the rigorous arguments preferred by mathematicians, although specification formalisms at level 2 may provide deduction rules that could in principle lead to formalising such arguments; the transition to level 3 is therefore marked by the provision of theorem provers and the fully formal specification languages alluded to which are firmly rooted in mathematical logic (making mechanical support a practical necessity), and which have demonstrably sound axiomatisations.

Formal methods could be used in different ways:

- *writing formal specifications:* The production of specifications which are then the basis for a conventional system development. In this case, specifications are used as a precise documentation medium which has the advantages of manipulability, abstraction and conciseness.

- *proving properties about the specification:* Consistency and property checks and animation of specification could be performed at this stage with the aid of the associated supporting tools. Therefore, formal methods allow us to find errors in the specification phase.
- *deriving implementations from a given specification:* Once a specification has been set up and one has figured out that it is indeed what is desired, it would be helpful to have a design method that could automatically derive a system's implementation that fulfills the given requirements. However, specifications are often given in a declarative manner and not in a constructive manner. This means that these specifications only describe what the system should do, but not how this function can be achieved. It is certainly not possible to derive correct programs from declarative specifications since these problems are intrinsically undecidable so the tools can never solve them. Although the tool can provide some sort of help and directions like suggesting possible patterns for refinement, but the construction of appropriate implementations will always remain a creative task for human beings.
- *verifying specifications with respect to a given implementation:* The design steps that are used to refine the system's specification must not affect the validity of the specification. Therefore, through correctness preserving refinement rules, we should be able to check whether each refinement step preserves the correctness of previous step system. This will give the developed system a degree of certainty and trustworthiness.

2.3.1 Classification of Formal Methods

Based of different criteria like syntax, underlying semantics or targeted systems , formal methods could be classified in different classes. A very general classification is presented in [137], which classify formal methods to three main classes, namely *Model-oriented*, Property-oriented and *Visual languages*.

- ***Model-oriented:*** In model-oriented method, a specifier defines a system's behavior directly by constructing a model of the system in terms of mathematical structures such as tuples, relations, functions, sets, and sequences.
- ***Property-oriented:*** Using a property-oriented method, a specifier defines the system's behavior indirectly by stating a set of properties, usually in the form of a set of axioms, that the system must satisfy.
- ***Visual languages:***Visual methods include any whose language contains graphical elements in their syntactic domains.

A more detailed and precise classification of formal methods is presented in [81, 80]which formal methods have been classified into the following five classes or types, i.e. ***Model-based, Logic-based, Algebraic, Process Algebra and Net-based (Graphical)*** methods. In the following subsections we will briefly discuss each of these approaches.

Model-based Methods.

A system is modelled by explicitly giving definition of states and operations that transform the system from a state to another. In this approach, there is no explicit representation of concurrency. Non-functional requirements (such as temporal requirement) could be expressed in some cases. There are three most popular model-based formalisms:

- Z [128].
- VDM [69].
- B-Method [2]

Logic-based Approach.

In this approach logics are used to describe system desired properties, including low-level specification, temporal and probabilistic behaviours. The validity of these properties is achieved using the associated axiom system of the related logic. In some cases, a subset of the logic can be executed, for example the Tempura system [100]. The executable specification can then be used for simulation and rapid prototyping purposes. Logic can be augmented with some concrete programming constructs to obtain what is known as wide spectrum formalism. The development of systems in this case is achieved by a set of correctness preserving refinement steps. Examples of this form are TAM [120] and the Refinement Calculus [119]. Below some popular logic-based formalisms have been listed:

- Hoare Logic [42].
- Modal Logic [104].
- Temporal Logic [90].
- TLA And TLA+ [76, 77].
- RTTL (Real-Time Temporal Logic) [105].

Algebraic Approach.

In this approach, an explicit definition of operations is given by relating the behaviour of different operations without defining states. This is similar to the model-based approach where there is no explicit representation of concurrency. Below some popular algebraic-based formalisms have been listed:

- OBJ [68].
- LARCH [60].

Process Algebra Approach.

In this approach, explicit representation of concurrent processes is allowed. System behaviour is represented by constraints on all allowable observable communication between processes. Below some popular process algebra-based formalisms have been listed:

- CSP(Communicating Sequential Processes) [64].
- CCS(Calculus of Communicating Systems) [96].
- ACP(Algebra of Communicating Processes) [19].
- LOTOS [67].

Net-Based Approach.

Graphical notations are popular notations for specifying systems as they are easier to comprehend and, hence, more accessible to nonspecialists. In this approach, graphical languages with a formal semantics are used, which bring special advantages in system development and re-engineering. Below some popular Net-Based formalisms have been listed:

- Petri Net [112].
- Statecharts [133].

Comprehensive comparison of different formal methods has been presented in [81, 80]. The results have been illustrated in the form of some tables which for each specific method shows to what extend it could be suited with respect to some given criteria like automated tool support, reliability, industrial strength and so on. Based on this study the B-Method is very good in the term of automated tool support and reliability with real industrial applications (such as the Paris Metro Line 14). As we consider the use of B-Method for our system modelling in the next section we present a more in depth discussion about it.

Here we should emphasise that providing a universal classification of formal methods is difficult because there is a wide range of possible criteria upon which the classification could be based. Any analysis may apply a different set of criteria. Possibly one useful criterion is the domain applicability of a method, but the available data on the application of formal methods is scant and poorly coordinated. Another attempt to provide a classification based on the theoretical basis of methods could be found in [14].

Another common classification of formal approaches from behavioural viewpoint is to partition them into *state-based* (e.g. B-Method, TLA or Z), usually rooted in logics, and *event-based* (e.g. CSP, CCS or LOTOS), with algebraic roots [1, 29, 28]. In the following paragraphs definitions of these two different approaches have been provided:

State-based: The focus is on capturing the system state at the right level of abstraction. In a state-based approach, an execution of a system is viewed as a sequence of states, where a state is an assignment of values to some set of components. The model of the system should also include an initial state and a precondition for each operation.

Event-based: The focus is on identifying all the relevant events of the system and then describing in what order these events are allowed to happen. To specify an event-based system, we must determine: the collection of events relevant to the system; the initial enabled events of the system (events which are immediately possible). Event-based approaches are suitable for modeling distributed and concurrent systems such as mail servers, telephony, communication protocols etc. The event-based style has become prevalent for large-scale distributed applications due to the inherent loose coupling of the participants. It facilitates the clear separation of communication from computation and carries the potential for easy integration of autonomous, heterogeneous components into complex systems that are easy to evolve and scale [48].

It is well-known that the two frameworks are interchangeable [1, 29]. For instance, an action can be encoded as a change in state variables, and likewise one can model a state change with different actions to reflect different values of its internal variables. However, converting from one representation to the other often leads to a significant enlargement of the state space. Moreover, neither approach on its own is practical when it comes to modular software, in which actions are often data-dependent: considerable domain expertise is then required to annotate the program and to specify proper claims.

In our approach we consider the B-Method as a Model-Based method. Considering the fact that we have two different version of the B-Method known as *standard B* and *Event-B* [8, 38] provides a sound justification for this decision. We will provide more details about these versions in the latter sections.

2.4 An overview of the B Method

The B method, invented by J.-R. Abrial [2], is a Model-based method built on set theory and predicate logic and extended by generalized substitutions. Classical B was initially developed for specifying, designing and coding software systems. The Specification, which is represented by an abstract machine, could be refined in a stepwise manor to produce a more concrete models. A machine encapsulates operations and state, the latter being determined by a set of variables. Development proceeds in a layered fashion, where higher level specifications are implemented using lower level constructs. Generalised Substitutions are used to describe state modifications, the refinement calculus is used to relate models at different levels of abstraction, and there are a number of structuring mechanisms like machine, refinement and implementation which are used to construct the softwares in a layered faction. The first version of the B method is extensively described in The B-Book [2].

Atelier-B [15] and the B-Toolkit [16] are two development environments that support the B Method. There are some differences between the two implementations of the B which these tools support. These tools provide mechanised support for syntax and semantic analysis, type checking, and mechanised theorem based proof checking. Therefore with reference to the classification given in the beginning of this chapter, B is classified as level 3 of formalism.

The B-Method is one of the most recently developed formal methods which it has several important features that distinguish it from other formalisms. Some of these features can be enumerated as following:

- The B-method is a mathematical method. It is based on logic and set theory. At the highest-level the system can be specified using Abstract Machine Notation(AMN) and generalised substitution statements. Formal methods utilize either a property-oriented or model-oriented approach and have different levels of rigor. Model-oriented formal methods specify system behaviour by the construction of a mathematical model with an underlying state (data) and a collection of operations on that states. As we mentioned earlier the B-method is belongs to the Model-Oriented approach of formal modelling. Therefore the B-method is very suitable for system-level modelling.
- The B-Method has a rich set of notations that preserve simplicity and readability. In particular, there is no real distinction between the specification notation and the programming notation. In B the specification notation is a restricted subset of whole set notation.
- In B, a specification is an abstract mathematical model of the required behaviour of a system. The abstract specification will then be transformed through a sequence

of formally defined refinement steps toward a concrete implementation. During the refinement process there is a number of proof obligations that must be discharged. In addition separate techniques are provided within the method to support the development of large systems specifications.

- As we highlighted before, the B-method is supported by two tools. These tools provide system developers the facility to develop the model using constructs that are described by precise mathematical theories. These models capture the behaviour in a complete application domain. As specification is developed into implementation, the tools can produce proof obligations that basically describe the complete set of tests that confirm that the behaviour of the specification and the design are consistent with implementation. Therefore, discharging the proof obligation is the counterpart of testing in other engineering disciplines and it dominates through the complete domain of system development rather than having testing at a single point.

The B-method has been used for modelling of different systems in both University research works and industrial developments [122, 44]. During the last years there were several attempts to apply B Method to modelling of distributed systems [31, 8]. It seems that the standard B-Method has some limitations in modelling this sort of systems. Therefore there are varieties of suggested extensions that can be apply to the standard B [5, 38].

2.5 Extending the B method

The B method was initially designed for formal software development and has been successfully used in many cases. With the increasing complexity of systems, and bearing in mind the fact that these systems are mostly distributed, adaptation of B method for system modelling has been considered by many researchers. A number of these research are presented here very briefly.

Another area of working with the B-method is to combine it with other formalism and add some useful notation to it. Since diagrammatic notations offer a visual presentation of systems and it could be quite useful in early stages of a new system development to have an overall understanding of the whole system; therefore these type of notation have a wide spared usage especially in industrial application. Among the graphical notations, [129] is a widely accepted notation for system and software development. Integration of B-method with UML is reported in [127, 78, 132] which is another area that we are exploring in this report.

2.5.1 Extending B with new features

Extending B and introducing dynamic constraints in B where presented in [5, 3] is one of the most important attempts to introduce new constructs into the B-method to make it much more suitable for distributed systems modelling. The main idea conveyed in the above mentioned papers is that B abstract machines, normally used to specify and develop software modules, might also be used to model the evolution of the *global* system's state. In this approach a machine can model a complete networked system. The *operations* of this machine represent the events in the system, which can occur *spontaneously* rather that being invoked, as is the case with a conventional abstract machine operation. Some events could even model communication between different agents situated in various parts of the network.

Instead of pre-condition operation, guarded events with certain predicates have been suggested. Each guard explicitly states the condition under which the related event can be enabled. Then within any time interval one of the enabled events could be selected and executed. Finally, new events, which were not explicitly present in an abstract specification, could be introduced in the successive refinement. Such events are all supposed to refine the *skip-operation* in the abstract model. The gradual introduction of new events in successive refinement steps makes it possible to develop a system by starting from a single abstract machine and ending-up eventually with a completely concrete distributed realisation.

Dynamic constraints such as liveness constraints, deadlock-freedom and eventuality properties are the essential issues in event-based systems. For handling these types of constraints some new clauses like *Modalities* and *variant* have been introduced. Furthermore refinement and decomposition are the main strategies for tackling the problem of complexity. The main difficulty in decomposition of a system to a number of subsystems is variable splitting. Suppose that we have some variables in the original model which are being used in more than one event. During the splitting process these shared variables could be a source of problems regarding the sub-system consistency. Abrial has suggested variable sharing during system decomposition and event splitting in [9].

The Event-B approach has been considered as an imperative method for describing distributed systems in [6, 7]. A distributed system contains a number of concurrent components, each of them being subjected to many transactions occurring, most of the time asynchronously. Among these components some of them are supposed to control other parts. Such control parts need to accumulate some information about the overall status of the whole system. Information need to be transmitted to or from control components through the communication links which are not instantaneous.

Further assumptions on the execution of events have been made. First of all, the execution of an event, which describes certain observable transaction of the state variables, is considered to take no time. As an immediate consequence, no two events of the same machine can occur simultaneously. When more than one guarded event is enable, then non-deterministically one of the enabled events can be executed and when no event is enable, then the machine execution stops.

2.5.2 Action-system based approach to distributed system

One of the early attempts at a formal presentation of a distributed system with B-AMN was introduced in [33]. The approach presented in that paper used an event-based view of action systems. More precisely the author has shown the similarities between B-AMN and action systems. This includes the fact that both approaches could be specified as a model which consists of some states and some operations acting on those states. More significantly it has been presented how reactive refinement and decomposition of action systems could be applied to abstract machines in the B-method. The approach can fit very closely with the stepwise refinement method of B where a single specification can be refined into several concrete models. Introducing so-called new internal events is the main practice before the decomposition stage.

In [34] the author has shown that despite the fact that a distributed system consists of different parts like control, communication link and so on, we could start the formal specification with a simple single model. As a next step by means of stepwise refinement we make this abstract model more detailed and concrete. Refining a model consists of refining its state's variables and its events. A refined model should present a much more concrete and accurate behaviour that the abstract one. The state of a refined model is linked to the state of the abstract model through *gluing invariants*. Each event of the abstract model is refined into a corresponding event of the concrete one. Another frequent method of refining an event system consists of adding a new event in the refined model which in that case the new event will refine a skip operation in the abstract model. The new events that are introduced at some level should maintain some specific constraints. For example they should not take control for ever.

Clearly, that model may now be quite large and thus difficult to develop, because it may have a lot of state variables and events incorporated with these state variables. That is the right moment to envisage decomposing the model into several sub-models. Decomposition is the best way to mastering the inherited complexity of distributed systems. A natural decomposition is clearly is one where we have a sub-model for each physical part, communication link and control unit. We could have several control unit as well as several physical parts communicating with each other. The role of the decomposition is clear. Once a sub-model separated from the main body it could be refined further independently from the rest of the system.

In [33] some novel ideas have been provided for system decomposition. These ideas are mostly inspired by the process algebra approach like CSP (Communicating Sequential Processes) [65]. For better understanding of this style of decomposition it seem to be necessary to have a quick review of related concepts in CSP.

In CSP, systems are modelled with some *Processes*. Interaction between a system and its environment is represented with some *events*. The set of all events which a process could engage is called the alphabet of the process. The behaviour of a process is specified in term of a small set of algebraic notation like prefixing ($\rightarrow$) and the sequence of events. In CSP notation two processes can be composed in parallel. Parallel composition of two processes P and Q is written as $P \| Q$. The two composed processes interact by synchronisation over shared events. For describing communication between CSP processes and their environments we use the notion of channel. There are two forms of channel, input and output channels. A channel named c is represented by a set of events in the form $c.i$. Occurrence of an event $c.i$ represents communication of value i over channel c. A process can accept an input value over input channel c and we use the notion of $c?x$ for it, or it can offer to its environment an output value over channel c and we use the notion of $c!x$ for it. Now by parallel composition of two processes with one shared event name in each of them, which should be an input event in the first process and an output event in the second process, we can model the value-passing communication between the two processes.

Based on the above mentioned strategy in CSP the concept of synchronisation between different subsystems based on shared event rather than shared variables has been introduced by Butler in [33, 32]. Unlike the initial style of events suggested by Abrial, in the Butler style events that are involved in communication can have input and output parameters. Decomposition based on value passing communication could ease the refinement process of distributed systems.

2.5.3 UML and B-method Integration The Unified Modelling Language

UML has become a de-facto standard notation for describing, analysing and designing object-oriented software systems. The graphical description of models helps developers and their customers to easily grasp the general structure of the modelled software and thus have a good basis for discussing user requirements and their possible implementation. However, the fact that UML suffers from the lack of a precise semantics is a major drawback of UML models for critical system modelling. On the other hand B is a formal software development method that covers the whole software process from abstract specification to the final implementation with good tools support like Atelier-B and B-Toolkit. These tools provide a framework for animation and mechanised proof. But in practice it could be difficult to learn and use B, especially for communicating with ordinary users. An appropriate combination of UML and B can give rise to a practical and rigorous software development. A promising approach is to start the modelling task with some UML specifications like UML use-case or class diagrams and derive B specification from them for formal proof and verification. Upon detecting a defect in the B model we can go back to the UML model and amend it. Several works related to the derivation of B specification from UML models have been reported in [127, 78, 132]. A tool for translating from UML to B [126] is available. The modelling could be done by the use of class diagrams and state-charts and it produces the appropriate B specification.

2.6 Formal Development of Web-based Applications and Related Works

There has been some limited work on specific approaches to formally representing certain aspects of Web applications, though this has tended to focus again on content and navigational issues to the exclusion of functionality. For example, Hadez [55, 54] looks at the use of formal methods (using the Z notation) to specify conceptual, structural and perspective schemas. Other approaches have focused on specification of timing constraints [40, 108] rather than content structure. Again, however, the focus is very narrow and fails to couple the specifications with broader application requirements.

In the next chapter we consider the application of B Method for formal specification of Web-based application with emphasis on operational aspect of applications. As we mentioned in beginning of this chapter, many of current web applications tend to be mission and business critical system. Therefore we believe that formal development could provide a sound bases for developing appropriate Web applications.

Chapter 3

Some Guidelines for Formal Development of Web-based Applications in the B-Method

3.1 An Introduction to Web-Based Systems

Web-based applications are distributed systems that can be accessed using a Web browser. During recent years the extent and scope of their use has grown rapidly, significantly affecting all aspects of our lives. Industries such as manufacturing, travel and hospitality, banking, education, and government are Web-enabled to improve and enhance their operations. E-commerce has expanded quickly, cutting across national boundaries. Even traditional legacy systems have migrated to the Web. The scope and complexity of current Web applications varies widely: from small-scale, short-lived services to large-scale enterprise applications distributed across the Internet, and corporate intranets and extranets.

Although numerous Web-based systems are in use now and many of us rely on them, the manner in which they are developed raises serious concerns [101, 116, 138]; they need to be reliable and perform well. To build such systems, Web-based system developers need a sound methodology, a disciplined process and a set of good guidelines. Due to the new demands, Web applications are evolving continually and the complexity of these systems is increasing rapidly. Therefore the use of a rigorous method becomes more important.

Formal methods use mathematical notation to describe systems in a clear and rigorous manner. Abstraction and stepwise refinement employed by formal methods is a valuable approach for developing complex Web-based systems. The B-Method is a well-known formal method [2] which has been applied to several software development missions including academic and industrial projects [122, 44, 31].

Our aim in this chapter, through the modelling of this specimen Web-based system, is to identify some challenging aspects of these types of systems and propose an approach to their formal representation. We hope to provide a set of guidelines which could serve as a basis for further work. In the rest of this chapter we present the travel agency case study and briefly discuss its initial aims and objectives. The chosen case study has been selected to be inclusive enough to represent the main properties and functionality of typical Web applications. By developing formal models in B we have extracted some generic and essential patterns. These patterns are considered to model some common properties and functionality shared by a broad category of Web applications. In the next step we have tried to find some appropriate formal refinements for these abstract patterns which could be provable within the framework of the B prover tool [4, 15, 17]. As Web applications are distributed systems, the decomposition of primary refinement models into subsystems and introducing suitable formal models for communication links are other objectives. The last section concludes the chapter with recommendations for further work and discussions.

3.2 Informal Representation of the Case Study

In this section we outline the main requirements and sketch the overall architecture of the system. The aim is to develop a Web-based Travel Agency system to enable potential users to access it through an Internet connection using a standard Web browser to perform one or more of the following tasks:

- Book a flight
- Cancel a booked flight
- Book a room in a hotel
- Cancel a booked room
- Hire a car
- Cancel a hired car

The Travel Agency Web-based system is hosted on the Travel Agency Server which is responsible for processing the Web-clients' requests. These messages are produced and sent by the client browser through Internet links and based on HTTP or other similar standards. The travel agency system relies on a group of secondary agencies' servers like flight agencies to accomplish the client requests. The travel agency system use Internet links to communicate with the secondary servers. A simple architecture of this system is depicted in Figure 3.1.

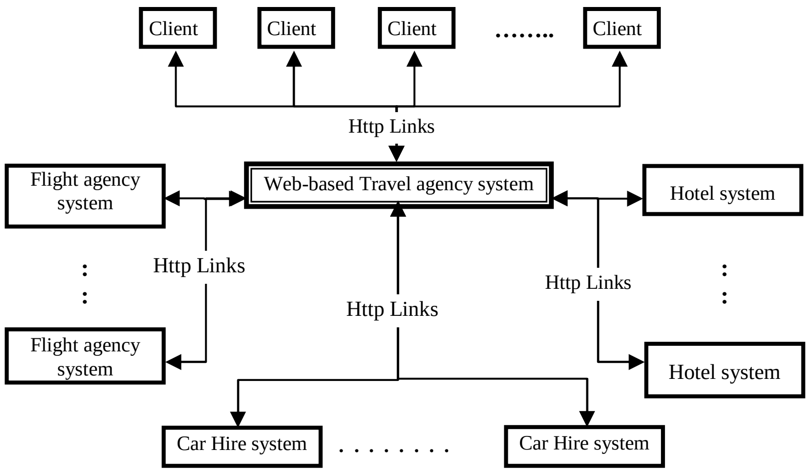


FIGURE 3.1: A Simple Architecture of the System.

It is apparent from Figure 3.1 that more than one client could communicate with the travel agency system simultaneously. The travel agency system will manage the status of different sessions by some state variables, stored in a local database. For booking requests like flight booking, a message which includes necessary details about the request will be broadcast to all related agencies servers by travel agency system. The number of responses which the travel agency should expect could vary from zero to the number of all secondary agencies in the best situation. The collected response will be sent by the travel agency system to the appropriate client. In some other cases, like cancelling a previously booked flight, the request will be sent directly to the related flight agency. Also it is quite convenient to assume a local database in the travel agency server for representing all booked services. This database could reduce the amount of communication and complexity of un-booking process.

3.3 An Overview of the Formal Development Process

As we mentioned previously our main objective, in applying formal method to this case study, was to identify some common challenging issues and propose some formal models for them. Therefore instead of detailed presentation of formal models, in this section we have summarised the formal development process.

This work is based on the Event-B style for development of distributed systems [3, 5]. Unlike standard B, which is used to specify and develop software modules in B, Event-B was introduced for modelling of distributed systems. In the Event-B style operations are then called "events" which may occur spontaneously rather than being invoked. Those events are no longer pre-conditioned, but guarded by a predicate, which express the condition under which the event can be enabled. When we refine a model, we can either refine an existing event by strengthening the guard or/and the before-after predicate (removing non-determinism or applying data refinement), or add a new event which is supposed to refine the skip event. The introduction of new events is supported by the superposition method [135, 18]. In superposition refinement, some new functionality is added to an existing model in the form of additional variables and assignments to these variables as new operations, while the original computation is preserved.

In the first stage of formal process an abstract model based on Event-B style has been produced. The abstract model is a single B-machine which encloses some operations to model the main functionality of the travel agency system from the viewpoint of the users. In the second step we have refined the abstract specification by introducing client side operations based on the superposition methodology. Operations of the abstract model have been classified as the server side operations at this stage. Some operations of the abstract model which are influenced by the introduction of client operations have been refined by adding extra guards and removing non-determinism. A full list of this model is presented in Appendix A.

Operations of the secondary agencies servers have been introduced in the second refinement model. In this stage some formal definitions for distributed databases have been added. In fact each secondary server has a local database which contains information about available service that this agency can offer to its costumers. Data distribution among secondary servers and the travel agency system leads to distribution of processing between different servers. In other words, introducing new operations which finally reside on secondary servers for manipulating distributed data resulted in further refinement of the travel agency operations in this stage. Now in the second refinement we have operations of the clients, the travel agency system and the secondary servers.

Decomposition is the main strategy to tackle the complexity of the model in Event-B style. Introducing communication links between different parts is a pre-stage to the decomposition process. Therefore in the third refinement stage we have introduced communication links. The main challenging questions which we have identified during the above mentioned development processes are:

- Session and State Management in Both Client and Server side
- Inter-Server Interactions
- Refinement of Complex Data types
- Abstraction of Distributed Databases
- Formal Modelling of Communications Links

In the following sections we have examined these issues in detail and we have presented some solutions for them. Although we have used the travel agency case study to discuss the main properties of a Web application and to clarify the key issues in developing a

B-model for them, the identified aspects and proposed solutions could be applied to a wide range of Web applications.

3.4 State Representation in Web-based Systems

The Web started as a means for sharing documents among scientists. Its designers have built the underlying technology (e.g., HTTP and HTML) with these goals in mind. Since then, people have realised the Web's potential as an application delivery medium and have started to exploit it. With the growth of e-business applications, the Web is rapidly being transformed into an application-intensive environment. In Web-based applications the core functionality of system, the business logic, is handled by the server. Most web applications need to maintain communication sessions with their client, and monitor each client's individual status and activities. The communication protocol between web browser and web server (HTTP) is stateless and it does not provide the functionality on session control. Therefore it is not trivial to maintain information about each client interaction with server. The server-centric architecture of current Web applications makes a server-side session the natural choice. In the following sections we have examined this subject in detail.

3.4.1 Session Handling and State Management in Server side

State maintenance is one of the major issues in many applications, such as e-commerce and banking applications. As transactions between Web clients and Web servers occur in a stateless environment, state must somehow be passed from one transaction to the next in a Web application. Keeping state data on the server side is generally considered the safest and most appropriate technique when handling information of a sensitive nature.

The server uses a session's state variables to identify a user, process the input data provided by a client and determine user rights or the type of access to be offered to a user. Furthermore, based on the information which has been provided by the client, the server can set state variables to determine the next possible execution path.

Challenge: How do you represent the state information related to a user's interaction with a Web application?

Guideline: We have used explicit state variables to represent session state information on the server side. By defining two reference sets for *state* and *session-ID* and a mapping function from a *session-ID* to session *state* we can manage each session in the server side identically. In other words each session has a session identifier *sid* which could be used as an index to access session information on the server side. A new *sid* could be allocated to a new client as soon as the server received the first request from this client and afterward the client can use this *sid* on subsequent interactions.

To clarify the guideline we have presented a snapshot of the specification machine for the Travel agency case study in Figure 3.2. We have introduced the set *STATE* and *SESSION*. The first definition represents the possible states for a client session and the second one serves as a typing reference for sessions' ID. The *session_state* variable maps each client session to its related state. The variable *session* represents the set of all current active sessions. The operation *StartNewSession* models the creation of a new session by the travel agency system. This operation allocates a free session ID for the newly created session and sets the necessary environmental variables for it. Any changes in a session's state variable could enable a operation and execution of an operation could resulted in some changes in a state variable. For example, the *SelectService* operation is enabled when the session state is *fresh* and its execution changes the state of related session to one of *booking*, *unbooking* or *signed_in* state. The *SelectService* operation models the interaction of the clients with the system, when they select an available service.

```
MACHINE TravelAgency
SETS
  SESSION;
STATE={fresh,booking,unbooking,service_selct,options_ret,choice_made,
         signed_in,certified,valid,invalid,booking_ret,unbooked_sel};
DEFINITIONS
  freshSESSION ≙ SESSION - session;
VARIABLES
  session, session_state,
INVARIANT
  session ⊆ SESSION ∧ session_state∈ session → STATE ∧ ...
INITIALISATION
  session := ∅ || session_state := ∅ || ...
OPERATIONS
 StartNewSession ≙
  ANY sid WHERE sid∈ freshSESSION THEN
    session := session ∪ {sid} ||
    session_state(sid) := fresh
  END;
 SelectService ≙
  ANY sid WHERE sid∈session ∧ session_state(sid)=fresh THEN
    SELECT (……. ) THEN
      session_state(sid):= booking
    WHEN (…….) THEN
      session_state(sid):= unbooking
    WHEN (…….)THEN
        session_state(sid):= signed_in
    END || ...
  END;
 FlightRequest ≙
  ANY sid WHERE sid∈ session ∧ session_state(sid)= booking THEN
    session_state(sid):= service_selct
  END;
```

FIGURE 3.2: Abstract model of the travel agency system.

3.4.2 State Management in client side

In Web based applications Web clients generally are classified as thin clients. This implies that processing in the client side usually is not significant. Web clients take input from users, perform type checking and simple data validation and in some cases carry out data encryption if necessary. Web clients use the application through Web browsers, over the Internet. They interact with system concurrently, independently, in an asynchronous manner. You can't control what they're doing and when they do it. Although the browser and underlying mechanism do not support state handling, still some coordination mechanism and state passing between server and client operations is necessary.

Challenge: How do you maintain the state information in the client side and perform coordination between different clients and the Web server.

Guideline: We have used a message-based mechanism for this purpose. Each message is mapped to a session ID which relates the message to a specific client session. The message-based mechanism could be considered as an implicit state representation in the client side. Therefore from this viewpoint we can assume that two different approaches have been taken for state representation in the server and the client side. We have found that the main advantage of this approach is to avoid shared state variables among clients and the Web server which in its turn could lead to further complication.

We have presented some operation of the clients along with the server's operations from first refinement of the case study in Figure 3.3 to illustrate the guideline. We have used comments to make a distinction between the server and newly introduced client's operations. The server operations use explicit state variables for state representation. On the other hand, the client operations employ an implicit message-based method for state representation and coordination with the server operations.

The session ID, *sid*, plays a central role to convey state information between client and server. However there is a situation that a client has triggered a new session but it has not obtained a session ID yet. In this step the client should use a temporary identification mechanism which for example could be the IP address plus some extra information. The *Client_ReqSession* operation in Figure 3.3 depicts this situation. We have defined a new variable named *handle* to use it as temporary index to represent a client request for a new session. When in the *StartNewSession* operation the server has processed this request it allocates a new session ID for this specific client session and replies to the client by placing the new session ID in the *new_client* message buffer. In the *Get_SessionID* operation the client receives this allocated *sid* and it will use it through the rest of session to communicate with the travel agency server. For example in the *PicService* operation we have a message buffer named *reqsevice_buf* which has been defined as a mapping from *session* to *REQUEST* to carry the client's requests to the Server.

As we have mentioned in section 3, we have used superposition refinement to introduce client operation. This means that we retain the variables and operations of the abstract specification and introduce new operations which have no effect on the previous variables. Some new variables which can be exploited by both the clients' operations as well as the Web server have been introduced in this stage. New variables are used as message buffers to exchange data between client and server operations. The introduction of these new variables has some implication on the Web server's operations.

In the abstract model some operations use nondeterministically chosen input values which need to satisfy just some typing and basic state conditions. In the refinement model some changes have been made in the operations' guards. These are related to the refinement of the nondeterministic choice of input parameters to the values in the related message buffers which are provided by clients. As we use superposition refinement in this stage, we do not require any gluing invariant which implies an easier set of proof obligations.

3.4.3 Conducting Inter-Server Interactions

Coordination and communication management is an important issue in modelling interactions between two or more servers. In the case of inter-server communications, unlike client and server communication, both parties which are involved in a session are providing some services. Interaction between the travel agency system and secondary servers is an example of such inter-server communication. For example the travel agency system can ask a flight Agency server for available flight options and the flight agency server will reply with available options.

Challenge: What is the best way to model inter-server interactions?

Guideline: Considering the fact that the servers are independent, any approach to modelling their interaction, should provide a solution with minimum possible cohesion between these subsystems. Using the message-based approach seems to be a good candidate for this purpose and furthermore it complies with common web services technologies.The messages are defined as mapping from a session ID to the requested information.

The message-based approach could be exploited to exchange both data and state information between servers. As server to server communications are mostly asynchronous, message-based communication is an appropriate candidate.

```
SETS
   HANDLE
DEFINITIONS
 freshHANDLE ≘ HANDLE - dom(new_client)
INVARIANT
 /*  Client Variables  */
   new_handle ⊆ HANDLE ∧ new_client ∈ HANDLE ⤔ SESSION ∧
   token ⊆ SESSION ∧ fresh_session ⊆ SESSION ∧
   reqsevice_buf ∈ SESSION ⇸ REQUEST
OPERATIONS
 Client_ReqSession ≘      /* Client Operation */
   ANY handle WHERE handle ∈ freshHANDLE THEN
     new_handle:= new_handle ∪{handle}
   END;
 StartNewSession ≘      /* Server Operation */
  ANY sid, handle WHERE sid ∈ freshSESSION ∧ handle ∈ new_handle  THEN
    session:= session ∪ {sid} ||
     session_state(sid) := fresh ||
     new_client(handle):= sid ||
    new_handle:= new_handle - {handle}
  END;
Get_SessionID ≘      /* Client Operation */
   ANY sid WHERE sid ∈ SESSION ∧ sid ∈ ran(new_client) THEN
     token:= token ∪ {sid} ||
     fresh_session:= fresh_session ∪ {sid} ||
     new_client:= new_client ⩥{sid}
   END;
PicService ≘      /* Client Operation */
  ANY sid, req WHERE sid ∈ fresh_session ∧ req ∈ REQUEST  THEN
      reqsevice_buf(sid):= req ||
      fresh_session:= fresh_session - {sid}
  END;
```

FIGURE 3.3: Some operation of the first refinement.

Some operations of the secondary servers and the travel agency system which involve communication are presented in Figure 3.4. In this model, *reqflight_buf* is used to transmit requests from the travel agency to flight agencies. Flight agencies use *respflight_buf* message buffer to send responses to the travel agency.

3.5 Abstraction and Refinement of Complex Data-types

In many Web applications we frequently need to represent some complex data types in different abstraction levels. For example this data could be a record with many fields containing all necessary information for a booking request. Refining abstract data types in a single step, especially when we do not need all details in this step, is not a good approach to refinement; because it swiftly turns our simple abstract model into an over-complicated refined model. Therefore we need to find a mechanism for stepwise refinement of the abstract data types.

```
INVARIANT
   /* Server's New Variables */
   reqflight_buf ∈ FLIGHT_AGENCY ⇸ (SESSION ⇸ FLIGHT_REQUEST) ∧
   /* Flight Agency Variables */
   respflight_buf ∈ SESSION ⇸ (FLIGHT_AGENCY ⇸ ℙ(FLIGHT_DETAIL))
OPERATIONS
Request_Flight ≙       /* Server Operation */
  ANY sid,fr WHERE
   sid ∈ SESSION ∧
   fr ∈ FLIGHT_REQUEST
  THEN
   reqflight_buf:= λ fa . (fa∈ FLIGHT_AGENCY | reqflight_buf(fa) ∪{sid↦fr})
  END;
Resp_FlightReqs ≙       /* Flight Agency Server Operation */
  ANY sid,fa,fr, xx WHERE
   sid ∈ session ∧
   fa ∈FLIGHT_AGENCY ∧
   fr ∈ FLIGHT_REQUEST
   (sid ↦ fr) ∈. reqflight_buf(fa)
   xx ∈ ℙ(FLIGHT_DETAIL) ∧
   xx ⊆ Matchflight(fr ↦ flight_db1(fa))
  THEN
    respflight_buf(sid):= respflight_buf(sid) ∪ {fa ↦ xx}  ||
    reqflight_buf(fa):= reqflight_buf(fa)- {sid ↦ fr}
   END;
```

FIGURE 3.4: Some operations of the secondary servers.

Challenge: What is a proper abstraction for data structures like records and how can we refine an abstract representation of a record in a step-wise manner?

Guideline: We found that most details could be abstracted away by defining some simple data types in the form of set definitions in the specification level. In refinement stage to overcome the problem of unnecessary detail we found that, instead of direct refinement of abstract data types, some constant mapping could be used. A mapping defines a relation from an abstract data type to the required additional detail. By employing this method we introduce fields into the refined model when it is necessary.

The abstract data types make operations very simple and understandable at the specification level and help us to have a clearer picture of the overall functionality of the system. But we need to introduce the necessary details into these abstract data types in the refinement level. Using constant mappings to introduce new fields of a previously defined abstract type could help to avoid unnecessary complication in the early stage of refinement and postpone the detailed refinement of abstract data types to after decomposition.

Using constant mapping to refine an abstract record may present some ambiguity to the reader. So we will try to make some clarification here. Let assume that we have an abstract record, REC in the specification level. We want to refine this abstract record by introducing two new fields of it, namely *afield* and *bfield*. We can define these two fields as a constant mapping from REC to two arbitrary types $SETA$ and $SETB$ respectively. Now we can assert that for any aa and bb such that aa belongs to $SETA$ and bb belongs to $SETB$ we can define a record that belongs to REC. Performing record refinement with a constant mapping rather than a variable mapping simply means that this information is global to all subsystems. Using constant mapping does not have any restrictive impact on records manipulation. To clarify this issue we have presented an example operation in Figure 3.5 that adds a record to a database.

```
MACHINE Database
SETS
   REC; SETA; SETB
CONSTANTS
  afield, bfield
PROPERTIES
  afield ∈ REC→SETA  ∧ bfield ∈ REC→SETB ∧
  ∀ (aa, bb•((aa ∈ SETA ∧ bb ∈ SETB) ⇒
  ∃ rr. (rr ∈ REC ∧ afield(rr) = aa ∧ bfield(rr) = bb)))
VARIABLES
  db
INVARIANT
  db ∈ ℙ(REC)
INITIALISATION
   db:=∅
OPERATIONS
 Add_Database ≘
  ANY af,bf,rn WHERE
   af ∈ SETA ∧
   bf ∈ SETB ∧
   rn ∈ REC ∧
   afield(rn)=af ∧ bfield(rn) = bf
  THEN
    db:= db ∪ {rn}
  END
END
```

FIGURE 3.5: An example of constant mapping.

An example from the case study is provided in Figure 3.6. We have two abstract data types; the first one is an abstraction for a record which contains all the necessary information for a flight request and the second one is the abstraction of a record which contains all details about an offered flight by a flight agency. We have used two abstract set definitions *FLIGHT_REQUEST* and *FLIGHT_DETAIL* for these two abstract records respectively. In the refinement stage we need to access the flight agency that has provided a flight. We assume that the flight agency identifier is a part of the *FLIGHT_DETAIL* record. Instead of direct refinement of the abstract data type, we have defined a constant mapping from *FLIGHT_DETAIL* to *FLIGHT_AGENCY* which could satisfy our requirement in this stage. The definition of this constant mapping is presented in Figure 3.6. The use of a constant function provides a way of modelling a record's field in B.

```
SETS
  FLIGHT_REQUEST; FLIGHT_DETAIL; FLIGHT_AGENCY;
CONSTANTS
  flightagency
PROPERTIES
  flightagency ∈ FLIGHT_ DETAIL → FLIGHT_AGENCY
```

Figure 3.6: An example of constant mapping from the case study.

By using similar techniques we are able to introduce any extra detail which might be necessary in successive refinement steps. Obviously at the implementation stage we have to replace these constant mapping with an actual data field; but the fact that we could postpone this step until after decomposition is helpful.

3.6 Abstraction and Refinement of Distributed Databases

Data that is shared between Web components and persistent between invocations of a Web application is usually maintained by one or more databases. These databases generally are distributed over different servers. Developing a formal abstract model and refinement for them is another challenge that we examine in this section. This issue has a close relation with process distribution; therefore we consider process distribution and distributed databases together.

We can assume different functionalities for a database system. For example the simplest case is a database which allows its contents to be viewed by different parts of the Web application. On the other hand a complex database could support different type of queries and permits updating current information or removing some records from it. As the system is distributed it means that when a server makes some changes in its database which could affects another part of the Web application, it takes some time for the other part to know about it.

Challenge: How we can represent a proper abstraction and refinement of certain distributed database operations?

Guideline: In a distributed setting involving multiple clients, the high level specification of a transaction such as confirming a flight booking needs to include the possibility of failure. Also query operations involving multiple databases should be specified very loosely at the abstract level.

To understand the complicated relation between process and Database from refinement viewpoint we need some examples. In the travel agency system as depicted in Figure 3.1 we have a set of secondary servers which store some information about their available services. Based on web clients' requests the travel agency server occasionally initiates and sends a distributed query to these secondary servers for information lookup. Later it should collect and send available services to related Web clients. Obviously in the specification level we need an abstract formal representation of these distributed processes and databases.

The first abstract model is presented in Figure 3.7. In this specification *Matchflight* is a constant function type definition. It takes *FLIGHT_REQUEST* as an abstraction for user request and an abstract database which contains some *FLIGHT_DETAIL* records and returns a set of *FLIGHT_DETAIL* records which match the user request.

```
CONSTANTS
 Matchflight
PROPERTIES
 Matchflight∈ FLIGHT_REQUEST × ℙ(FLIGHT_DETAIL)→ℙ(FLIGHT_DETAIL)
INVARIANT
 flight_db ∈ ℙ(FLIGHT_DETAIL) ∧
 flight_option ∈ SESSION⇸ℙ(FLIGHT_DETAIL)

Retrieve_FlightOptions≘       /* Server Operation */
 ANY sid, fr, xx WHERE
       sid ∈ session ∧
       fr ∈ FLIGHT_REQUEST  ∧
       xx ∈ ℙ(FLIGHT_DETAIL) ∧
       xx ⊆ Matchflight(fr ↦flight_db)
 THEN
   flight_options(sid) := xx
 END;
```

FIGURE 3.7: An abstract model of the database operation.

In this abstract model we have defined the virtual database, *flight_db*, as an abstract representation for a set of distributed databases which reside on secondary servers. As we mentioned earlier the content of these distributed databases could change independently from the travel agency system. Based on the above assumption we have defined the operation *Retrieve_FlightOptions* which is an abstraction for collecting secondary servers' responses to a distributed query for a service. Obviously we have not introduced secondary servers and their related databases in the abstract model to avoid making the model over-complicated.

An intended refinement of the abstract model is presented in Figure 3.8. In this refinement based on the superposition technique we have introduced some new operations. The *Request_Flight* operation models the travel agency side event that initiates a query broadcast to a set of secondary servers. Equally when a secondary server receives a query for a service, it responds if it has any available option(s). This is demonstrated in *Resp_FlightReqs* operation. The virtual database definition has been replaced by actual databases which are distributed among secondary servers and we have defined these by a mapping from *FLIGHT_AGENCY* to power set of *FLIGHT_DETAIL*. The *Retrieve_FlightOptions* has been refined in response to the introduction of the new operations and now clearly reflects the fact that it should collect the secondary servers' responses to reply the initial service query.

```
CONSTANTS
  Matchflight
PROPERTIES
 Matchflight ∈ FLIGHT_REQUEST × ℙ(FLIGHT_DETAIL) → ℙ(FLIGHT_DETAIL)
INVARIANT
   flight_db1 ∈ FLIGHT_AGENCY → ℙ(FLIGHT_DETAIL) ∧
   reqflight_buf ∈ FLIGHT_AGENCY ⇸( SESSION⇸ FLIGHT_REQUEST)  ∧
   respflight_buf ∈ SESSION⇸( FLIGHT_AGENCY ⇸ℙ(FLIGHT_DETAIL)) ∧
   flight_option ∈ SESSION⇸ℙ(FLIGHT_DETAIL)

 OPERATIONS
  Request_Flight ≘           /* Server Operation */
   ANY sid,fr WHERE sid ∈ session ∧ fr ∈ FLIGHT_REQUEST THEN
     reqflight_buf:= λfa . (fa ∈ FLIGHT_AGENCY | reqflight_buf(fa) ∪ {sid↦fr})
   END;
  Resp_FlightReqs ≘         /* Flight Agency Server Operation */
   ANY sid,fa,fr, xx WHERE
     sid ∈ SESSION ∧
     fa ∈ FLIGHT_AGENCY ∧
     fr ∈ FLIGHT_REQUEST ∧
     xx ∈ ℙ(FLIGHT_DETAIL) ∧
     xx ⊆ Matchflight(fr ↦ flight_db (fa))
   THEN
       respflight_buf(sid):= respflight_buf(sid) ∪ {fa ↦ xx}
   END;
  Retrieve_FlightOptions ≘          /* Server Operation */
   ANY sid WHERE sid ∈ session THEN
    flight_options(sid):= ⋃ fa.(fa ∈ FLIGHT_AGENCY ∧
                      fa∈ dom(respflight_buf(sid)) | respflight_buf(sid)(fa))
   END;
```

Figure 3.8: A refinement of the database operations.

Our intention is that the abstract database is an abstraction of the union of all of the distributed databases. The response to a client request is formed from the union of the responses from each of the agencies so this may seem like a reasonable abstraction. However, we faced some difficulties when we tried to prove that the model in Figure 3.8 is a valid refinement of the abstract model in Figure 3.7. The problem is that the abstract specification of *Retrieve_FlightOptions* is based on the value of (the abstraction of) all the flight agency databases at the point at which the results are collated by the travel agency. But the results collated in the refined version will have been generated by the individual flight agencies at earlier points in time. If the flight agency databases didn't change in between the point at which they respond to a flight request and the point at which those responses are collated by the travel agency, then our refinement would be valid. However, this is clearly an unrealistic restriction. The fact that the user gets information about an available flight is no guarantee that that flight will still be available when they try to book it. In principle the value of a flight agency database at the point of generating a response might be completely different to its value at the point at which that response is collated with other responses.

One possible abstract specification for this kind of distributed database query is presented in Figure 3.9. Although it appears to be a very loose specification but it is the strongest specification that we could introduce in the abstract level. In this specification we do not use definitions like *Matchflight* and virtual database *flight_db*.

```
INVARIANT
 flight_option ∈ SESSION⇸ℙ(FLIGHT_DETAIL)
Retrieve_FlightOptions ≙      /* Server Operation */
 ANY sid , xx WHERE
  sid ∈ session  ∧
  xx : ( xx ∈ ℙ(FLIGHT_DETAIL))
 THEN
   flight_options(sid):= xx
 END;
```

FIGURE 3.9: A corrected version of abstraction in Fig 3.7

As we mentioned earlier data and process distribution have a reciprocal effect on each other. We present another scenario from the travel agency case study to clarify this issue further. During the booking process when a web client receives some available options from the travel agency system, it can select one of them and send back its selected service to the travel agency system. Now the travel agency system will know which secondary server has offered this service and then send a booking request to this specific secondary server. In the meantime this service could have been offered to another Web client and is no longer available. Therefore in general the travel agency system could expect either a successful or a failed response for a requested service booking. If the travel agency system receives a confirmation for service booking it will add an appropriate record to it local database for booked services. In either case of success or fail, it should reply to the related Web client with a suitable response.

Developing an abstract formal specification for this case is not a straightforward task. In the abstract level we have not introduced secondary servers, just to avoid complication, but we have to find a mechanism to model the system behaviour. Using nondeterministic "choice" could be an acceptable approach to model this case in the abstract level. This solution is depicted in Figure 3.10. It should be emphasised that in the actual system the booking process is a two stage process. If the requested service is still available on a specific secondary server, then the first stage takes place on that secondary server. In the second stage, when the travel agency system receives a message from this specific secondary server denoting successful booking in the first stage, then the travel agency system will add this booking to its database. Therefore the booking database on each secondary server just stores booked services which have been offered by this specific server. On the other hand the booking database on the travel agency system stores all booked services of its users. The *Flight_Booking* operation in Figure 3.10 demonstrates the booking process in the travel agency system. This operation is defined as a nondeterministic choice of two possible outcomes. The first case illustrates a successful booking, while in the second (failed) case, no booking is made.

```
INVARIANT
  session_state ∈ session → STATE ∧
  flight_booking ∈ USER ↔ FLIGHT_DETAIL ∧
  selctflight_buf ∈ SESSION ⇸ FLIGHT_DETAIL
Flight_Booking ≙
 ANY sid,fd WHERE
   sid ∈session ∧
   fd ∈ FLIGHT_DETAIL ∧
  session_state(sid)= valid ∧
  (sid ↦ fd) ∈ selctflight_buf
 THEN
  CHOICE
    flight_booking := flight_booking ∪ {session_user(sid) ↦ fd} ||
    selctflight_buf := {sid}⩤ selctflight_buf ||
    session_state(sid):= fresh ||
    session_request(sid):= none
  OR
    selctflight_buf:= {sid}⩤ selctflight_buf ||
    session_state(sid) := fresh ||
    session_request(sid):= none
  END
 END;
```

Figure 3.10: Modelling the possibility of failure.

In the refined model when we introduced databases in the secondary servers, now the booking process in the Travel agency system is no longer nondeterministic and it depend on the state of these databases. Therefore the refined operation could be modelled as we presented in Figure 3.11. Here the *Agency_flight_booking* shows the first stage of booking in the secondary server and the *Flight_Booking* has been refined accordingly.

3.7 Developing Formal Models for Communication Links

Communication links are the medium for interaction between different parts of distributed systems. In Web-based systems communication links connect a client to a Web server or a Web server to another Web server or a data server. Although communication in different levels could be based on different protocols and standards, but in general a message-based approach is a widely accepted method in Web based application. This approach is flexible and general enough to be implemented in the context of available standards like XML based technologies and tools. In Event-B developments introducing communication is an important stage before decomposition of a single model to several sub-models. In the following sections we discuss the process of developing a formal model for communication links.

```
CONSTANTS
  flght_agency
PROPERTIES
  flght_agency ∈ FLIGHT_DETAIL → FLIGHT_AGENCY
INVARIANT
   fa_booking ∈ FLIGHT_AGENCY ⇸ (USER ↔ FLIGHT_DETAIL) ∧
   flight_db1 ∈ FLIGHT_AGENCY → ℙ(FLIGHT_DETAIL) ∧
   taf_booking ∈ ℙ(USER * FLIGHT_DETAIL * FLIGHT_AGENCY) ∧
   selectflight_buf1 ∈ FLIGHT_AGENCY⇸ (SESSION ⇸ FLIGHT_DETAIL)
Agency_flight_booking ≙        /* Flight_agency Server Operation*/
  ANY fa,sid,fd WHERE fa ∈ FLIGHT_AGENCY ∧
          sid ∈ SESSION ∧ fd ∈ FLIGHT_DETAIL  THEN
    SELECT fd ∈ flight_db1(fa) THEN
      ANY fdb WHERE fdb ∈ ℙ(FLIGHT_DETAIL) ∧ fdb ⊆ flight_db1(fa) THEN
        /* Updating original Database that maybe affected by booking */
        flight_db1(fa):= fdb
      END ||
       fa_booking(fa):= fa_booking(fa) ∪ {(fd ↦ session_user(sid))} ||
       flightbookingresp(sid) := success
    WHEN fd ∉ flight_db1(fa) THEN
        flightbookingresp(sid) := failed
    END
  END;
Flight_Booking ≙       /* Server Operation */
 ANY sid,fa,fd WHERE
         sid ∈session ∧ fd ∈ FLIGHT_DETAIL ∧ fa∈ FLIGHT_AGENCY
 THEN
      SELECT sid↦success∈ flightbookingresp THEN
        taf_booking:= taf_booking ∪ {(session_user(sid) ↦fd ↦fa)} ||
        suc_session:=suc_session ∪ {sid}
      WHEN sid↦failed∈ flightbookingresp THEN
       selectflight_buf1(fa):= selectflight_buf1(fa) - {sid ↦fd} ||
       unsuc_session:=unsuc_session ∪ {sid}
      END
 END;
```

FIGURE 3.11: Refined model after introduction of secondary servers.

3.7.1 Formal models of Synchronised Communication Links

Synchronised communication is a common pattern of communication between Web clients and Web servers. In other words generally the communication between clients and the Web sever follows the send-process-receive pattern.

Challenge: What is an appropriate abstract model and refinement for communication links between clients and the Web Server?

Guideline: At the abstract level it is convenient to model communication link a one-place buffer. But this causes problems with model decomposition. So we present a pattern for refining a communication link involving a one-place buffer by an unbounded buffer.

To exemplify this issue we have presented some operation of the travel case study in Figure 3.12. We have used a function definition to present a single place buffer for data communication between each client and the travel agency server.

```
INVARIANT
  reqsevice_buf ∈ SESSION ⇸ REQUEST ∧
  resp_buf ∈ SESSION ⇸ RESPOSE
PicService ≙        /* Client Operation */
 ANY sid, req WHERE
   sid ∈ fresh_session ∧
   sid∉dom(reqsevice_buf) ∧
   req∈ REQUEST ∧ req≠ none
 THEN
   reqsevice_buf(sid):= req ||
   fresh_session:= fresh_session - {sid}
 END;
SelectService ≙        /* Server Operation */
 ANY sid, req WHERE
   sid ∈ session ∧
   req∈ REQUEST ∧
   resp∈RESPONSE ∧
   sid ∈ dom(reqsevice_buf)
 THEN
   session_request(sid):= req ||
   reqsevice_buf:= {sid}⩤ reqsevice_buf ||
   resp_buf(sid):= resp
 END;
Submit_Servic_Dtail ≙        /* Client Operation */
 ANY sid,resp WHERE
   sid ∈ dom(resp_buf) ∧
   resp ∈ RESPONSE ∧
   resp_buf(sid):= resp
 THEN
   resp_buf(sid):=  {sid}⩤ resp_buf(sid)
 END;
```

FIGURE 3.12: Abstract model with one-place buffers.

In this model the *reqsevice_buf* and *resp_buf* define a single-place buffer from *session* to *REQUEST* and *RESPONSE* respectively. The *PicService* is a client operation which puts a request in the *reqsevice_buf*. The client then waits for the server response, i.e., the first client operation is no longer enabled for this session and the second client operation is enabled when a response appears in the response buffer. On the server side the *SelectService* operation takes the request from the buffer and then produces a response for the client by placing a response in the *resp_buf*. Later the client's operation *Submit_Servic_Dtail* can take this response from buffer when received it.

Before the decomposition step we have to refine each buffer by splitting it to three buffers and distribute them between the client, the communication and the server machines. But when we replace a single-place buffer with three single-place buffers we face difficulty. We should be able to demonstrate that all buffers are empty when a web client's operation produces a new message, before placing it in its buffer. This condition arises from the gluing invariant which relates buffers in the refined model to the previous abstract model. Clearly this is not a practical solution since, for example, a client cannot see whether or not a buffer on the server side is empty. To overcome this difficulty we consider the refinement of the one-place buffers with unbounded buffers based on using sequence definition in B-method.

This intermediate refinement would help to split the buffers between different machines and without too much restriction discharges prove obligations associated with this distribution. Using unbounded buffers resolves the need for condition that distributed buffers should be empty when we add a new message.

The intermediate refinement for the above model is presented in Figure 3.13. Here the single-place buffers of Figure 3.12 have been replaced by unbounded buffers. Part of the necessary gluing invariant are illustrated as well. The gluing invariant was constructed using an iterative approach in combination with the B prover as described in [47]. We first considered the case of a single implicit session. This simplification means that the invariant has no universal quantifiers and the proof is much more automatic. We start with a trivial invariant containing type information. We then generate and attempt to prove the refinement proof obligations. Those that cannot be proved lead to a clause in the invariant. The additional invariant clauses result in further proof obligations which may in turn lead to further invariant clauses. In this case a sufficient invariant was constructed in three iterations and the proof was completely automatic (for the case without universal quantification). The invariant is then generalised to multiple sessions and the proof goes through, though not completely automatically.

The above refinement indicates that single-place buffers could be refined by multi-place buffers. The refinement works because of the request-response protocol that the client and server follow. Multi-place buffers allow having more than one message at the same time in different buffers. Although in this model message duplication is impossible due to error and delay in communication links, message duplication is very likely in reality and it could be taken in to account in later refinements. The next step refinement involves splitting each unbounded buffer into three unbounded buffers and introducing new operations for communications between these. These three buffers will be distributed between client, communication and server respectively. This decomposition process is a straightforward task with a simple gluing invariant which states that the order concatenation of the sub-buffers should be equal to the original buffer. Due to space restriction we have not presented this refinement here.

Using sequences to represent communication buffers imposes ordering of messages. In other words it assumes that the communication link should guarantee message delivery in the order which they been sent out by sender. This implication could be considered as a restriction and in some cases it might be necessary to use a more general model to

```
INVARIANT
  sreq_buf∈ SESSION ⇸ seq(REQUEST) ∧
  sresp_buf∈ SESSION ⇸ seq(RESPONSE) ∧
  /* Gluing Invariant */
  ∀sid.(sid ∈ fresh_session ⇒ reqsevice_buf(sid) =∅ ) ∧
  ∀sid.(sid ∈dom(sreq_buf) ∧ sreq_buf(sid)≠[] ⇒
     first(sreq_buf(sid))∈ reqsevice_buf(sid) ) ∧ ...

OPERATIONS
 PicService ≙        /* Client Operation */
  ANY sid, req WHERE
    sid ∈ fresh_session ∧
    sid ∉ dom(sreq_buf) ∧
    req ∈ REQUEST ∧
    req≠ none
  THEN
     sreq_buf (sid):= sreq_buf ⌢ [req] ||
     fresh_session:= fresh_session - {sid}
  END;
 SelectService ≙        /* Server Operation */
  ANY sid,req WHERE
     sid ∈ session ∧
     req ∈ REQUEST ∧
     resp ∈ RESPONSE ∧
     sid ∈dom(sreq_buf) ∧
     sreq_buf(sid)≠[] ∧
     first(sreq_buf(sid)) = req
  THEN
    session_request(sid):= req ||
    sreq_buf:= tail(sreq_buf) ||
    sresp_buf(sid):= sresp_buf(sid) ⌢ [resp]
  END;
Submit_Servic_Dtail ≙          /* Client Operation */
 ANY sid,resp WHERE
   sid ∈dom(resp_buf) ∧
   resp ∈ RESPONSE ∧
   sresp_buf(sid)≠ [] ∧
   first(sresp_buf(sid)):= resp
 THEN
  sresp_buf(sid):=  tail(sresp_buf(sid))
 END;
```

Figure 3.13: Refined model with unbounded buffers.

represent communication buffers. Therefore a different model based on using unordered multi-place buffers can be used. This approach is presented in the following section.

3.7.2 A More General Model of Communication Links

It is obvious that some communication protocols do not preserve message ordering. In these cases using sequences is not appropriate. Therefore a different model based on using unordered multi-place buffers which is inspired by [32] has been introduced in this section. This unordered buffer had been named as a *Bag* which is a collection of elements that may have a multiple occurrences of any element. Bag representation does not guarantee to output messages in the order in which they are input. In Figure 3.14 some operations which could be associated with a bag and a model of communication link based on using bags is presented.

```
DEFINITIONS
   BAG(T) ≙ (T → NAT);
   emptybag(T) ≙  λxx.( xx∈T | 0 ) ;
   add(b,x) ≙ b ⩤ { x ↦ b(x)+1 } ;
   rem(b,x) ≙ b ⩤ { x ↦ b(x)-1 } ;

INVARIANT
  sreq_buf ∈ SESSION ⇸ BAG(REQUEST) ∧
  sresp_buf ∈ SESSION ⇸ BAG(RESPOSE)
OPERATIONS
PicService ≙        /* Client Operation */
 ANY sid, req WHERE
   sid ∈ fresh_session ∧
   req∈ REQUEST ∧
   req≠ none
 THEN
    reqsevice_buf(sid):=add(reqsevice_buf(sid),req) ) ||
    fresh_session:= fresh_session - {sid}
 END;
SelectService ≙        /* Server Operation */
 ANY sid, req WHERE
   sid ∈ session ∧
   req ∈ REQUEST
   sid ∈ dom(reqsevice_buf) ∧
   reqsevice_buf (sid)(req) > 0
 THEN
   session_request(sid):= add(session_request(sid), req) ||
   reqsevice_buf(sid):= rem(reqsevice_buf(sid),req) )
END;
Submit_Servic_Dtail ≙        /* Client Operation */
 ANY sid,resp WHERE
   sid ∈ dom(resp_buf) ∧
   resp ∈RESPONSE ∧
   sresp_buf(sid)(resp)≠0
 THEN
   sresp_buf(sid):= add(sresp_buf(sid),resp)
   sresp_buf(sid):= rem(sresp_buf(sid),resp)
 END;
```

FIGURE 3.14: Modelling unbounded buffers with Bag.

This representation does not put any restriction on order which messages could be delivered to receiver and from this viewpoint it is more general that the previous model.

3.8 Summary of Results, Conclusions and Further Work

We have identified some key issues in formal modelling of Web-based systems like state representation in server and client side, distributed database system abstraction and refinement, handling complex data types and formal model for communication links.

We have proposed some solutions for these aspects which have been exemplified with event-B models of a Travel agency case study.

In formal modelling we have considered only the safety properties and we have not tackled the liveness issue. Although our work has been influenced by mainstream work in Web-based system modelling and implementation, our models require further refinement to implementation level.

Furthermore Web-based systems are constructed from distributed subsystems which could operate concurrently and are connected with communication links. The fact that the rich and complicated nature of such systems could not be completely enclosed by a single B machine reveals the importance of decomposition as a next step in formal development process. Decomposition is also an essential strategy for tackling the rapid growth of system models' complexity. Decomposition strategies could be based on CSP style value passing channels which has been developed in [32] and applied to other types of distributed systems [113].

Finally by investigating different examples of Web-based system we would expect to identify some other challenging issues. By recognising these issues and identifying some proper models a platform for formal modelling of Web-based systems could be proposed. Web Services as a standardised derivation of Web-based system is another potential area which could be examined by formal method practitioners. In the next chapter we will attempt to develop more formal representations of the refinement patterns we have identified and used here. These should make it easier to recognise and apply the patterns and to provide tool support for their application. Ideally this should include the automatic construction of appropriate gluing invariants when applying a pattern.

Chapter 4

Pattern Based Formal Modelling

4.1 An introduction

Building reliable Web-based application is hard. Building mission-critical Web applications is even harder. Current Web applications consist of a hybrid of distributed, multi-tier, concurrent systems. In addition to these a typical Web application should deploy numerous functionalities. All these aspects require tackling complex issues. Since Web-based applications serve as the front line of modern e-Business, modelling these types of systems requires dealing with dependability, heterogeneity, scalability, security, high-availability, short time-to-market and technology-neutral issues.

Considering the above mentioned facts reveals that the formal specification of substantial Web applications could be very complicated and tiresome. A desirable situation is to have a set of generic formal patterns to apply them to the problem in hand and built the entire solution by using some mechanisms like instantiation and composition. These Formal Patterns codify the repeatable experience and knowledge that has been attained from similar tasks before. Patterns not only document solutions to common reappearing problems, but also point out pitfalls that should be avoided. In addition, creating software from existing resources is a well-established part of programming and software engineering for reasons of quality, productivity, rapid development and deployment. Progress on this broad approach to reuse began at the lowest levels of programming, such as code, and has slowly reached toward the highest levels of software development process, such as architectural design. The ability to reuse software assets is a vital step in the effective and efficient development of new systems and solutions.

The challenging aspects of Web-based applications make the idea of developing some generic patterns for these type of systems very attractive. Patterns for Web-based applications would help developers understand some of the problems beforehand, as well as show how to solve them. In addition it will help developers to build systems with a shorter time to market. The idea of using patterns for Web application is not new and it is well-known practice that some famous venders like IBM very much devoted to it [11].

Based on our experience with developing formal models for the Travel Agency System in the previous chapter, we have identified some common patterns which could be seen in many Web applications . In this chapter we first define these patterns informally. In the second stage we develop a very generic formal specification of these patterns. Through stepwise refinement we introduce more details to each model to specialise and refine them to some design patterns. The formal pattern approach could enable system developers to argue not only about the applicability of different patterns, but also it should support implementation of successful solutions through the re-use of a single pattern or a suitable combination of generic patterns to build new systems. In other words we anticipate that these formal models could be used in two different ways:

1. As a detailed example and prescriptive approach, following the mappings and guidance provided
2. As a way to design more complex systems, to compose several patterns together for more complex system architectures.

For informal presentation of each pattern we can adapt the common style of pattern representation in the field of software patterns. Although there is no universal convention on pattern presentation [50, 49], but the following elements considered to be essential [107] in pattern documentation:

1. **Name**: This is a meaningful name which consist of a single word or short phrase to refer to the problem and its solution.
2. **Context**: This part describes how the problem occurs and under which conditions the proposed solution is applicable.
3. **Problem**: A statement of the problem which describes its intent. It should clarify the goal and objectives of the pattern within the given context and forces. In reality often the forces oppose the objectives as well as each other.
4. **Forces**: A description of the relevant forces and constraints and how maybe they interact or conflict with each other and with goals.
5. **Solution**: The structure of the solution part for our formal patterns is different from general pattern representation. Although the solution usually starts with informal textual or graphical representation but the actual solution here consist of an abstract formal specification and a number of stepwise related refinements. Therefore the structure of a formal solution for a pattern essentially is much more comprehensive than the informal pattern.

6. **Examples**: This part is optional but it provides some related real world examples which could help the reader to understand the pattern's use and applicability.
7. **Resulting Context**: This section describes the result, benefits and consequences of applying the pattern. It also shows how the forces were balanced or resolved.

4.2 Formal Web applications Patterns

In this section we present three generic patterns which model interactions between interested parties and a Web Application. Interested parties include Web Clients or secondary servers which communicate with the Web application. Our main emphasis in the specification level is to present these patterns in a generic form. This ensures that we can specialise them later through refinement process. In many practical systems these generic pattern could be closely related. Therefore pattern instantiation and composition usually are the next steps which could be envisaged. Also it is not our intention to investigate this issue in this chapter but during formal pattern modelling maybe in some cases it would be important to consider how different patterns could match together to build a larger pattern to demonstrate a part of a real system functionality and behaviour.

As we mentioned earlier, we present three generic patterns in this chapter. The first pattern is about *session creation* and it is concerned with preserving client state across several interactions with the Web application. Considering the fact that the underlying HTTP protocol is stateless and does not provide any support for state tracking and on the other hand many Web applications are state-full applications, makes the session creation pattern an essential part of many Web applications.

The second pattern represents the general model of interaction between a typical Web client and the Web server which is the request-process-response pattern. We have generalised the formal specification of this pattern in such a way which could demonstrate the essence of a whole class of similar interactions. Clearly when we consider applying this pattern to a specific case, some specialisation should be envisaged.

In the third pattern, we present the idea of communicating servers. In fact this pattern could be considered as an extension of the previous pattern. In this pattern the Web application server in pattern two, has been replaced by a main Web server and a number of secondary servers. The main server relies on services of secondary servers to fulfill the Web clients Requests.

Before starting with formal presentations of the above patterns, in the next section we discuss the common conventions and some general background to our modelling approach.

4.2.1 General approach to Formal Patterns

Across the rest of this chapter we have used some common definition like sets, constants, and variable definitions which are used in several occasions. to avoid any redundant explanation we discuss them in this section. In addition to this we use a general approach in modelling the communication links between components. This approach is also specified here.

The Web clients are sending requests and receiving responses, therefore we need to model the request and response objects. Request and response are structured variables and we have to use an appropriate model for them. Here we have used a constant mapping to define a record-like structure for both request and response. This mechanism which introduced in the previous chapter and has been described in [46] in detail. The main advantage of this approach is that flexible enough to allow further refinement of the structure by introducing extra fields in later stages.

A *request* record at least has three fields, namely *ReqID*, *ReqSID* and*Servc*. The *ReqID* field represent the sender of the request. To represent clients we have defined a reference set and it named as $AGENT\_ID$. For each client session with the Web server we allocate a fresh unique ID from this set. All the requests which send by this session will contain this ID as their first field.

When the Web server receives a request form a client, it will check the second field of the request for a valid session ID. If it does not include a session ID, then the Web server assumes that it has received the request from a new client session. Therefore the Web server creates a new session ID. The session ID will help the Web server to retrieve the client related specific information on receiving subsequent requests from each client.

The third field in the request record represents the requested service by the client. It could have further details, but in this stage we have abstracted away all detail in order to have a generic pattern as well as avoiding any unnecessary complication in the proof obligations.

The structure of the *response* record is very similar to the *request* record. The first field *RespID* indicates which client should receive the response. Like a *request* record, the second field *RespSID* carries the *session ID* which should be used with the next request. The third field contains the server response to the requested service which previously has been made by the client. Again in the implementation level this part could have much more details but it have been abstracted away for the same reason we have mentioned for the request record. The definitions of *request* and *response* are illustrated in Figure 4.1.

Another issue that we interested in is the modelling of communication links. In chapter 3 we presented a detailed approach to refine the communication links. In specification level we started with single-place buffers. This is perfectly reflects the nature of client interactions with the Web server. When the client sends a request, it should normally wait for a response from the Web server, before sending another request. In the previous chapter it has been pointed out that by using this approach we face some complication in the refinement level. Because we should refine the single-place buffers to multi-place buffers to be able to discharge the proof obligations and proceed to decomposition stage. Although this approach works perfectly, it is very complex and to avoid this issue we propose another solution.

In this chapter we use a different approach to model the communication links in a simpler way. The key concept in this approach is a history-recorder which guarantee that no repeated request or response will be put in the communications buffers. Both on the clients and server operations there are guards that enforce this requirement. The history recorder for requests is named as *req_hist* and it is *resp_hist* for response. By using this simple mechanism we can avoid some complications like refining single-place buffers before decomposition.

4.2.2 Web Based Session Creation

Context: Many typical Web applications like E-commerce shopping applications need to identify different users and maintain user data within a session. A Session is a series of requests that occur during a time-period from the same user. The stateless nature of the HTTP protocol, which is employed for communication, means that the Web application should handle the state information. Basically, a Web server handles each request independently from each other and does not have any knowledge about the preceding requests from the same user. To overcome the problem, Web applications should implement a session management policy. This session management policy should guarantee that all user interactions could be managed coherently in a session. To manage a session, a server should save traces of user requests temporarily and maintains the session state of each user. Above all, a server should identify the user who sends a request.
Problem: How we should develop an effective session management policy in Web applications?
Forces:

1. HTTP is a stateless Protocol.
2. The Web server has no control over the Web clients' behaviour.
3. Web applications are usually dealing with more than one client at a time, therefore they need to identify each client correctly.
4. The Web application should handles multiple transactions within a single session. To complete a transaction, it may interact with a client by transferring several

web pages and gathers several user specific information like credit card number and delivery address from the client.

Solution: There have been various methods to identify clients from their requests. Using *session ID* is the most common method used to track user sessions. The *session IDs* are typically generated and associated with each new requests which the server receives. In fact upon receiving each request, the server checks whether it contains a valid *session ID*. If the received request does not contain a valid ID, it assumed to be a new session and then a new *session ID* will be created. This *session ID* along with the initial page in the form of the HTTP response will be send back to the Web client. Otherwise if the request does contain a valid *session ID*, the server application will use this *session ID* to retrieve the particular user date which is associated with the *session ID*. All user data is stored on the server either in a temporary file or database.

Formal Specification: Sets, constants and variables definitions of this pattern are presented in Figure 4.1. The *req_buf* models the output links from clients to the Web server. Each client puts its request in this buffer and the Web server retrieves it form this buffer. The *current* variable represent the current active browser windows on all clients computers. When a client open a new browser window, a new identifer for this window will be add to this set.

The *session* variable is representing the set of valid sessions. When the Web server receives a request without a valid session ID, it assumes that a new client has joined the system and the server will allocate a new session ID for it and adds it to the *session*. The *resp_buf* play the similar role to the *req_buf*, but it stores responses from the Web server to clients.

A simple formal representation of the scenario, which described in the first part of solution, is illustrated in 4.2. In this specification we assumed that multiple users could interact with the Web application. In addition to that each user is allowed to open more than a single browser window and have multiple connections with the Web application server. The act of opening a new browser window and typing a specific URL (Uniform Resource Locator) by the user has been modelled in the *Client_CreateAgent* operation. Here *aid* is a unique handle to identify each opened browser window on the client computer. The *Server_CreateSession* represents the server side actions after receiving a new request from a client. Through the operation guard *ReqSID(req)= null*, the new request would be checked to examine that it does not contain a valid *session ID*. The next part of the guard $sid \in SESSION \wedge sid \notin session$ represents the server allocating a valid new *session ID*. The new *session ID* will be associated with the request in the body of the operation. This task has been accomplished by building a response for the client by using the request handler ID and a new *session ID*.

The HTTP link between clients and the server here has been modelled with a set. We discussed this in the previous section and there is no need to repeat it here. In

```
SETS
 SESSION; REQUEST; RESPONSE;
 AGENT_ID; SERVICES; SRVC_RESP

CONSTANTS
   null, ReqID, ReqSID, Srvc,
/* REQUEST ==
   ReqID       ∈ AGENT_ID
   ReqSID      ∈ SESSION
   Srvc        ∈ SERVICES        */
   RespID, RespSID, Srvc_resp
/* RESPONSE ==
   RespID      ∈ AGENT_ID
   RespSID     ∈ SESSION
   Srvc_resp   ∈ SRVC_RESP       */

PROPERTIES
 null    ∈ SESSION                         ∧
 /* REQUEST Record Definition */
 ReqID ∈ REQUEST → AGENT_ID                ∧
 ReqSID ∈ REQUEST → SESSION                ∧
 Srvc ∈ REQUEST → SERVICES                 ∧

/* RESPONSE  Record Definition */
 RespID ∈ RESPONSE → AGENT_ID              ∧
 RespSID ∈ RESPONSE → SESSION              ∧
 Srvc_resp ∈ RESPONSE→SRVC_RESP

VARIABLES
    req_buf, current, session, resp_buf,
    req_hist, resp_hist

INVARIANT
  req_buf        ∈ ℙ(REQUEST)     ∧
  current        ∈ ℙ(AGENT_ID)    ∧
  session        ∈ ℙ(SESSION)     ∧
  resp_buf       ∈ ℙ(RESPONSE)    ∧
  req_hist       ∈ ℙ(REQUEST)     ∧
  resp_hist      ∈ ℙ(REQUEST)

INITIALISATION
  req_buf :=   ∅         ||
  current:=    ∅         ||
  session :=   ∅         ||
  resp_buf :=  ∅         ||
  req_hist:=   ∅         ||
  resp_hist := ∅
```

FIGURE 4.1: Sets and Variables Definitions of the Session Creation Pattern

addition to the buffers there are two operations for modelling the communication process which are namely *Convey_SessionReq* and *Convey_SessionID*. These operations have been introduced in the refinement. The invariants for refinement are presented in the Figure 4.3 and a part of the refinement which contains these operations is illustrated in the Figure 4.4. Maintaining simplicity is the main reason for postponing the introduction of the communication link to the refinement stage. The *req_buf* of the specification is divided into two buffers in the refinement. The same process is applied to the response buffer *resp_buf*. The splitting process is a pre-requisite for the decomposition process in the later refinement stages.

Resulting Context: Although the session creation pattern is an essential part of almost all of Web applications but it usually applied in conjunction with other patterns. In the next section we present another pattern which could be composed with this pattern.

4.2.3 User-to-Web application pattern

Context: The User-to-Web application pattern is applicable to situations where users interact with a single Web application. The core idea of this pattern is around the simple configuration of Request-Processing-Response. The scheme stars with a single request from a client to the Web application. When the request received by the Web application it will process the request and produce an appropriate response. By generalising the

```
Client_CreateAgent ≙            /* Client Operation */
 ANY aid, req WHERE
    aid ∈ AGENT_ID
    ∧ aid ∉ current
    ∧ req ∈ REQUEST
    ∧ req ∉ req_hist
    ∧ ReqID(req) = aid
    ∧ ReqSID(req) = null
 THEN
   current := current ∪ {aid}
   || req_buf := req_buf ∪ {req}
   || req_hist := req_hist ∪ {req}
 END;

Server_CreateSession ≙       /* Server Operation */
 ANY req, resp, sid WHERE
    req ∈ REQUEST
    ∧ req ∈ req_buf
    ∧ ReqSID(req) = null
    ∧ resp ∈ RESPONSE
    ∧ RespID(resp) = ReqID(req)
    ∧ sid ∈ SESSION ∧ sid ∉ session
    ∧ RespSID(resp) = sid
    ∧ resp ∉ resp_hist
 THEN
   resp_buf := resp_buf ∪ {resp}
   || session := session ∪ {sid}
   || session_state := session_state ∪ {sid ↦ NS}
   || req_buf:= req_buf - {req}
   || resp_hist := resp_hist ∪ {resp}
 END;
```

FIGURE 4.2: Formal Specification of the Session Creation Pattern

```
INVARIANT
   req_buf1     ∈ ℙ(REQUEST)
  ∧ req_buf2    ∈ ℙ(REQUEST)
  ∧ req_hist1   ∈ ℙ(REQUEST)
  ∧ req_hist2   ∈ ℙ(REQUEST)
  ∧ resp_buf1   ∈ ℙ(RESPONSE)
  ∧ resp_buf2   ∈ ℙ(RESPONSE)
  ∧ resp_hist1  ∈ ℙ(RESPONSE)
  ∧ resp_hist2  ∈ ℙ(RESPONSE)
```

```
/* Gluing Invarints */
  ∧ req_buf = req_buf1 ∪ req_buf2
  ∧ req_buf1 ∩ req_buf2 = ∅
  ∧ resp_buf = resp_buf1 ∪ resp_buf2
  ∧ resp_buf1 ∩ resp_buf2 = ∅
  ∧ req_buf1 ⊆ ReqID^-1[current]
  ∧ req_buf2 ⊆ ReqID^-1[current]
  ∧ resp_buf1 ⊆ RespSID^-1[session]
  ∧ resp_buf2 ⊆ RespSID^-1[session]
```

FIGURE 4.3: Invariants in Refinement of the Session Creation Pattern

processing part, this pattern could be applied to many common cases like login, search and query, selecting an option and submitting selected choices.

Problem: How the client-Web application interactions could be specified in abstract form such that can be refined easily to represent different cases which they comply with the generic form of the Request-Processing-Response pattern.

```
OPERATIONS
 Client_CreateAgent ≙              /* Client Operation */
  ANY aid, req WHERE
     aid ∈ AGENT_ID
     ∧ aid ∉ current
     ∧ req ∈ REQUEST
     ∧ ReqID(req) = aid
     ∧ ReqSID(req) = null
     ∧ req ∉ req_hist1
  THEN
    current := current ∪ {aid}
    || req_buf1 := req_buf1 ∪ {req}
    || req_hist1 := req_hist1 ∪ {req}
  END;
 Convey_SessionReq ≙                 /* Communication Operation */
  ANY req WHERE req ∈ REQUEST ∧ req ∈ req_buf1
  THEN
     req_buf2 := req_buf2 ∪ {req}
     || req_buf1 := req_buf1 - {req}
  END;
 Server_CreateSession ≙        /* Server Operation */
  ANY req, resp, sid WHERE
     req ∈ REQUEST
     ∧ req ∈ req_buf2
     ∧ ReqSID(req) = null
     ∧ resp ∈ RESPONSE
     ∧ RespID(resp)= ReqID(req)
     ∧ sid ∈ SESSION
     ∧ sid ∉ session
     ∧ RespSID(resp) = sid
     ∧ resp ∉ resp_hist1
  THEN
    resp_buf1 := resp_buf1 ∪ {resp}
    || session := session ∪ {sid}
    || req_buf2 := req_buf2 - {req}
    || resp_hist1 := resp_hist1 ∪ {resp}
  END;
 Convey_SessionID ≙       /* Communication Operation */
  ANY resp WHERE resp ∈ RESPONSE ∧ resp ∈ resp_buf1
  THEN
     resp_buf2:= resp_buf2 ∪ {resp}
     || resp_buf1:= resp_buf1 - {resp}
  END
END
```

Figure 4.4: Refinement of the Session Creation Pattern

Solution: The User-to-Web application Pattern is commonly observed in e-business solutions that provide users with the ability to access their information and change it by interacting directly with core application and databases. A simple topology of this pattern is illustrated in Figure 4.5. This pattern captures the essence of direct interactions between users and the Web application.

Such interactions can range from simple static information lookup to complex updates involving enterprise data. Examples of applications that use this pattern include the following:

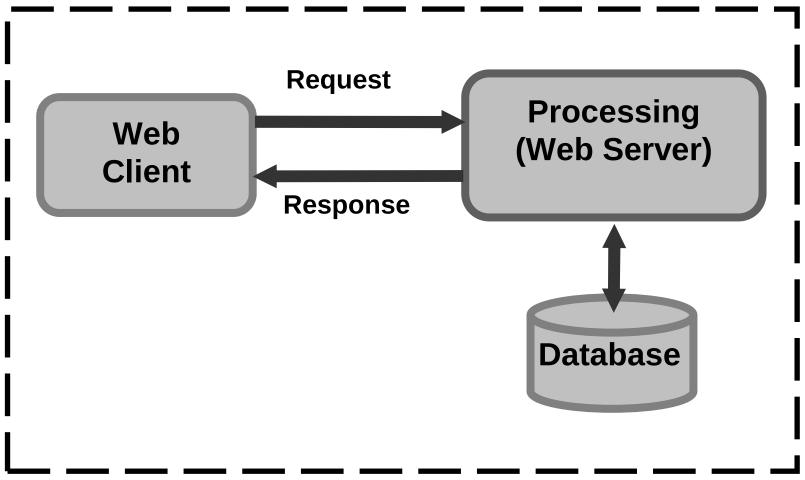


FIGURE 4.5: Architecture of the User-to-Web Applications Pattern

1. Applications such as an Online Broker application that allows customers to manage their portfolios and add/change/remove services and bookings across the Web.
2. Web based retailers that allow customers to shop for and buy retail goods by accessing a catalogue of items and order entry functions from their browsers across the Internet.
3. Convenience Banking which allow clients to view account balances, view recent transactions, pay bills/transfer funds, stop payments and manage bank card.

A Sequence diagram that illustrate the communication link between a client and the Web Application is depicted in Figure 4.6. It is clear from this diagram that when a client submits a request, they should wait to receive a response before sending the next request. Although in practise users can resubmit requests, the Web server could have a mechanism to discard the repeated request.

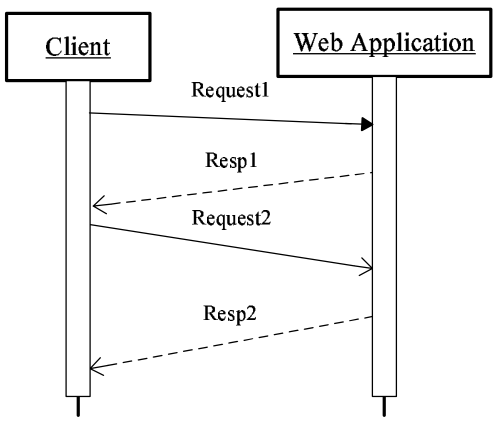


FIGURE 4.6: Message Sequencing of the User-to-Web Applications Pattern

Formal Specification: In this section we provide a formal model for the user-to-Web application pattern. The sets, constants and variables definitions of the pattern are presented in Figure 4.7. The structure of the request and response objects are the same as the previous pattern and they serve the same purpose. The only difference is that requests in this pattern always contain valid *session ID*. We have defined two new constant definition which are *Resp_func* and *Update_func*. The significance of these will be discussed very shortly, but here we can highlight the point that theses two definition along with the simple definition of database contribute toward a more simpler and generic pattern. Another major issue is the initialisation of the *current* and *session* variables. Instead of initialising these variable with empty set like previous pattern, they have been initialised with a subset of *AGENT_ID* and *SESSION*. This shows that the second pattern should be build on the top of the *session creation* pattern.

```
SETS
  SESSION; REQUEST;
  RESPONSE; AGENT_ID;
  SERVICES; SRVC_RESP; DB
CONSTANTS
  ReqID, ReqSID, Srvc,
/* REQUEST ==
  ReqID      ∈ AGENT_ID
  ReqSID     ∈ SESSION
  Srvc       ∈ SERVICES         */
  RespID, RespSID, Srvc_resp,
/* RESPONSE ==
  RespID     ∈ AGENT_ID
   RespSID   ∈ SESSION
   Srvc_resp ∈ SRVC_RESP        */
  Resp_func, Update_func
PROPERTIES
  /* REQUEST Record Definition */
  ReqID      ∈ REQUEST → AGENT_ID   ∧
  ReqSID     ∈ REQUEST → SESSION    ∧
  Srvc       ∈ REQUEST → SERVICES   ∧
  * RESPONSE  Record Definition */
  RespID     ∈ RESPONSE → AGENT_ID  ∧
  RespSID    ∈ RESPONSE → SESSION   ∧
  Srvc_resp  ∈ RESPONSE → SRVC_RESP ∧

  Update_func ∈ (DB × SERVICES) → DB ∧
  Resp_func ∈ (DB × SERVICES) → RESPONSE

VARIABLES
  req_hist, current, req_buf,
  session, resp_hist, resp_buf, db
INVARIANT
  req_buf    ∈ ℙ(REQUEST)    ∧
  req_hist   ∈ ℙ(REQUEST)    ∧
  req_buf    ⊆ req_hist  ∧
  current    ∈ ℙ(AGENT_ID)   ∧
  session    ∈ ℙ(SESSION)    ∧
  resp_buf   ∈ ℙ(RESPONSE)   ∧
  resp_hist  ∈ ℙ(RESPONSE)   ∧
  resp_buf   ⊆ resp_hist ∧
  db ∈ DB

INITIALISATION
  req_buf    := ∅  ||
  req_hist   := ∅  ||
  current    :∈ ℙ(AGENT_ID)  ||
  session    :∈ ℙ(SESSION)   ||
  resp_buf   := ∅  ||
  resp_hist  := ∅  ||
  db         :∈ DB
```

FIGURE 4.7: Sets, Constants and Variables Definitions of the User-to-Web Applications Pattern

The operations of the User-to-Web pattern could be viewed in Figure 4.8. In order to make our model more generic and applicable to wider cases we have used two other constant definitions to define an essential part of the pattern which is related to querying a database and producing a response for the clients. As it has been depicted in Figure 4.5, to provide the client with a response, the Web application usually should access a database. The precise method of producing the response and the way that it effects the content of the database are dependent on both the request and the initial content of the database. Therefore it is very desirable to have a general specification instead of strict definition in the pattern. This general specification could be refined when the pattern applied to the specific cases.

```
MakeRequest ≙        /* Client Operation */
  ANY req WHERE
      req ∈ REQUEST
      ∧ ReqID(req) ∈ current
      ∧ req ∉ req_hist
  THEN
     req_buf := req_buf ∪ {req}   ||
     req_hist := req_hist ∪ {req}
  END;

ProcessRequest ≙   /* Server Operation */
  ANY req, resp WHERE
      req ∈ REQUEST
      ∧ req ∈  req_buf
      ∧ ReqSID(req) ∈ session
      ∧ resp ∈ RESPONSE
      ∧ RespID(resp) = ReqID(req)
      ∧ RespSID(resp)  = ReqSID(req)
      ∧ resp  = Resp_func(db, Srvc(req))
      ∧ resp ∉ resp_hist
  THEN
     resp_buf := resp_buf ∪ {resp}      ||
     resp_hist := resp_hist ∪ {resp}   ||
     db:= Update_func(db, Srvc(req))    ||
     req_buf:= req_buf - {req}
  END;

GetResponse ≙       /* Client Operation */
 ANY resp WHERE
    resp ∈ RESPONSE
    ∧ resp ∈ resp_buf
 THEN
   resp_buf :=  resp_buf- {resp}
 END
```

Figure 4.8: Abstract Specification for of the User-to-Web Applications Pattern

The first constant mapping we mentioned in the previous paragraph is the *Resp_func*, that defines a mapping between the initial state of the database, the service which is requested and the response produced by the Web server. In other words the response is a function of the state of the database and the requested service. The second definition is *Update_func*, that defines a relation between the initial and the final state of the database based on the requested service. Later in the operation body of the pattern we should see how these definition contribute toward a succinct definition of the pattern.

The formal specification consists of three operations for making requests, processing requests and receiving the result of processing on the client side. These operations have been presented in Figure 4.8. To clarify the pattern further it is worth mentioning the following points:

1. In the *MakeRequest* operation the *req_hist* variable has been used for history tracking mechanism. This approach, that also has been used in previous pattern, is to

guarantee the uniqueness of each request.

2. To model the communication between the client and the Web server we have used the set-based buffer approach.

3. The constant mapping function have been used in the guard and body of the *ProcessRequest* operation to avoid any implementation details.

A refined B Model for the User-to-Web Applications Pattern has been produced. In the refined model we have introduced new operations to explicitly model the communication links. The new operations' details are very similar to what we had in the previous pattern's refinement, therefore here we avoid to present it but the full B codes could be found in the Appendix B.

Resulting Context: As we mentioned earlier, this pattern could be used in conjunction with the session creation pattern, because for any kid of interaction between a client and the Web application a session should be established first. As it has been pointed out earlier the composition of the pattern could be seen in the form of variable sharing. Another important issue which should be considered is that, this pattern is applicable to cases which involve simple lookup from database that does not change its content and to cases which involve changing the database, therefore when the pattern applied to a specific case, some instantiation would be necessary.

4.2.4 Distributed Processing pattern

Context: In many Web applications the main Web server relies on one or more secondary servers to provide clients with their requested service. The main server's collaboration with secondary servers also could be applied to scenarios where there is the need for integration with legacy or third-party applications. In this case the new application is not built as a stand alone solution, but instead it has to communicate with some other applications. Therefore unlike the previous pattern, in this pattern, the processing is distributed between different servers. As a result of user requests the Web application may need to interact with one or more third-party applications or back-end data system by sending some service request. In the second stage, to accomplish user's requests, it has to collect the secondary servers responses and compile the response page for the client.

Problem: How the client and Web application interactions which involves a main Web server and some secondary servers could be specified formally. This should carried out in such a way that demonstrate the following essential points:

- The distribution of processing between the main server and secondary servers

- The link between the initial client requests and broadcasting a service request by the main server
- Compiling the final response for each client from secondary servers' responses

Solution: The distributed processing Pattern is observed in many e-business solutions which for example involve credit checking and money transactions. When a client does online shopping, the main Web application should check the user details and credit with a bank or credit institution server. The process could proceed based on the type of response that the main server receives from the secondary server/s. Another example of this pattern could be found in the Travel Agency example of the previous chapter. The main Web application provides an integrated airlines booking, accommodation reservation and even car hiring facility. The system facilitates potential users to carry out different queries and make all different bookings as an integrated solution. A simple topology of this pattern is illustrated in Figure 4.9.

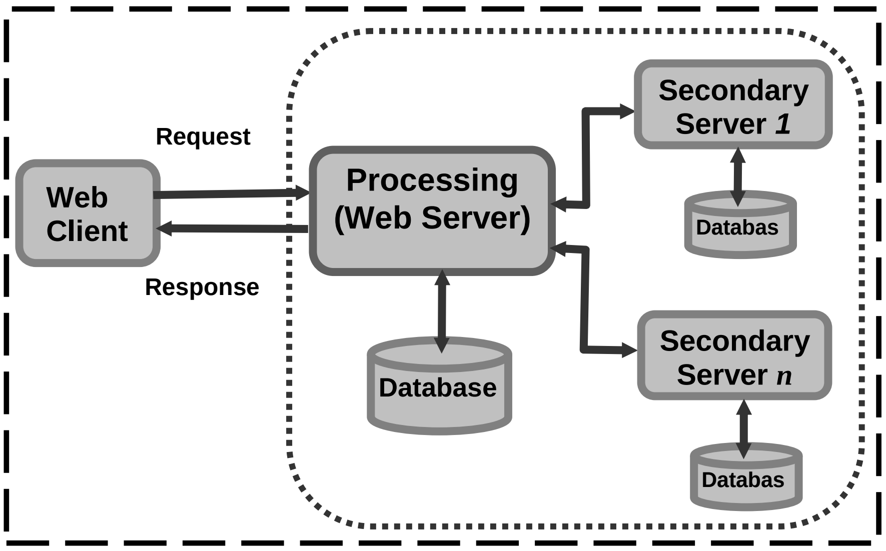


FIGURE 4.9: Architecture of Distributed Processing Pattern

In fact this pattern could be considered as an extension of the previous pattern that the central processing has been replaced with a distributed one. This extended configuration allows the Web application to access new application business logic or data and to provide a higher level of services to the Web application's clients.

Unlike the communication link between the client and the Web application, which is synchronous, the link between the Web application and secondary servers usually is asynchronous. The communication is considered to be asynchronous in the sense that the main server broadcast the service request and then continue with other jobs. The main advantage of this approach is that the main server does not need to wait for response. Another feature that could be considered is whether the Web server establishes the links with secondary servers in parallel or serial manner. In other words whether the Web server sends its request to all secondary servers merely based on a client's request or it could be depend on another secondary server's response. In the later case the request would be sent just after a specific response has been received from another server. As a result different derivations of this pattern could be considered based on the above situation. The message sequencing and communication between a client, the Web application and secondary applications is depicted in Figure 4.10. Based on this sequence diagram regarding any client's request we can envisage two different scenarios. As it depicted in the first case, a client request resulted in a situation that the Web application initiates and sends a series of requests to third parties applications and wait for appropriate responses before replying the client's request. It should be noticed that we have a single Web server, but there multiple secondary servers. Another important issue that should be taken into consideration is that the web server does not establish a permanent session with secondary servers. For this reason there is no need to have a notion like session ID that we use for client user interactions. As we mentioned earlier communication between the Web server and secondary server is single-shout and asynchronous.

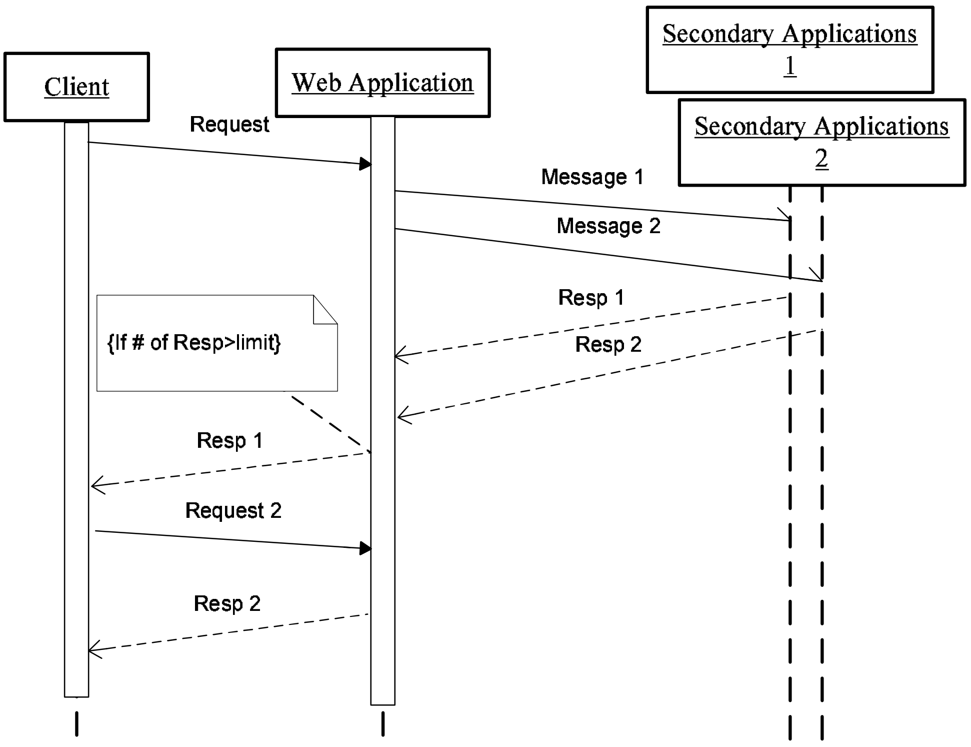


FIGURE 4.10: Message Sequencing Distributed Processing Pattern

Formal Specification: The formal specification of this pattern consist of three main operations. These operations are representing the main Web server asking for a service, secondary server responding to a request for a service and main Web server compiling the final response for the client respectively. The definitions and formal specification have been presented in Figure 4.11 and 4.12. In the *Ask_for_Service* operation the Web server sends a request for service to a set of available secondary servers. This has been modelled by placing requests in buffers associated with different secondary servers. Here we have assumed that the Web server accomplishes this task simultaneously, but obviously, in practice the middleware will send the request to each secondary server separately. This fact could been taken into account in later refinement stages when the middleware model added. To indicate that the Web server has sent the service request to the secondary servers and it is waiting to receive some responses, the client request has been added to the *pending_reqs* set. Like previous pattern using *req_hist* a mechanism has been developed to guarantee that no repeated request is been send to secondary servers.

```
SETS
  REQUEST; RESPONSE; REC_ID; USERS;
  SERVICES; SERVERS; SRVC_RESP
CONSTANTS
  Limit,
  ReqID, User, Srvc,
/* REQUEST ==
  ReqID       ∈ REC_ID
  User        ∈ USERS
  Srvc        ∈ SERVICES          */
  RespID, Provider, Srvc_resp
/* RESPONSE ==
  RespID      ∈ REC_ID
  Provider    ∈ SERVERS
  Srvc_resp ∈ SRVC_RESP           */
PROPERTIES
  Limit = 3              ∧
  /* REQUEST Record Definition */
  ReqID     ∈ REQUEST → REC_ID      ∧
  User      ∈ REQUEST → USERS       ∧
  Srvc      ∈ REQUEST → SERVICES    ∧
  /* RESPONSE  Record Definition */
  RespID    ∈ RESPONSE → REC_ID     ∧
  Provider  ∈ RESPONSE → SERVERS    ∧
  Srvc_resp ∈ RESPONSE → SRVC_RESP
```

```
VARIABLES
  req_hist, serv_req, pending_reqs,
  resp_hist, serv_resp, completed_reqs,
  final_resp
INVARIANT
  req_hist    ∈ ℙ(REQUEST)                  ∧
  serv_req    ∈ SERVERS ↔ REQUEST           ∧
  ran(serv_req)      ⊆ req_hist             ∧
  pending_reqs      ∈ ℙ(REQUEST)            ∧
  resp_hist   ∈ ℙ(RESPONSE)       ∧
  serv_resp   ∈ ℙ(RESPONSE)       ∧
  serv_resp   ⊆  resp_hist        ∧
  completed_reqs ∈ ℙ(REQUEST)     ∧
  final_resp  ∈ ℙ(RESPONSE)
INITIALISATION
  req_hist          := ∅         ||
  serv_req          := ∅         ||
  pending_reqs      := ∅         ||
  resp_hist         := ∅         ||
  serv_resp         := ∅         ||
  completed_reqs    := ∅         ||
  final_resp        := ∅
```

FIGURE 4.11: Definitions of Distributed Processing Pattern

The *Provide_Service* operation illustrates how a secondary server responds to a request for service from the Web server. In order to have a generic model we have avoided including any application specific details about how the response would be constructed. Instead of that we have used the common nondeterministic pattern for representing the the secondary server action. Nonetheless it is worth to mention that this could be replaced with a more precise design level substitution when we apply the pattern to specific cases. It is also very important to notice that we have not explicitly defined the databases in the secondary servers. The reason for this is already provided in 3.6.

In a realistic situation it is very likely that only a subset of all available secondary servers will responde to each request for service. Therefore the Web application should designed in such a way to proceed to the next step when it received the minimum

```
Ask_for_Service ≙      /* Main Server Operation */
  ANY req WHERE
     req ∈ REQUEST
     ∧ req ∉ req_hist
  THEN
    serv_req := serv_req ∪ SERVERS * {req}
    || req_hist := req_hist ∪ {req}
    || pending_reqs := pending_reqs ∪ {req}
  END;

Provide_Service ≙      /*secondary Server Operation */
  ANY serv, req , srvc_resp, resp WHERE
     serv ∈ SERVERS
     ∧ req ∈ REQUEST
     ∧ srvc_resp ∈ SRVC_RESP
     ∧ resp ∈ RESPONSE
     ∧ resp  ∉ resp_hist
     ∧ (serv ↦ req) ∈ serv_req
     ∧ RespID(resp) = ReqID(req)
     ∧ Provider(resp) = serv
     ∧ Srvc_resp(resp) =  srvc_resp
  THEN
    serv_resp := serv_resp ∪ {resp}
    || resp_hist := resp_hist ∪ {resp}
    || serv_req := serv_req - {serv ↦ req}
  END;

Complete_Req ≙      /* Main Server Operation */
  ANY req, resp WHERE
     req ∈ REQUEST
     ∧ req ∈ pending_reqs
     ∧ resp ⊆ serv_resp ∩ {rs | rs ∈ RESPONSE ∧ RespID(rs) = ReqID(req)}
     ∧ card(resp) ≥ Limit
  THEN
    final_resp := final_resp ∪ resp
    || completed_reqs := completed_reqs ∪ {req}
    || pending_reqs := pending_reqs - {req}
  END
```

FIGURE 4.12: Formal Specification of Distributed Processing Pattern

necessary number of responses. The next step is compiling the final response for the client. The optimal threshold for the minimum acceptable number of responses could vary and depend on different factors. Here the simple constant *Limit* has been defined to represent this threshold. In the *Complete_Req*, which is specified as a part of the Web application server, the operation's guard checks the number of received responses for each single request against this limit. If the limit has been satisfied, the guard is enabled and the final response would be complied. After that the state of the request should be changed from *pending* to *completed* which has been accomplished by removing the request from the *pending_reqs* set and adding it to the *completed_reqs* set.

As it was the case for previous patterns, a refinement of this pattern has been produced. This refinement mainly deals with the issue of the communication layer and the related gluing invariants. Like previous patterns it wrap the first specification model with a communication layer which could helps to carry out the decomposition process smoothly. The decomposition process is an essential part of complex distributed systems modelling which the Web applications are an eminent example of such systems. In Web application, the decomposition process make it possible to distinguish between functionality of different parts and distribute them over different architectural elements.

4.3 Devising next stages of formal patterns

The formal patterns which we have developed in the previous sections are high level architectural and platform-independent patterns. Therefore each of these patterns could be refined to one or more platform-specific design patterns directly linked to the chosen platform. In Web application implementation, one of the most widely used platform is the Sun Microsystems, Java 2 Platform Enterprise Edition (J2EE) standards. The J2EE defines the standard for developing multi-tier enterprise and Web applications. The J2EE platform is based on modular and standardised component definitions which should boost the applicability of pattern-based approach.

Another logical trend, as a continuation of this chapter, is instantiation and composition of generic pattern to produce more sophisticated patterns for formal specifications and designs. Devising rules and guidelines for formal patterns composition and pattern applicability in the form of a framework could be considered as a natural continuation of the formal patterns. In the next chapters we investigate these issues in depth.

4.4 Concluding and Results

Patterns describe successful solutions to known problems and using patterns in software development is a well-known approach. Patterns have proven useful to help developer to reuse successful practices. In addition patterns teach useful techniques, they help people communicate better, and reason about different solutions.

Based on our experiences in the Travel Agency case study in the previous chapter, we have extracted some generic patterns. In this chapter we presented these patterns formally. Formalising patterns provide a well-founded support for reuse. Furthermore, formalised patterns are a step toward defining a framework for developing mission-critical Web-based Applications. As a continuation of our work in the first step we developed some formal models for generic patterns that we presented in this chapter.

In the next chapter we will apply our findings in this chapter to some real case studies and assess our model. In applying these patterns the instantiation and composition of patterns as a main strategy that will be considered. The outcomes will show to what extend these formal patterns could assist the formal development process. The final findings could serve as the basis for development of new generation of tools for the B-Method and extending it to support automatic or semi-automatic refinement.

Chapter 5

Specification Partitioning and Composition Techniques

5.1 Introduction

In conventional B development we start the formal specification process with a single B machine which usually contains few operations and related variables. This single specification could be refined in a stepwise manner by introducing extra events' definitions and their related variables to produce a more concrete model. During the refinement process when the complexity of the concrete model reaches a specific level that makes it difficult to manage and understand the model we can decompose the refined model to a number of sub-models. After this stage each sub-model could be refined independently.

Based on our experiences in developing a formal specification for the *Travel Agency System* in Chapter 3 we found that the following characteristics of real Web applications make conventional B development unsatisfactory.

- *Multi-layer Architecture*: The multi-layer architecture of this kind systems which put a lot of emphasis on separation of layers' specification and design is not compatible with the idea of a single B machine for the whole system specification. Using a single B machine in early stage of specification and refinement resulted in mixing up functionalities which in most cases are independent. The major design criteria like modularity, manageability and comprehensibility are in favour of separation between the specifications of different layers.
- *Substantial Requirements*: The substantial list of requirements for an actual Web application could makes the specification and refinements process very complicated. Unlike simple exploratory case studies, it is not very convenient to start the formal specification of these systems with a few operations and variables. The

main reason is that we have a comprehensive list of closely related requirements with the same precedence. The close connection between requirements simply means that it is not possible to specify one requirement without introducing all other related properties and it is the point where the complexity is laying. Here according to Abrial's view [10] may argue that in many systems it is possible to classify the initial requirements to a number of subsets of closely linked requirements. Then in the second phase we can pick up each subset one at a time and incorporate it into our model through superposition refinement. Based on our experiences, especially with the travel case study, even when this approach is possible it is not convenient. A major problem associated with this approach is that it could be a long-drawn-out process before reaching the point to have the full specification and starting the actual refinement. Another weakness of this approach is that any changes in one model may effects all previous models and bringing them inline with the changes could be very time consuming and tedious job.

To overcome these complications, we devise a new development approach in this chapter. Our alternative approach is based on *specification partitioning* in the early stage of formal specification. In this approach the system specification comprises a few separated but closely connected B machines. The relation between different models is defined by some *composition relations* which are an essential part of this new approach. Each B machine, articulates a specific aspect, layer or part of the system. In Web applications different models should be devised in such a way that should match with the multi-layer architecture and underlying platforms.

With reference to indispensable role of composition in our approach, in the next part of this chapter we first explore the bases of the composition mechanism. In the later sections we extend the composition mechanism based on different scenarios in the case study.

In this chapter we pursue a practical approach rather than a theoretical one. For this purpose we have chosen an online English auction system to develop our ideas and demonstrate how they work in practise. A short informal presentation of the case study in addition to the overview of the system architecture are presented in section 5.3.

Another important aspect of our work in this chapter is to apply the formal specification patterns which we developed in the previous chapter. Assessing the suitability of these patterns for real cases is an essential aspect of pattern based development. An important issue which can arise by our new approach to the formal modelling, is how the specification partitioning could effect the formal patterns of Chapter 4. In section 5.4.1 and during formal development process these issues will be discussed in detail.

5.2 Composition Techniques for Event-B

In this section we examine the existing formal background for devising an effective composition mechanism in the Event-B. In Event-B a system is specified as an abstract machine containing some state variable and a number of events. The events are guarded operations that can perform some actions when their guard enabled. The execution of an event can effects the state variables.

There are two different approaches on how the behaviour of a system could be viewed. The fist approach is called *state-based* view and it is based on this approach that the behaviour of the system is defined in terms of its state and how they are changing. In the second approach which is called *event-based* view, the behaviour of the system would be defined in terms of its events and their execution sequence. Butler in his early works of B and CSP [32, 31] has illustrated that the event-based view of Event-B corresponds to the way in which the system behaviour is modelled in process algebra formalisms like CSP.

In CSP we define a system based on a number of independent *Processes*. Each process can interact with other processes or more generally with its environment by engaging in synchronous atomic events. According to Butler's approach a machine in B could be considered as a process in CSP and each guarded operation in that machine could be viewed as an event of the corresponding process. In addition to this, based on the notion of channel and value passing, which defines the method that CSP processes can communicate values with their environment, the idea of events with input-output parameters has formed.

Based on the above approach we can borrow the idea of process composition and adapt it for specification composition or very similarly for distributed system decomposition. There are some notions which shared by both composition and decomposition. Furthermore in many situation they could be considered as the reverse of the each other but in this chapter we are only concerned with specification composition. More specifically we are interested in parallel composition and hence in the next section we review this issue based on some recent work from Butler which appears in RODIN deliverable D19 [114].

The CSP formalism supports different forms of composition including serial and parallel composition. In specification composition we are interested in parallel composition. Parallel composition as it has been defined in Butler's works could have different variations. As we have devised our specification partition and composition mechanism based on the idea of parallel composition in the reminder of this section we review this idea.

5.2.1 Basic Parallel Composition Mechanism

The parallel composition of two machine is defined by fusing the shared event of both machines together. The parallel operator $\|$ defines a synchronous connection between shared events and leaving independent events independent. The synchronisation between shared events means that the composed system can engage in the composite events when the guard of both event are enabled. In practice the parallel composition models simultaneous execution of the the shared events in both system. Here it is helpful emphasising that shared events are defined based on having the same names as it is the case in CSP, but later in this chapter we show that this could be extended by defining composition between events with different names.

To define the basic parallel composition in more precise formal style, let assume we have two machines M and N with disjoint state variables m and n respectively. The event ev_M denotes the event from the machine M that should be fused or composed with the event ev_N in the machine N. The parallel composition of these machines' events could be defined as follows:

$$\begin{array}{l}
ev_M \;\widehat{=}\; WHEN\ G(m)\ THEN\ S(m)\ END \\
ev_N \;\widehat{=}\; WHEN\ H(n)\ THEN\ T(n)\ END \\
ev_M \;\|\; ev_N \;\widehat{=}\; WHEN\ G(m) \;\wedge\; H(n)\ THEN\ S(m) \;\|\; T(n)\ END
\end{array}$$

or

$$\begin{array}{l}
ev_M \;\widehat{=}\; ANY\ x\ WHERE\ G(x,m)\ THEN\ S(x,m)\ END \\
ev_N \;\widehat{=}\; ANY\ y\ WHERE\ H(y,n)\ THEN\ T(y,n)\ END
\end{array}$$

$$ev_M \;\|\; ev_N \;\widehat{=}\; \begin{array}{l}
ANY\ x,y\ WHERE \\
\ G(x,m) \;\wedge\; H(y,n) \\
THEN \\
\ S(x,m) \;\|\; T(y,n) \\
END
\end{array}$$

5.2.2 Parallel Composition with Value-Passing

In many situations the composed events need to exchange parameters, but the definition of *Basic Parallel Composition* does not support this kind of communication. Based on the idea of communicating channels with input/output parameters the definition of parallel composition in Event-B could be extended to deal with this issue.

In most basic form we can compose a single output event from one Machine with an input event from another machine. To be able to distinguish input and output parameters from ordinary variables of the systems we can use the same convention which been used in CSP. For marking a parameter as input parameter we add the ? sign in the front of it and for output parameter we use use ! instead. More formally, *Parallel Composition with Value-Passing* could be be defined as following:

$$\begin{aligned} ev_M \;&\widehat{=}\; ANY\ x!\ WHERE\ G(x!,m)\ THEN\ S(x!,m)\ END \\ ev_N \;&\widehat{=}\; ANY\ x?\ WHERE\ H(x?,n)\ THEN\ T(x?,n)\ END \end{aligned}$$

$$ev_M \parallel ev_N \;\widehat{=}\; \begin{array}{l} ANY\ x!\ WHERE \\ \ G(x!,m)\ \wedge\ H(x!,n) \\ THEN \\ \ S(x!,m)\ \parallel\ T(x!,n) \\ END \end{array}$$

Once again ev_M represents the output event from machine M and ev_N corresponds to the input event in machine N. The output parameter in ev_M denoted by $x!$ and the input parameter in event ev_N is $x?$. It is very important to note that the composed parameter becomes an output parameter. Another possible extension is to have events with independent parameters along with input/output parameters.

$$\begin{aligned} ev_M \;&\widehat{=}\; ANY\ x!, y\ WHERE\ G(x!,y,m)\ THEN\ S(x!,y,m)\ END \\ ev_N \;&\widehat{=}\; ANY\ x?, z\ WHERE\ H(x?,z,n)\ THEN\ T(x?,z,n)\ END \end{aligned}$$

$$ev_M \parallel ev_N \;\widehat{=}\; \begin{array}{l} ANY\ x!, y, z\ WHERE \\ \ G(x!,y,m)\ \wedge\ H(x!,z,n) \\ THEN \\ \ S(x!,y,m)\ \parallel\ T(x!,z,n) \\ END \end{array}$$

It is worth emphasising that the above extensions to Event-B not supported by current tools. In the later sections of this chapter we try to adopt and extend current composition mechanisms to devise the ideas of *Specification Composition* and later the *Architectural Composition.*

As we mentioned it in the introduction of this chapter we follow a practical approach and therefore we introduce our ideas by the means of a case study. Before starting the formal modelling based on our new approach in the next section we introduce the case study informally and very briefly.

5.3 Online Auction System

An online auction system is a typical example of Web-based applications. These type of applications demonstrate a very rich nature of distributed, multi-threaded and transactional-based e-commerce systems. Therefore it makes very desirable to apply our formal patterns to it.

The online auction system is intended to facilitate online transactions between buyers and sellers. The system allows the clients to buy and sell items by means of auctions. Different types of auctions exist, but we consider the English auction because of its popularity. In the English auction the item for sale is put up for auction starting at a relatively low minimum price. Bidders are then allowed to place their bids until the auction closes. In most cases the duration of the auction is fixed in advance, e.g. 30 days.

It is expected that any user with a standard web browser, an internet connection and a basic knowledge of computing will be able to not only shop for items on the auction Website but also set up their own online auctions with ease.

For using the main functionalities of the system the user should register with the system, also it allow all users to brows auctions list. After providing necessary information and registering with the auction system a user can login to the system. The auction system creates an account for each registered user and they have to transfer some funds into their account before bidding for any item. The other main functionalities of the system are login, starting an auction, bidding for an item and transferring from/to the personal account with the auction system.

The auction system has some internal mechanism to determine the end of each running auction, closing it, informing the winner if there is any, and finally transferring funds from the buyer account to the seller account and deducting the related commission. In addition to that some security procedure like blocking a user account after a number of unsuccessful attempts to login or for other legal reasons and log out an inactive client after some period of time may have been envisaged. For our proposes here this short induction of the system should be enough but for interested readers a longer list of informal requirements could be found in the appendix section.

5.3.1 The Architecture of The Auction System

Before starting with the actual formal specification of the auction system, it is necessary to discuss the architecture of underlying platform which the final system would be built on top it. Although the the final implementation of the auction system is not a part of our mission in this chapter but it is very important to acknowledge the significance of platform architecture on design and even specification level modelling. In fact to have an appropriate set of specification and design models, the modelling approach should comply with the proposed architecture which recommended by the platform. Hence it is very important to have a short review of the system architecture. In this chapter we have opted for Java EE [95] as our choice of platform.

Java Platform, Enterprise Edition or Java EE(formerly known as Java 2 Enterprise Edition or J2EE up to version 1.4), is one of the most widely used Platform for developing and running distributed multitier Web applications. Java Web applications are largely based on modular software components running on an application server. The Java EE platform defined a layered architecture for developing application. This layered model consist of different parts as following:

- Client-tier components run on the client machine.
- Web-tier components run on the Java EE server.
- Business-tier components run on the Java EE server.
- Enterprise information system (EIS)-tier software runs on the EIS server.

Although a Java EE application can consist of the four tiers shown in Figure 5.1, Java EE multi-tiered applications are generally considered to be three-tiered applications because they are distributed over three locations: client machines, the Java EE server machine, and the database or legacy machines at the back end.

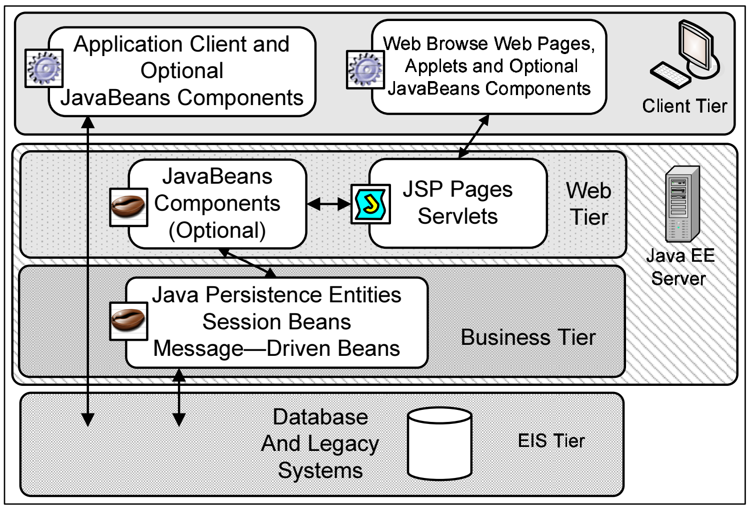

Application Client and Optional JavaBeans Components
Web Browse Web Pages, Applets and Optional JavaBeans Components
Client Tier
JavaBeans Components (Optional)
JSP Pages Servlets
Web Tier
Java EE Server
Java Persistence Entities Session Beans Message—Driven Beans
Business Tier
Database And Legacy Systems
EIS Tier

FIGURE 5.1: Java EE Application Model, Presented in [94]

Here we are not intended to present details of Java EE 5 and we refer the interested readers to the above official Sun documents. But we would like to point out an important issue which is the separation between the interface layer and the application or business logic layer. In the later stage we see how this layering mechanism effects the patterns applicability and our approach to formal modelling of the system.

Based on the above general application architecture we have devised the following block diagram which shows the main parts of the auction system and how they related to the other elements.

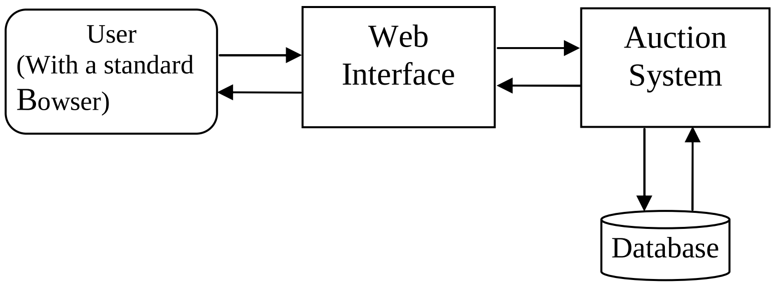


FIGURE 5.2: Architecture of Auction System

The interface provided for the users will be a series of web pages that they can easily access through a standard web browser such as Microsoft Internet Explorer. The separation between the Web interface and the core business logic of the system by using the layering approach provides a mechanism for partitioning the specification. In other words later we see that the specification of the system is comprised of two or more partitions. When we dealing with a large complex system specification this approach should result in a more manageable and clearer specification. Furthermore the combination of this method with the pattern based approach of the previous chapter should lead to a shorter and yet more efficient development process.

In summary, in the rest of this chapter we mainly concentrate on the two following issues and their implications on the formal development process :

- Partitioning the formal specification of the system in some machine in such a way that the formal specification comply more closely with the layered architecture of Java EE.
- Using formal specification patterns to construct our specification machines to achieve higher efficiency and faster development time.

Both of the above subjects have some implications on the formal development process. The partitioning mechanism raises the issue of the composition between different models and this in turn effects the whole specification and refinement approach in B method. In addition to that partitioning mechanism effects the applicability of the specification patterns as well. For example now we have to investigate how the pattern definition could be extended to match the layering mechanism and specification partitioning.

5.4 Specification Partitioning and Composition Techniques

As we discussed in preceding sections, specification partitioning for Web application seems to be very attractive. Breaking the system specification into different parts brings some advantages like modularity and simplicity of models. Another important aim of this approach is to speed up the specification process through parallelism and reusability and we intend to boost the later one by means of formal specification patterns as well.

Based on three-tier architecture of Web applications, it has long being advocated that the *Application Layer* could be analysed and modelled separately from the *Web Layer*. This means that the *Business Analysis* or *Problem Domain Analysis*, as it is also known, could be carried out independent of the *Web Layer* specification and design. In addition to that the *Application Layer* commonly has to be persistent and persistence is usually achieved with a database. Based on this approach we can divide the Auction system specification at least into two separated B machine along with their composition mechanism. The first one models the operations of the *Web Layer* and the second one, the *Application Layer*, models the business logic of the auction system along with the database.

Web Applications are *Event Driven* systems. The Web server receives HTTP Requests from the client. Each *HTTP Request* encapsulate an *Event* which in fact has been defined by the previous page that client's browser has received. When the Web server extracts an event from the request, it must evaluate it and decide how to respond to it. This evaluation process may involve interaction with the *Application Layer*.

The *Web Layer* is also responsible for sending an HTML output stream to the client. Each time that the Web server sends a new Page (or Frame) of HTML to the client , we can consider that as the changing of the state of the client side and this illustrated by the Browser is displaying a new page. Again in this stage to build the appropriate HTML output, the *Web Layer* may need to access application information which can be accomplished by calling the *Application Layer* and acquiring such information from the database. It can then use that data to build an appropriate HTML page or form populated with the necessary details and send it out to the client for display in the browser. These interactions between the two layers in both direction forms the basis for the composition mechanism between the *Web Layer* and the *Application Layer*.

It is destined to use the formal patterns of the previous chapter for the formal development of the auction system whenever it is appropriate. But as a result of the specification partitioning approach, the alignment of the patterns in some cases might be necessary. In this relation two different situations are possible.

In the first case we have a scenario which includes a number of events that they have no connections with the operations of the other machine. In this case we should be able to use our formal patterns with a minimal amendment. On the other hand it is feasible to have a scenario where a part of pattern should be separated out to another machine. In this case there are some connections between operations of the two models and we need to extend the applied formal pattern by devising a suitable composition mechanism.

5.4.1 Combining Formal Patterns and Specification Partitioning

To facilitate pattern application, first we have to identify suitable scenarios in the requirements document. In the second phase each scenario should be analysed to find out how it could be modelled with a pattern or a combination of patterns. In this section, two different scenarios have been chosen to demonstrate our approach. The first case concerns with the *Initial Client Interaction* and how a session could be established between a new client and the auction system. The second case demonstrate the *login* scenario for a previously registered user.

The *Web Layer* is the first point of contact in Web applications and it is responsible for sequencing, state handling and interaction with clients. Therefore we start the specification of the system with this layer. As we proceed, the specification of the *Application Layer* and database in different machine would be introduced and consequently the composition mechanisms could be defined between these layers' specification.

Before starting with the formal specification of different scenarios, it is necessary to discuss some definitions and conventions that we have used throughout the modelling process. The key concepts that we use them frequently are *HTTP Request*, *HTTP Response* and *HTTP Session*. For defining the first two items in the B specification we have used a property-based record style. The detailed structure of these two item is presented in 5.3. The first field of *HTTP Request*, the *ReqID* determines the sender of the request uniquely. We assume that for each newly opened browser window on the client computer the browser produce a unique ID and it assigns to every requests that should be send by this client session. Similarly every produced response by the Web server should be linked to a specific client session. Therefore we use the same Id In the *HTTP Response* and it has been named as *RespID*.

On the server side for each new client session, the Web server creates an object which called *HTTP Session* and links all of this client's information and state variables to this object. The formal specification of this process is presented in the next section. In the B we define the a reference set for *HTTP Session* objects and we identify each instance of this set with a *session ID* or more concisely *SID*. Beside the first request from the each client session all other requests and response should contain a valid session ID. Therefore both *HTTP Request* and *HTTP Response* have a field for session ID and they been named *ReqSID* and *RespSID* respectively.

Browsers work based on the notion of the page. The client browser renders a received response in the form of active viewing page. Within the active page commonly there are

```
HTTP_REQUEST ::
    ReqID         ∈ AGENT_ID
    ReqSID        ∈ HTTP_SESSION
    Req_Page      ∈ PAGE
    Req_Service   ∈ SERVICES
    Data_Field    ∈ DATA

 HTTP_RESPONSE ::
    RespID        ∈ AGENT_ID
     RespSID      ∈ HTTP_SESSION
    Resp_Page     ∈ PAGE
    Resp_Service  ∈ ℙ(SERVICE)
```

Figure 5.3: The Structure of HTTP Request and Response

some options which lined to the available services on that specific page. The third and fourth fields in both *HTTP Request* and *HTTP Response* models the page and services. The only difference between them is that request could include only one requested service. A requested service in the auction system for example could be *Register*, *Login* or *Create_NewAuction*. In most cases with a requested service the client has to provide some information. For example in the case of *Login* these are the *User_Name* and *Password*. The browser stores these items in the *Data_Field* of the *HTTP Request*.

As the record style that we use is not supported by the current B tools, the property based style of these records with set definitions are presented in the Figure 5.4.

```
SETS
  AGENT_ID; HTTP_REQ; HTTP_RESP; HTTP_SESSION; DATA;
  USER; NAME;   SERVICE = {Register, Login, Brows};
  PAGE = {Index_Page, Main_Page, Denial_Page};
  MESSAGE

PROPERTIES
  ∧ Req_ID         ∈ HTTP_REQ → AGENT_ID
  ∧ Req_SID        ∈ HTTP_REQ → HTTP_SESSION
  ∧ Req_Page       ∈ HTTP_REQ → PAGE
  ∧ Req_Service    ∈ HTTP_REQ → SERVICE
  ∧ Data_Field     ∈ HTTP_REQ → DATA

  ∧ Resp_ID        ∈ HTTP_RESP → AGENT_ID
  ∧ Resp_SID       ∈ HTTP_RESP → HTTP_SESSION
  ∧ Resp_Page      ∈ HTTP_RESP → PAGE
  ∧ Resp_Service   ∈ HTTP_RESP → ℙ(SERVICE)
  ∧ Data_Item      ∈ HTTP_RESP → ℙ(AUCTION_CATEGORY)
  ∧ Resp_Item      ∈ HTTP_RESP → ℙ(AUCTION)
```

Figure 5.4: The Property-Based style of HTTP Request and Response

As we mentioned earlier to present our approach in detail in the following sections we have selected two scenarios from the auction system. In the first stage we analyse each scenario to determine whether it should be specified only in the *Web Layer* or it needs to be separated out to the *Application Layer*. If it is so, then the composition mechanism between different parts should formalised.

5.4.2 Modelling Initial Client Interaction and Establishing Session

In the auction system, as is the case in most of Web applications, multiple users interact with the application server simultaneously. Clearly, we need a mechanism to identify each user separately and track their requests over the whole course of conversation. Therefore we can apply the session creation pattern in the same form that we have presented it in Chapter 4. The new user interacts with the system by typing in the initial URL of the auction system. When the server receives initial request which does not contain a valid session ID, it assumes that a new user has joined to the system. Therefore the server creates a new session ID and the related management structure for it. In the next stage it produces a response page, which in this case, is the initial or introductory page. The newly created session ID embedded in the page would be sent to the user. Each subsequent request form that particular client will include the session ID, making the user tracking possible for the Auction system. The B code presented in figure 5.5 gives us a snapshot of this scenario.

On the server side we keep a list of valid session IDs. When the auction system receives a new HTTP request which contain a session ID, it will check it against this list. In the first place the server has to find out whether it is a valid ID and secondly if it is a valid ID to retrieve the related state information for this particular client. All parts of this pattern resides in the *Web Layer* and there no need for messaging with the *Application Layer* in this stage. Therefore no composition mechanism needs to be devised.

5.4.3 Modelling Login Scenario

When the previous stage is accomplished and a new session ID allocated for the new client, the new view should be built and sent back to the client. After the new view is received and rendered by the browser it will provide a new set of options to the client. In the case of the auction system one of the options could be login. The client should provide a user name and password. When the client types in their user name and password and presses the *Login button*, the following sequence of events will take place.

- The browser will send an HTTP request to the Web application. This request contains an event name which is *Login* and in addition it has two parameters which are the *User Name* and the *Password*.

```
MakeHTTP_Request ≙                        /* Client Operation */
  ANY aid , http_req WHERE
       aid ∈ AGENT_ID - used_ ids
     ∧ http_req ∈ HTTP_REQ
     ∧ Req_ID(http_req)= aid
     ∧ Req_SID(http_req)= Null
     ∧ Req_Page(http_req)= Empty
     ∧ http_req ∉ req_hist
  THEN
     used_ids := used_ids ∪ {aid}
   || req_buf := req_buf ∪ {http_req}
   || req_hist := req_hist ∪ {http_req}
  END;
ProcessHTTP_INiRequest ≙             /* Web Server Operation */
  ANY http_req, sid , http_resp WHERE
       http_req ∈ HTTP_REQ
     ∧ http_req ∈ req_buf
     ∧ Req_SID(http_req)= Null
     ∧ Req_Page(http_req)= Empty
     ∧ sid ∈ HTTP_SESSION - current_session
     ∧ http_resp ∈ HTTP_RESP
     ∧ Resp_SID(http_resp) =sid
     ∧ Resp_ID(http_resp) = Req_ID(http_req)
     ∧ Resp_Page(http_resp) = Index_Page
     ∧ http_resp ∉ resp_hist
  THEN
     current_session := current_session ∪ {sid}
   || resp_buf := resp_buf ∪ {http_resp}
   || req_buf := req_buf - {http_req}
   || resp_hist := resp_hist ∪ {http_resp}
  END;
```

FIGURE 5.5: Session creation in the Auction system

- When the HTTP request arrived at the Web Server, it invokes the Servlet. This Servlet evaluates the request specially for a valid session ID and when it found, the controller in the Servlet retrieves related state data for this session.
- The event name and accompanied parameters along with the retrieved state data in the form of a message would be sent to the *Application Layer*.
- Based on the data received by the *Application Layer*, it will produce a response of either *success* or *fail*. In the case of failure, the response could contain extra information about the exact nature of the fault. This response will be send back to the *Web Layer*. To produce this response the *Application Layer* needs to consult the stored data in the database.
- The Web interface layer based on the received message from the *Application Layer*, should decide which view should be build and send to the client. If the *Application Layer*'s response was positive the next view would be the main page of the auction system with available options for a logged-in user. Otherwise the client would be

asked to retry or if the maximum number of attempt has been reached the service would be denied.

- Although it might not be the case for the*Login Response Page*, but in some other cases when the *Web Layer* is building up the response page it may need to communicate with the *Application Layer* for some extra necessary data to construct the response page.

The above scenario is compatible with the formal definition of our generic *User-to-Web application* pattern. But clearly here we have to extend this pattern to deal with the new layout that processing has divided between the Web server and the *Application Layer*. The formal specification consists of two separate models and the composition mechanism definition as well.

The first model in Figure 5.6 specifies the data and operations of the *Web Layer*. This formal specification defines how the incoming requests form different clients are being linked to state data on the server side. In the next stage embedded events and their related parameters should be extracted from the initial *HTTP_Request*. After preliminary checks, if the received event and its parameters were valid, the *Web Layer* will pass them to the *Application Layer*. In the *Application Layer* based on stored data a decision will be made about the outcome of the requested event and a response will be produced and sent back to the *Web Layer* in the form of a message. The *Web Layer* based on the received response from the *Application Layer*, will built the appropriate page for the client and send it in the form an HTTP response to that client.

The main complexity of this model is layering in the *Login_User* event. A new syntax has been devised for the *ANY* substitution which is not supported by the current B tools. This style has been inspired by the CSP style of value passing based event synchronisation as we discussed it in section 5.2. In this operation we have three types of parameters. The first parameter, *http_req* is an ordinary parameter. The next three parameters, which are *sid!*, *uu!* and *pp!* are output parameters. The exclamation mark has been borrowed from CSP to indicate the type of variables that can send out some value to another event. In this case we want to send these parameters to an event in the *Application Layer*. The third type of parameter is *mesg?* where the question mark denotes that this is an input parameter. In this case the input is being provided by some event in the *Application Layer*. As it is evident from the specification the *Web Layer* response to the client is being build based on this input parameter in the body of the *Login_User* event. The exact detail of synchronisation mechanism will be discussed when we introduce the other elements. But it is important to notice that here in this operation we have both input and output parameters and this is an extension of parallel composition which defined by Butler in [114].

The second model encapsulates the underlying business logic of the login process in the auction system. This model is presented in 5.7. Based on the actual value of the input

```
Client_Request_Login ≙        /* Client Operation */
 ANY http_req WHERE
    http_req ∈ HTTP_REQ
    ∧ Req_Service(http_req) = Login
    ∧ http_req ∉ req_hist
 THEN
    req_buf := req_buf ∪ {http_req}
    || req_hist := req_hist ∪ {http_req}
 END;

Login_User ≙           /* Synchronising Operation */
 ANY http_req, sid! , uu!, pp!, mesg? WHERE
    http_req ∈ req_buf
    ∧ sid! = Req_SID(http_req)
    ∧ sid! ∈ current_session
    ∧ Req_Page(http_req) = view(sid!)
    ∧ Req_Service(http_req) = Login
    ∧ uu! = User(Data_Field(http_req))
    ∧ pp! = Name(Data_Field(http_req))
    ∧ mesg? ∈ MESSAGE
 THEN
   pending_session(sid!):= Req_ID(http_req)
   || req_buf := req_buf - {http_req}
   || view(sid!) := Next_View(view(sid!), mesg?)
 END;

Respond_Login  ≙         /* Web Server Operation */
 ANY sid, http_resp WHERE
    sid ∈ dom(pending_session)
    ∧ http_resp ∈ HTTP_RESP
    ∧ Resp_ID(http_resp) = pending_session(sid)
    ∧ Resp_SID(http_resp) =sid
    ∧ Resp_Page(http_resp) = view(sid)
    ∧ http_resp ∉ resp_hist
 THEN
   resp_buf := resp_buf ∪ {http_resp}
   || pending_session := {sid} ⩤ pending_session
   || resp_hist := resp_hist ∪ {http_resp}
 END
```

FIGURE 5.6: Web Layer - First Part of Login Scenario

parameters different scenarios are possible. Each possible scenarios has been modelled with a separate event. The *Login_Success* is the only case of success and the rest of operations represent the different failure scenarios. The guards in each failure operation demonstrate the condition which could lead to the failure of the login process.

All operations in this segment of the model have the same set of parameters. The input and output parameters have been indicated by the question and exclamation mark in the end of parameters respectively. The parameters in the *Application Layer* are counterparts of operation of the *Web Layer*. More precisely each input parameter in the *Web Layer*'s operations at least has an output counterpart parameter in the *Application Layer*'s operation and vice versa. The exact connection between operations

```
Login_Success ≙
 ANY sid? , uu?, pp? ,mesg! WHERE
    sid?   ∈ current_session
    ∧ uu?  ∈ registered_user
    ∧ pp?  ∈ PASSWORD
    ∧ pp?  = passwords(uu?)
    ∧ uu?  ∉ loggedon_user
    ∧ uu?  ∉ blocked_user
    ∧ mesg! = LOGIN_SUCCESS
 THEN
   loggedon_user := loggedon_user ∪ {uu?}
   || try_count(sid?)  :=  0
   || session_user(sid?)     := uu?
 END;
Login_Failed_notRegistered ≙
 ANY sid?,uu?, pp?, mesg! WHERE
    sid?   ∈ current_session
    ∧ uu?  ∈ USER
    ∧ pp?  ∈ PASSWORD
    ∧ uu?  ∉ registered_user
    ∧ try_count(sid?) < 3
    ∧ mesg! = USER_NOT_REGISTERED
 THEN
   try_count(sid?) := try_count(sid?) + 1
 END;
 Login_Failed_IncorrectPass ≙
 ANY sid?,uu?, pp?, mesg! WHERE
    sid?   ∈ current_session
    ∧ uu?  ∈ USER
    ∧ pp?  ∈ PASSWORD
    ∧ passwords(uu?) ≠ pp?
    ∧ try_count(sid?) < 3
    ∧ mesg! = PASSWORD_NOT_CORRECT
 THEN
   try_count(sid?) := try_count(sid?) + 1
 END

Login_Failed_HasLogedin ≙
 ANY sid?,uu?, pp?, mesg! WHERE
    sid?   ∈ current_session
    ∧ uu?  ∈ USER
    ∧ pp?  ∈ PASSWORD
    ∧ uu?  ∈ loggedon_user
    ∧ try_count(sid?) < 3
    ∧ mesg! = USER_HAS_LOGGEDIN
 THEN
   try_count(sid?) := try_count(sid?) + 1
 END;
 Login_Failed_UserBlocked ≙
 ANY sid?,uu?, pp?, mesg! WHERE
    sid?   ∈ current_session
    ∧ uu?  ∈ USER
    ∧ pp?  ∈ PASSWORD
    ∧ uu?  ∈ blocked_user
    ∧ try_count(sid?) < 3
    ∧ mesg! = USER_IS_BLOCKED
 THEN
   try_count(sid?) := try_count(sid?) + 1
 END;
Login_Denied ≙
 ANY sid?, uu?, pp?,mesg! WHERE
  ( sid?  ∈ current_session
  ∧ uu? ∈ USER ∧ pp? ∈ PASSWORD
  ∧ try_count(sid?) >= 3
  ∧ mesg! = MAXTRY_SERVICE_DENIED )
  ∧
   ( uu?  ∉ registered_user
   ∨ passwords(uu?) ≠ pp?
   ∨ uu? ∈ loggedon_user
    ∨ uu? ∈ blocked_user )

 THEN
   skip
 END
```

Figure 5.7: Application Layer - Second Part of Login Scenario

of two machines and their input and output parameters should be defined by the third component which is the composition mechanism definition.

The composition mechanism which defines how different parts of the system should be linked together can have different forms. As we discussed earlier in section 5.2 this approach has been inspired by value passing event in CSP formalism.

In initial parallel composition mechanism an output event from one system is composed with a corresponding input event from another system. The composition has been done in such a way that the output value from one event becomes the input value for the other event. The formal semantics of composed system with output parameters passing value to input variables could be presented by a single joint parameter.

In our work here we have extended the idea of fused or composed pairs of events in three different aspects. The first aspect allows some relaxation on the naming convention in such a way that different names for fused events could be used. The second aspect concerns communication links in which one way communication has been extended to bidirectional communication. The third and the most important aspect is the extension of one-to-one composition to a one-to-many composition. We discuss these three aspect in more details in the following.

In the initial composition approach the name of composed event in the two different system should be identical, but here we consider to compose events with different names. For example we want to compose the *Login_User* event in the *Web Layer* specification with the *Login_Success* event in the *Application Layer* specification. Although the relaxation of the shared name convention gives us a great degree of freedom, it introduces some new challenging issues both semantically and syntactically. We can solve the semantics issue by the means of renaming mechanism similar to what we have in CSP. The syntax issue should be dealt with by incorporating a third element into system specification which is the definition of composition mechanism. A simple way for defining the composition mechanism could be a lookup table that defines the input and output events. In the reminder of this section we should discuss this issue in more details.

The second aspect of extension in the initial parallel composition is related to the communication. The communication in the initial work [114] is one directional. In other words one event is an output event with one or more output parameters and possibly some independent parameters and the counterpart event is an input event with input parameter(s). In our approach we allow bidirectional communication in the sense that we have both input an output parameters in a single event. For example in the *Login_User* event we have three output parameters which are *Session ID*, *User Name* and *Password* and we have one input parameter which is a *Message* from the *Application Layer*. A similar situation could be seen in the *Login_Success* event in the *Application Layer* specification that we have three input parameters and one output parameter.

The idea of having simultaneous input and output parameters in a single event makes the specification composition very brief and yet comprehensive. It enables us to fuse systems and events together without being concerned about underlaying architectural complexity. But it should be highlighted that simultaneity between input and output is not possible in the implementation level. As a result we have to refine the specification level composition to an architectural or design level composition in the later stages of formal development process. In the next section we examine this issue in detail.

The last and the most important aspect of extension of the initial parallel composition which we introduce here is the one-to-many composition. Replacing one-to-one parallel composition with a table-based defined composition gives us a great amount of flexibility that we need need to model more sophisticated composition scenarios. A good example of such cases could be seen in the *Login* operation. In the *Web Layer* specification of the *Login* operation we have one event which is the *Login_User* event. The three output parameters of this event should be seen by all six events of the *Application Layer* which are *Login_Success*, *Login_Failed_notRegistered*, *Login_Failed_IncorrectPass*, *Login_Failed_HasLogedin*, *Login_Failed_UserBlocked* and *Login_Denied*. This could be observed through the corresponding input and output parameters in composed events. On the other hand based on actual value of parameters in runtime situation and regarding the mutual nature of the events' guards in the *Application Layer* only one event should be enabled. When the enable event executed, it should provide the output parameter for the *Login_User* event in the *Web Layer*. Here we can see that this type of composition has a nondeterministic nature that could be modelled with a choice.

As a direct result of the above extensions, the naming convention that defines the composition mechanism implicitly no longer could be sufficient. Therefore we need a third element in our modelling approach to define and store the explicit definition of the extended composition mechanism. In the next section we examine this issue in detail.

Defining the Specification Composition

Considering the fact that extended composition mechanism now is a substantial element, and it could not be comprehended directly from the specification of the *Web* or *Application Layer*, it seems inevitable to have a new part that defines the composition. To have a better understanding of the different aspect of the extended composition mechanism in the *Login* scenario, we have depicted it informally in Figure 5.8.

Form Figure 5.8 it could be comprehendible that the composition mechanism definition should includes the following aspects:

- The name of the input and output event(s)
- The exact nature of communication between composed events, i.e. one-directional or bi-directional, for example from Figure 5.8 we can see that the *Login_User* event in the *Web Layer* is both input and output event in the same time.
- The type of composition for example one-to-one parallel or one-to-many composition, in the case of login scenario, the composition of the *Login_User* event with multiple events in the *Application Layer* is one-to-many composition.
- The input and output parameter(s)

As we mentioned in the previous section this information could be stored in a lookup table in a separate file a new element of the modelling. The tools can retrieve this information and check it against the definition of the composed events in different machines to produce and discharge the the proof obligations.

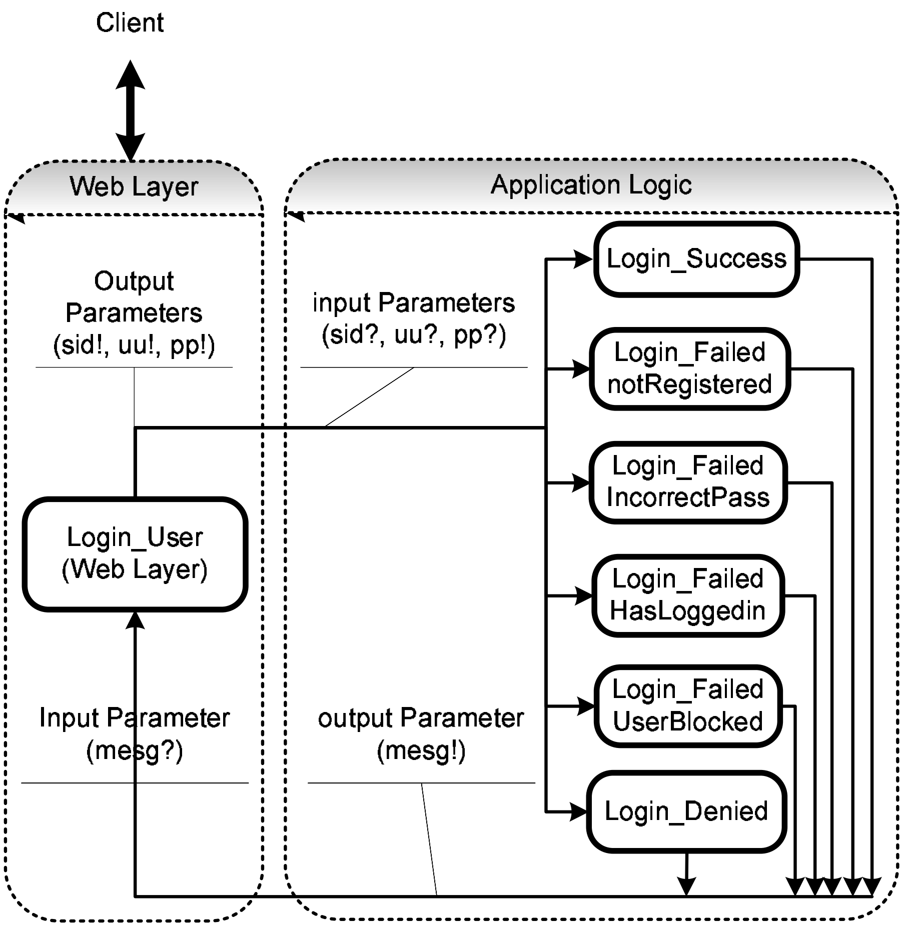


Figure 5.8: Informal Illustration of Composition Mechanism for Login Scenario

To provides a better view on formal definition of the composition mechanism and how events in different part of a system are composed, an abstract formal model of the composition for*Login* scenario has been presented in Figure 5.9. In the *Web Layer* specification the *Client_Request_Login* and *Respond_Login* are independent operations and they should remain unchanged in the composed system. The *Login_User* operation in the *Web Layer* has to be fused to all the events in the *Application Layer*. In the composed events the new guard are conjunction of the initial events' guards and the body of composed events are the parallel composition of both event's statements. Another important issue that should be noticed is that the composition of input and output parameters are becoming outputs.

The specification composition which has been introduced in this section initiate a new prospect for modelling of complex multilayered systems, but as we mentioned previously is not directly implementable. Therefore in new section we try to refine it in such away to make it more realistic.

```
 :
Client_Request_Login ≙
   /* Client Operation */
 ANY http_req WHERE
  $G_{M1}$ (http_req, …)
 THEN
  $S_{M1}$(http_req, …)
 END;

Login_User ≙
 ANY http_req, sid! , uu!, pp!, mesg?
WHERE
 $G_{M2}$ (http_req , sid! , uu!, pp!, mesg?)
 THEN
  $S_{M2}$(http_req, sid! , uu!, pp!, mesg?)
 END;

Respond_Login ≙
 ANY sid, http_resp WHERE
 $G_{M3}$ (sid, http_resp)
 THEN
  $S_{M3}$(sid, http_resp, …)
 END;   :
```

(a) The Web Layer's Operations

```
Login_Success ≙
 ANY sid? , uu?, pp? ,mesg! WHERE
   $G_{N1}$(sid? , uu?, pp? ,mesg!)
 THEN
   $S_{N1}$(sid? , uu?, pp? ,mesg!)
 END;
Login_Failed_notRegistered ≙
 ANY sid?,uu?, pp?, mesg! WHERE
   $G_{N2}$ (sid? , uu?, pp? ,mesg!)
 THEN
   $S_{N2}$(sid? , uu?, pp? ,mesg!)
 END;
:
:
Login_Denied ≙
 ANY sid?, uu?, pp?,mesg! WHERE
   $G_{N6}$(sid? , uu?, pp? ,mesg!)
 THEN
   $S_{N6}$(sid? , uu?, pp? ,mesg!)
 END
```

(b) The Application Layer's Operations

```
 :
Client_Request_Login ≙
   /* Client Operation */
 ANY http_req WHERE
  $G_{M1}$ (http_req, …)
 THEN
  $S_{M1}$(http_req, …)
 END;
Login_User || Login_Success ≙
 ANY http_req,  sid! , uu!, pp!,mesg! WHERE
  $G_{M2}$ (http_req , sid! , uu!, pp!,mesg!)
  ∧  $G_{N1}$(sid! , uu!, pp!,mesg!)
 THEN
  $S_{M2}$(http_req, sid! , uu!, pp!,mesg!)
  || $S_{N1}$(sid! , uu!, pp!,mesg!)
 END;
Login_User || Login_Failed_notRegistered ≙
 ANY http_req,  sid! , uu!, pp!,mesg!  WHERE
  $G_{M2}$ (http_req , sid! , uu!, pp!,mesg!)
  ∧ $G_{N2}$ (sid! , uu!, pp!,mesg!)
 THEN
  $S_{M2}$(http_req, sid! , uu!, pp!,mesg!)
  || $S_{N2}$(sid! , uu!, pp!,mesg!)
 END;
```

```
Login_User || Login_Failed_IncorrectPass ≙
 ANY http_req,  sid! , uu!, pp!,mesg! WHERE
  $G_{M2}$ (http_req , sid! , uu!, pp!,mesg!)
 ∧ $G_{N3}$(sid! , uu!, pp!,mesg!)
 THEN
  $S_{M2}$(http_req, sid! , uu!, pp!,mesg!)
  || $S_{N3}$(sid! , uu!, pp!,mesg!)
 END
 :
 :
Login_User || Login_Denied ≙
 ANY http_req,  sid! , uu!, pp!,mesg! WHERE
  $G_{M2}$ (http_req , sid! , uu!, pp!, mesg!)
 ∧ $G_{N6}$(sid! , uu!, pp!,mesg!)
 THEN
  $S_{M2}$(http_req, sid! , uu!, pp!,mesg!)
  || $S_{N6}$(sid! , uu!, pp!,mesg!)
 END
Respond_Login ≙
 ANY sid, http_resp WHERE
 $G_{M3}$ (sid, http_resp)
 THEN
  $S_{M3}$(sid, http_resp,…)
 END;
```

(c) The Composed System's Operations

FIGURE 5.9: Formal Presentation of Composition

Architectural Level Composition

In previous section it has been pointed out that bi-directional composition with simultaneous input-output is not implementable. This is due to the atomicity of events that are producing input and output in the same time. In reality producing a response could not be simultaneous with receiving inputs, because for producing a response it may need other parts to involved. For example in the case of *Login* when the *Application Layer* receives the input parameters from the *Web Layer* they need to be checked against the information in the database. Obviously this process takes time and the *Application Layer*'s events could not produce a response, or equivalently the output parameters, simultaneous with receiving the input parameters. For this reason it is necessary to refine the specification composition in such a way to allow intermediary processing to happen in between. We call this composition the *Architectural Level Composition* or simply *Architectural Composition.*

To illustrate the idea of the architectural composition we use the login scenario again. The refined formal model of *Web Layer* for login has been presented in 5.10. One major difference between this model and the previous one, is the new event *Request_for_Login.* The main motivation behind this change is to separate the input and output composition. This in turn makes it possible to have intermediary processing in between composed events and over the problem of atomicity.

```
Client_Request_Login ≙
 /* Client Operation */
 ANY http_req WHERE
    http_req ∈ HTTP_REQ
    ∧ Req_Service(http_req) = Login
    ∧ http_req ∉ req_hist
 THEN
  req_buf := req_buf ∪ {http_req}
  || req_hist := req_hist ∪ {http_req}
 END;

Request_for_Login ≙
/* Web Server Operation */
 ANY http_req, sid!, uu!, pp! WHERE
  http_req   ∈ req_buf
  ∧ sid!      = Req_SID(http_req)
  ∧ sid!      ∈ current_session
  ∧ Req_Page(http_req) = view(sid!)
  ∧ Req_Service(http_req) = Login
  ∧ uu! = User(Data_Field(http_req))
  ∧ pp! = Name(Data_Field(http_req))
 THEN
  pending_session(sid!):= Req_ID(http_req)
  || req_buf := req_buf - {http_req}
 END;
```

```
Login_User ≙
/* Web Server Operation */
 ANY sid?, mesg? WHERE
    sid?     ∈ HTTP_SESSION
    ∧ sid?    ∈ dom(pending_session)
    ∧ mesg?  ∈ MESSAGE
 THEN
  completed(sid?) := pending_session(sid?)
  || view(sid?) := Next_View(view(sid?), mesg?)
  || pending_session:= {sid?}⩤ pending_session
 END;

Respond_Login ≙
 /* Web Server Operation */
 ANY sid, http_resp WHERE
    sid ∈ dom(completed)
    ∧ http_resp ∈ HTTP_RESP
    ∧ Resp_ID(http_resp) = completed(sid)
    ∧ Resp_SID(http_resp)  = sid
    ∧ Resp_Page(http_resp) = view(sid)
    ∧ http_resp ∉ resp_hist
 THEN
   resp_buf := resp_buf ∪ {http_resp}
   || completed := {sid} ⩤ completed
   || resp_hist := resp_hist ∪ {http_resp}
 END
```

FIGURE 5.10: Refined Web Layer Model for *Login*

The change in composed events brings some other changes in to the models. For instance the number of parameter in the *Login_User* event 5.10 has changed as a result of introducing the new *Request_for_Login* event. In the new layout we do not need both input and output parameters in the same event any longer. For example all parameters beside one ordinary parameter in the *Request_for_Login* event are output parameters.

The new formal model for the *Application Layer* has been presented in Figure 5.11. In this model we have a new event which is the corresponding event for *Request_for_Login* in the *Web Layer* with the same name. Obviously because the inbound and outbound composition have bean separated the event in the *Application Layer* only has input parameters. As a result here we have a much simpler one-to-one parallel inbound composition between the two *Request_for_Login* events in the *Web Layer* and *Application Layer* respectively. The outbound composition also has changed in the sense that we on longer

```
Request_for_Login ≙
 ANY sid?, uu?, pp? WHERE
    sid? ∈ current_session
    ∧ uu? ∈ USER
    ∧ pp? ∈ PASSWORD
 THEN
   session_user(sid?)     := uu?
   || user_password(sid?) := pp?
 END;
Login_Success ≙
 ANY sid!, uu, pp, mesg! WHERE
  …
    ∧ (sid! ↦ uu) ∈ session_user
    ∧ (sid! ↦ pp) ∈ user_password
    ∧ uu ∈ registered_user
    ∧ passwords(uu) = pp
    ∧ uu ∉ loggedon_user
    ∧ uu ∉ blocked_user
    ∧ mesg != LOGIN_SUCCESS
 THEN
   loggedon_user := loggedon_user ∪ {uu}
   || try_count(sid!)  :=  0
   || session_user := {sid!} ⩤ session_user
   || user_password := {sid!} ⩤ user_password
 END;
Login_Failed_notRegistered ≙
 ANY sid!,uu, pp, mesg! WHERE
..
    ∧ (sid! ↦ uu) ∈ session_user
    ∧ (sid! ↦ pp) ∈ user_password
    ∧ uu ∉ registered_user
    ∧ try_count(sid!) < 3
    ∧ mesg! = USER_¬_REGISTERED
 THEN
   try_count(sid!) := try_count(sid!) + 1
   || session_user := {sid!} ⩤ session_user
   || user_password := {sid!} ⩤ user_password
 END;
```

```
Login_Failed_IncorrectPass ≙
 ANY sid!,uu, pp, mesg! WHERE
…
    ∧ (sid! ↦ uu) ∈ session_user
    ∧ (sid! ↦ pp) ∈ user_password
    ∧ passwords(uu) ≠ pp
    ∧ try_count(sid!) < 3
    ∧ mesg! = PASSWORD_¬_CORRECT
 THEN
   try_count(sid!) := try_count(sid!) + 1
   || session_user := {sid!} ⩤ session_user
   || user_password := {sid!} ⩤ user_password
 END;
 Login_Failed_HasLogedin ≙
 ANY sid!,uu, pp, mesg! WHERE
…
    ∧ (sid! ↦ uu) ∈ session_user
    ∧ (sid! ↦ pp) ∈ user_password
    ∧ uu ∈ loggedon_user
    ∧ try_count(sid!) < 3
    ∧ mesg! = USER_HAS_LOGGEDIN
 THEN
   try_count(sid!) := try_count(sid!) + 1
   || session_user := {sid!} ⩤ session_user
   || user_password := {sid!} ⩤ user_password
 END;
 Login_Failed_UserBlocked ≙
 ANY sid!,uu, pp, mesg! WHERE
….
    ∧ (sid! ↦ pp) ∈ user_password
    ∧ uu ∈ blocked_user
    ∧ try_count(sid!) < 3
    ∧ mesg! = USER_IS_BLOCKED
 THEN
   try_count(sid!) := try_count(sid!) + 1
   || session_user := {sid!} ⩤ session_user
   || user_password := {sid!} ⩤ user_password
 END;
```

FIGURE 5.11: Refined Application Layer Model for *Login*

have both input and output parameters in the same event. In addition to that, the number of output parameter in composed events has changed, but the composition pattern remans unchange. The new composition mechanism has been depicted in Figure 5.12 informally. As it could be seen from Figure 5.12 the inbound composition Between the *Web Layer* and the *Application Layer* is very simple *one-to-one parallel composition.* On the other hand outbound composition is the more complex *Many-to-One Composition.* The *Many-to-One Composition* here is very similar to the composition that we had in

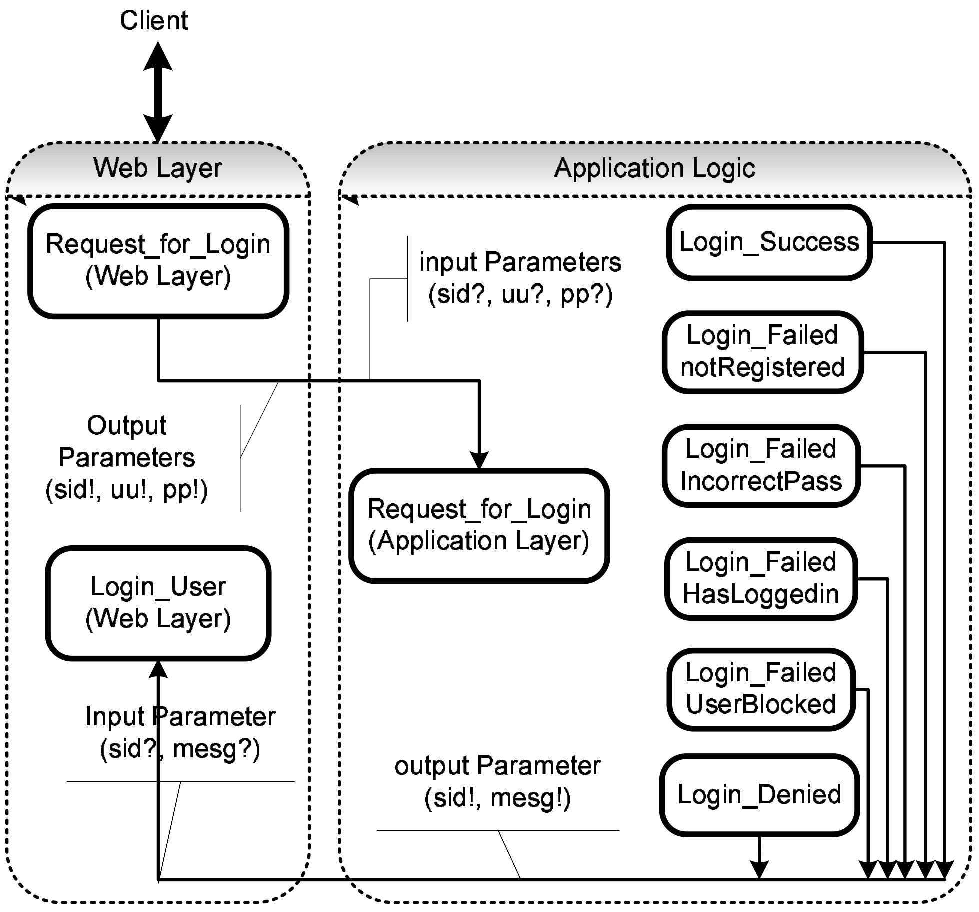

Client
Web Layer
Application Logic
Request_for_Login (Web Layer)
Output Parameters (sid!, uu!, pp!)
Login_User (Web Layer)
Input Parameter (sid?, mesg?)
input Parameters (sid?, uu?, pp?)
Request_for_Login (Application Layer)
output Parameter (sid!, mesg!)
Login_Success
Login_Failed notRegistered
Login_Failed IncorrectPass
Login_Failed HasLoggedin
Login_Failed UserBlocked
Login_Denied

FIGURE 5.12: Informal Illustration of the Refined Composition for the Login Scenario

the specification level. In fact the formal definition of them is almost identical and this similarity leads us to the new concept of *Composition Patterns.* The refined composition mechanism is modelled by a lookup table similar to the specification composition. In the next section we try to sketch out the bases of this idea and how it could be merged with the specification pattern to enable us to reach a higher level of productivity and reusability.

5.5 Composition Patterns

A composition mechanism defines the way that different part of a system could be fused together. To construct a formal model for our Auction System, in the last two section, we devised some new composition mechanisms. These composition mechanisms extending the preliminary ideas of composition which we presented them in section 5.2. It seems perfectly justifiable to consider the reusability of composition mechanisms including those we have reviewed in section 5.2. To endorse the reusability of composition mechanisms we prefer to define them as *Composition Patterns.* The idea of composition patterns along with the specification patterns that we defined in the previous chapter could serve as a framework for future works on formal modelling of Web applications and Web services. In this section we want to review all patterns including those that has been presented in section 5.2.

5.5.1 Basic Parallel Composition

In this form of composition events in two different model could be fused together. This a simple synchronisation mechanism between a pair of event without any value passing. The joint event guard is the conjunction of both events guards and the composed system will engaged in the composed event when both guards are enable. This pattern has been defined in Butler works and it is presented in section 5.2.

5.5.2 Parallel Composition with Value-Passing

The previous simple parallel composition could be extended by introducing the idea of communicating values. In this form of composition we have a pair composed event that can synchronise by value passing. One event could produce one or more output that could be accepted by the second event as input. We simply call the first event as output event and the later one as input event. As an accepted variant each event could deal with a number of independent variables in addition to input or output parameters. This pattern has been defined in Butler works and it is presented in section 5.2.

5.5.3 Broadcasting Composition

The *Broadcasting Composition* pattern or *Multiple-Parallel Composition with Value-Passing* could be seen as an extension of the previous pattern in some way. In this pattern we have a single output event which could synchronise with more that one input event. The input events in the initial pattern have the same name but the belong to different machines. This pattern has been defined in [114] but we did not present it in section 5.2, because it has not been used in the auction case study. This pattern is very useful for situation where a subsystem should ask multiple secondary subsystems or systems for a service. A good example of this scenario could be observed in the travel case study of Chapter 3 where the main server has to query the secondary servers for a service by broadcasting a request for service.

The formal definition of this pattern could be presented as following:

Here the ev_{M_1} is an output event in the machine M_1 and the rest of events are input event each belong to a separate machine.

All the above composition patterns were simple in the sense that they could be defined implicitly by the naming convention i.e. the composed events had the same name. In

$$
\begin{aligned}
ev_{M_1} &\mathrel{\widehat{=}} ANY\ x!\ WHERE\ G_1(x!, m_1)\ THEN\ S_1(x!, m_1)\ END \\
ev_{M_2} &\mathrel{\widehat{=}} ANY\ x?\ WHERE\ G_2(x?, m_2)\ THEN\ S_2(x?, m_2)\ END \\
&.. \\
ev_{M_n} &\mathrel{\widehat{=}} ANY\ x?\ WHERE\ G_n(x?, m_n)\ THEN\ S_n(x?, m_n)\ END
\end{aligned}
$$

$$
ev_{M_1} \parallel ev_{M_2} \parallel .. \parallel ev_{M_n} \mathrel{\widehat{=}} \begin{array}{l} ANY\ x!\ WHERE \\ \ G_1(x!, m_1) \wedge G_2(x!, m_2) \wedge .. \wedge G_n(x!, m_n) \\ THEN \\ \ S_1(x!, m_1) \parallel S_2(x!, m_2) \parallel .. \parallel S_n(x!, m_n) \\ END \end{array}
$$

addition to that we were using the same names for input and output parameters hence there was no need for *Explicit Formal Definition* of the composition mechanism. But in the rest of following patterns this no longer is the case.

5.5.4 Choice Composition

We used this composition pattern in login case scenario for the first time, without naming it. In the first look this composition seems very similar to the previous composition pattern, but its semantics is different. Three forms of it could be envisaged.

(a) Fusing an output event in the first model/machine with multiple input events in the second model/machine

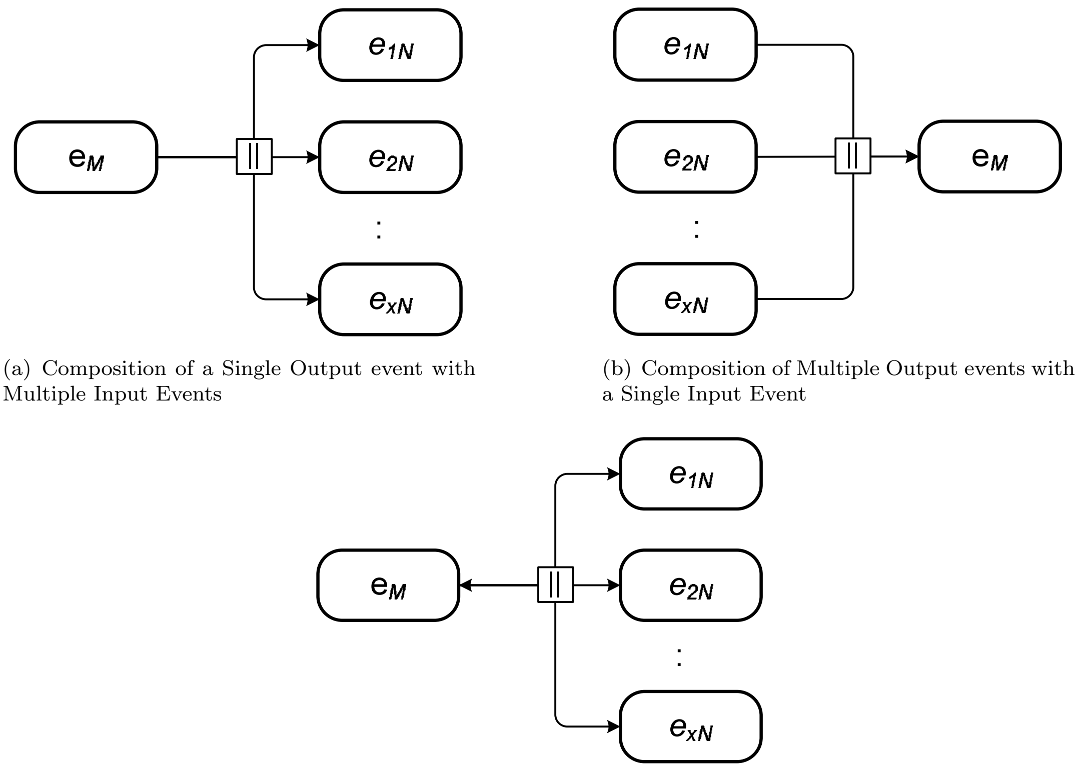
(b) Fusing multiple output events in the first model/machine with a single input event in the second model/machine

(c) Combination of the above case in the form composing an input/output event with multiple input/output events in the second model/machine.

An informal presentation of all three forms is depicted in Figure 5.13.

This pattern has the following differences with the broadcasting pattern:

1. Here the composition is just between two machines/models.
2. The naming convention could not be applied, because we have multiple events in one machine that participate in the composition.
3. Composition between a single input event and multiple output event is possible. Despite the hint in the [114], that this may leads to deadlock, here there is no such danger. The reason for that in the mutuality of the output events' guard. The hidden fact in this pattern is this: the guards in the grouped events are mutually



(a) Composition of a Single Output event with Multiple Input Events

(b) Composition of Multiple Output events with a Single Input Event

(c) Composition of an Input/Output event with Multiple Input/Output Events

FIGURE 5.13: Informal Presentation of the Choice Composition

exclusive. Therefore in any possible state only one of the them will have a valid guard and could be activated.

Considering the fact that in runtime only one pair of events including one input and one output will be composed in parallel manner, make this pattern very similar to the *Internal Choice* in CSP. The name of this pattern is chosen to reflect this fact. In the following Figure the formal definition for the first form of this pattern is presented. Deriving a formal definition for the other two forms is very similar.

$$ev_M \;\widehat{=}\; ANY\ x!\ WHERE\ G_M(x!, m)\ THEN\ S_M(x!, m)\ END \;\in\; M$$

$$\left.\begin{array}{l} ev1_N \;\widehat{=}\; ANY\ x?\ WHERE\ G_{1N}(x?, n_1)\ THEN\ S_1(x?, n_1)\ END \\ : \\ : \\ evx_N \;\widehat{=}\; ANY\ x?\ WHERE\ G_{xN}(x?, n_x)\ THEN\ S_x(x?, n_x)\ END \end{array}\right\} \;\in\; N$$

$$ev_M \boxed{\|} \{ev1_N, ev2_N, ..., evx_N\} \;\widehat{=}\; \left\{ \begin{array}{l} CHOICE \\ \quad ev_M \;\|\; ev1_N \\ OR \\ \quad ev_M \;\|\; ev2_N \\ : \\ OR \\ \quad ev_M \;\|\; evx_N \\ END \end{array} \right.$$

5.5.5 Conclusions and Future Work

In this chapter we introduced a new approach for formal development of Web based application that could be equally beneficial for modelling other type of complex systems which consist of many parts. In this new approach which is based on specification partitioning and composition, we start the specification process by constructing a number of specification models. Each specification model represents a part or a subsystem of the whole system. The relations between different specification models are defined by some composition mechanisms or patterns. In general the formal definitions of composition patterns should be given explicitly, although in simple cases the composition could be defined implicitly by the naming convention like CSP.

The main advantages of this approach are that it promote modularity, reusability and team based system development. Because in this approach the system specification includes several parts, each part should model a specific set of coherent requirements and the modelling could be done by an individual in parallel with other part of the system. Specification partitioning could assist with the idea of separation of concerns that intended toward concentrating on one issue at a time. This in turn should resulted in cleaner specification and design with higher reusability.

As it is the case in many engineering processes, the above advantages could not be achieved without any cost. The cost of this approach is the overhead of defining the composition mechanism in separate explicit model and obviously, the extension of current tools to support this approach.

Chapter 6

Conclusions

This chapter summarises the main ideas presented in this thesis. In addition, future research directions in the formal development of Web applications are discussed. Using the B Method for development of distributed systems in general, and Web applications particulary, has some implications on the B method itself. These could be in the form of some suggested syntax and semantics extensions. In the following sections these issues are discussed.

6.1 An Overview of the Research

The opening chapter provides the introduction, motivations and contributions of this thesis. Chapter 1 gives an overview of the history of the Web from a document sharing medium to a platform for the development of large scale distributed applications. We explained that the need for development of more sophisticated systems with a higher level of functionality was a real force behind the need for appropriate analysis and modelling before any real implementation. As a result of this real need, different modelling approaches emerged. Some of these modeling approaches had roots in hypermedia modelling while others derived form the general field of software engineering. The need for a more comprehensive approach leads to the merging of some previously introduced modelling approaches. It assumed that the combined approaches should provide a higher level of modelling, like architectural and business level modelling.

Web applications are critical to the day to day operation and success of many organizations. Many Web applications in that sense could be classified as *mission critical systems* and therefore they should be highly dependable. To have a dependable system we need to have a verifiable modelling approach as well as tool support for it. The survey in Chapter 1 shows that although there has been a great advance in Web modelling approaches during last few years, but almost all of approaches fail to address the issue of verifiability. The need for dependable web application with verifiable development process and tool support lead us toward the use of formal method for Web application development as the ultimate aim of this research.

Chapter 2 surveys dependable software engineering and its different aspects. It identifies that the use of formal methods is an eminent way to develop dependable systems. A review of different formalisms and the domain of their applicability has been discussed. Assessing some essential aspects like tool support, the completeness of notation and the extent of their previous use in developing practical systems lead us to the selection of B method. A short review of the B method notation, and some proposed extensions to the standard B method to make it more suitable for system modelling have been presented.

Combination of the B method with other formal or semiformal notations has been discussed in many papers. In this relation the process algebraic notation, CSP, has a central role to inspire some suggested extensions to this research. Therefore a short review of the CSP is provided in the end of Chapter 2.

In this thesis we follow a practical approach towards formalising Web based applications. To have a better understanding of these systems a case study, the travel agency, is chosen in Chapter 3. By developing a formal B specification for the travel agency case study we tried to achieve two goals. First, the formal specification should serve as a firm base for understanding , further discussion and designing the system. The second goal was to identify the challenging aspects of formal development and propose some solutions for overcoming them.

A number of issues like requirement handling in complex system modelling, refinement of structured data types, specification of distributed databases and their refinement, and modelling of communication links have been identified as challenging aspects. We tried to provide some solutions for these issues in the rest of Chapter 3.

In Chapter 4 based on our experience in the previous chapter we have explored the idea of specification and refinement patterns. Although in a complex Web application we may face a long list of requirements but in most cases the user interactions with the system could be matched by a few patterns. Based on this fact we have identified a number of generic patterns in Chapter 4 and developed some formal specification and refinement models for them.

The formal specification and refinement patterns could serve as a basis for reusability and faster system development. They also can alleviate the burden of proof obligation discharging by suggesting some useful approaches. We tried as much as possible to make the patterns' definitions generic. This make it possible to apply them to a wider number of cases but it is associated with a negative aspect. This negative aspect is that now we have to specialise patterns for each applied case, which may in turn resulted in different and difficult proof obligations.

Finally in Chapter 5 we introduced specification partitioning as another means to tackle the complexity of formal development process for Web applications. In line with our practical approach, we used another case study in this chapter to discuss the main aspect of specification partitioning. We discussed that composition mechanisms are the major techniques to construct the complete systems specification from subsystems. We explored some example cases and related devised composition techniques for them.

The composition mechanisms or patterns as we labeled them later in Chapter 5 could be subject to refinement in correlation with the whole system refinement. In this relation we presented the idea of composition in different levels like specification and architectural composition. In addition to that we found that the idea of composition pattern could provide yet a better framework for development of formal web applications.

6.2 Major Thesis Contributions

Applying formal methods specially to complex distributed systems like Web application is a very challenging task. When we started this research we were able only to find few previous works relating to the formalisation of Web applications, none of them on applying B to the Web applications.

Based on our practical approach we identified some interesting issues in Chapter 3. Although these cases were extracted from the given case study, but they can be reappear in many other Web applications. We provided some B models for the following cases that can serve as a set of guidelines for future developments:

- A practical approach based on superposition refinement for incorporating requirements into system specifications in a stepwise manner
- Introducing a property based approach for stepwise refinement of structured data types in the B method.
- Specification and refinement of communication links by starting from single-place buffers refined by multi-place buffers with order preservation and later unordered multi-place buffers. The introduction of communication links is a pre-requisite for successful decomposition of large models, which is in turn a major technique to tackle the complexity.
- Develop an appropriate strategy for specification and refinement of distributed databases. The specification model should be general enough to allow proper refinement that reflects the complex nature of the database interactions.

Aiming at providing a framework for developing Web application we defined three specification and refinement patterns in Chapter 4. These patterns are selected based on the experiences gained in the previous chapter and they should be general enough to be reused in other systems developments.

Specification partitioning and composition techniques are other main contributions of this research towards a more effective way to model Web application, that we presented in chapter 5. Some extensions and new aspects that we presented in this thesis are not supported by the current B tools. Therefore they open a new front for extending current tools to provide better support for Web applications by incorporating the new constructs.

6.3 Future Research

The future of software development is tied very closely to the Web. Web applications are becoming more and more common nowadays. Software engineering research must address the concern of developing dependable Web applications. Formal methods are our major tool to tackle this issue and tool support for formal methods is an essential factor. As a result of the nature of web development projects, current constructs in a formal method, like B, are not enough for Web applications. Our research has shown that modification and new extensions are needed to adapt the B method for development of Web applications. This research is the first step in an emerging research field that addresses the needs for formal Web application development. In the following, we point out some areas for future research that could extend the work presented in this thesis.

- Developing formal models for more case studies to identify further interesting and challenging aspects. These extra cases can enrich the field and provide a wider framework for real systems development.
- Extending the idea of specification and refinement patterns by incorporating more new patterns definitions.
- Refining the patterns including new patterns to reach a more detailed technical, and platform specific level, based on current underlaying technologies in Web based applications.
- Developing the idea of specification composition by investigating the new cases and identifying more composition mechanisms.
- Extending the B tools to incorporate the new extensions aiming at providing a more supportive and productive development environment for dependable systems.

Bibliography

[1] Martín Abadi and Leslie Lamport. Composing specifications. *ACM Trans. Program. Lang. Syst.*, 15(1):73–132, 1993.

[2] J. R. Abrial. *The B book - Assigning Programs to Meanings*. Cambridge University Press, 1996.

[3] J.-R. Abrial. Extending b without changing it (for developing distributed systems). In Henry Abrias, editor, *Proceedings of the 1st Conference on the B Method*, pages 169–191, November 1996.

[4] J.-R. Abrial and D. Cansell. Click'n'Prove-Interactive Proofs Within Set Theory, Version 23, May 2003. http://www.loria.fr/ cansell/cnp.html.

[5] J.-R. Abrial and L. Mussat. Introducing dynamic constraints in b. In *B'98 : The 2nd International B Conference, Recent Advances in the Development and Use of the B Method*, pages 83–128, April 1998.

[6] Jean-Raymond Abrial. Event driven system construction. `http://www.atelierb.societe.com/documents_en.htm`, 1999.

[7] Jean-Raymond Abrial. Guidelines to formal system studies. `http://www.atelierb.societe.com/documents_en.htm`, 2000.

[8] Jean-Raymond Abrial. Event driven distributed program construction. MATISSE project, Aug 2001.

[9] Jean-Raymond Abrial. Discrete system models. `http://i12www.ira.uka.de/~keller/Uni-Page/Lecture-Abrial.htm`, 2002.

[10] Jean-Raymond Abrial and Stefan Hallerstede. Refinement, decomposition, and instantiation of discrete models: Application to Event-B. *Fundamenta Informaticae*, XXI, 2006.

[11] Jonathan Adams, Srinivas Koushik, George Galambos, and Guru Vasudeva. *Patterns for e-business: A Strategy for Reuse.* IBM Press, 2001.

[12] Brian Henderson-Sellers Alice Gu and David Lowe. Web Modelling Languages: the gap between requirements and current exemplars, 2002.

[13] Robert Allen and David Garlan. A formal basis for architectural connection. *ACM Trans. Softw. Eng. Methodol.*, 6(3):213–249, 1997.

[14] S O Anderson, R E Bloomfield, and G L Cleland. Guidance on the use of formal methods in the development and assurance of high integrity industrial computer systems part iii a directory of formal methods. Technical report, EWICS(EUROPEAN WORKSHOP ON INDUSTRIAL COMPUTER SYSTEMS), June 1998. Available online at www.ewics.org.

[15] Atelier B Web Page. http://www.atelierb.societe.com/.

[16] B-Core(UK) Ltd. B-Toolkit. http://www.b-core.com/btoolkit.html.

[17] B4free Web Page. http://www.b4free.com/.

[18] R. Back and K. Sere. Superposition refinement of reactive systems. *Formal Aspects of Computing*, 8(3):324–346, 1996.

[19] Jos C. M. Baeten and Jan A. Bergstra. Real time process algebra. *Formal Asp. Comput.*, 3(2):142–188, 1991.

[20] M. R. Barbacci and C. B. Weinstock. Mapping metah into acme. Technical Report CMU/SEI-98-SR- 006, Carneggie Mellon University / Software Engineering Institute, Computer Science Department, Fanstord, California, July 1998.

[21] Luciano Baresi, Franca Garzotto, and Paolo Paolini. From web sites to web applications: New issues for conceptual modeling. In Stephen W. Liddle, Heinrich C. Mayr, and Bernhard Thalheim, editors, *ER (Workshops)*, volume 1921 of *Lecture Notes in Computer Science*, pages 89–100. Springer, 2000.

[22] Luciano Baresi, Franca Garzotto, Paolo Paolini, and Sara Valenti. Hdm2000: The hdm hypertext design model revisited. Technical report, Politecnico di Milano, 2000.

[23] Don Batory, Lou Coglianese, Mark Goodwin, and Steve Shafer. Creating reference architectures: an example from avionics. In *SSR '95: Proceedings of the 1995 Symposium on Software reusability*, pages 27–37. ACM Press, 1995.

[24] Hubert Baumeister, Nora Koch, and Luis Mandel. Towards a UML Extension for Hypermedia Design. In Robert B. France and Bernhard Rumpe, editors, *UML*, volume 1723 of *Lecture Notes in Computer Science*, pages 614–629. Springer, 1999.

[25] Zohra Bellahsene, Dilip Patel, and Colette Rolland, editors. *Object-Oriented. Information Systems, 8th International Conference, OOIS 2002, Montpellier, France, September 2-5, 2002, Proceedings*, volume 2425 of *Lecture Notes in Computer Science*. Springer, 2002.

[26] Pam Binns and Steve Vestal. Formal real-time architecture specification and analysis. In *RTOSS '93: Proceedings of the tenth IEEE workshop on Real-time operating systems and software*, pages 104–108. IEEE Computer Society, 1993.

[27] Mario A. Bochicchio and Antonella Longo. UWA+: bridging web systems design and business process modeling. In *Hypermedia Development & Web Engineering Principles and Techniques: Put them in use International Workshop on Web Engineering*, in conjunction with ACM Hypertext 2004, Santa Cruz, August 2004.

[28] Tommaso Bolognesi. Composing event constraints in state-based specification. In David de Frutos-Escrig and Manuel Núñez, editors, *FORTE*, volume 3235 of *Lecture Notes in Computer Science*, pages 13–32. Springer, 2004.

[29] Tommaso Bolognesi. A conceptual framework for state-based and event-based formal behavioural specification languages. In *ICECCS*, pages 107–116. IEEE Computer Society, 2004.

[30] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture - A System of Patterns*, page 35. John Wiley & Sons Ltd., Chichester, England, 1996.

[31] M. Butler and M. Waldén. Distributed system development in b. In *Proceedings of the 1st Conference on the B Method*, pages 155–168, Nantes, France, November 1996.

[32] Michael J. Butler. Stepwise refinement of communicating systems. *Sci. Comput. Program.*, 27(2):139–173, 1996.

[33] Michael J. Butler. An approach to the design of distributed systems with b amn. In Jonathan P. Bowen, Michael G. Hinchey, and David Till, editors, *ZUM*, volume 1212 of *Lecture Notes in Computer Science*, pages 223–241. Springer, 1997.

[34] Michael J. Butler. A system-based approach to the formal development of embedded controllers for a railway. *Design Automation for Embedded Systems*, 6(4), July 2002. ISSN 0929-5585.

[35] Cristina Cachero and Jaime Gómez. Advanced conceptual modeling of web applications: Embedding operation interfaces in navigation design. In Matilde Celma, Oscar Pastor, Natalia Juristo Juzgado, and Juan José Moreno-Navarro, editors, *JISBD*, pages 235–248, 2002.

[36] Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks*, 33(1-6):137–157, 2000.

[37] Peter P. Chen, David W. Embley, Jacques Kouloumdjian, Stephen W. Liddle, and John F. Roddick, editors. *Advances in Conceptual Modeling: ER '99 Workshops*

on Evolution and Change in Data Management, Reverse Engineering in Information Systems, and the World Wide Web and Conceptual Modeling, Paris, France, November 15-18, 1999, Proceedings, volume 1727 of *Lecture Notes in Computer Science*. Springer, 1999.

[38] ClearSy. Event B reference manual, June 2001.

[39] Jim Conallen. *Building Web applications with UML*. Addison-Wesley Longman Publishing Co., Inc., 2000.

[40] J. P. Courtiat, M. Diaz, R.C. De Oliveira, and P. Senac. Formal methods for the description of timed behaviors of multimedia and hypermedia distributed systems. *Computer Communications*, 19:1134–1150, 1996.

[41] Feras T. Dabous, Fethi A. Rabhi, and Hairong Yu. Using software architectures and design patterns for developing distributed applications. In *Australian Software Engineering Conference*, pages 290–299. IEEE Computer Society, 2004.

[42] Edsger W. Dijkstra and Carel S. Scholten. *Predicate calculus and program semantics*. Springer-Verlag New York, Inc., 1990.

[43] Alexander Egyed and Nenad Medvidovic. Consistent architectural refinement and evolution using the unified modeling language. In *Proc. of the 1st Workshop on Describing Software Architecture with UML, co-located with ICSE 2001*, pages 83–87, Toronto, Canada, 2001.

[44] P. Luigia et al. A methodology for integrating of formal methods in a healthcare case study. Technical Report 436, TUCS, December 2001.

[45] Andy Evans, Stuart Kent, and Bran Selic, editors. *UML 2000 - The Unified Modeling Language, Advancing the Standard, Third International Conference, York, UK, October 2-6, 2000, Proceedings*, volume 1939 of *Lecture Notes in Computer Science*. Springer, 2000.

[46] N. Evans and M. Butler. A proposal for records in event-B. *In Proceedings of Formal Methods 2006 (in press)*, Augsst 2006.

[47] C. Ferreira and M. Butler. Using b refinement to analyse compensating business processes. In *ZB 2003: Formal Specification and Development in Z and B: Third International Conference of B and Z Users*, LNCS 2651, Turku, Finland, 2003. Springer.

[48] Ludger Fiege, Gero Mühl, and Felix C. Gärtner. Modular event-based systems. *Knowl. Eng. Rev.*, 17(4):359–388, 2002.

[49] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[50] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[51] David Garlan, Robert Allen, and John Ockerbloom. Exploiting style in architectural design environments. In *SIGSOFT '94: Proceedings of the 2nd ACM SIGSOFT symposium on Foundations of software engineering*, pages 175–188. ACM Press, 1994.

[52] David Garlan and Andrew Kompanek. Reconciling the needs of architectural description with object-modeling notations. In Evans et al. [45], pages 498–512.

[53] David Garlan, Robert T. Monroe, and David Wile. Acme: Architectural description of component-based systems. In Gary T. Leavens and Murali Sitaraman, editors, *Foundations of Component-Based Systems*, pages 47–68. Cambridge University Press, 2000.

[54] Daniel M. Germán. *HadeZ, a Framework for the Specification and Verification of Hypermedia Applications.* PhD thesis, University of Waterloo, 2000.

[55] Daniel M. Germán and Donald D. Cowan. Formalizing the specification of web applications. In Chen et al. [37], pages 281–292.

[56] Athula Ginige and San Murugesan. Guest Editors' Introduction: Web Engineering - An Introduction. *IEEE MultiMedia*, 8(1):14–18, 2001.

[57] Jaime Gómez and Cristina Cachero. *OO-H Method: extending UML to model web interfaces*, pages 144–173. Idea Group Publishing, 2003.

[58] Jaime Gómez, Cristina Cachero, and Oscar Pastor. Conceptual modeling of device-independent web applications. *IEEE MultiMedia*, 8(2):26–39, 2001.

[59] M. Goulo and F. Abreu. Bridging the gap between Acme and UML 2.0 for CBD. In *Specification and Verification of Component-Based Systems (SAVCBS'2003)*, 2003.

[60] John V. Guttag and James J. Horning. *Larch: languages and tools for formal specification.* Springer-Verlag New York, Inc., 1993.

[61] Brendan Haire, David Lowe, and Brian Henderson-Sellers. Supporting web development in the open process: Additional roles and techniques. In Bellahsene et al. [25], pages 82–94.

[62] Ahmed E. Hassan. Architecture recovery of web applications. Master's thesis, University of Waterloo, 2002.

[63] Rolf Hennicker and Nora Koch. A uml-based methodology for hypermedia design. In Evans et al. [45], pages 410–424.

[64] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[65] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice/Hall International, 1985.

[66] Ian Sommerville. *Software Engineering*, chapter 3. Addison-Wesley, 7/E edition, 2004.

[67] ISO. Information systems processingopen systems interconnectionlotos. Technical Report DIS 8807, International Standards Organisation, 1987.

[68] Goguen J.A. and Tardo J. An introduction to obj: a language for writing and testing software specifications. In *Specification of Reliable Software*, pages 170–189. IEEE Press, 1979.

[69] Cliff B. Jones. *Systematic software development using VDM (2nd ed.).* Prentice-Hall, Inc., 1990.

[70] Cris Kobryn. Modeling enterprise software architectures using UML. In Cris Kobryn, editor, *Proceedings of The Second International Enterprise Distributed Object Computing Workshop.* IEEE, 1998.

[71] N. Koch and A. Kraus.

[72] Nora Koch, Andreas Kraus, Cristina Cachero, and Santiago Meli. Modeling web business processes with oo-h and uwe. In D. Schwabe, O. Pastor, G. Rossi, and L. Olsina, editors, *In Third International Workshop on Web-oriented Software Technology (IWWOST03)*, pages 27–50, July 2003.

[73] Xiaoying Kong and Li Liu. A web application architecture framework. In *AusWeb04, The Tenth Australian World Wide Web Conference*, Seaworld Nara Resort, Gold Coast, July 2004.

[74] Philippe Kruchten. The 4+1 View Model of Architecture. *IEEE Software*, 12(6):42–50, 1995.

[75] Wing Lam and Venky Shankararaman. An enterprise integration methodology. *IT Professional*, 6(2):40– 48, 2004.

[76] Leslie Lamport. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16(3):872–923, 1994.

[77] Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers.* Addison-Wesley, July 2002.

[78] Hung Ledang and Jeanine Souquières. Contributions for Modelling UML State-Charts in B. In Michael J. Butler, Luigia Petre, and Kaisa Sere, editors, *IFM*, volume 2335 of *Lecture Notes in Computer Science*, pages 109–127. Springer, 2002.

[79] Ying Liang. Generation of object models for information systems from business system models. In Bellahsene et al. [25], pages 255–266.

[80] Xiaodong Liu, Zhiqiang Chen, Hongji Yang, Hussein Zedan, and William C. Chu. A design framework for system re-engineering. In *APSEC*, pages 342–. IEEE Computer Society, 1997.

[81] Xiaodong Liu, Hongji Yang, and Hussein Zedan. Formal methods for the re-engineering of computing systems: A comparison. In *COMPSAC*, pages 409–. IEEE Computer Society, 1997.

[82] David Lowe. Web system requirements: an overview. *Requir. Eng.*, 8(2):102–113, 2003.

[83] David Lowe and John Eklund. Client needs and the design process in web projects. *J. Web Eng.*, 1(1):23–36, 2002.

[84] David Lowe and Brian Henderson-Sellers. Characteristics of web development processes. In *SSGRR-2001: International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, August 2001.

[85] David Lowe, Brian Henderson-Sellers, and Alice Gu. Web Extensions to UML: Using the MVC Triad. In Stefano Spaccapietra, Salvatore T. March, and Yahiko Kambayashi, editors, *ER*, volume 2503 of *Lecture Notes in Computer Science*, pages 105–119. Springer, 2002.

[86] David Lowe and Rachatrin Tongrungrojana. WebML+ for Communication of Information Flows: An Empirical Study. In Juan Manuel Cueva Lovelle, Bernardo Martín González Rodríguez, Luis Joyanes Aguilar, José Emilio Labra Gayo, and María del Puerto Paule Ruíz, editors, *ICWE*, volume 2722 of *Lecture Notes in Computer Science*, pages 218–221. Springer, 2003.

[87] David Lowe and Rachatrin Tongrungrojana. Web Information Exchange Diagrams for UML. In Xiaofang Zhou, Stanley Y. W. Su, Mike P. Papazoglou, Maria E. Orlowska, and Keith G. Jeffery, editors, *WISE*, volume 3306 of *Lecture Notes in Computer Science*, pages 29–40. Springer, 2004.

[88] David C. Luckham, John J. Kenney, Larry M. Augustin, James Vera, D. Bryan, and Walter Mann. Specification and analysis of system architecture using rapide. *IEEE Trans. Software Eng.*, 21(4):336–355, 1995.

[89] Jeff Magee, Naranker Dulay, Susan Eisenbach, and Jeff Kramer. Specifying distributed software architectures. In Wilhelm Schäfer and Pere Botella, editors, *ESEC*, volume 989 of *Lecture Notes in Computer Science*, pages 137–153. Springer, 1995.

[90] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-Verlag New York, Inc., 1992.

[91] Daniel Schwabe Mark D. Jacyntho and Gustavo Rossi. A software architecture for structuring complex web applications. *J. Web Eng.*, 1(1):37–60, 2002.

[92] Nenad Medvidovic, Peyman Oreizy, Jason E. Robbins, and Richard N. Taylor. Using object-oriented typing to support architectural design in the c2 style. In *SIGSOFT FSE*, pages 24–32, 1996.

[93] Nenad Medvidovic and David S. Rosenblum. Assessing the suitability of a standard design method for modeling software architectures. In *WICSA1: Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1)*, pages 161–182. Kluwer, B.V., 1999.

[94] Sun Microsystems. The Java EE 5 online tutorial. http://java.sun.com/javaee/5/docs/tutorial/doc/.

[95] Sun Microsystems. Java Platform, Enterprise Edition 5 Web Pages. http://java.sun.com/javaee/technologies/javaee5.jsp.

[96] Robin Milner. *Communication and concurrency.* Prentice Hall, Hemel Hempstead, United Kingdom, 1st edition, 1989.

[97] Susana Montero, Paloma Díaz, and Ignacio Aedo. A framework for the analysis and comparison of hypermedia design methods. In M. H. Hamza, editor, *Applied Informatics*, pages 1053–1058. IASTED/ACTA Press, 2003.

[98] Mark Moriconi, Xiaolei Qian, and Robert A. Riemenschneider. Correct architecture refinement. *IEEE Trans. Software Eng.*, 21(4):356–372, 1995.

[99] Matthew John Morley. *Safety Assurance in Interlocking Design.* PhD thesis, Department of Computer Science, the University of Edinburgh, 1996.

[100] Ben Moszkowski. *Executing temporal logic programs.* Cambridge University Press, 1986.

[101] San Murugesan and Yogesh Deshpande, editors. *Web Engineering, Software Engineering and Web Application Development*, Lecture Notes in Computer Science 2016. Springer, 2001.

[102] OMG. UML Resource Page. http://www.uml.org/.

[103] Object-Oriented Hypermedia. http://gplsi.dlsi.ua.es/iwad/ooh project/.

[104] Mehmet A. Orgun and Wanli Ma. An overview of temporal and modal logic programming. In *ICTL '94: Proceedings of the First International Conference on Temporal Logic*, pages 445–479. Springer-Verlag, 1994.

[105] Jonathan S. Ostroff. *Temporal Logic for Real-Time Systems.* Advanced Software Development. John Wiley & Sons, 1989.

[106] Scott P. Overmyer. What's different about requirements engineering for web sites? *Requir. Eng.*, 5(1):62–65, 2000.

[107] Patterns Home Page. http://www.hillside.net/patterns.

[108] Fabiano Borges Paulo, Marcelo Augusto Santos Turine, Maria Cristina Ferreira de Oliveira, and Paulo Cesar Masiero. Xhmbs: A formal model to support hypermedia specification. In *Hypertext*, pages 161–170. ACM, 1998.

[109] Dewayne E. Perry and Alexander L. Wolf. Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17(4):40–52, 1992.

[110] Charles J. Petrie and Akhil Sahai. Guest editors' introduction: Business processes on the web. *IEEE Internet Computing*, 8(1):28–29, 2004.

[111] Thomas A. Powell. *Web Site Engineering: Beyond Web page Design.* Prentice-Hall, 1998.

[112] Wolfgang Reisig. *Petri nets: an introduction.* Springer-Verlag New York, Inc., 1985.

[113] A. Rezazadeh and Michael Butler. Event-based modelling and refinement of distributed monitoring and control systems. In *Refinement of Critical Systems (RCS'03)*, Turku, June 2003.

[114] RODIN Web Page. http://rodin.cs.ncl.ac.uk/.

[115] Gustavo Rossi, Daniel Schwabe, and Fernando Lyardet. Web Application Models Are More Than Conceptual Models. In Chen et al. [37], pages 239–253.

[116] et al. S. Murugesan. Web engineering: A new discipline for development of web-based systems. In *Proceedings of the First ICSE Workshop on Web Engineering*, LNCS 1189, Los Angeles, June 1999.

[117] Hans Albrecht Schmid and Oliver Herfort. A behavioral semantics of oohdm core features and of its business process extension. In Nora Koch, Piero Fraternali, and Martin Wirsing, editors, *ICWE*, volume 3140 of *Lecture Notes in Computer Science*, pages 74–87. Springer, 2004.

[118] Hans Albrecht Schmid and Gustavo Rossi. Modeling and designing processes in e-commerce applications. *IEEE Internet Computing*, 8(1):19–27, 2004.

[119] D. J. Scholefield. *A Refinement Calculus for RealTime Systems.* PhD thesis, Department of Computer Science, University of York, 1992.

[120] David Scholefield and Hussein S. M. Zedan. Tam: A formal framework for the development of distributed real-time systems. In Jan Vytopil, editor, *FTRTFT*, volume 571 of *Lecture Notes in Computer Science*, pages 411–428. Springer, 1992.

[121] Daniel Schwabe and Gustavo Rossi. An object oriented approach to web-based applications design. *TAPOS*, 4(4):207–225, 1998.

[122] Emil Sekerinski and Kaisa Sere. *Program Development by Refinement: Case Studies Using the B Method*. Springer-Verlag, 1999.

[123] Bran Selic. Using uml for modeling complex real-time systems. In Frank Mueller and Azer Bestavros, editors, *LCTES*, volume 1474 of *Lecture Notes in Computer Science*, pages 250–260. Springer, 1998.

[124] Mary Shaw, Robert DeLine, Daniel V. Klein, Theodore L. Ross, David M. Young, and Gregory Zelesnik. Abstractions for software architecture and tools to support them. *IEEE Trans. Software Eng.*, 21(4):314–335, 1995.

[125] Mary Shaw and David Garlan. *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, Inc., 1996.

[126] Colin Snook and Michael Butler. UML-B formal modelling with UML.

[127] Colin Snook and Michael Butler. Verifying dynamic properties of uml models by translation to the b language and toolkit. In Proceedings of UML 2000 Workshop, Dynamic Behaviour in UML Models: Semantic Questions, 2000.

[128] J. M. Spivey. *Understanding Z: a specification language and its formal semantics*. Cambridge University Press, 1988.

[129] Unified Modelling Language Specification. http://www.omg.org/uml/.

[130] Rachatrin Tongrungrojana and David Lowe. WebML+: connecting business models to information designs. In *SEKE: Fifteenth International Conference on Software Engineering and Knowledge Engineering*, pages 17–24, Knowledge Systems Institute, Skokie, IL, San Francisco, USA, 2003.

[131] Rachatrin Tongrungrojana and David Lowe. Wied: A web modelling language for modelling architectural-level information flows. *J. Digit. Inf.*, 5(2), 2004.

[132] Helen Treharne. Supplementing a uml development process with b. In Lars-Henrik Eriksson and Peter A. Lindsay, editors, *FME*, volume 2391 of *Lecture Notes in Computer Science*, pages 568–586. Springer, 2002.

[133] Andrew C. Uselton and Scott A. Smolka. A compositional semantics for statecharts using labeled transition systems. In *CONCUR '94: Proceedings of the Concurrency Theory*, pages 2–17. Springer-Verlag, 1994.

[134] UWA Consortium. http://www.uwaproject.org.

[135] M. Waldén and K. Sere. Reasoning about action systems using the b-method. *Formal Methods in Systems Design*, 13:5–35, 1998.

[136] Jos Warmer and Anneke Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 1998.

[137] Jeannette M. Wing. A specifier's introduction to formal methods. *IEEE Computer*, 23(9):8–24, 1990.

[138] et al. Y. Deshpande. Web engineering: Beyond cs, is and se. In *Proceedings of the First ICSE Workshop on Web Engineering*, pages 171–176, Los Angeles, June 1999.

[139] Weiquan Zhao and David A. Kearney. Deriving architectures of web-based applications. In Xiaofang Zhou, Yanchun Zhang, and Maria E. Orlowska, editors, *APWeb*, volume 2642 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 2003.

Appendix A

The Travel Agency System

A.1 Specification of The Travel Agency System

```
MACHINE  TravelAgency

SETS
  SESSION; USER; PASS; CARD_DETAIL;
  FLIGHT_REQUEST; FLIGHT_DETAIL;
  ROOM_REQUEST; ROOM_DETAIL;
  CAR_REQUEST; CAR_DETAIL;
  STATE={fresh,booking,unbooking,service_selct,options_ret,choice_made,
          signed_in,certified,valid,invalid,booking_ret,unbooked_sel};

    REQUEST={bf, br, bc, uf, ur, uc, none}

DEFINITIONS
    freshSESSION == SESSION - session;
    freshUSER == USER - dom(current_user) ;
    freshPASS == PASS - ran(current_user)

CONSTANTS
    unnamed

PROPERTIES
    unnamed: USER

VARIABLES
    session, session_user, session_state,
    session_request, current_user,
    flight_booking, flight_options,selctflight_buf,
    booked_flight,ubselctflight_buf,
    room_booking, room_options,selctroom_buf,
    booked_room, ubselctroom_buf,
    car_booking, car_options, selctcar_buf,
    booked_car,ubselctcar_buf

INVARIANT

    session <: SESSION &
    session_user: session --> USER &
    session_state: session --> STATE &
    session_request: session --> REQUEST &
    current_user: USER +-> PASS &
    flight_booking: USER <-> FLIGHT_DETAIL &
    flight_options: SESSION +-> POW(FLIGHT_DETAIL) &
    selctflight_buf: SESSION +-> FLIGHT_DETAIL &
    booked_flight: SESSION +->POW(FLIGHT_DETAIL) &
    ubselctflight_buf: SESSION +-> FLIGHT_DETAIL &
    room_booking: USER <-> ROOM_DETAIL &
    room_options: SESSION +-> POW(ROOM_DETAIL) &
    selctroom_buf: SESSION +-> ROOM_DETAIL &
    booked_room: SESSION +-> POW(ROOM_DETAIL) &
    ubselctroom_buf: SESSION +-> ROOM_DETAIL &
    car_booking: USER <-> CAR_DETAIL &
    car_options: SESSION +-> POW(CAR_DETAIL) &
    selctcar_buf: SESSION +-> CAR_DETAIL &
    booked_car: SESSION +-> POW(CAR_DETAIL) &
    ubselctcar_buf: SESSION +-> CAR_DETAIL

INITIALISATION
    session := {} || session_user := {} ||  session_state := {} ||
    session_request := {} || current_user:={}||
    flight_booking:={} ||flight_options:= {} || selctflight_buf:={} ||
    booked_flight:= {} || ubselctflight_buf:= {} ||
    room_booking:={} || room_options:= {} || selctroom_buf:= {} ||
    booked_room:= {} || ubselctroom_buf := {} ||
    car_booking:={} ||car_options:= {} || selctcar_buf:= {} ||
    booked_car:= {} || ubselctcar_buf := {}

OPERATIONS
```

```
Client_ReqSession=        /* Client Operation */
  skip;

StartNewSession =
  ANY sid WHERE  sid: freshSESSION  THEN
   session := session \/ {sid} ||
    session_user(sid) := unnamed  ||
    session_request(sid) := none ||
   session_state(sid) := fresh
  END;

Get_SessionID=        /* Client Operation */
  skip;

PicService=         /* Client Operation */
    skip;

SelectService =
 ANY sid, req WHERE
     sid:session &
     req: REQUEST &
     req/=none &
     session_state(sid)=fresh
 THEN
    SELECT (req=bf or req=br or req= bc) THEN
       session_state(sid):= booking
    WHEN (req=uf or req=ur or req= uc) THEN
      SELECT session_user(sid) = unnamed THEN
       session_state(sid):= unbooking
      WHEN session_user(sid) /= unnamed THEN
      /* If User has signed-in before there is no need for re-login.*/
        session_state(sid):= signed_in
       END
    END ||
  session_request(sid):= req
 END;

Submit_FlightRequest=       /* Client Operation */
    skip;
```

```
Submit_RoomRequest =       /* Client Operation */
    skip;

Submit_CarRequest =       /* Client Operation */
    skip;

Get_FlightRequest =       /* Server Operation */
  ANY sid WHERE sid: session &
      session_state(sid)= booking &
      session_request(sid)= bf
  THEN
     session_state(sid):= service_selct
  END;

Get_RoomRequest =       /* Server Operation */
  ANY sid WHERE sid: session &
      session_state(sid)= booking &
      session_request(sid)= br
  THEN
     session_state(sid):= service_selct
  END;

Get_CarRequest =       /* Server Operation */
  ANY sid WHERE sid: session &
     session_state(sid)= booking &
     session_request(sid)= bc
  THEN
     session_state(sid):= service_selct
  END;

Request_Flight =       /* Server Operation */
 skip;

Request_Room =       /* Server Operation */
 skip;

Request_Car =       /* Server Operation */
 skip;

Resp_FlightReqs =       /* Flight Agency Server Operation */
  skip;
```

```
Resp_RoomReqs =       /* Hotel Server Operation */
  skip;

Resp_CarReqs =       /* Car Agency Server Operation */
  skip;

Retrieve_FlightOptions =       /* Server Operation */
 ANY sid WHERE sid: session &
     session_state(sid) = service_selct &
     session_request(sid)= bf
 THEN
   ANY xx WHERE  xx: POW(FLIGHT_DETAIL) THEN
      flight_options(sid):= xx
   END ||
    session_state(sid):= options_ret
 END;

Retrieve_RoomOptions=
 ANY sid WHERE sid: session &
     session_state(sid)= service_selct &
     session_request(sid)= br
 THEN
   ANY xx WHERE  xx: POW(ROOM_DETAIL) THEN
      room_options(sid):= xx
   END ||
   session_state(sid):= options_ret
 END;

Retrieve_CarOptions=
 ANY sid WHERE sid: session &
     session_state(sid)= service_selct &
     session_request(sid)= bc
 THEN
   ANY xx WHERE  xx: POW(CAR_DETAIL) THEN
      car_options(sid):= xx
   END ||
   session_state(sid):= options_ret
 END;

Select_Flight=       /* Client Operation */
```

```
    skip;

  Select_Room=       /* Client Operation */
    skip;

  Select_Car=       /* Client Operation */
    skip;

  GetSelected_Flight=       /* Server Operation */
   ANY sid,fd WHERE sid:session &
       session_state(sid)=options_ret &
       session_request(sid)=bf &
       sid:dom(flight_options) &
       fd:FLIGHT_DETAIL &
       fd: flight_options(sid)
   THEN
    selctflight_buf(sid):= fd ||
    flight_options:= {sid}<<| flight_options ||
     SELECT session_user(sid) = unnamed THEN
        session_state(sid):= choice_made
     WHEN session_user(sid) /= unnamed THEN
        session_state(sid):= signed_in
     END
   END;

  GetSelected_Room=       /* Server Operation */
   ANY sid,rd WHERE sid: session &
       session_state(sid)=options_ret &
       session_request(sid)=br &
       sid:dom(room_options) &
       rd: ROOM_DETAIL & rd: room_options(sid)
   THEN
    selctroom_buf(sid):= rd ||
    room_options:= {sid}<<| room_options ||
     SELECT session_user(sid) = unnamed THEN
        session_state(sid):= choice_made
     WHEN session_user(sid) /= unnamed THEN
        session_state(sid):= signed_in
     END
   END;
```

```
GetSelected_Car=       /* Server Operation */
 ANY sid,cd WHERE sid:session &
     session_state(sid)=options_ret &
     session_request(sid)=bc &
     sid:dom(car_options) &
     cd: CAR_DETAIL &
     cd: car_options(sid)
 THEN
   selctcar_buf(sid):= cd ||
   car_options:= {sid}<<|car_options ||
   SELECT session_user(sid) = unnamed THEN
      session_state(sid):= choice_made
   WHEN session_user(sid) /= unnamed THEN
      session_state(sid):= signed_in
   END
 END;

Request_UserInfo=         /* Server Operation */
  skip;

Client_Register =       /* Client Operation */
  skip;

Register =
 ANY sid,username,pass WHERE sid: session &
     session_state(sid)= choice_made &
     username: freshUSER &
     pass: freshPASS
 THEN
   current_user:= current_user \/ {username|->pass} ||
   session_user(sid) := username ||
   session_state(sid) := signed_in
 END;

Client_login =       /* Client Operation */
  skip;

Login =
 ANY sid, username, pass WHERE sid: session  &
     username: USER & pass: PASS &
     (session_state(sid)= choice_made or
     session_state(sid)= unbooking) &
     (username|-> pass): current_user
 THEN
    session_user(sid) := username ||
    CASE session_state(sid) OF
     EITHER  choice_made THEN
      session_state(sid) := signed_in
     OR unbooking THEN
      session_state(sid) := certified
     END
    END
 END;

EnterCard =       /* Client Operation */
   skip;

Card_Validate=
 ANY sid WHERE sid: session &
     session_state(sid)= signed_in
 THEN
     CHOICE
      session_state(sid) := valid
     OR
      session_state(sid) := invalid
     END
  END;

Restart_invalid=
 ANY sid WHERE sid: session &
     session_state(sid)= invalid
 THEN
      session_state(sid) := fresh ||
      session_request(sid):= none
 END;

Send_SelectedFlight=        /* Server Operation */
  skip;

Send_SelectedRoom=        /* Server Operation */
  skip;
```

```
Send_SelectedCar=        /* Server Operation */
  skip;

Agency_flight_booking=        /* Flight_agency Server Operation*/
  skip;

Hotel_room_booking=        /* Hotel Server Operation*/
  skip;

Agency_car_booking=        /* Car_agency Server Operation*/
  skip;

Flight_Booking =        /* Server Operation */
 ANY sid,fd WHERE sid:session &
     fd: FLIGHT_DETAIL &
     session_state(sid)=valid &
     sid: dom(selctflight_buf)  &
     selctflight_buf(sid)= fd
 THEN
  CHOICE
   flight_booking:= flight_booking \/ {session_user(sid)|->fd} ||
   selctflight_buf:= {sid}<<| selctflight_buf ||
   session_state(sid):= fresh ||
   session_request(sid):= none
  OR
   selctflight_buf:= {sid}<<| selctflight_buf ||
   session_state(sid) := fresh ||
   session_request(sid):= none
  END
 END;

 Room_Booking =        /* Server Operation */
 ANY sid,rd WHERE sid: session &
     rd: ROOM_DETAIL &
     session_state(sid)=valid &
     sid: dom(selctroom_buf) &
     selctroom_buf(sid)= rd
 THEN
   CHOICE
     room_booking:= room_booking \/{session_user(sid)|->rd} ||
      selctroom_buf:= {sid}<<| selctroom_buf ||
      session_state(sid):= fresh ||
     session_request(sid):= none
   OR
     selctroom_buf:= {sid}<<| selctroom_buf ||
      session_state(sid):= fresh ||
     session_request(sid):= none
   END
 END;

Car_Booking =         /* Server Operation */
 ANY sid,cd WHERE sid: session &
     cd:CAR_DETAIL &
     session_state(sid)=valid &
     sid:dom(selctcar_buf) &
     selctcar_buf(sid)= cd
 THEN
   CHOICE
     car_booking:= car_booking \/{session_user(sid)|->cd} ||
      selctcar_buf:= {sid}<<| selctcar_buf ||
      session_state(sid):= fresh ||
     session_request(sid):= none
   OR
     selctcar_buf:= {sid}<<| selctcar_buf ||
      session_state(sid):= fresh ||
     session_request(sid):= none
   END
 END;

Retrieve_BookedFlight=
 ANY sid WHERE sid: session &
     session_state(sid) = certified &
     session_request(sid)= uf
 THEN
   booked_flight(sid):= flight_booking[{session_user(sid)}]  ||
   session_state(sid):= booking_ret
 END;

Retrieve_BookedRoom=
 ANY sid WHERE sid: session &
     session_state(sid) = certified &
     session_request(sid)= ur
```

```
 THEN
   booked_room(sid):=  room_booking[{session_user(sid)}] ||
   session_state(sid):= booking_ret
 END;

Retrieve_BookedCar=
 ANY sid WHERE sid: session &
     session_state(sid) = certified &
     session_request(sid)= uc
 THEN
   booked_car(sid):= car_booking [{session_user(sid)}] ||
   session_state(sid):= booking_ret
 END;

Select_UnBbookedFlight=       /* Client Operation */
  skip;

Select_UnBbookedRoom=       /* Client Operation */
  skip;

Select_UnBbookedCar=       /* Client Operation */
  skip;

GetSelected_UBFlight=       /* Server Operation */
 ANY sid,fd WHERE sid:session &
     session_state(sid)=booking_ret &
     session_request(sid)=uf &
     sid:dom(booked_flight) &
     fd:FLIGHT_DETAIL &
     fd: booked_flight(sid)
 THEN
  ubselctflight_buf(sid):= fd ||
  booked_flight:= {sid}<<| booked_flight ||
  session_state(sid):= unbooked_sel
 END;

GetSelected_UBRoom=       /* Server Operation */
 ANY sid,rd WHERE sid: session &
     session_state(sid)=booking_ret &
     session_request(sid)=ur &
     sid:dom(booked_room) &
```

```
        rd: ROOM_DETAIL &
        rd: booked_room(sid)
    THEN
     ubselctroom_buf(sid):= rd ||
     booked_room:= {sid}<<|booked_room ||
     session_state(sid):= unbooked_sel
    END;

  GetSelected_UBCar=       /* Server Operation */
    ANY sid,cd WHERE sid:session &
        session_state(sid)=booking_ret &
        session_request(sid)=uc  &
        sid:dom(booked_car) &
        cd: CAR_DETAIL &
        cd: booked_car(sid)
    THEN
      ubselctcar_buf(sid):= cd ||
      booked_car:= {sid}<<|booked_car ||
      session_state(sid):= unbooked_sel
    END;

  Send_UBFlight=        /* Server Operation */
    skip;

  Send_UBRoom=        /* Server Operation */
    skip;

  Send_UBCar=        /* Server Operation */
    skip;

  Agency_Flight_Unbooking=        /* Flight_agency Server Operation*/
    skip;

  Hotel_Room_Unbooking=        /* Hotel Server Operation*/
    skip;

  Agency_Car_Unbooking=        /* Car_agency Server Operation*/
    skip;

  Unbook_Flight =
    ANY sid,fd WHERE sid: session &
```

```
        fd:FLIGHT_DETAIL &
        session_state(sid)=unbooked_sel &
        sid: dom(ubselctflight_buf) &
        ubselctflight_buf(sid)= fd
    THEN
     CHOICE
      flight_booking:= flight_booking - {session_user(sid)|->fd} ||
       ubselctflight_buf:= {sid}<<|ubselctflight_buf ||
        session_state(sid):= fresh ||
      session_request(sid):= none
     OR
        ubselctflight_buf:= {sid}<<| ubselctflight_buf ||
         session_state(sid):= fresh ||
        session_request(sid):= none
      END
    END;

  Unbook_Room =
    ANY sid,rd WHERE sid: session &
        rd: ROOM_DETAIL & session_state(sid)=unbooked_sel &
        sid: dom(ubselctroom_buf) &
        ubselctroom_buf(sid)= rd
    THEN
      CHOICE
        room_booking:= room_booking - {session_user(sid)|->rd} ||
         ubselctroom_buf:= {sid}<<| ubselctroom_buf ||
         session_state(sid):= fresh ||
        session_request(sid):= none
      OR
        ubselctroom_buf:= {sid}<<| ubselctroom_buf ||
         session_state(sid):= fresh ||
        session_request(sid):= none
      END
    END;

  Unbook_Car =
    ANY sid,cd WHERE sid: session &
        cd: CAR_DETAIL &
        session_state(sid)=unbooked_sel &
        sid: dom(ubselctcar_buf) &
        ubselctcar_buf(sid)= cd
```

```
   THEN
     CHOICE
       car_booking:= car_booking - {session_user(sid)|-> cd} ||
        ubselctcar_buf:= {sid}<<| ubselctcar_buf ||
        session_state(sid):= fresh ||
       session_request(sid):= none
     OR
       ubselctcar_buf:= {sid}<<| ubselctcar_buf ||
        session_state(sid):= fresh ||
       session_request(sid):= none
     END
   END;

  Client_Recv_Reply=       /* Client Operation */
    skip;

  Client_logout=       /* Client Operation */
    skip;

  Logout=
    ANY sid WHERE sid:session  THEN
      session:= session - {sid} ||
       session_user:= {sid}<<| session_user ||
        session_state:= {sid}<<| session_state ||
        session_request:= {sid} <<| session_request ||
        flight_options:= {sid}<<|flight_options ||
        selctflight_buf:= {sid}<<| selctflight_buf ||
        booked_flight:= {sid}<<| booked_flight ||
        ubselctflight_buf:= {sid}<<|ubselctflight_buf ||
        room_options:= {sid}<<|room_options ||
        selctroom_buf:= {sid}<<| selctroom_buf ||
        booked_room:= {sid}<<|booked_room ||
        ubselctroom_buf:= {sid}<<| ubselctroom_buf ||
        car_options:= {sid}<<|car_options ||
        selctcar_buf:= {sid}<<| selctcar_buf ||
       booked_car:= {sid}<<|booked_car ||
      ubselctcar_buf:= {sid}<<| ubselctcar_buf
    END

END
```

A.2 First Refinement of The Travel Agency System—Separating Clients Operations from the Travel Agency Server

```
REFINEMENT  TravelAgency1
REFINES     TravelAgency

SETS
    TAG; HANDLE;RESP={failed, succeed}

 DEFINITIONS
    freshSESSION == SESSION - session ;
    freshUSER == USER - dom(current_user) ;
    freshPASS == PASS - ran(current_user);
    freshHANDLE == HANDLE - dom(new_client)

CONSTANTS
    Reg, Log

PROPERTIES
    Reg: TAG & Log: TAG

VARIABLES

 /*  Server Variables  */
    session, session_user, session_state,
    session_request, current_user,
    flight_booking, flight_options,selctflight_buf,
    booked_flight,ubselctflight_buf,
    room_booking, room_options,selctroom_buf,
    booked_room, ubselctroom_buf,
    car_booking, car_options, selctcar_buf,
    booked_car,ubselctcar_buf,

 /*  Server New Variables  */
    availflight_buf, availroom_buf, availcar_buf,resp_buf,
    bookedflight_buf, bookedroom_buf, bookedcar_buf,

 /*  Client Variables  */
    new_handle, new_client, token,
    fresh_session, reqsevice_buf,
    reqFD_buf, reqRD_buf, reqCD_buf,
    flightReq_buf, roomReq_buf, carReq_buf,
    pikedflight_buf, pikedroom_buf, pikedcar_buf,
    unnamed_buf,  unsigned_buf,userInfo_buf, reqCard_buf,
    card_buf, ubflight_buf, ubroom_buf, ubcar_buf, logout_buf

INVARIANT
 /*  Server New Variables  */
    availflight_buf: SESSION +-> POW(FLIGHT_DETAIL) &
    availroom_buf: SESSION +-> POW(ROOM_DETAIL) &
    availcar_buf: SESSION +-> POW(CAR_DETAIL) &
    resp_buf: SESSION +-> RESP &
    bookedflight_buf: SESSION +-> POW(FLIGHT_DETAIL) &
    bookedroom_buf: SESSION +-> POW(ROOM_DETAIL) &
    bookedcar_buf: SESSION +-> POW(CAR_DETAIL) &

 /*  Client Variables  */
    new_handle<: HANDLE &
    new_client: HANDLE+->SESSION &
    token <: SESSION &
    fresh_session<: SESSION &
    reqsevice_buf: SESSION +-> REQUEST &
    reqFD_buf<: SESSION &
    reqRD_buf<: SESSION &
    reqCD_buf<: SESSION &
    flightReq_buf: SESSION +->FLIGHT_REQUEST &
    roomReq_buf: SESSION +->ROOM_REQUEST &
    carReq_buf: SESSION +->CAR_REQUEST &
    pikedflight_buf: SESSION +->FLIGHT_DETAIL &
    pikedroom_buf: SESSION +->ROOM_DETAIL &
    pikedcar_buf: SESSION +->CAR_DETAIL &
    unnamed_buf<: SESSION &
    unsigned_buf<: SESSION &
    userInfo_buf: SESSION +->(USER*PASS*TAG) &
    reqCard_buf<: SESSION &
    card_buf: SESSION+->(CARD_DETAIL)&
    ubflight_buf: SESSION+->FLIGHT_DETAIL &
    ubroom_buf: SESSION+->ROOM_DETAIL &
    ubcar_buf: SESSION +->CAR_DETAIL &
    logout_buf<: SESSION

INITIALISATION
```

```
/*  Server Variables  */
   session := {} ||
   session_user := {} ||
   session_state := {} ||
   session_request := {} ||
   current_user:={}||
   flight_booking:={} ||
   flight_options:= {} ||
   selctflight_buf:= {} ||
   booked_flight:= {} ||
   ubselctflight_buf:= {} ||
   room_booking:={} ||
   room_options:= {} ||
   selctroom_buf:= {} ||
   booked_room:= {} ||
   ubselctroom_buf := {} ||
   car_booking:={} ||
   car_options:= {} ||
   selctcar_buf:= {} ||
   booked_car:= {} ||
   ubselctcar_buf := {} ||

/*  Server New Variables  */
   availflight_buf:= {} ||
   availroom_buf:= {} ||
   availcar_buf:= {} ||
   resp_buf:= {} ||
   bookedflight_buf:= {} ||
   bookedroom_buf:= {} ||
   bookedcar_buf:= {} ||

/*  Client Variables  */
   new_handle:={} ||
   new_client:={} ||
   token:={} ||
   fresh_session:={} ||
   reqsevice_buf:={} ||
   reqFD_buf:={} ||
   reqRD_buf:={} ||
   reqCD_buf:={} ||
   flightReq_buf:={} ||
    roomReq_buf:={} ||
    carReq_buf:={} ||
    pikedflight_buf:= {} ||
    pikedroom_buf:= {} ||
    pikedcar_buf:= {} ||
    unnamed_buf:={}||
    unsigned_buf:={}||
    userInfo_buf:={}||
    reqCard_buf:={}||
    card_buf:= {} ||
    ubflight_buf:= {} ||
    ubroom_buf:= {} ||
    ubcar_buf:= {} ||
    logout_buf:= {}


OPERATIONS

  Client_ReqSession=        /* Client Operation */
    ANY handle WHERE handle: freshHANDLE THEN
      new_handle:= new_handle \/{handle}
    END;

  StartNewSession =       /* Server Operation */
    ANY sid, handle WHERE  sid: freshSESSION &
        handle: new_handle
    THEN
     session:= session \/{sid}||
      session_user(sid) := unnamed  ||
      session_request(sid) := none ||
      session_state(sid) := fresh ||
      new_client(handle):= sid ||
     new_handle:= new_handle - {handle}
    END;

  Get_SessionID=        /* Client Operation */
    ANY sid WHERE sid: SESSION &
        sid: ran(new_client)
    THEN
      token:= token \/ {sid} ||
      fresh_session:= fresh_session \/ {sid} ||
      new_client:= new_client |>>{sid}
    END;

  PicService=         /* Client Operation */
   ANY sid, req WHERE sid: fresh_session &
       req: REQUEST & req/= none
   THEN
        reqsevice_buf(sid):= req ||
        fresh_session:= fresh_session - {sid}
   END;

  SelectService =        /* Server Operation */
   ANY sid, req WHERE sid: session &
       session_state(sid)=fresh &
       sid:dom(reqsevice_buf) &
       req: REQUEST &
       req/=none &
       reqsevice_buf(sid)=req
   THEN
    session_request(sid):= req ||
    reqsevice_buf:= {sid}<<| reqsevice_buf ||
    CASE req OF
     EITHER  bf THEN
      reqFD_buf:= reqFD_buf \/ {sid}||
      session_state(sid):= booking
     OR  br THEN
      reqRD_buf:= reqRD_buf \/ {sid} ||
      session_state(sid):= booking
     OR  bc THEN
      reqCD_buf:= reqCD_buf \/ {sid} ||
      session_state(sid):= booking
     OR uf,ur,uc THEN
      SELECT session_user(sid) = unnamed THEN
       session_state(sid):= unbooking
      WHEN session_user(sid) /= unnamed THEN
      /* If User has signed-in before there is no need for relogin.*/
       session_state(sid):= signed_in
      END
     END
    END
  END;
```

```
Submit_FlightRequest =        /* Client Operation */
 ANY sid, fr WHERE  sid: token &
     sid: reqFD_buf &
     fr: FLIGHT_REQUEST
 THEN
   flightReq_buf(sid):= fr ||
   reqFD_buf:= reqFD_buf - {sid}
 END;

Submit_RoomRequest =       /* Client Operation */
 ANY sid, rr WHERE sid: token &
     sid: reqRD_buf &
     rr: ROOM_REQUEST
 THEN
   roomReq_buf(sid):=rr ||
   reqRD_buf:= reqRD_buf -{sid}
 END;

Submit_CarRequest =       /* Client Operation */
 ANY sid, cr WHERE sid: token &
     sid: reqCD_buf &
     cr: CAR_REQUEST
 THEN
   carReq_buf(sid):=cr ||
   reqCD_buf:= reqCD_buf - {sid}
 END;


Get_FlightRequest =       /* Server Operation */
  ANY sid WHERE sid: session &
      session_state(sid)= booking &
      session_request(sid)= bf &
      sid: dom(flightReq_buf)
  THEN
   flightReq_buf:= {sid}<<| flightReq_buf ||
   session_state(sid):= service_selct
  END;

Get_RoomRequest =       /* Server Operation */
  ANY sid WHERE sid: session &
```

```
        session_state(sid)= booking &
        session_request(sid)= br &
        sid: dom(roomReq_buf)
    THEN
      roomReq_buf:={sid}<<|roomReq_buf ||
      session_state(sid):= service_selct
    END;

  Get_CarRequest =        /* Server Operation */
    ANY sid WHERE sid: session &
        session_state(sid)= booking &
        session_request(sid)= bc &
        sid: dom(carReq_buf)
    THEN
      carReq_buf:={sid}<<| carReq_buf ||
      session_state(sid):= service_selct
    END;

  Request_Flight =        /* Server Operation */
    skip;

  Request_Room =        /* Server Operation */
    skip;

  Request_Car =        /* Server Operation */
    skip;

  Resp_FlightReqs =   /* Flight Agency Server Operation */
    skip;

  Resp_RoomReqs =     /* Hotel Server Operation */
    skip;

  Resp_CarReqs =      /* Car Agency Server Operation */
    skip;

  Retrieve_FlightOptions =     /* Server Operation */
   ANY sid WHERE sid: session &
       session_state(sid) = service_selct &
       session_request(sid)= bf
   THEN
```

```
    ANY xx WHERE  xx:POW(FLIGHT_DETAIL) THEN
       flight_options(sid):= xx
    END ||
    availflight_buf(sid):= flight_options(sid) ||
    session_state(sid):= options_ret
  END;

Retrieve_RoomOptions=   /* Server Operation */
  ANY sid WHERE sid: session &
      session_state(sid)= service_selct &
      session_request(sid)= br
  THEN
    ANY xx WHERE  xx:POW(ROOM_DETAIL) THEN
       room_options(sid):= xx
    END ||
    availroom_buf(sid):= room_options(sid) ||
    session_state(sid):= options_ret
  END;

Retrieve_CarOptions=     /* Server Operation */
  ANY sid WHERE sid: session &
      session_state(sid)= service_selct &
      session_request(sid)= bc
  THEN
    ANY xx WHERE  xx:POW(CAR_DETAIL) THEN
       car_options(sid):= xx
    END ||
    availcar_buf(sid):= car_options(sid) ||
    session_state(sid):= options_ret
  END;

Select_Flight=       /* Client Operation */
   ANY sid, fd WHERE sid: token &
       sid: dom(availflight_buf) &
       fd: FLIGHT_DETAIL &
       fd: availflight_buf(sid)
   THEN
     pikedflight_buf(sid):= fd ||
     availflight_buf:= {sid}<<| availflight_buf
   END;
```

```
Select_Room=       /* Client Operation */
  ANY sid, rd WHERE sid: token &
      sid: dom(availroom_buf) &
      rd: ROOM_DETAIL &
      rd: availroom_buf(sid)
  THEN
    pikedroom_buf(sid):= rd ||
    availroom_buf:= {sid}<<| availroom_buf
  END;

Select_Car=       /* Client Operation */
  ANY sid, cd WHERE sid: token &
      sid: dom(availcar_buf) &
      cd: CAR_DETAIL &
      cd: availcar_buf(sid)
  THEN
    pikedcar_buf(sid):= cd ||
    availcar_buf:= {sid}<<| availcar_buf
  END;

GetSelected_Flight=       /* Server Operation */
 ANY sid,fd WHERE sid:session &
     fd:FLIGHT_DETAIL &
     session_state(sid)=options_ret &
     session_request(sid)=bf &
     sid:dom(flight_options) &
     fd: flight_options(sid) &
     sid:dom(pikedflight_buf) &
     pikedflight_buf(sid)= fd
 THEN
  selctflight_buf(sid):= fd ||
  flight_options:= {sid}<<|flight_options ||
  pikedflight_buf:={sid}<<| pikedflight_buf ||
   SELECT session_user(sid) = unnamed THEN
      session_state(sid):= choice_made
   WHEN session_user(sid) /= unnamed THEN
   /* If User has signed-in before there is no need for re-login.*/
      session_state(sid):= signed_in
   END
 END;
```

```
GetSelected_Room=       /* Server Operation */
 ANY sid,rd WHERE sid: session &
     rd: ROOM_DETAIL &
     session_state(sid)=options_ret &
     session_request(sid)=br &
     sid:dom(room_options) &
     rd: room_options(sid) &
     sid:dom(pikedroom_buf) &
     pikedroom_buf(sid)= rd
 THEN
  selctroom_buf(sid):= rd ||
  room_options:= {sid}<<|room_options ||
  pikedroom_buf:= {sid}<<| pikedroom_buf ||
   SELECT session_user(sid) = unnamed THEN
      session_state(sid):= choice_made
   WHEN session_user(sid) /= unnamed THEN
   /* If User has signed-in before there is no need for relogin. */
      session_state(sid):= signed_in
   END
 END;

GetSelected_Car=       /* Server Operation */
 ANY sid,cd WHERE sid:session &
     cd: CAR_DETAIL &
     session_state(sid)=options_ret &
     session_request(sid)=bc &
     sid:dom(car_options) &
     cd: car_options(sid)  &
     sid:dom(pikedcar_buf) &
     pikedcar_buf(sid)= cd
 THEN
   selctcar_buf(sid):= cd ||
   car_options:= {sid}<<|car_options ||
   pikedcar_buf:= {sid}<<| pikedcar_buf ||
   SELECT session_user(sid) = unnamed THEN
      session_state(sid):= choice_made
   WHEN session_user(sid) /= unnamed THEN
   /* If User has signed-in before there is no need for relogin. */
      session_state(sid):= signed_in
   END
 END;
```

```
Request_UserInfo=         /* Server Operation */
 ANY sid WHERE sid: session &
     (session_state(sid)= choice_made or
      session_state(sid)= unbooking )
 THEN
   SELECT session_state(sid)= choice_made THEN
      unnamed_buf:= unnamed_buf \/ {sid}
   WHEN session_state(sid)= unbooking THEN
  /* There is a distinction between a user that requests
     an unbooking and booking session ,because the
     former has registered before so it can just can
     login, but for booking session it can either
     login or register if it has not registered */
      unsigned_buf:= unsigned_buf \/ {sid}
   END
 END;

Client_Register =       /* Client Operation */
  ANY sid, username,pass,confpass WHERE sid: token &
      sid: unnamed_buf &
      username:freshUSER &
      pass:freshPASS &
      confpass:freshPASS &
      pass=confpass
  THEN
   userInfo_buf(sid):= (username|-> pass|-> Reg) ||
   unnamed_buf:= unnamed_buf - {sid}
 END;

Register =        /* Server Operation */
 ANY sid,username,pass WHERE sid: session &
     session_state(sid)= choice_made &
     username:freshUSER &
     pass:freshPASS &
     sid:dom(userInfo_buf) &
     userInfo_buf(sid)= (username|->pass|->Reg)
 THEN
   current_user:= current_user \/ {username|->pass} ||
   session_user(sid) := username ||
   userInfo_buf := {sid}<<| userInfo_buf ||
   session_state(sid) := signed_in ||
   reqCard_buf:= reqCard_buf \/ {sid}
 END;

Client_login =       /* Client Operation */
 ANY sid, username,pass WHERE sid: token &
     username: USER & pass: PASS &
     (username|->pass): current_user &
     (sid: unnamed_buf or sid: unsigned_buf)
 THEN
    userInfo_buf(sid):= (username|->pass|->Log) ||
    SELECT sid: unnamed_buf THEN
      unnamed_buf:= unnamed_buf - {sid}
    WHEN sid: unsigned_buf THEN
      unsigned_buf:= unsigned_buf - {sid}
    END
 END;

Login  =        /* Server Operation */
 ANY sid,username, pass WHERE sid: session &
    (session_state(sid)= choice_made or
     session_state(sid)= unbooking) &
     username: USER & pass: PASS &
    (username|-> pass): current_user &
     sid: dom(userInfo_buf) &
     userInfo_buf(sid)= (username|->pass|->Log)
 THEN
   session_user(sid) := username ||
   userInfo_buf:= {sid}<<| userInfo_buf ||
    CASE session_state(sid) OF
     EITHER  choice_made THEN
      session_state(sid) := signed_in ||
      reqCard_buf:= reqCard_buf \/ {sid}
     OR unbooking THEN
      session_state(sid) := certified
     END
    END
 END;

EnterCard =       /* Client Operation */
 ANY sid,xcard WHERE sid:token &
```

```
     sid: reqCard_buf &
     xcard: CARD_DETAIL
 THEN
      card_buf(sid):= xcard ||
      reqCard_buf:= reqCard_buf - {sid}
 END;

Card_Validate=        /* Server Operation */
  ANY sid WHERE sid:session &
      session_state(sid)=signed_in &
      sid:dom(card_buf)
  THEN
     CHOICE
      session_state(sid) := valid
     OR
      session_state(sid) := invalid
     END ||
      card_buf:= {sid}<<| card_buf
  END;

Restart_invalid=
 ANY sid WHERE sid: session & session_state(sid)= invalid THEN
      session_state(sid) := fresh ||
      session_request(sid):= none
 END;

Send_SelectedFlight=        /* Server Operation */
  skip;

Send_SelectedRoom=        /* Server Operation */
  skip;

Send_SelectedCar=        /* Server Operation */
  skip;

Agency_flight_booking=        /* Flight_agency Server Operation*/
  skip;

Hotel_room_booking=        /* Hotel Server Operation*/
  skip;
```

```
Agency_car_booking=       /* Car_agency Server Operation*/
  skip;

Flight_Booking =       /* Server Operation */
 ANY sid,fd WHERE sid:session &
     fd: FLIGHT_DETAIL &
     session_state(sid)=valid &
     sid: dom(selctflight_buf)  &
     selctflight_buf(sid)= fd
 THEN
  CHOICE
   flight_booking:= flight_booking \/ {session_user(sid)|->fd} ||
    selctflight_buf:= {sid}<<| selctflight_buf ||
    resp_buf(sid):= succeed ||
    session_state(sid):= fresh ||
   session_request(sid):= none
  OR
     selctflight_buf:= {sid}<<| selctflight_buf ||
      resp_buf(sid):= failed ||
     session_state(sid) := fresh ||
     session_request(sid):= none
  END
 END;

 Room_Booking =       /* Server Operation */
 ANY sid,rd WHERE sid: session &
     rd: ROOM_DETAIL &
     session_state(sid)=valid &
     sid: dom(selctroom_buf) &
     selctroom_buf(sid)= rd
 THEN
   CHOICE
     room_booking:= room_booking \/
     {session_user(sid)|->selctroom_buf(sid)} ||
      selctroom_buf:= {sid}<<| selctroom_buf ||
      resp_buf(sid):= succeed ||
      session_state(sid):= fresh ||
     session_request(sid):= none
   OR
     selctroom_buf:= {sid}<<| selctroom_buf ||
      resp_buf(sid):= failed ||
      session_state(sid):= fresh ||
     session_request(sid):= none
   END
 END;

Car_Booking =        /* Server Operation */
 ANY sid,cd WHERE sid: session &
     cd:CAR_DETAIL &
     session_state(sid)=valid &
     sid:dom(selctcar_buf) &
     selctcar_buf(sid)= cd
 THEN
  CHOICE
   car_booking:= car_booking \/{session_user(sid)|->cd} ||
    selctcar_buf:= {sid}<<| selctcar_buf ||
    resp_buf(sid):= succeed ||
    session_state(sid):= fresh ||
   session_request(sid):= none
  OR
   selctcar_buf:= {sid}<<| selctcar_buf ||
    resp_buf(sid):= failed ||
    session_state(sid):= fresh ||
   session_request(sid):= none
  END
 END;

Retrieve_BookedFlight=        /* Server Operation */
 ANY sid WHERE sid: session &
     session_state(sid) = certified &
     session_request(sid)=uf
 THEN
  booked_flight(sid):= flight_booking[{session_user(sid)}] ||
  bookedflight_buf(sid):= booked_flight(sid) ||
  session_state(sid):= booking_ret
 END;

Retrieve_BookedRoom=        /* Server Operation */
 ANY sid WHERE sid: session &
     session_state(sid) = certified &
     session_request(sid)=ur
 THEN
```

```
    booked_room(sid):=  room_booking[{session_user(sid)}] ||
    bookedroom_buf(sid):= booked_room(sid) ||
    session_state(sid):= booking_ret
  END;

Retrieve_BookedCar=          /* Server Operation */
  ANY sid WHERE sid: session &
      session_state(sid) = certified &
      session_request(sid)= uc
  THEN
    booked_car(sid):= car_booking [{session_user(sid)}] ||
    bookedcar_buf(sid):= booked_car(sid) ||
    session_state(sid):= booking_ret
  END;

Select_UnBbookedFlight=        /* Client Operation */
   ANY sid, fd WHERE sid: token &
       sid: dom(bookedflight_buf) &
       fd: FLIGHT_DETAIL &
       fd: bookedflight_buf(sid)
   THEN
     ubflight_buf(sid):= fd ||
     bookedflight_buf:= {sid}<<| bookedflight_buf
   END;

Select_UnBbookedRoom=        /* Client Operation */
   ANY sid, rd WHERE sid: token &
       sid: dom(bookedroom_buf) &
       rd: ROOM_DETAIL &
       rd: bookedroom_buf(sid)
   THEN
     ubroom_buf(sid):= rd ||
     bookedroom_buf:= {sid}<<| bookedroom_buf
   END;

Select_UnBbookedCar=        /* Client Operation */
   ANY sid, cd WHERE sid: token &
       sid: dom(bookedcar_buf) &
       cd: CAR_DETAIL &
       cd: bookedcar_buf(sid)
   THEN
```

```
     ubcar_buf(sid):= cd ||
     bookedcar_buf:= {sid}<<| bookedcar_buf
   END;

 GetSelected_UBFlight=       /* Server Operation */
  ANY sid,fd WHERE sid:session &
      fd:FLIGHT_DETAIL &
      session_state(sid)=booking_ret &
      session_request(sid)=uf &
      sid:dom(booked_flight) &
      fd: booked_flight(sid) &
      sid: dom(ubflight_buf) &
      ubflight_buf(sid)= fd
  THEN
   ubselctflight_buf(sid):= fd ||
    booked_flight:= {sid}<<| booked_flight ||
    ubflight_buf:= {sid}<<| ubflight_buf ||
   session_state(sid):= unbooked_sel
  END;

 GetSelected_UBRoom=       /* Server Operation */
  ANY sid,rd WHERE sid: session &
      rd: ROOM_DETAIL &
      session_state(sid)=booking_ret &
      session_request(sid)=ur &
      sid:dom(booked_room) &
      rd: booked_room(sid) &
      sid: dom(ubroom_buf) &
      ubroom_buf(sid)= rd
  THEN
   ubselctroom_buf(sid):= rd ||
    booked_room:= {sid}<<|booked_room ||
    ubroom_buf:= {sid}<<| ubroom_buf ||
   session_state(sid):= unbooked_sel
  END;

 GetSelected_UBCar=       /* Server Operation */
  ANY sid,cd WHERE sid:session &
      cd: CAR_DETAIL &
      session_state(sid)=booking_ret &
      session_request(sid)=uc  &
      sid:dom(booked_car) &
      cd: booked_car(sid) &
      sid: dom(ubcar_buf) &
      ubcar_buf(sid)= cd
  THEN
    ubselctcar_buf(sid):= cd ||
     booked_car:= {sid}<<|booked_car ||
     ubcar_buf:= {sid}<<| ubcar_buf ||
    session_state(sid):= unbooked_sel
  END;

 Send_UBFlight=        /* Server Operation */
   skip;

 Send_UBRoom=        /* Server Operation */
   skip;

 Send_UBCar=        /* Server Operation */
   skip;

 Agency_Flight_Unbooking=   /* Flight_agency Server Operation*/
   skip;

 Hotel_Room_Unbooking=        /* Hotel Server Operation*/
   skip;

 Agency_Car_Unbooking=        /* Car_agency Server Operation*/
   skip;

 Unbook_Flight =         /* Server Operation */
  ANY sid,fd WHERE sid: session &
      fd:FLIGHT_DETAIL &
      session_state(sid)=unbooked_sel &
      sid: dom(ubselctflight_buf) &
      ubselctflight_buf(sid)= fd
  THEN
    CHOICE
      flight_booking:= flight_booking - {session_user(sid)|->fd} ||
       ubselctflight_buf:= {sid}<<|ubselctflight_buf ||
       resp_buf(sid):= succeed ||
       session_state(sid):= fresh ||
      session_request(sid):= none
    OR
      ubselctflight_buf:= {sid}<<| ubselctflight_buf ||
       resp_buf(sid):= failed ||
       session_state(sid):= fresh ||
      session_request(sid):= none
    END
  END;

Unbook_Room =          /* Server Operation */
  ANY sid,rd WHERE sid: session &
      rd: ROOM_DETAIL &
      session_state(sid)=unbooked_sel  &
      sid: dom(ubselctroom_buf) & ubselctroom_buf(sid)= rd
  THEN
    CHOICE
      room_booking:= room_booking - {session_user(sid)|->rd} ||
       ubselctroom_buf:= {sid}<<| ubselctroom_buf ||
       resp_buf(sid):= succeed ||
       session_state(sid):= fresh ||
      session_request(sid):= none
    OR
      ubselctroom_buf:= {sid}<<| ubselctroom_buf ||
       resp_buf(sid):= failed ||
       session_state(sid):= fresh ||
      session_request(sid):= none
    END
  END;

Unbook_Car =
  ANY sid,cd WHERE sid: session &
      cd: CAR_DETAIL &
      session_state(sid)=unbooked_sel
      & sid: dom(ubselctcar_buf) &
      ubselctcar_buf(sid)= cd
  THEN
   CHOICE
    car_booking:= car_booking - {session_user(sid)|-> cd} ||
     ubselctcar_buf:= {sid}<<| ubselctcar_buf ||
      resp_buf(sid):= succeed ||
     session_state(sid):= fresh ||
    session_request(sid):= none
   OR
    ubselctcar_buf:= {sid}<<| ubselctcar_buf ||
     resp_buf(sid):= failed ||
     session_state(sid):= fresh ||
    session_request(sid):= none
   END
  END;

 Client_Recv_Reply=       /* Client Operation */
   ANY sid WHERE sid: token & sid: dom(resp_buf) THEN
    resp_buf:= {sid}<<|resp_buf
   END;

 Client_logout=       /* Client Operation */
   ANY sid WHERE sid: token THEN
     reqsevice_buf:= {sid}<<| reqsevice_buf ||
     new_client:= new_client |>>{sid} ||
      fresh_session:= fresh_session - {sid} ||
       reqFD_buf:= reqFD_buf - {sid} ||
       reqRD_buf:= reqRD_buf - {sid} ||
       reqCD_buf:= reqCD_buf- {sid} ||
       flightReq_buf:= {sid}<<| flightReq_buf ||
       roomReq_buf:= {sid}<<| roomReq_buf ||
       carReq_buf:= {sid}<<| carReq_buf ||
       pikedflight_buf:= {sid}<<|pikedflight_buf ||
       pikedroom_buf:= {sid}<<|pikedroom_buf ||
       pikedcar_buf:= {sid}<<|pikedcar_buf ||
       unnamed_buf:= unnamed_buf - {sid} ||
       unsigned_buf:= unsigned_buf - {sid} ||
       userInfo_buf := {sid}<<| userInfo_buf ||
       reqCard_buf:= reqCard_buf - {sid} ||
       card_buf:= {sid}<<| card_buf ||
       ubflight_buf:= {sid}<<| ubflight_buf ||
      ubroom_buf:= {sid}<<| ubroom_buf||
     ubcar_buf:= {sid}<<|ubcar_buf ||
    logout_buf:= logout_buf \/ {sid}
   END;

 Logout=         /* Server Operation */
   ANY sid WHERE sid: session & sid: logout_buf  THEN
```

```
      session:= session - {sid} ||
       session_user:= {sid}<<| session_user ||
        session_state:= {sid}<<| session_state ||
        session_request:= {sid} <<| session_request ||
        flight_options:= {sid}<<|flight_options ||
        selctflight_buf:= {sid}<<| selctflight_buf ||
        booked_flight:= {sid}<<| booked_flight ||
        ubselctflight_buf:= {sid}<<|ubselctflight_buf ||
        room_options:= {sid}<<|room_options ||
        selctroom_buf:= {sid}<<| selctroom_buf ||
        booked_room:= {sid}<<|booked_room ||
        ubselctroom_buf:= {sid}<<| ubselctroom_buf ||
        car_options:= {sid}<<|car_options ||
        selctcar_buf:= {sid}<<| selctcar_buf ||
        booked_car:= {sid}<<|booked_car ||
        ubselctcar_buf:= {sid}<<| ubselctcar_buf ||
        availflight_buf:= {sid}<<| availflight_buf ||
        availroom_buf:= {sid}<<| availroom_buf ||
        availcar_buf:= {sid}<<| availcar_buf ||
        resp_buf:= {sid}<<|resp_buf ||
        bookedflight_buf:= {sid}<<| bookedflight_buf ||
        bookedroom_buf:= {sid}<<| bookedroom_buf ||
       bookedcar_buf:= {sid}<<| bookedcar_buf ||
     logout_buf:= logout_buf - {sid}
    END
END
```

A.3 Second Refinement—Introduction of Secondary Servers into The Model

```
 REFINEMENT  TravelAgency2F
 REFINES     TravelAgency1F

 SETS
    SESSION; USER; PASS; CARD_DETAIL ;
    FLIGHT_REQUEST; FLIGHT_DETAIL;
    ROOM_REQUEST; ROOM_DETAIL;
    CAR_REQUEST; CAR_DETAIL;
    REQUEST={bf, br, bc, uf, ur, uc, none};
    TAG= { Reg, Login, Logout}; HANDLE;

    FLIGHT_AGENCY; HOTEL; CAR_AGENCY

 DEFINITIONS
    freshSESSION == SESSION - session;
    newSESSION(new) == session \/{new} ;
    freshUSER== USER - dom(current_user) ;
    freshPASS== PASS - ran(current_user) ;
    freshHANDLE == HANDEL - dom(new_client)

 CONSTANTS
    unnamed
    Matchflight, Matchroom, Matchcar

 PROPERTIES
    unnamed: USER &
    Matchflight: FLIGHT_REQUEST * POW(FLIGHT_DETAIL)-->POW(FLIGHT_DETAIL) &
    Matchroom: ROOM_REQUEST * POW(ROOM_DETAIL)-->POW(ROOM_DETAIL) &
    Matchcar: CAR_REQUEST * POW(CAR_DETAIL)-->POW(CAR_DETAIL)

VARIABLES

    /*  Server Variables  */

    session, session_user, session_state,
    session_request, current_user,
    flight_booking, flight_options, booked_flight,
    room_db, room_booking, room_options, booked_room,
    car_db, car_booking, car_options, booked_car,

    /*  Client Variables  */

    new_handle, new_client, token,
    fresh_session, req_sevicebuf,
    reqFD_buf, reqRD_buf, reqCD_buf,
    unbooking_session, unsigned_sessinon,
    unnamed_session, booking_session,
    flightReq_buf, roomReq_buf, carReq_buf,
    flight_select, room_select, car_select,
    userInfo_buf, reqCard_buf, certified_session,
    card_buf, valid_session, invalid_session,
    ubflight_buf, ubroom_buf, ubcar_buf,
    logout_buf

    /* Server's New Variables */

    reqflight_buf, reqroom_buf, reqcar_buf,
    ret_session,selectflight_buf, selectroom_buf,
    selectcar_buf, taf_booking,
    tar_booking, tac_booking,
    pro_session1, failed_session1,
    suc_session, unsuc_session,
    selectUBflight_buf, selectUBroom_buf,
    selectUBcar_buf,
    pro_session2, failed_session2,
    send_session,

    /* Flight Agency Variables */
     flight_db1, respflight_buf,
     fa_booking,

    /* Hotels  Variables */
      room_db1,  resproom_buf,

    /* Car Agency Variables */
    car_db1, respcar_buf

INVARIANT
    /* Server's New Variables */
    reqflight_buf: FLIGHT_AGENCY +-> (SESSION +-> FLIGHT_REQUEST) &
    reqroom_buf: HOTEL +-> (SESSION +-> ROOM_REQUEST) &
    reqcar_buf: CAR_AGENCY +-> (SESSION +-> CAR_REQUEST) &
    ret_session<: session &
    selectflight_buf: FLIGHT_AGENCY +->(SESSION +->FLIGHT_DETAIL)&
    selectroom_buf: HOTEL +->(SESSION +->ROOM_DETAIL )&
    selectcar_buf: CAR_AGENCY +-> (SESSION +->CAR_DETAIL) &
    taf_booking: POW(FLIGHT_AGENCY*FLIGHT_DETAIL*USER) &
    tar_booking: POW(HOTEL*ROOM_DETAIL*USER) &
    tac_booking: POW(CAR_AGENCY*CAR_DETAIL*USER) &
    pro_session1<: session &
    failed_session1<: session &
    suc_session<: session &
    unsuc_session<: session &
    selectUBflight_buf: FLIGHT_AGENCY +->(FLIGHT_DETAIL+->USER) &
    selectUBroom_buf: HOTEL +->(ROOM_DETAIL+->USER) &
    selectUBcar_buf: CAR_AGENCY +->(CAR_DETAIL+->USER) &
    pro_session2<: session &
    failed_session2<: session &
    send_session<: session &

    /* Flight Agency Variables */
    flight_db1: FLIGHT_AGENCY --> POW(FLIGHT_DETAIL) &
    ! fa1,fa2 . (fa1:FLIGHT_AGENCY & fa2:FLIGHT_AGENCY & fa1/=fa2 =>
                 flight_db1(fa1) /\ flight_db1(fa2)={}) &
    flight_db = UNION(fa).(fa: FLIGHT_AGENCY |flight_db1(fa)) &
    respflight_buf: SESSION --> (FLIGHT_AGENCY <->  FLIGHT_DETAIL) &
    fa_booking: FLIGHT_AGENCY +->POW(FLIGHT_DETAIL * USER) &

    /* Hotels  Variables */
    room_db1: HOTEL --> POW(ROOM_DETAIL) &
    ! hh1,hh2 . (hh1: HOTEL & hh2: HOTEL & hh1/=hh2 =>
                 room_db1(hh1) /\ room_db1(hh2)={}) &
    room_db = UNION(hh).(hh: HOTEL |room_db1(hh)) &
    resproom_buf: SESSION --> (HOTEL <->  ROOM_DETAIL) &
    hotel_booking: HOTEL +->POW(ROOM_DETAIL * USER) &

    /* Car Agency Variables */
    car_db1: CAR_AGENCY --> POW(CAR_DETAIL) &
    car_db = UNION(ca).(ca: CAR_AGENCY |car_db1(ca)) &
   ! ca1,ca2 . (ca1:CAR_AGENCY & ca2:CAR_AGENCY & ca1/=ca2 =>
                 car_db1(ca1) /\ car_db1(ca2)={}) &
    respcar_buf: SESSION --> POW(CAR_AGENCY <->  CAR_DETAIL) &
    ca_booking: CAR_AGENCY +->POW(CAR_DETAIL * USER)

INITIALISATION
    session := {} || session_user := {} ||
    session_request := {} || current_user:= {} ||
    flight_db:= POW(FLIGHT_DETAIL) || flight_booking:={} ||
    flight_options:= {} || booked_flight:= {} ||
    room_db:= POW(ROOM_DETAIL) || room_booking:={} ||
    room_options:= {} || booked_room:= {} ||
    car_db:= POW(CAR_DETAIL) || car_booking:={} ||
    car_options:= {} || booked_car:= {}

    new_handle:={} || new_client:={} || token:={} ||
    fresh_session:={} || req_sevicebuf:={} ||
    reqFD_buf:={} || reqRD_buf:={} || reqCD_buf:={} ||
    unbooking_session:={} || unsigned_session:={} ||
    unnamed_session:={} || booking_session:={} ||
    flightReq_buf:={} || roomReq_buf:={} || carReq_buf:={} ||
    flight_select:= {} || room_select:= {} || car_select:= {} ||
    userInfo_buf:={}|| reqCard_buf:={}|| certified_session:={}||
    card_buf:= {} || valid_session:={}|| invalid_session:={} ||
    ubflight_buf:= {} || ubroom_buf:= {} || ubcar_buf:= {} ||
    logout_buf:= {}

    /* New Variables Initialisation */
    reqflight_buf:= {} || reqroom_buf:= {} || reqcar_buf:= {} ||
    ret_session:= {} || selectflight_buf:= {} || selectroom_buf:= {} ||
    selectcar_buf:= {} || taf_booking:= {} ||
    tar_booking:= {} || tac_booking:= {} ||
    pro_session1:= {} || failed_session1:= {} ||
    suc_session:= {} || unsuc_session:= {} ||
    selectUBflight_buf:= {} || selectUBroom_buf:= {} ||
    selectUBcar_buf: := {} ||
    pro_session2:= {} || failed_session2:= {} ||
    send_session:= {} ||

    flight_db1:= FLIGHT_AGENCY -->POW(FLIGHT_DETAIL) ||
    respflight_buf:= {} || fa_booking:= {} ||

    room_db1:= HOTEL --> POW(ROOM_DETAIL) ||
    resproom_buf:= {} || hotel_booking::= {} ||

    car_db1:= CAR_AGENCY --> POW(CAR_DETAIL) ||
    respcar_buf:= {} || ca_booking:= {} ||

OPERATIONS

  Client_ReqSession=        /* Client Operation */
    ANY handle WHERE handle: freshHANDLE THEN
      new_handle:= new_handle \/{handle}
    END;
```

```
StartNewSession =       /* Server Operation */
  ANY sid, handle WHERE  sid: freshSESSION & handle: new_handle  THEN

   newSESSION(sid)||
    session_user(sid) := unnamed  ||
    session_request(sid) := none ||
    new_client(handle):= sid ||
   new_handle:= new_handle - handle
  END;

Get_SessionID=        /* Client Operation */
  ANY sid WHERE sid: SESSION & sid: ran(new_client) THEN
    token:= token \/ sid ||
    fresh_session:= fresh_session \/ sid ||
    new_client:= new_client |>>{sid}
  END;

PicService(sid,req)=         /* Client Operation */
 PRE sid: SESSION & req: REQUEST THEN
   SELECT sid: fresh_session & req: REQUEST & req/= none THEN
      req_sevicebuf(sid):= req ||
      fresh_session:= fresh_session - sid
   END
 END;

SelectService =        /* Server Operation */
 ANY sid, req WHERE sid: session & req: REQUEST & req/= none
          sid: dom(req_sevicebuf) & req_sevicebuf(sid)= req THEN

   session_request(sid):= req_sevicebuf(sid) ||
    req_sevicebuf:= sid<<| req_sevicebuf ||
    CASE req OF
     EITHER  bf THEN
      reqFD_buf:= reqFD_buf \/ {sid}
     OR  br THEN
      reqRD_buf:= reqRD_buf \/ {sid}
     OR  bc THEN
      reqCD_buf:= reqCD_buf \/ {sid}
     OR uf,ur,uc THEN
      unbooking_session:= unbooking_session \/ sid
     END
 END;

Submit_FlightDetail(sid,fr) =       /* Client Operation */
  PRE sid: SESSION & fr: FLIGHT_REQUEST THEN
   SELECT sid: token & sid: reqFD_buf & fr: FLIGHT_REQUEST  THEN
      flightReq_buf(sid):= fr ||
      reqFD_buf:= {sid}<<| reqFD_buf
   END
  END;

Submit_RoomDetail(sid,rr) =       /* Client Operation */
  PRE sid: SESSION & rr: ROOM_REQUEST THEN
   SELECT sid: token & sid: reqRD_buf & fr: ROOM_REQUEST  THEN
      roomReq_buf(sid):=rr ||
      reqRD_buf:= {sid}<<| reqRD_buf
   END
  END;

Submit_CarDetail(sid,cr) =       /* Client Operation */
  PRE sid: SESSION & cr: CAR_REQUEST THEN
   SELECT sid: token & sid: reqCD_buf & fr: CAR_REQUEST  THEN
      carReq_buf(sid):=cr ||
      reqCD_buf:= {sid}<<| reqCD_buf
   END
  END;

Request_Flight =       /* Server Operation */
  ANY sid,fr WHERE sid: SESSION & sid: dom(flightReq_buf) &
                   fr: FLIGHT_REQUEST & flightReq_buf(sid)= fr  THEN
   reqflight_buf:= %fa . (fa: FLIGHT_AGENCY | reqflight_buf(fa) \/{sid|->fr}) ||
   flightReq_buf:= {sid}<<| flightReq_buf
  END

Request_Room =       /* Server Operation */
  ANY sid,rr WHERE sid: session & sid: dom(roomReq_buf)
                     rr: ROOM_REQUEST & roomReq_buf(sid)= rr  THEN
   reqroom_buf:= %hh . (hh: HOTEL | reqroom_buf(hh) \/{sid|->rr}) ||
   roomReq_buf:= {sid}<<| roomReq_buf
 END
```

```
Request_Car =       /* Server Operation */
  ANY sid,cr WHERE sid: session & sid: dom(carReq_buf) &
                   cr: CAR_REQUEST  & carReq_buf(sid)= cr THEN

   reqcar_buf:= %ca . (ca: CAR_AGENCY | reqcar_buf(ca) \/{sid|->cr}) ||
   carReq_buf:= {sid}<<| carReq_buf
  END;

Resp_FlightReqs =       /* Flight Agency Server Operation */
 ANY sid,fa,fr WHERE sid: session &  fa:FLIGHT_AGENCY & fa: dom(reqflight_buf)
                     & fr: FLIGHT_REQUEST & {sid|->fr}: reqflight_buf(fa) THEN

   ANY xx WHERE xx <: Matchflight(fr|->flight_db1(fa)) THEN
     respflight_buf(sid):= respflight_buf(sid) \/ {fa |-> xx}
   END    ||
  reqflight_buf(fa):= reqflight_buf(fa)- {sid|->fr}
 END;

Resp_RoomReqs =       /* Hotel Server Operation */
 ANY sid,hh,rr WHERE sid:session & hh: HOTEL & hh: dom(reqroom_buf) &
                     rr: ROOM_REQUEST & (sid|->rr):reqroom_buf(hh) THEN

   ANY xx WHERE xx <: Matchroom(rr|->room_db1(hh)) THEN
     resproom_buf(sid):= resproom_buf(sid) \/ {hh |-> xx}
   END    ||
  reqroom_buf(hh):= reqroom_buf(hh) - {sid|->rr}
 END;

Resp_CarReqs =       /* Car Agency Server Operation */
 ANY sid,ca,cr WHERE sid: session & ca: CAR_AGENCY & ca: dom(reqcar_buf) &
                     cr: CAR_REQUEST & {sid|->cr}: ran(reqcar_buf(ca)) THEN

   ANY xx WHERE xx <: Matchcar(cr|->car_db1(ca)) THEN
     respcar_buf(sid):= respcar_buf(sid) \/ {ca |-> xx}
   END    ||
  reqcar_buf(ca):= reqcar_buf(ca) - {sid|->cr}
 END;

Retrieve_FlightOptions=         /* Server Operation */
 ANY sid WHERE sid: session & sid: dom(respflight_buf)& sid/: ret_session &
                                       card(dom(respflight_buf(sid)))>= 3 THEN
     flight_options(sid):= ran(respflight_buf(sid)) ||
     ret_session:= ret_session \/ {sid}
 END;

Retrieve_RoomOptions=         /* Server Operation */
 ANY sid WHERE sid: session & sid: dom(resproom_buf)& sid/: ret_session &
                                       card(dom(resproom_buf(sid)))>= 3 THEN

     room_options(sid):= ran(resproom_buf(sid)) ||
     ret_session:= ret_session \/ {sid}
 END;

Retrieve_CarOptions=         /* Server Operation */
 ANY sid WHERE sid: session & sid: dom(respcar_buf)& sid/: ret_session &
                                       card(dom(respcar_buf(sid)))>= 3 THEN

     car_options(sid):= ran(respcar_buf(sid)) ||
     ret_session:= ret_session \/ {sid}
 END;


Select_Flight=       /* Client Operation */
  ANY sid, fd WHERE sid: token & sid: dom(flight_options) &
                    fd: FLIGHT_DETAIL & fd: flight_options(sid) THEN
    flight_select(sid):= fd ||
    booking_session:= booking_session \/ sid ||
    flight_options:= {sid}<<| flight_options
  END;

Select_Room=       /* Client Operation */
  ANY sid, rd WHERE sid: token & sid: dom(room_options) &
                    rd: ROOM_DETAIL & fd: room_options(sid) THEN
    room_select(sid):= rd ||
    booking_session:= booking_session \/ sid ||
    room_options:= {sid}<<| room_options
  END;

Select_Car=       /* Client Operation */
  ANY sid, cd WHERE sid: token & sid: dom(car_options) &
                    cd: CAR_DETAIL & fd: car_options(sid) THEN
    car_select(sid):= cd ||
    booking_session:= booking_session \/ sid ||
    car_options:= {sid}<<| car_options
  END;

Request_UserInfo=         /* Server Operation */
 ANY sid WHERE sid: session & (sid: booking_session or sid: unbooking_session) THEN
   SELECT sid: unbooking_session THEN         /* If User has signed-in before there*/
     IF  session_user(sid)= unnamed THEN      /* is no nedd for relogin. */
       unnamed_session:= unnamed_session \/ sid ||
       unbooking_session:= unbooking_session - sid  /* There is also a distinction */
     ELSE                                          /* between a user that requests */
       reqCard_buf:= reqCard_buf \/ sid ||        /* an unbooking and othere one's */
       unbooking_session:= unbooking_session - sid    /* because the former has to */
     END                                              /* registered before so it */
                                               /* can just can login, but
                          for booking session it can either login or register */
   WHEN sid: booking_session THEN
     IF  session_user(sid)= unnamed THEN
       unsigned_session:= unsigned_session \/ sid ||
       booking_session:= booking_session - sid
     ELSE
       reqCard_buf:= reqCard_buf \/ sid ||
       booking_session:= booking_session - sid
     END
   END
 END;

Client_Register(sid,name,pass,confpass) =       /* Client Operation */
 PRE sid: SESSION & name: USER & pass: PASS & confpass: PASS THEN
  SELECT sid: token & sid: unsigned_session  & name: freshUSER &
                              pass: freshPASS & pass= confpass THEN
   userInfo_buf(sid):= {username|->pass|->Reg} ||
   unsigned_session:= unsigned_session - {sid}
  END
 END;

Register =        /* Server Operation */
 ANY sid,username,pass WHERE sid: session & sid:dom(userInfo_buf) &
                             username: freshUSER & pass: freshPASS &
                          userInfo_buf(sid)= (username|->pass|->Reg) THEN
   current_user:= current_user \/ {username|->pass} ||
   session_user(sid) := username ||
   reqCard_buf:= reqCard_buf \/ sid ||
   userInfo_buf := {sid}<<| userInfo_buf
 END;

Client_login(sid,username,pass) =       /* Client Operation */
 PRE sid: SESSION & name: USER & pass: PASS  THEN
  SELECT sid: token & (sid: unsigned_session or sid:unnamed_session) &
                                  & {username|->pass}: current_user  THEN
   userInfo_buf(sid):= {username|->pass|->Login} ||
   IF sid: unsigned_session THEN
      unsigned_session:= unsigned_session - {sid}
   ELSE
      unnamed_session:= unnamed_session - {sid}
   END
  END
 END;

Login  =        /* Server Operation */
 ANY sid,username, pass WHERE sid: session & sid: dom(userInfo_buf) &
   {username|->pass}:current_user & userInfo_buf(sid)={username|->pass|->Login} THE

   session_user(sid) := username ||
    CASE session_request(sid) OF
     EITHER  bf, br, bc THEN
       reqCard_buf:= reqCard_buf \/ sid
     OR uf,ur,uc THEN
       certified_session:= certifed_session \/ {sid}
     END ||
   userInfo_buf:= {sid}<<| userInfo_buf
 END;

EnterCard(sid, xcard) =       /* Client Operation */
  PRE sid: SESSION & xcard: CARD_DETAIL  THEN
   SELECT sid:token & sid: reqCard_buf & xcard: CARD_DETAIL  THEN

     card_buf(sid):= xcard ||
     reqCard_buf:= {sid}<<| reqCard_buf
   END
```

```
 END;

Card_Validate=        /* Server Operation */
  ANY sid WHERE sid:session & sid: dom(card_buf) THEN
     CHOICE
      valid_session:= valid_session \/ {sid}
     OR
      invalid_session:= invalid_session \/ {sid}
     END ||
      card_buf:= {sid}<<| card_buf
  END;

Restart_invalid=        /* Server Operation */
 ANY sid WHERE sid: session & sid: invalid_session THEN
      fresh_session:=fresh_session \/ {sid} ||
      invalid_session:= invalid_session - {sid}
 END;

Send_SelectedFlight=        /* Server Operation */
 ANY sid,fd WHERE sid:session & sid: valid_session & session_request(sid):=bf &
       sid: dom(flight_select) & fd: FLIGHT_DETAIL &  flight_select(sid)= fd THEN
   ANY fa WHERE fa: FLIGHT_AGENCY & {fa|-> fd}: respflight_buf(sid) &
                                                          sid: ret_session THEN
     selectflight_buf(fa):= selectflight_buf(fa) \/ {sid|->fd} ||
     ret_session:= ret_session - {sid} ||
     respflight_buf:= {sid}<<| respflight_buf
   END ||
   valid_session:= valid_session - {sid} ||
 END;

Send_SelectedRoom=        /* Server Operation */
 ANY sid,rd WHERE sid:session & sid: valid_session & session_request(sid):=br &
       sid: dom(room_select) & rd: ROOM_DETAIL &  room_select(sid)= rd THEN
   ANY hh WHERE hh: HOTEL & {hh|-> rd}: resproom_buf(sid) &
                                                          sid: ret_session THEN
     selectroom_buf(fa):= {sid|->rd} ||
     ret_session:= ret_session - {sid} ||
     resproom_buf:= {sid}<<| resproom_buf
   END ||
   valid_session:= valid_session - {sid} ||
 END;
```

```
Send_SelectedCar=        /* Server Operation */
 ANY sid,cd WHERE sid:session & sid: valid_session & session_request(sid):=bc &
       sid: dom(car_select) & cd: CAR_DETAIL &  car_select(sid)= cd THEN
   ANY ca WHERE ca: CAR_AGENCY & {ca|-> cd}: respcar_buf(sid) &
                                                         sid: ret_session THEN
     selectcar_buf(ca):= {sid|->cd} ||
     ret_session:= ret_session - {sid} ||
     respcar_buf:= {sid}<<| respcar_buf
   END ||
   valid_session:= valid_session - {sid} ||
 END;

Agency_flight_booking=        /* Flight_agency Server Operation*/
  ANY fa,sid,fd WHERE fa: FLIGHT_AGENCY & fa: dom(selectflight_buf) & sid: SESSION
                  & sid/: pro_session & sid/: failed_session & fd: FLIGHT_DETAIL
                                                & {sid|->fd}: selectflight_buf(fa) THEM
     IF fd: flight_db1(fa) THEN
       ANY fdb WHERE fdb<: flight_db1(fa) THEN
       /* Updating original Database that maybe affected by booking */
          flight_db1(fa):= fdb
       END ||
        fa_booking(fa):= fa_booking(fa) \/{fd|->session_user(sid)} ||
       pro_session1:=pro_session1 \/ {sid}
     ELSE
      failed_session1:= failed_session1 \/ {sid}
     END
  END;

Hotel_room_booking=        /* Hotel Server Operation*/
  ANY hh,sid,rd WHERE hh: HOTE & hh: dom(selectroom_buf) & sid: SESSION
                  & sid/: pro_session & sid/: failed_session & rd: ROOM_DETAIL
                                                & {sid|->rd}: selectroom_buf(hh) THEM
     IF rd: room_db1(hh) THEN
       ANY rdb WHERE rdb<: room_db1(hh) THEN
       /* Updating original Database that maybe affected by booking */
          room_db1(hh):= rdb
       END ||
        hotel_booking(hh):= hotel_booking(hh) \/{rd|->session_user(sid)} ||
       pro_session1:=pro_session1 \/ {sid}
     ELSE
```

```
      failed_session1:= failed_session1 \/ {sid}
     END
  END;

Agency_car_booking=        /* Car_agency Server Operation*/
  ANY ca,sid,cd WHERE ca: CAR_AGENCY & ca: dom(selectcar_buf) & sid: SESSION
                  & sid/: pro_session & sid/: failed_session & rd: ROOM_DETAIL
                                          & {sid|->cd}: selectcar_buf(ca) THEM
     IF cd: car_db1(ca) THEN
       ANY cdb WHERE cdb<: car_db1(ca) THEN
       /* Updating original Database that maybe affected by booking */
          car_db1(ca):= cdb
       END ||
        ca_booking(ca):= ca_booking(ca) \/{cd|->session_user(sid)} ||
       pro_session1:=pro_session1 \/ {sid}
     ELSE
      failed_session1:= failed_session1 \/ {sid}
     END
  END;

Flight_Booking =        /* Server Operation */
 ANY sid,fa,fd WHERE sid:session & (sid: pro_session or sid: failed_session) &
     sid: dom(flight_select) & fd: FLIGHT_DETAIL &  flight_select(sid)= fd &
   fa: FLIGHT_AGENCY & fa: dom(selectflight_buf) & {sid|->fd}: selectflight_buf(fa) THEN

   IF sid: pro_session THEN
     taf_booking:= taf_booking \/{fa|->fd|->session_user(sid)} ||
      selectflight_buf(fa):= selectflight_buf(fa) - {sid|->fd} ||
       flight_select:= {sid}<<| flight_select||
       pro_session1:= pro_session1 - {sid} ||
      suc_session:=suc_session \/ {sid} ||
     session_request(sid):= none
   ELSE
      selectflight_buf(fa):= selectflight_buf(fa) - {sid|->fd} ||
       flight_select:= {sid}<<| flight_select||
        failed_session1:= failed_session1 - {sid} ||
       unsuc_session:=unsuc_session \/ {sid} ||
      session_request(sid):= none
   END
 END;
```

```
Room_Booking =        /* Server Operation */
 ANY sid,hh,rd WHERE sid: session & (sid: pro_session or sid: failed_session) &
             sid: dom(room_select) & rd: ROOM_DETAIL & room_select(sid)= rd
             hh: HOTEL & hh: dom(selectroom_buf) & {sid|->rd}: selectroom_buf(hh)

   IF sid: pro_session THEN
     tar_booking:= tar_booking \/{hh|->rd|->session_user(sid)} ||
      selectroom_buf(hh):= selectroom_buf(hh) - {sid|->rd} ||
       room_select:= {sid}<<| room_select||
       pro_session1:= pro_session1 - {sid} ||
      suc_session:=suc_session \/ {sid} ||
     session_request(sid):= none
   ELSE
      selectroom_buf(hh):= selectflight_buf(hh) - {sid|->rd} ||
       room_select:= {sid}<<| room_select||
        failed_session1:= failed_session1 - {sid} ||
       unsuc_session:=unsuc_session \/ {sid} ||
      session_request(sid):= none
   END
 END;

Car_Booking =        /* Server Operation */
 ANY sid,cd WHERE sid: session & sid: pro_session or sid: failed_session) &
                 sid: dom(car_select) &  cd:CAR_DETAIL & car_select(sid)= cd
   ca: CAR_AGENCY & ca: dom(selectcar_buf) & {sid|->cd}: selectcar_buf(fa) THEN

   IF sid: pro_session THEN
     tac_booking:= tac_booking \/{ca|->cd|->session_user(sid)} ||
      selectcar_buf(ca):= selectcar_buf(ca) - {sid|->cd} ||
       car_select:= {sid}<<| car_select||
       pro_session1:= pro_session1 - {sid} ||
      suc_session:=suc_session \/ {sid} ||
     session_request(sid):= none
   ELSE
      selectcar_buf(ca):= selectcar_buf(ca) - {sid|->cd} ||
       car_select:= {sid}<<| car_select||
        failed_session1:= failed_session1 - {sid} ||
       unsuc_session:=unsuc_session \/ {sid} ||
      session_request(sid):= none
   END
 END;
```

```
Retrieve_BookedFlight=         /* Server Operation */
 ANY sid, fa WHERE sid: session & sid: certified_session &
                      fa: FLIGHT_AGENCY &  session_request(sid):=uf THEN
   booked_flight(sid):=  %fd . (fd: FLIGHT_DETAIL &
                     {fa|->fd|->session_user(sid)}:taf_booking| fd) ||
   valid_session:= valid_session - sid
 END;

Retrieve_BookedRoom=         /* Server Operation */
 ANY sid, hh WHERE sid: session & sid: certified_session &
                     hh: HOTEL & session_request(sid):=ur THEN
   booked_room(sid):=  %rd . (rd: ROOM_DETAIL &
                   {hh|->rd|->session_user(sid)}:tar_booking | rd) ||
   valid_session:= valid_session - sid
  END;

Retrieve_BookedCar=         /* Server Operation */
 ANY sid, ca WHERE sid: session & sid: certified_session &
                     ca: CAR_AGENCY &  session_request(sid):=uc THEN

   booked_car(sid):=  %cd . (cd: CAR_DETAIL &
                   (ca|->cd|->session_user(sid)):tac_booking | cd) ||
   valid_session:= valid_session - sid
 END;

Select_UnBbookedFlight=       /* Client Operation */
  ANY sid, fd WHERE sid: token & sid: dom(booked_flight) &
                   fd: FLIGHT_DETAIL & fd: booked_flight(sid) THEN
    ubflight_buf(sid):= fd ||
    booked_flight:= {sid}<<| booked_flight
  END;

Select_UnBbookedRoom=       /* Client Operation */
  ANY sid, rd WHERE sid: token & sid: dom(booked_room) &
                   rd: ROOM_DETAIL & fd: booked_room(sid) THEN
    ubroom_buf(sid):= rd ||
    booked_room:= {sid}<<| booked_room
  END;

Select_UnBbookedCar=       /* Client Operation */
```

```
    ANY sid, cd WHERE sid: token & sid: dom(booked_car) &
                      cd: CAR_DETAIL & fd: booked_car(sid) THEN
      ubcar_buf(sid):= cd ||
      booked_car:= {sid}<<| booked_car
    END;

  Send_UBFlight=        /* Server Operation */
   ANY sid, fd WHERE sid: session & sid: dom(ubflight_buf) & sid/: send_session &
                                        fd: FLIGHT_DETAIL & ubflight_buf(sid)=fd THEN
     ANY fa WHERE fa: FLIGHT_AGENCY & {fa|->fd|->session_user(sid)}:taf_booking THEN
       selectUBflight_buf(fa):= selectUBflight_buf(fa) \/ {fd|->session_user(sid)} ||
       send_session:= send_session \/ {sid}
     END
   END;

  Send_UBRoom=        /* Server Operation */
   ANY sid, rd WHERE sid: session & sid: dom(ubroom_buf) & sid/: send_session &
                                        rd: ROOM_DETAIL & ubroom_buf(sid)=rd THEN
     ANY hh WHERE hh: HOTEL & {hh|->rd|->session_user(sid)}:tar_booking THEN
       selectUBroom_buf(hh):= selectUBroom_buf(hh) \/ {rd|->session_user(sid)} ||
       send_session:= send_session \/ {sid}
     END
   END;

  Send_UBCar=        /* Server Operation */
   ANY sid, cd WHERE sid: session & sid: dom(ubcar_buf) &  sid/: send_session &
                                          cd: CAR_DETAIL & ubcar_buf(sid)=cd THEN
     ANY ca WHERE ca: CAR_AGENCY & {ca|->cd|->session_user(sid)}:tac_booking THEN
       selectUBcar_buf(ca):= selectUBcar_buf(ca) \/ {cd|->session_user(sid)} ||
       send_session:= send_session \/ {sid}
     END
   END;

  Agency_Flight_Unbooking=        /* Flight_agency Server Operation*/
    ANY fa,fd,uu WHERE fa: FLIGHT_AGENCY & fa: dom(selectUBflight_buf) &
        uu: USER & fd: FLIGHT_DETAIL & {fd|->uu}: selectUBflight_buf(fa) THEM
       IF fd: fa_booking(fa) THEN
         ANY fdb WHERE flight_db1(fa)<: fdb  THEN
         /* Updating original Database that maybe affected by Unbooking */
            flight_db1(fa):= fdb
         END ||
          fa_booking(fa):= fa_booking(fa) - {fd|->uu} ||
         pro_session2:=pro_session2 \/ {sid}
       ELSE
        failed_session2:= failed_session2 \/ {sid}
       END
    END;

  Hotel_Room_Unbooking=        /* Hotel Server Operation*/
    ANY hh,rd,uu WHERE hh: HOTE & hh: dom(selectUBroom_buf) & uu: USER &
                     rd: ROOM_DETAIL & {rd|->uu}: selectUBroom_buf(hh) THEM
       IF rd: hotel_booking(hh) THEN
         ANY rdb WHERE room_db1(hh)<: rdb  THEN
         /* Updating original Database that maybe affected by Unbooking */
            room_db1(hh):= rdb
         END ||
          hotel_booking(hh):= hotel_booking(hh) - {rd|->uu} ||
         pro_session2:=pro_session2 \/ {sid}
       ELSE
        failed_session2:= failed_session2 \/ {sid}
       END
    END;

  Agency_Car_Unbooking=        /* Car_agency Server Operation*/
    ANY ca,cd,uu WHERE ca: CAR_AGENCY & ca: dom(selectcar_buf) & uu: USER &
                       rd: ROOM_DETAIL & {cd|->uu}: selectUBcar_buf(ca) THEM
       IF cd: ca_booking(ca THEN
         ANY cdb WHERE car_db1<:(ca)cdb  THEN
         /* Updating original Database that maybe affected by Unbooking */
            car_db1(ca):= cdb
         END ||
          ca_booking(ca):= ca_booking(ca)- {cd|->|->uu} ||
         pro_session2:= pro_session2 \/ {sid}
       ELSE
        failed_session2:= failed_session2 \/ {sid}
       END
    END;

  Unbook_Flight =         /* Server Operation */
   ANY sid, fd WHERE sid: session & (sid: pro_session2 or sid:failed_session2) &
      sid: send_session & sid: dom(ubflight_buf) & fd: FLIGHT_DETAIL &
                                                       ubflight_buf(sid)=fd  THEN
     ANY fa WHERE fa: FLIGHT_AGENCY & {fa|->fd|->session_user(sid)}:taf_booking THEN
       taf_booking:= taf_booking - {fa|->fd|->session_user(sid)} ||
       send_session:= send_session - {sid} ||
       ubflight_buf:= {sid} <<| ubflight_buf ||
       session_request(sid):= none ||
        IF sid: pro_session2 THEN
          pro_session2:= pro_session2- {sid} ||
          suc_session:= suc_session \/ {sid}
        ELSE
          failed_session2:= failed_session2- {sid} ||
          unsuc_session:= unsuc_session \/ {sid}
        END
     END
   END;

  Unbook_Room =         /* Server Operation */
   ANY sid, rd WHERE sid: session & (sid: pro_session2 or sid:failed_session2) &
      sid: send_session & sid: dom(ubroom_buf) & rd: ROOM_DETAIL &
                                                          ubroom_buf(sid)= rd THEN
     ANY hh WHERE hh: HOTEL & {hh|->rd|->session_user(sid)}:tar_booking THEN
       tar_booking:= tar_booking - {hh|->rd|->session_user(sid)} ||
       send_session:= send_session - {sid} ||
       ubroom_buf:= {sid} <<| ubroom_buf ||
       session_request(sid):= none ||
        IF sid: pro_session2 THEN
          pro_session2:= pro_session2- {sid} ||
          suc_session:= suc_session \/ {sid}
        ELSE
          failed_session2:= failed_session2- {sid} ||
          unsuc_session:= unsuc_session \/ {sid}
        END
     END
   END;

  Unbook_Car =         /* Server Operation */
   ANY sid, cd WHERE sid: session & (sid: pro_session2 or sid:failed_session2) &
      sid: send_session & sid: dom(ubcar_buf) & cd: CAR_DETAIL &
                                                        ubcar_buf(sid)=cd  THEN
     ANY ca WHERE ca: CAR_AGENCY & {ca|->cd|->session_user(sid)}:tac_booking THEN
       tac_booking:= tac_booking - {ca|->cd|->session_user(sid)} ||
       send_session:= send_session - {sid} ||
       ubcar_buf:= {sid} <<| ubcar_buf ||
       session_request(sid):= none ||
        IF sid: pro_session2 THEN
          pro_session2:= pro_session2- {sid} ||
          suc_session:= suc_session \/ {sid}
        ELSE
          failed_session2:= failed_session2- {sid} ||
          unsuc_session:= unsuc_session \/ {sid}
        END
     END
   END;

  Client_Recv_Reply=       /* Client Operation */
    ANY sid WHERE sid: token & sid: dom(resp_buf) THEN
     resp_buf:= {sid}<<|resp_buf
    END;


  Client_logout=       /* Client Operation */
    ANY sid WHERE sid: token THEN
      reqsevice_buf:= {sid}<<| reqsevice_buf ||
      new_client:= new_client |>>{sid} ||
       fresh_session:= fresh_session - {sid} ||
        reqFD_buf:= reqFD_buf - {sid} ||
        reqRD_buf:= reqRD_buf - {sid} ||
        reqCD_buf:= reqCD_buf- {sid} ||
        flightReq_buf:= {sid}<<| flightReq_buf ||
        roomReq_buf:= {sid}<<| roomReq_buf ||
        carReq_buf:= {sid}<<| carReq_buf ||
        pikedflight_buf:= {sid}<<|pikedflight_buf ||
        pikedroom_buf:= {sid}<<|pikedroom_buf ||
        pikedcar_buf:= {sid}<<|pikedcar_buf ||
        unnamed_buf:= unnamed_buf - {sid} ||
        unsigned_buf:= unsigned_buf - {sid} ||
        userInfo_buf := {sid}<<| userInfo_buf ||
        reqCard_buf:= reqCard_buf - {sid} ||
        card_buf:= {sid}<<| card_buf ||
        ubflight_buf:= {sid}<<| ubflight_buf ||
       ubroom_buf:= {sid}<<| ubroom_buf||
      ubcar_buf:= {sid}<<|ubcar_buf ||
     logout_buf:= logout_buf \/ {sid}
    END;

  Logout=         /* Server Operation */
    ANY sid WHERE sid: session & sid: logout_buf  THEN
      session:= session - {sid} ||
       session_user:= {sid}<<| session_user ||
        session_state:= {sid}<<| session_state ||
        session_request:= {sid} <<| session_request ||
        flight_options:= {sid}<<|flight_options ||
        selctflight_buf:= {sid}<<| selctflight_buf ||
        booked_flight:= {sid}<<| booked_flight ||
        ubselctflight_buf:= {sid}<<|ubselctflight_buf ||
        room_options:= {sid}<<|room_options ||
        selctroom_buf:= {sid}<<| selctroom_buf ||
        booked_room:= {sid}<<|booked_room ||
        ubselctroom_buf:= {sid}<<| ubselctroom_buf ||
        car_options:= {sid}<<|car_options ||
        selctcar_buf:= {sid}<<| selctcar_buf ||
        booked_car:= {sid}<<|booked_car ||
        ubselctcar_buf:= {sid}<<| ubselctcar_buf ||
        availflight_buf:= {sid}<<| availflight_buf ||
        availroom_buf:= {sid}<<| availroom_buf ||
        availcar_buf:= {sid}<<| availcar_buf ||
        resp_buf:= {sid}<<|resp_buf ||
        bookedflight_buf:= {sid}<<| bookedflight_buf ||
        bookedroom_buf:= {sid}<<| bookedroom_buf ||
       bookedcar_buf:= {sid}<<| bookedcar_buf ||
     logout_buf:= logout_buf - {sid}
    END
END
```

Appendix B

Specification and Refinement of Patterns

B.1 Specification of The Session Creation pattern

```
MODEL
   Session_Creat_Spec

SETS
    SESSION; STATE = {NS, RM, RP};
    REQUEST; RESPONSE; AGENT_ID;
    SERVICES; SRVC_RESP

CONSTANTS
    null,
    ReqID, ReqSID, Srvc,
/* REQUEST ==
    ReqID   : AGENT_ID
    ReqSID  : SESSION
    Srvc        : SERVICES
*/
    RespID, RespSID, Srvc_resp
/* RESPONSE ==
    RespID  : AGENT_ID
    RespSID : SESSION
    Srvc_resp   : SRVC_RESP
*/
```

```
PROPERTIES
    null    : SESSION       &

    /* REQUEST Record Definition */
    ReqID       : REQUEST --> AGENT_ID  &
    ReqSID  : REQUEST --> SESSION   &
    Srvc        : REQUEST --> SERVICES  &

    /* RESPONSE  Record Definition */
    RespID      : RESPONSE --> AGENT_ID     &
    RespSID        : RESPONSE --> SESSION   &
    Srvc_resp   : RESPONSE --> SRVC_RESP

VARIABLES
    req_buf, current, req_hist, session, resp_buf,
    session_state, resp_hist

INVARIANT
    req_buf     : POW(REQUEST)
    & current   : POW(AGENT_ID)
    & req_hist  : POW(REQUEST)
    & session   : POW(SESSION)
    & resp_buf  : POW(RESPONSE)
    & session_state : session--> STATE
    & resp_hist : POW(RESPONSE)


INITIALISATION
    req_buf := {}
    || current := {}
    || req_hist := {}
    || session := {}
    || resp_buf := {}
    || session_state := {}
    || resp_hist := {}

OPERATIONS

  Client_CreateAgent =              /* Client Operation */
    ANY aid, req WHERE
        aid : AGENT_ID
        & aid /: current
        & req : REQUEST
        & req /: req_hist
        & ReqID(req) = aid
        & ReqSID(req) = null
    THEN
      current := current \/ {aid}
      || req_buf := req_buf \/ {req}
      || req_hist := req_hist \/ {req}
    END;

  Convey_SessionReq =           /* Middleware Operation */
     skip;



 Server_CreateSession =        /* Server Operation */
   ANY req, resp, sid WHERE
       req : REQUEST
       & req : req_buf
       & ReqSID(req) = null
       & resp : RESPONSE
       & RespID(resp)= ReqID(req)
       & sid : SESSION & sid /: session
       & RespSID(resp)= sid
       & resp /: resp_hist
   THEN
    resp_buf := resp_buf \/ {resp}
    || session := session \/ {sid}
    || session_state := session_state \/ {sid |-> NS}
    || req_buf:= req_buf - {req}
    || resp_hist := resp_hist \/ {resp}
   END;

  Convey_SessionID =        /* Middleware Operation */
     skip

END
```

B.2 Refinement of The Session Creation pattern

```
REFINEMENT
   Session_Creat_Ref1

REFINES
    Session_Creat_Spec

VARIABLES
    current, session,
    req_buf1, req_buf2,
    req_hist1, req_hist2,
    resp_buf1, resp_buf2,
    resp_hist1, resp_hist2

INVARIANT
    req_buf1      : POW(REQUEST)
    & req_buf2    : POW(REQUEST)
    & req_hist1   : POW(REQUEST)
    & req_hist2   : POW(REQUEST)
    & resp_buf1   : POW(RESPONSE)
    & resp_buf2   : POW(RESPONSE)
    & resp_hist1  : POW(RESPONSE)
    & resp_hist2  : POW(RESPONSE)
/* Gluing Invarints */
    & req_buf = req_buf1 \/ req_buf2
    & req_buf1 /\ req_buf2 = {}
    & resp_buf = resp_buf1 \/ resp_buf2
    & resp_buf1 /\ resp_buf2 = {}
    & req_buf1 <: ReqID~[current]
    & req_buf2 <: ReqID~[current]
    & resp_buf1 <: RespSID~[session]
    & resp_buf2 <: RespSID~[session]
/*
 & ! rq .( rq : REQUEST & rq : req_buf1 => ReqID(rq) : current)
 & ! rq .( rq : REQUEST & rq : req_buf2 => ReqID(rq) : current)
 & ! rsp . ( rsp : RESPONSE & rsp : resp_buf1 => RespSID(rsp) : session)
 & ! rsp . ( rsp : RESPONSE & rsp : resp_buf2 => RespSID(rsp) : session)
*/
INITIALISATION
    current        := {}
    || session     := {}
    || req_buf1    := {}
    || req_buf2    := {}
    || req_hist1   := {}
    || req_hist2   := {}
    || resp_buf1   := {}
    || resp_buf2   := {}
    || resp_hist1  := {}
    || resp_hist2  := {}

OPERATIONS

 Client_CreateAgent  =               /* Client Operation */
   ANY aid, req WHERE
       aid : AGENT_ID
       & aid /: current
       & req : REQUEST
       & ReqID(req) = aid
       & ReqSID(req) = null
       & req /: req_hist1
   THEN
    current:= current \/ {aid}
    || req_buf1 := req_buf1 \/ {req}
    || req_hist1 := req_hist1 \/ {req}
   END;

 Convey_SessionReq =                    /* Client Operation */
   ANY req WHERE req : REQUEST & req : req_buf1
    THEN
       req_buf2 := req_buf2 \/ {req} ||
       req_buf1 := req_buf1 - {req}
    END;

 Server_CreateSession =        /* Server Operation */
   ANY req, resp, sid WHERE
       req : REQUEST
       & req : req_buf2
       & ReqSID(req) = null
       & resp : RESPONSE
       & RespID(resp)= ReqID(req)
       & sid : SESSION & sid /: session
       & RespSID(resp)= sid
       & resp /: resp_hist1
   THEN
    resp_buf1 := resp_buf1 \/ {resp}
    || session := session \/ {sid}
    || req_buf2 := req_buf2 - {req}
    || resp_hist1 := resp_hist1 \/ {resp}
   END;

 Convey_SessionID =       /* Server Operation */
   ANY resp WHERE resp : RESPONSE & resp : resp_buf1
    THEN
       resp_buf2:= resp_buf2 \/ {resp} ||
       resp_buf1:= resp_buf1 - {resp}
    END
END
```

B.3 Specification of The User-to-Web Applications Pattern

```
MODEL
   User_to_Web_Spec

SETS
    SESSION; REQUEST; RESPONSE; AGENT_ID;
    SERVICES; SRVC_RESP; DB

CONSTANTS
    ReqID, ReqSID, Srvc,
/* REQUEST ==
    ReqID   : AGENT_ID
    ReqSID  : SESSION
    Srvc        : SERVICES
*/
    RespID, RespSID, Srvc_resp,
/* RESPONSE ==
    RespID  : AGENT_ID
    RespSID : SESSION
    Srvc_resp   : SRVC_RESP
*/
```

```
    Resp_func, Update_func

PROPERTIES

    /* REQUEST Record Definition */
    ReqID       : REQUEST --> AGENT_ID  &
    ReqSID  : REQUEST --> SESSION   &
    Srvc        : REQUEST --> SERVICES  &

    /* RESPONSE  Record Definition */
    RespID      : RESPONSE --> AGENT_ID     &
    RespSID     : RESPONSE --> SESSION      &
    Srvc_resp   : RESPONSE --> SRVC_RESP        &

    Resp_func   : (DB * SERVICES) --> RESPONSE  &
    Update_func : (DB * SERVICES) --> DB

VARIABLES
    req_hist, current, req_buf,
    session, resp_hist, resp_buf, db

INVARIANT
    req_buf     : POW(REQUEST)      &
    req_hist    : POW(REQUEST)      &
    req_buf <: req_hist         &
    current : POW(AGENT_ID)     &
    session     : POW(SESSION)      &
    resp_buf    : POW(RESPONSE)     &
    resp_hist   : POW(RESPONSE)     &
    resp_buf    <: resp_hist        &
    db : DB

INITIALISATION
    req_buf     := {}           ||
    req_hist    := {}           ||
    current     :: POW(AGENT_ID)    ||
    session     :: POW(SESSION) ||
    resp_buf    := {}       ||
    resp_hist   := {}           ||
    db      :: DB
```

```
OPERATIONS

 MakeRequest =         /* Client Operation */
   ANY req WHERE
       req : REQUEST
       & ReqID(req) : current
       & req /: req_hist
   THEN
     req_buf := req_buf \/ {req}   ||
     req_hist := req_hist \/ {req}
   END;

 Convey_Request =       /* Middleware Operation */
   skip;

 ProcessRequest =    /* Server Operation */
   ANY req, resp WHERE
       req : REQUEST
       & req :  req_buf
       & ReqSID(req) : session
       & resp : RESPONSE
       & RespID(resp) = ReqID(req)
       & RespSID(resp)  = ReqSID(req)
       & resp  = Resp_func(db, Srvc(req))
       & resp /: resp_hist
   THEN
    resp_buf := resp_buf \/ {resp}      ||
    resp_hist := resp_hist \/ {resp}   ||
    db:= Update_func(db, Srvc(req))    ||
    req_buf:= req_buf - {req}
   END;

 Convey_Response =      /* Middleware Operation */
  skip;

 GetResponse =        /* Client Operation */
  ANY resp WHERE
      resp : RESPONSE
      & resp : resp_buf
  THEN
```

```
    resp_buf :=  resp_buf- {resp}
  END
END
```

B.4 Refinement of The User-to-Web Applications Pattern

```
REFINEMENT
    User_to_Web_Ref1

REFINES
     User_to_Web_Spec

VARIABLES
    req_hist, current,
    req_buf1,  req_buf2,

    resp_hist, session, db,
    resp_buf1, resp_buf2

INVARIANT
    req_buf1    : POW(REQUEST)      &
    req_buf2    : POW(REQUEST)      &
    resp_buf1   : POW(RESPONSE)     &
    resp_buf2   : POW(RESPONSE)     &

/* Gluing Invarints */
    req_buf = req_buf1 \/ req_buf2  &
    req_buf1 /\ req_buf2 = {}       &
    ! rq .( rq : REQUEST & rq : req_buf1 => ReqID(rq) : current) &
    ! rq .( rq : REQUEST & rq : req_buf2 => ReqID(rq) : current) &
    req_buf1 <: req_hist    &
    req_buf2 <: req_hist    &

    resp_buf = resp_buf1 \/ resp_buf2   &
    resp_buf1 /\ resp_buf2 = {}     &
    ! rsp . ( rsp : RESPONSE & rsp : resp_buf1 => RespSID(rsp) : session) &
    ! rsp . ( rsp : RESPONSE & rsp : resp_buf2 => RespSID(rsp) : session) &
    resp_buf1  <: resp_hist &
    resp_buf2 <: resp_hist
```

```
INITIALISATION
    req_hist := {}          ||
    current :: POW(AGENT_ID)    ||
    req_buf1 := {}              ||
    req_buf2 := {}              ||

    resp_hist := {}         ||
    session :: POW(SESSION) ||
    db :: DB                ||

    resp_buf1 := {}             ||
    resp_buf2 := {}

OPERATIONS

 MakeRequest =        /* Client Operation */
   ANY req WHERE
       req : REQUEST
       & ReqID(req) : current
       & req /: req_hist
   THEN
     req_buf1:= req_buf1 \/ {req}  ||
     req_hist:= req_hist \/ {req}
   END;

 Convey_Request =       /* Middleware Operation */
    ANY   req WHERE req : REQUEST  &  req : req_buf1
     THEN
        req_buf2 := req_buf2 \/ {req}
        || req_buf1 := req_buf1 - {req}
     END;

 ProcessRequest =    /* Server Operation */
   ANY req, resp WHERE
       req : REQUEST
       & req :  req_buf2
       & ReqSID(req) : session
       & resp : RESPONSE
       & RespID(resp) = ReqID(req)
       & RespSID(resp)  = ReqSID(req)
       & resp  = Resp_func(db, Srvc(req))
       & resp /: resp_hist
   THEN
     resp_buf1 := resp_buf1 \/ {resp}
     || resp_hist := resp_hist \/ {resp}
     || db:= Update_func(db, Srvc(req))
     || req_buf2:= req_buf2 - {req}
   END;

 Convey_Response=       /* Middleware Operation */
   ANY resp WHERE
       resp : RESPONSE
       & resp : resp_buf1
   THEN
       resp_buf2 := resp_buf2  \/ {resp}
       || resp_buf1 := resp_buf1 -   {resp}
   END;

 GetResponse =        /* Client Operation */
   ANY resp WHERE
       resp : RESPONSE
       & resp : resp_buf2
   THEN
    resp_buf2 :=  resp_buf2- {resp}
   END
END
```

B.5 Specification of The Distributed Processing pattern

```
MODEL
   Distributed_Proc_Spec

/* Main Server interaction with secondary servers Model */
/* In The following model we have multiple user and
multiple secondary servers */



SETS
    REQUEST; RESPONSE; REC_ID;
    USERS; SERVICES; SERVERS; SRVC_RESP
```

```
CONSTANTS
    Limit,
    ReqID, User, Srvc,
/* REQUEST ==
    ReqID : REC_ID
    User  : USERS
    Srvc  : SERVICES
*/

    RespID, Provider, Srvc_resp
/* RESPONSE ==
    RespID  :   REC_ID
    Provider  :     SERVERS
    Srvc_resp :     SRVC_RESP
*/

PROPERTIES
    Limit = 3          &
    /* REQUEST Record Definition */
    ReqID : REQUEST --> REC_ID  &
    User    : REQUEST --> USERS &
    Srvc    : REQUEST --> SERVICES  &
    /* RESPONSE  Record Definition */
    RespID : RESPONSE --> REC_ID    &
    Provider : RESPONSE --> SERVERS &
    Srvc_resp : RESPONSE --> SRVC_RESP

VARIABLES
    req_hist, serv_req, pending_reqs,
     resp_hist, serv_resp, completed_reqs,
    final_resp

INVARIANT
    req_hist : POW(REQUEST)         &
    serv_req : SERVERS <-> REQUEST      &
    ran(serv_req) <: req_hist           &
    pending_reqs : POW(REQUEST)         &
    resp_hist : POW(RESPONSE)           &
    serv_resp : POW(RESPONSE)           &
    serv_resp <:  resp_hist             &
    completed_reqs : POW(REQUEST)       &
    final_resp : POW(RESPONSE)

INITIALISATION
    req_hist := {}          ||
    serv_req := {}          ||
    pending_reqs := {}      ||
    resp_hist := {}     ||
    serv_resp := {}         ||
    completed_reqs := {}        ||
    final_resp := {}

OPERATIONS

 Ask_for_Service =      /* Main Server Operation */
   ANY req WHERE
       req : REQUEST
       & req /: req_hist
   THEN
     serv_req := serv_req \/ SERVERS * {req}
     || req_hist := req_hist \/ {req}
     || pending_reqs := pending_reqs \/ {req}
   END;

Transmit_Req_For_Service =
    skip;

Provide_Service =        /*secondary Server Operation */
  ANY serv, req , srvc_resp, resp WHERE
      serv : SERVERS
      & req : REQUEST
      & srvc_resp : SRVC_RESP
      & resp : RESPONSE
      & resp  /: resp_hist
      & (serv |-> req) : serv_req
      & RespID(resp)= ReqID(req)
      & Provider(resp)= serv
      & Srvc_resp(resp)=  srvc_resp
  THEN
    serv_resp := serv_resp \/ {resp}
    || resp_hist := resp_hist \/ {resp}
    || serv_req := serv_req - {serv |-> req}
  END;

Transmit_Service_Resp =
    skip;

Complete_Req =       /* Main Server Operation */
  ANY req, resp WHERE
      req : REQUEST
      & req : pending_reqs
      & resp <: serv_resp /\ {rs | rs : RESPONSE
                          & RespID(rs)= ReqID(req)}
      & card(resp) >= Limit
  THEN
    final_resp := final_resp \/ resp
    || completed_reqs := completed_reqs \/ {req}
    || pending_reqs := pending_reqs - {req}
  END
END
```

B.6 Refinement of The Distributed Processing pattern

```
REFINEMENT
   Distributed_Proc_Ref1

REFINES
   Distributed_Proc_Spec

VARIABLES
    req_hist,
    serv_req1, pending_reqs, serv_req2,
    resp_hist1, serv_resp1, serv_resp2,
    completed_reqs, final_resp

INVARIANT
    serv_req1 : SERVERS <-> REQUEST     &
    ran(serv_req1) <: req_hist      &
    serv_req2 : SERVERS <-> REQUEST     &
    ran(serv_req2) <: req_hist      &
    resp_hist1 : POW(RESPONSE)      &
    serv_resp1 : POW(RESPONSE)      &
    serv_resp1 <: resp_hist1            &
    serv_resp2 : POW(RESPONSE)      &
    serv_resp2 <: resp_hist1            &
    serv_req1 /\ serv_req2 = {}     &
    serv_resp1 /\ serv_resp2= {}        &

/* Gluing Invariants */
    serv_req = serv_req1 \/ serv_req2   &
     resp_hist1 =  resp_hist        &
    serv_resp = serv_resp1 \/ serv_resp2



INITIALISATION
    req_hist := {}      ||
    serv_req1 := {}         ||
    pending_reqs := {}  ||
    serv_req2 := {}     ||
    resp_hist1 := {}        ||
    serv_resp1 := {}        ||
    serv_resp2 := {}        ||
    completed_reqs := {}    ||
    final_resp  := {}

OPERATIONS

Ask_for_Service =       /* Main Server Operation */
  ANY req WHERE
      req : REQUEST
      & req /: req_hist
  THEN
    serv_req1 := serv_req1 \/ SERVERS * {req}
    || req_hist := req_hist \/ {req}
    || pending_reqs := pending_reqs \/ {req}
  END;

Transmit_Req_For_Service =
  ANY serv,req WHERE
      serv : SERVERS
      & req : REQUEST
      & (serv |-> req) : serv_req1
  THEN
    serv_req2:= serv_req2 \/ {(serv |-> req)}
    || serv_req1 := serv_req1 -{(serv |-> req)}
  END;


Provide_Service =        /*secondary Server Operation */
  ANY serv, req , srvc_resp, resp WHERE
      serv : SERVERS
      & req : REQUEST
      & srvc_resp : SRVC_RESP
      & resp : RESPONSE
      & resp  /: resp_hist1
      & (serv |-> req) : serv_req2
      & RespID(resp)= ReqID(req)
      & Provider(resp)= serv
      & Srvc_resp(resp)=  srvc_resp
  THEN
    serv_resp1 := serv_resp1 \/ {resp}
    || resp_hist1 := resp_hist1 \/ {resp}
    || serv_req2 := serv_req2 - {(serv |-> req)}
  END;

 Transmit_Service_Resp =
  ANY resp WHERE
      resp :  RESPONSE
      & resp : serv_resp1
  THEN
    serv_resp2 := serv_resp2  \/ {resp}
    || serv_resp1 := serv_resp1  - {resp}
  END;

 Complete_Req =       /* Main Server Operation */
   ANY req, resp1 WHERE
       req : REQUEST
       & req : pending_reqs
       & resp1 = serv_resp2 /\ {rs | rs : RESPONSE
                                & RespID(rs)= ReqID(req)}
    /* resp1 = serv_resp2 /\ (RespID~[{ReqID(req)}]) */
       & card(resp1) >= Limit
   THEN
```

```
     final_resp := final_resp \/ resp1
     || completed_reqs := completed_reqs \/ {req}
     || pending_reqs := pending_reqs - {req}
   END
END
```