
*The Department of Computer Science
The University of Auckland
New Zealand*

Form-Oriented Security Analysis of The WrecDirect Web Application

Dong Zhang

July 2006

Supervisors:

Gerald Weber and Clark Thomborson



A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS OF BACHELOR OF SCIENCE (HON-
OURS)

Abstract

One area of web application security is application-level security, which refers to those vulnerabilities that are inherited from an application itself and independent of platforms. Conducting application-level security analysis for each web application individually can be a complicated task. We approach this problem by building a Form-Oriented model. A Form-Oriented model roughly consists of three parts—a dialog model (a.k.a formchart), a layered data model, and dialog specifications. These integrated parts together define the behaviors of a web application. We choose the WrecDirect registration module to apply Form-Oriented modeling. With this Form-Oriented model, we investigate security issues from three aspects: input validation, error handling and concurrent use. We have found some artifacts in Form-Oriented modeling such as bipartite structure and model refinement are of great value to the security analysis.

Acknowledgement

With a deep sense of gratitude, I wish to express my sincere thanks to both of my supervisors, Dr. Gerald Weber and Prof. Clark Thomborson, for their enthusiastic support and guidance throughout the research. What I know today about the process of research, I learned from Dr. Weber and Prof. Thomborson.

My sincere thanks are due to Mr. Barry Dowdeswell and Mr. Stephen McWilliams at AARN ltd., for providing me constant encouragement and support. They have invested a huge amount of their precious time in helping me understand WrecDirect application.

The episode of acknowledgement would not be complete without the mention of Barbara Thomborson. Her timely help during my writing stage is truly unforgettable. I enjoyed my work with the company of the SSG group members namely, Jasvir, Anirban, William, Jinho, Jun, Teng, Lei and Han.

Contents

1	Introduction	1
1.1	Challenges in Web Security	1
1.2	Need for a Web Model	3
1.3	Benefits of Modeling Security	4
1.4	Application Level Structure	6
1.5	Related Work in Web Modeling	8
1.6	Organization	11
2	Form-Oriented Analysis	13
2.1	Form-Oriented Information System Model	13
2.2	An Abstraction of Web Application	16
2.3	Model Decomposition	18
2.4	Model Refinement	19
3	Modeling WrecDirect Registration Module	21
3.1	WrecDirect On-line Tendering System	21
3.2	Registration Module	22
3.3	Formchart	27
3.4	Layered Data Model and Dialog Specifications	28
3.5	User Message Model and Dialog Specifications	30

3.6	Summary	38
4	Security Analysis	39
4.1	Input Validation	39
4.2	Error Handling	42
4.3	Multi-User Registration	44
4.4	Summary	45
5	Conclusion	47
5.1	Contributions to Security Modeling	47
5.2	Limitations of Form-Oriented Security Analysis	48
5.3	Future Work	49

1

Introduction

Security has widely been accepted as one of the major concerns in software developments. Many efforts have been made to address security issues right from design phase through to testing phase of the software development cycle. In the web application world, such requirements are even stronger, because web applications are vulnerable to attacks in the sense that all client/server interactions are across network. For example, unlike in a normal software application, where user inputs are generated locally, the messages received by web application servers are normally from a remote client site. Whether these messages have been spoofed and modified is always worth a question mark. Such problems need to be addressed since they do come up and cause damages in real life.

1.1 Challenges in Web Security

Successful intrusions into web sites may lead to various problems, such as unauthorized modifications to web content or leaks of user information. These possible attacks may adversely affect the day-to-day running of most on-line services, and they are even more harmful when it comes to web sites of critical tasks. E-commerce web sites are of this sort.

An article entitled *Security Hole Threatens British E-tailers* was published on 25th January, 2001 in a British newspaper. In this article, a “glitch” that may affect 40 percent of UK’s e-commerce sites was found. [1] By exploiting this “glitch”, the journalist managed to buy goods for less than intended prices. As the author rightly claimed this process “does not require any particular technical skill”, what was involved in the attacks was merely changing the downloaded HTML pages and loading them back into browsers. This case is not unique. Internet Security System (ISS) has listed 10 shopping cart applications that were and/or are still suffering from similar vulnerabilities [2].

Ranging from the above simple attack to the most sophisticated hacking plan, more and more vulnerabilities in our cyberspace are disclosed every day. According to Bhasin [3], seven different types of threats were identified. Each of these seven types contains a group of techniques that can be used to compromise a variety of targets. Following up the attacks, researcher have been able to summarize common security problems shared by most web platforms [4].

Despite the vast range of vulnerabilities that reside on different levels of a system, the focus of this dissertation is on so-called application-level web security. Application-level web security, as defined by Scott and Sharp, refers to “vulnerabilities inherent in the code of a web-application itself (irrespective of the technology in which it is implemented or the security of the web-server/back-end database on which it is built).” [5] The fact that this type of defects exist in business logic rather than the underlying technology, makes the process of identifying such vulnerabilities varies from application to application. Therefore, a specific investigation into a particular web application implementation is required to carry out application-level security analysis.

On the other hand, it is not hard to see application-level web security holes are widely recognized and damaging. In the *Top Ten Most Critical Web Application Security Vulnerabilities* published by OWASP community [6], seven of the ten vulnerabilities can be application related. For example, the top one “unvalidated input” is a potential security hole created by careless input validation. A possible consequence, as we have seen at the beginning, can result in unexpected loss in e-business transactions. In addition, a

SQL-injection that exploits this problem may expose confidential information.

Besides identifying security vulnerabilities, another aspect of web security is auditing. In a typical web application, clients' behaviors and data are recorded in a log file, which can be used for security analysis or recovery among other uses. If an attack or even an attempt to attack is detected and recorded, we must record it in an understandable way. In security auditing, a full understanding of the problem is necessary in order to close security holes that may lead to similar attacks in the future. The difficulties arise with the fact that more and more web applications tend to use dynamic content nowadays. In these web applications, components on a page are generated on the fly based on server states. So in order to render a transaction that happened some time ago, the server state at that point of time needs to be available to determine the action taken by the client. Since both server state and server actions are domain specific, web application auditing obviously involves application-level issues.

1.2 Need for a Web Model

Current industrial web application developments are still in an ad-hoc fashion. In the design phase, designers and developers prototype web pages that they have in mind. A brainstorm-like diagram (a.k.a page diagram) is constructed to show web pages and their outlook. Such a diagram is neither formal nor precise. After this, arrows are added to link up pages to present navigation paths. Because some old web applications mainly contain static HTML pages, this approach can be considered as a fast and effective way to present a design. Functions were merely jumping between web pages to display HTML content. Quality assurance, in this case, is an unchallenging task. All it has to do is to guarantee the use of well-formed HTML syntax.

Problems came along, though, when dealing with dynamic content and data submission. In dynamic web content scenario, a server action is invoked to generate replies. This action may depend the state of the server and/or the data received from the client. For example, in an on-line shopping application, when a customer requires the price of a

certain product, a product ID is passed on to server where both the product ID and client identity are taken into account to decide a discount rate and then the final price. This transaction obviously can not be represented easily by a simple arrow in the prototype.

Another key feature that should be included in a design is the validation of messages that are passed back and forth. The rationale for doing so is that clients' inputs must be consistent with other components of the system to make themselves understood. In the above on-line shopping example, a product ID may have to be in a certain format to match up a record in the product database. If the server starts processing the message without a validation step, arbitrary inputs are allowed into queries, which may lead to database exceptions, in the absence of an error handling mechanism.

Due to the lack of formalism and intention to detail, traditional approach generally resulted in web applications of poor quality. The rise of scripting technology makes any non-trivial applications have to be able to handle dynamic content. Current modeling techniques, as argued before, are mainly focusing on static web content. So there is a need for a state-of-the-art modeling technique in our on-line world.

1.3 Benefits of Modeling Security

From the previous discussion, we may conclude that a modeling technology for web application is desirable and necessary to achieve efficient development. The outcome of such a modeling process must be definitive, free-of-ambiguity as well as complete. More specifically, a model should be able to “display significant features and characteristics of the system, which one wishes to study, predict, modify, or control.” [7] On the other hand, having an application-level model at the early stage of web development does not only benefit development process [8], but also addresses some other important issues, such as reliability and security.

A simulation model, as defined by Kellner [7], is a computerized model that can represent the dynamic behaviors of a complex system. In the world of web application, such a model can be handy. The difficulties on application-level web modeling is the web

content is dynamic and domain specific. Security issues arise when the dynamic behaviors go out of control. A breach of confidentiality is a good example. Imagine a web page that can display sensitive information to a particular user but not to any others, and this piece of information is dynamic based on the client's inputs and identity. In such a scenario, the confidentiality and the risk of losing it can not be studied in any model which does not capture the semantics of client identity. Meanwhile, web applications have been found of a state-machine nature. The behaviors of a model are bounded by strict rules. Once a user-system interaction is known, a prediction of its consequence is possible even before examining the implementation code.

Application-level security concerns the vulnerabilities inherited from the application itself and irrelevant to the underlying technology [9], which means the security analysis done on one application is possibly applicable to others. The model is platform and technology independent, so security holes identified on this level may also appear in an implementation based on a different platform. For example, in the reported price modification case at the beginning of this dissertation, the very same failure occurred in many different e-commerce web sites. Our hope is that with the assistance of a model, vulnerabilities can be identified easily and have educational meanings with respect to the system we have modeled but other similar systems.

There are roughly two opposite directions in building a model, namely forward and backward engineering. In forward engineering, a model is built at first based on requirements, then issues like functionality and security get investigated according to the model. After several cycles of refinement, the model becomes the master blueprint for implementation and deployment. In contrast, the backward approach, also known as reverse engineering, is to abstract a model out of an existing implementation, and then analysis is done over it. Although the forward direction is generally preferred in development, for sake of minimizing rework effort, both approaches are valuable from security point of view. If a model is developed before an implementation, we have a base to do our security analysis at a very early stage. The advantages follow from the natural logic that the earlier security holes are discovered, the better they can be avoided. On the other hand, once an

implementation is done, it is also worth generating a corresponding security model, which can be used for vulnerability analysis to give confidence in the product's reliability (ie. assurance). In comparison to a post-implementation blind quality-assurance test, where arbitrary testing inputs are fed into the application, the abstraction can be expected to lead us to potential defects more easily and rapidly.

1.4 Application Level Structure

In a web environment, security can be compromised at different levels. In an article named *Internet security: firewalls and beyond* [10], investigations into three layers of the TCP/IP protocol stack were carried out. Proposals have been made to build a firewall that can protect the internet, transportation and application layer better. Surely if each protocol layer can be successfully guarded, fewer attacks will happen. But in real life, how much damage a malicious action can cause depends on the designs at all levels. Take web spoofing for example. The research presented by Edward et al. [11] has shown this traditional attacking method, although concentrating on stealing messages at protocol level, is closely related to the application level logic. In their study, a form submitted by a client can easily become a target. Possibilities were discussed, such as if messages are routed through a middle machine, attackers can capture the form content and breach client's privacy. Furthermore, if the form content is carefully edited before being passed on to the real server, the consequence may be server ends up in an unexpected state after accepting this message, given that the client is trusted and spoofing has gone undetected. It is fairly clear in the above situation, if a leak of confidential information is unavoidable, the damage to the server can be eliminated or limited by putting some constraints on incoming messages.

On the application level, a web system consist of two sides, the client and the server. Distinguished from other software systems, a web application is featured by request/response style transactions between these two sides. In a typical dynamic web application, a client would firstly receive a page containing forms where inputs can be entered. After

a client submits a completed form to the server, it may take some actions to change its own state and possibly generate new pages as replies to the client for the next cycle. In the book “Form-Oriented Analysis” [12], such a transaction is abstracted as a single *state transition function*:

$$stateTransitionFunction : message \times state \longrightarrow state \times listOf(message)$$

From the above formulation, a web transaction can be represented by only three components, client forms, server actions, and the messages being passed in between. Client forms are designed and generated by the web server; server actions happen on the sever side but may be dependent on incoming messages; and messages are created by either side, depending on which direction they are traveling at. Since all three key components can be tracked down to either client or server, we can take our abstraction further into two groups: client pages and server actions. Client pages are forms displayed on the client site and later submitted to a web server; server actions are transition functions executed on the web server. Based on this idea, messages naturally come in as directed links from client pages to server actions (or the other way around). This design was originally proposed by Draheim et al., and notations used by them are illustrated in Figure 1.1. The pattern of these interactions is a bipartite graph [14], in which all client pages belong to one group and server actions belong to the other. As a result, in order to get another web page from one page, a server action must be invoked.

From security aspect, this representation is very interesting, in the sense that serval key features are well captured. Firstly, navigation is represented in detail. All possible web

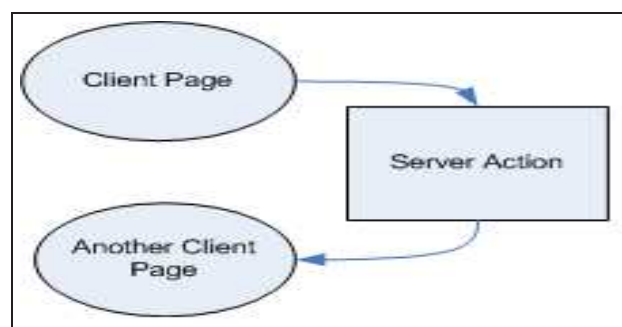


Figure 1.1: Bipartite Structure in Web Applications. After [13].

pages are included, though maybe with different content. A visiting flow is a directed path along the links. Secondly, semantics are precisely defined. Even though client pages are volatile, page changes are recorded in the form of data transmissions with the consequence of server state changes. Thirdly, type information is introduced and control boundaries are drawn out. In this bipartite diagram, every client page is decided by web server. Therefore, we may assume the submitted data must be of a certain format (type), then it is only the content left to be worried about. This mechanism can be considered as a way to create an object-oriented metaphor, where commonalities among objects are abstracted into types.

As argued before, simulation is crucial in web security assurance and application testing. A bipartite structure emphasizes the server actions that are exactly missing in naive page diagram. In a simulation process, we are able to observe the effects on the server side, given any incoming messages. This is doable because we take web applications as a state machine. On the other hand, analyzing server actions can lead us directly to potential defects, instead of searching for them blindly.

1.5 Related Work in Web Modeling

Application-level web security issues have been a research topic for several years, but most investigations have been feature-driven and technology focused. The concept of application level security was raised by Scott and Sharp in 2002 [9], when they also listed three common attacks as: form modification, SQL attacks and Cross-Site Scripting. Meanwhile, they proposed a tool [15] to counter these attacks. Similar attacks were also discussed by Huang et al [16]. In their writing, three types of security violations were defined, such as integrity violation, access rights violation and confidentiality violation; both static and runtime analysis were studied to detect security holes.

Another stream in this research area is focused on a particular domain of web application use. Security issues of an on-line medical system has been addressed by Gritzalis et al. [17]. Twelve threats were raised, which can all be characterized as well known attacks

such as access control, traffic spoofing and forged network address, although the architecture and security requirements of their system are special. In e-commerce world, where transactions take place over internet, five basic requirements were listed [18], which are confidentiality, authentication, data integrity, nonrepudiation and selective application of services. Vertical solutions from network to application level were offered to enhance web transaction security. On the application level, HTTPS and SHTTP standards were discussed as good candidates for securing HTTP transactions.

Orthogonal to domain specific research, some typical modules of web applications have received intensive study. Access control is one good example. It is designed to be the entrance guard of most web systems, and this fact makes it a well-known hot spot often targeted by various attacks. Due to its special role, researchers have been putting effort in creating formalized models. Role based access control [19] was born long ago and now is increasingly used in web applications. In addition, discretionary and mandatory access control models [20] were introduced to put more attention onto information assets.

In generic web application modeling, developments have given rise to a few techniques. One group came out from UML camp, including OOHDMM [21], OO-H [22], and WEBML [23]. Perhaps for the reason that UML is arguably the most widely accepted practice in traditional software modeling, quite a number of contributions can be found in promoting UML extended approaches. For instance, a navigation model was built using OOHDMM [24]; web services were attempted in UML standard [25]. Diversified from the UML-based methodologies, another branch has appeared based on the widely used Object Oriented modeling. In Gellersen and Gaedke' article [26], WebComposition Component Model was used as XML-based description to generate static HTML pages. Other research outcomes include WebGraph [27], a language that only models HTML pages. It is formal enough that even a model checking can be done. From our knowledge, the closest model to the web application structure discussed in previous section, has come from Baresi, Garzotto and Paolini. In their publication [28], the importance of capturing semantic associations was highlighted. They also defined the user operations and system operations in a web environment, which are comparable to our client pages and server

actions, but in their design, system operations were not intensively investigated because they argued those are implementation issues. With their goal of modeling functionality, the static-link type of system operations could suffice.

In our reading of the literature on web and OO modeling, we found very few security analysis. Most of these models lack detailed semantics, which would severely limit the range of security analysis that could be conducted. For example, hypermedia models typically focus on just two aspects of the system, its data structures and its navigation paths. A dynamic web application has another dimension: operation, which represents the state of a session. Although this feature has been included into modular scope modeling (e.g. access control), cases where it is incorporated at generic level are still rarely seen. One example is UMLsec, which has come close to what we will be doing. UMLsec features mostly-used security requirements; it uses associated constraints to evaluate specifications and indicate possible vulnerabilities; it ensures that stated security requirements enforce given security policy and that UML specification satisfies the defined requirements. In UMLsec, formal semantics are represented by a combination of activity diagrams, statecharts, sequence diagrams, static structure diagrams, deployment diagrams, and subsystems [29]. The use of these diagrams was illustrated in several publications in assistance to security analysis. [30] [31] [32]

To summarize, a large proportion of current web modeling research has an emphasis on static content modeling. These techniques are represented by OOHD, OO-H, WEBML. Formalism has been successfully achieved in this area, but an essential limitation has prevented them from being widely used in web security analysis. This limitation is that due to the lack of semantics, dynamic web systems can hardly be incorporated into models. The above argument is particularly true when most latest web applications contain dynamic content. A few techniques such as UMLsec have been suggested to complement them. In UMLsec, implementation levels issues like encryption and physical security are mandatory requirements. In our work with Form-Oriented modeling, we try to offer a pure application level security analysis.

1.6 Organization

Chapter two will give a background introduction of Form-Oriented Analysis. Chapter three will introduce the application itself and model this application in formcharts, data model and dialog specifications. The writing style follows the actual process of modeling, which means the model will get refined several times before being finalized. Chapter four will discuss the Form-Oriented model with respect to security issues. Meanwhile, proposals will be made to improve its security. Chapter five will conclude this research and indicate future work. Readers who are familiar with Form-Oriented modeling may want to start from Chapter Three to get a taste of the actual model.

2

Form-Oriented Analysis

After surveying quite a number of modeling standards, Form-Oriented Analysis was chosen to conduct this research. In this chapter we will give a general introduction to Form-Oriented Analysis and its applications.

2.1 Form-Oriented Information System Model

As Form-Oriented analysis can cope with many distinct types of systems and can be used for different purposes, we need to pick the most appropriate Form-Oriented model to serve our use. Our target application is a dynamic web application. In other words, it is a submit/response style system interface related to a layered data model. In Form-Oriented modeling, *information system model* [12] comes closest to our requirement. An information system model has a bipartite typed state machine and layered data model. For each of these two key components, there are several subelements that are used to detail a complete operating environment. See figure 2.1

Also as known as *formchart*, the dialog model is a user interface model. It includes client pages, server actions and navigation paths. Following the notations used by Draheim and Weber, we depict client pages by ellipses, server actions by rectangles and a

client/server transition by an arrow between an ellipse and a rectangle. Since formchart elements are allowed to carry the same names to depict same model elements, a dialog model can be repainted by different formcharts for readability purpose. As a result, the formcharts that come from one dialog model are in a weak isomorphic relationship with each other. Then the questions come down only to which one is tidier and more readable from presentation perspective. In our modeling, only one formchat is produced. Readability is no problem, for the number of elements is very limited.

Naming conventions were also suggested [12]. In our modeling, most of these conventions are strictly followed. It is worth mentioning that default transition names were used for simplicity. This can be explained in figure 2.2 The notations above the line precisely name a transition without ambiguity. Thus these notations can then be omitted on the formchart diagram to give tidiness. For instance, using the name **P TO A** can point us to the exact arcs by default, as shown in the graph.

Under the dialog model, there is a layered data model, which is made up of user message model, opaque identity model and information model. User message model defines the messages that are transferred between client pages and server actions. These mes-

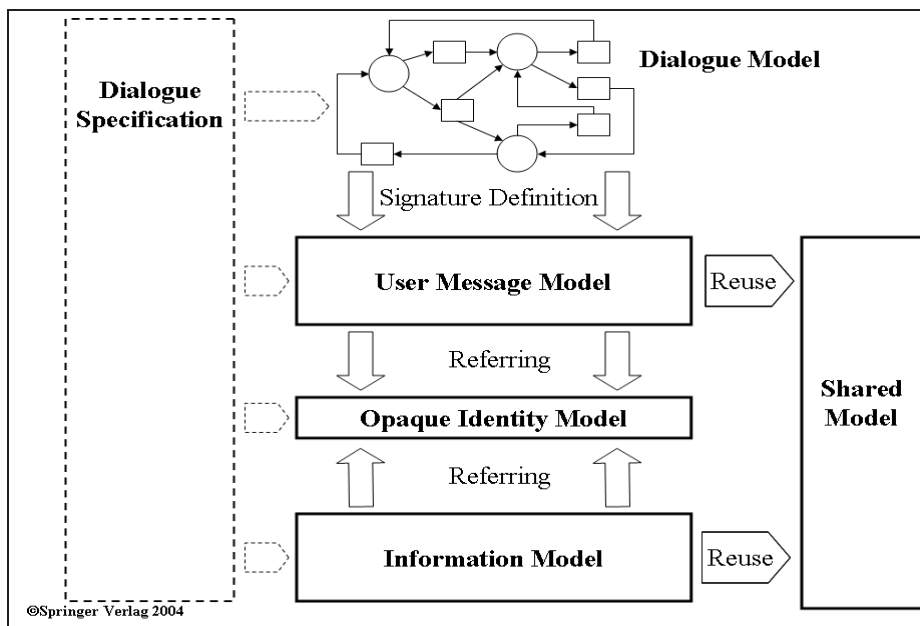


Figure 2.1: The information system model of form-oriented analysis. [12]

sages are structural data that form a hierarchical tree structure. One good example can be web registration forms, where individual fields such as address, contacts and names are entered on a client page. All parameters belong to one user, which can be taken as a top level object. This property can let us define strong typed user/system objects, similar to the concept “class” in object-oriented design. Since message exchanges happen on links, directions at which they are traveling must be specified. We use two key words ”source” and ”target” to indicate whether messages we are using reside at the beginning or the end of an arrow.

In a web environment, server state is crucial in determining how server actions behave. Therefore the information model is introduced to define the updateable portion of server state. This portion comprises persistent data and session data. In our research, we followed the advice in Draheim and Weber’s book to neglect the distinction between these two. Opaque identity model came in as a mapping facility that bridges user message model and information model. This tool, in the simplest words, is to help a client page object communicate with its correspondence in information model. The the third part is called shared model, where types can be defined. These types are supposed to be used by both user message model and information model. Throughout our modeling, a textual notation system was used to build the data model. In the notation, entities are represented by classes with attributes, navigated associations and multiplicities.

Once we have a navigation model and a data model, we can define user/system interactions using dialog specifications. In this specification, user capabilities and server

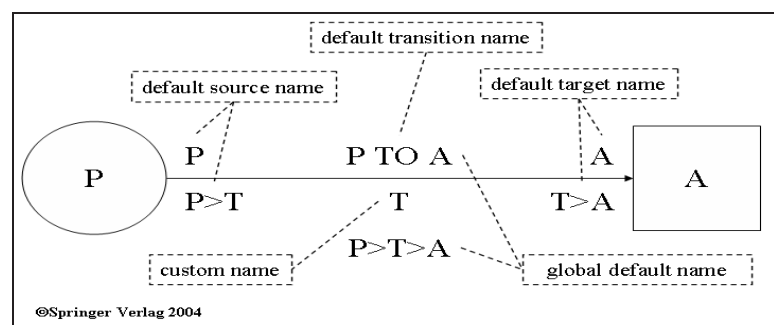


Figure 2.2: Formchart naming conventions . [12]

reactions are written with respect to Dialog Model and data model. Researchers have generalized seven key elements: Flow conditions, Server output specifications, Client input constraint, Enabling conditions, Client output constraints, Server input constraints and Side effect specifications (definitions in [12]). It is not hard to see that some of these specifications are only associated with certain dialog components while other may be applied at global level. Obviously, Flow conditions can be put only on transitions from server action to client page and side effect must belong to a particular server action. This finding is justified in our modeling experience.

Notation wise, a graphical representation is used for dialog model, which contains ellipses, rectangles and directed links in between. In the diagram, a strict bipartite is formed if all ellipses are taken into one group and rectangles are taken into the other. In the data model, textural class-like format is used to capture the system's data model. For dialog specifications, DCL, an extension to Object Constraint Language OCL was chosen.

2.2 An Abstraction of Web Application

The scope of Form-Oriented is clarified in two dimensions. Horizontally, Form-Oriented information system model allow partial modeling, with the support of model decomposition. Vertically, only pure application level information is collected into a model. This section is devoted to explaining how the scope is defined vertically and what bits in a web application actually come into the model.

From user perspective, a web page in real applications depends on many different environments. For example, the look-and-feel of a button may be related to browser and/or displaying device. Characters may be in different languages based on individual machine settings. All these can be considered as low level issues, and thus not included in our model. In fact, what we are really interested in is the navigation and the actual information that is delivered. In our Form-Oriented model, web pages become part of our formchart, where only their names are recorded as IDs. When a form is submitted, regardless whether triggered by a button or a link, a server action is executed in our model, as a transition from a client page to a server action. Similarly, no matter what language

a form content is entered in, our user message model will capture and format it to be understood by the system.

On the server side, activities are described by dialog specifications. In an access control module, a user name and password pair is passed to the server to execute a check. It may be the case that password is encrypted and server must at first run a decryption routine before doing a look-up in database. This detail is ignored in our model, because it is an implementation strategy to improve the security of data transfer. Similar cases can also be found in constraint design. In some technologies, user control constraints are automatically enforced. While in others, developers have to manually conduct constraint check in their code. But the Form-Oriented model again overlook the implementation of constraint checkings.

Since data are depicted in information models, the underlying data storage also becomes irrelevant. In a typical relational database, an entity may have attributes and relations to other entities. In order to successfully render this data table, we can create a class in the information model and then add attributes and multiplicities in associations with other entities. Although this abstraction makes the data independent from the underlying database, we still allow SQL commands to appear in dialog specifications for the convenience of usage. Departing away from real implementations, we then need to define primitive types in Form-Oriented Model. The most common ones include string, number, date etc. These basic units are at the bottom of type hierarchy and used across data models and dialog specifications.

This vertical abstraction with no doubt has the benefit of limiting our focus to application level issues only, but in security analysis, the room is restrained. For example, SQL injection, a well-known application level attack, exploits the lack of forethought in input validation. In our model, we can put a requirements in server input constraint to ensure a client input is valid. But how these requirements are met have gone missing.

2.3 Model Decomposition

The demand for model decomposition comes from the need for partial modeling. In most situations, an entire web application is too big or too complex to be modeled at once. This is particularly true when considering our model as a suite of formcharts, data models and associated specifications. All these components must co-exist to cover compulsory aspects of a system. If our interest is only in one module or one functional unit of the whole web application, it is highly desirable that only this part is modeled, while completeness, consistency and definitiveness are still maintained.

The discussion about Feature-Driven approach has brought into our sight interaction capabilities. An interaction capability “is offered similarly on different occasions, in different contexts , by several client pages consists of a set of typically loosely coupled or even uncoupled model states and transitions” [12]. It means in a web application, there may exist some parts that are independent from the rest of the system. One example would be the registration module. This functional unit can be found in a wide range of web applications like on-line enrollment, forum, payroll system etc. The design and implementation of this feature is not closely attached to its domain-specific host application. Regardless of where it appears, normally several web pages are used to offer an interface where user name/password pairs can be entered, and some database tables on the server side get involved to conduct authentication and authorization. Because of the independence and self-containment, it not difficult to isolate this module out from the other parts of the system. Some publications have already picked it out [33]. In our Form-Oriented modeling research, we restrict our focus with a submodel on this particular functional unit.

As stated before, we allow two nodes in different formcharts to point to one element in a model by using a same name. To decompose a formchart, we would just simply duplicate the nodes at boundaries and display them in splitted submodels with same names. This convention makes the composition and decomposition intuitive. As an example, figure 2.3 has decomposed a page-server-page transition into two subgraphs. Sine the server

action A is at the boundary, it is duplicated in each of the submodels but still point to the same element.

As formcharts are divided into subgraphs, the associated data model and dialog specifications can also be made local. In user message model, client pages and server actions are global signatures, but it doesn't harm if they reappear with their associated nodes in submodels, as long as they have the same definitions. For dialog specifications, most of their constraints are applied on transitions or server actions. Since these two elements have already been made local on formcharts, separating their specifications is a natural process.

2.4 Model Refinement

Three types of Form-Oriented models have been defined, [12], namely: "signature model", "server input declared model/server input safe model" and "multi-user safe model". In practice, we found the later two have been proved of great value from security perspective.

Server input constraints are the tricky bits in Form-Oriented analysis. In server input declared model, input constraints are specified as the conditions on incoming messages. Since the client end is not under system's control, these conditions may be violated by incorrect user inputs. An immediate question then, is to ask what will follow up if server input constraints are not satisfied. Basic error handling mechanism include redirections to a general error page, or simply the session is shut down. No matter what mechanism is used, server input constraints should be replaced. In a server input safe model, these

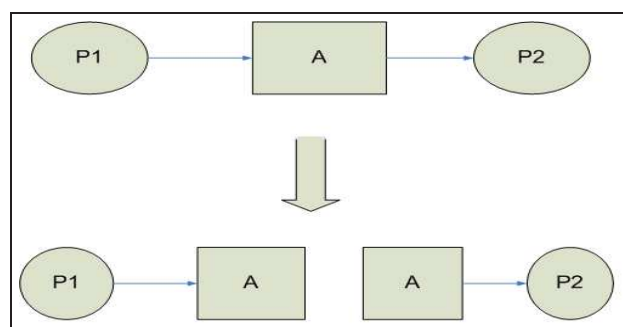


Figure 2.3: Formchart Decomposition

constraints are removed. Instead, error handling will come into play when invalid inputs are received.

Another situation that we sometimes face is, in a multi-user environment, we keep different user views separated in the model as long as they don't affect each other. We achieve this with model decomposition. Each client then has its own instance of server state machine. Since many clients may be acting on the same data model simultaneously, the changes made into data model by one client may in turn adversely affect another ongoing session. In multi-user safe model, transitions and their flow conditions are created to redirect navigation in case of this type of exceptions . This model explicitly targets at the cases where operations are valid at the beginning of a session, but become invalid towards the end, due to impacts from other clients to server states. In our experience, this could happen when a transaction is not atomic. We have found a good example in our model to further explain this point. Please refer to Chapter Five's multi-user registration security analysis.

3

Modeling WrecDirect Registration Module

In this chapter, we present the WrecDirect registration module, and its Form-Oriented information systems model. Our aim here is to illustrate how Form-Oriented modeling is conducted in practice. The discussion starts from the background of WrecDirect application. Then a model is built upon its raw implementation. In the process of building our Form-Oriented information system model, a reverse engineering approach was undertaken in general. The web application that was chosen to be modeled is the registration module of an on-line bidding system called WrecDirect. The model still got updated a few times in parallel with development progress. To the time when this dissertation is written, the implementation is in testing phase.

3.1 WrecDirect On-line Tendering System

WrecDirect is an on-line trading application targeted at damaged vehicle market. Like other tendering systems, it allows participants to list their inventories as well as bid for a stock. The payment transaction is not included in the system, which means after a tender is closed, trading parties need to seek for another channel to arrange actual payment. At

prototype phase, there are basically two types of players in the system: buyers and sellers. Registered sellers are able to list their vehicles, while buyer are able to browse, search and bid for vehicles. Vehicle details are also provided along the way, including its type, photo, statistics and condition. In terms of workflow, a buyer normally experiences browsing, bidding, winning and payment to finish a deal, and a vehicle is archived only after payment is confirmed from seller.

WrecDirect is special in two ways. First, it heavily uses non-web communication channels. For example, user registration is confirmed via email; payments are confirmed back to the system via email and winner is notified via email once a tender is closed. The existence of external communications makes defining our model scope rather important. From security viewpoint, it may not be appropriate to give assurance on channels that are excluded from our model. The second one is its operational environment that has an impact on security measurement. At registration step, all potential clients register their interests, but they don't get access to the system immediately. Instead, a manual process would come in to make decisions about who can become legitimate users. Passwords are generated once a potential client is granted access, and sent over through email. This logic has created a confidence interval in which users are more or less trusted, if we only choose clients from a closed community. The only open gate left to strangers is the initial registration step where people can freely register their interests. For whatever components hidden behind this gate, the likelihood of application level attacks (e.g. flood attack) will presumably reduce. Based on this assumption, we have chosen the registration module to exercise Form-Oriented modeling and security analysis.

3.2 Registration Module

WrecDirect Registration Module consists of five web pages, and a well defined process that a successful registration must go through. These five pages are explained as following, in the sequence they should be visited.

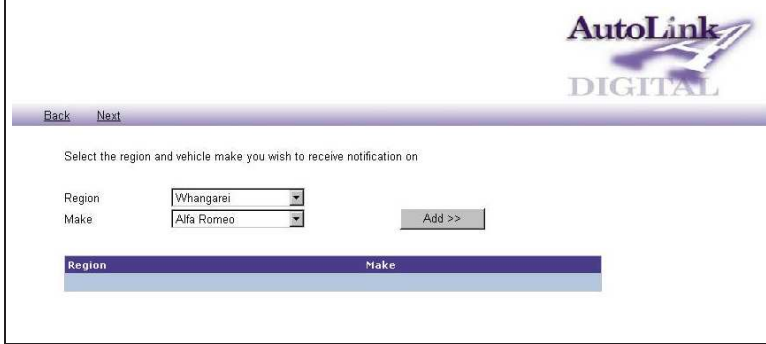
- **WelcomePage:** */WrecDirect/Default.aspx* On the welcome page, a link to regis-

Figure 3.1: WelcomePage

Figure 3.2: BasicInfoPage

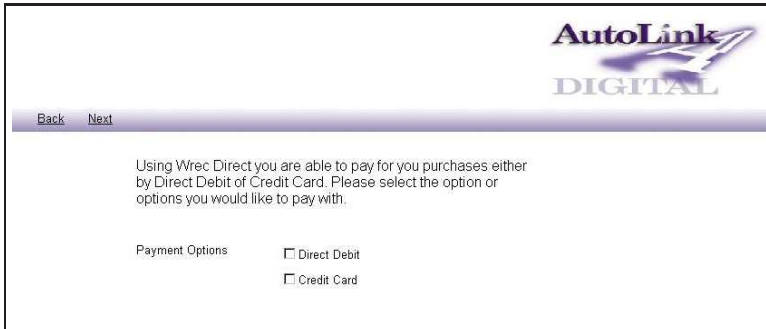
tration is provided. This is the entrance to registration process.

- **BasicInfoPage:** */WrecDirect/Registration.aspx* This the first page where users can input their address, contact and user type.
- **PreferencePage:** */WrecDirect/Registration.aspx* The third page let users select their interested vehicles and regions. They will get notified once a match is found in the tender list.
- **PaymentPage:** */WrecDirect/BuyerNotification.aspx* Users are able to select their preferred payment methods on this page.
- **ConfirmationPage:** */WrecDirect/RegistrationComplete.aspx* A successful registration is confirmed on this page.



The screenshot shows the 'PreferencePage' interface. At the top right is the 'AutoLink DIGITAL' logo. Below the logo is a navigation bar with 'Back' and 'Next' links. The main content area contains the instruction: 'Select the region and vehicle make you wish to receive notification on'. There are two dropdown menus: 'Region' with 'Whangarei' selected and 'Make' with 'Alfa Romeo' selected. An 'Add >>' button is positioned to the right of the 'Make' dropdown. Below these elements is a table with two columns: 'Region' and 'Make'.

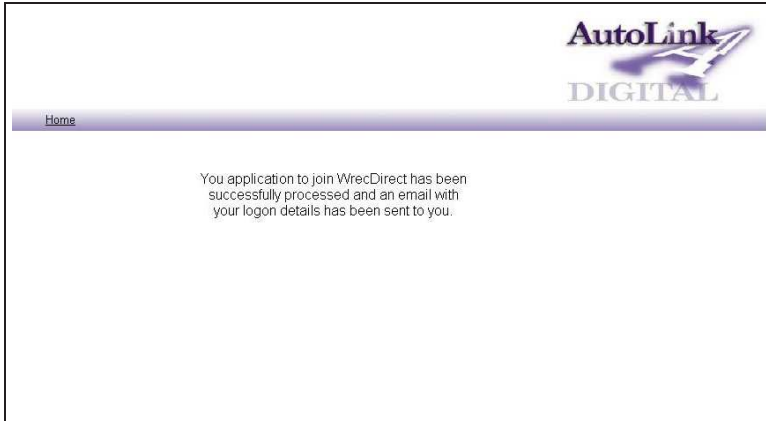
Figure 3.3: PreferencePage



The screenshot shows the 'PaymentPage' interface. At the top right is the 'AutoLink DIGITAL' logo. Below the logo is a navigation bar with 'Back' and 'Next' links. The main content area contains the instruction: 'Using Wrec Direct you are able to pay for you purchases either by Direct Debit of Credit Card. Please select the option or options you would like to pay with.' Below this text are two radio button options: 'Direct Debit' and 'Credit Card'.

Figure 3.4: PaymentPage

The WelcomePage is rendered in figure 3.1. The main window has two fields which registered users can input their username and passwords to login. For our interest, the “Register” link on the top can lead into registration process. Figure 3.2 is the first page of registration process, where information such as Logon name, address, contact and email can be collected. Business rules enforced here are: 1. logon name cannot be empty. 2.



The screenshot shows the 'ConfirmationPage' interface. At the top right is the 'AutoLink DIGITAL' logo. Below the logo is a navigation bar with a 'Home' link. The main content area contains the message: 'You application to join WrecDirect has been successfully processed and an email with your logon details has been sent to you.'

Figure 3.5: ConfirmationPage

name cannot be empty. 3. Email must not be empty. 4. For the reason that a drop down list is used, a registration must be of either buyer or seller type. If there is nothing wrong at this step, the “next” link can let us proceed to the PreferencePage (See figure 3.3). Here, the two drop down lists control the selection of interested regions and makes. Users are also allowed to add preferences by clicking the “Add” button. Finally, a user may choose none, one or both of the payment options.(figure 3.4) The “next” button on this page will try to complete the registration process. If everything goes well, a confirmation message is displayed to the user as in figure 3.5. There is a branch from BasicInfoPage for user type “seller”. If seller is selected from the user type combo box, “next” button will lead to the final ConfirmationPage.

On the server side, operations are managed in parallel with client actions. Starting from BasicInfoPage, user inputs are passed back and forth to simulate a session. For example, the information collected at BasicInfoPage will be passed back to client for PreferencePage (but not displayed) and resubmitted by client with additional fields from this page (in this case would be the preferred region and make). More intuitively, a new user credential is like ball being passed between client and server, with bits and pieces added at each stage, and a complete profile is constructed at final stage. In some of the web pages, user input validation are carried out, and in case of a violation, an error message is displayed. By clicking browsers’ back button, the last page can still be retrieved based on the message passing mechanism described before (last step is recaptured from server side incomplete user information). On most of the pages, there is a “back” link, used to point backwards to the previous page. This is equivalent to the back button on most browsers. On this event, inputs on current page are discarded, and will be populated later on when this page is revisited and inputs are reentered. The entire user information is attempted to be written into database only on clicking the “next” button of PaymentOptionPgae. In other words, if this page or any previous page is closed before finishing, all the inputs before that step will be lost. There is no session storage maintained on server side.

The data structure that is relevant to registration task is revealed by figure 3.6. Every user in our system must be of buyer or seller user type. If the same person acts both as a

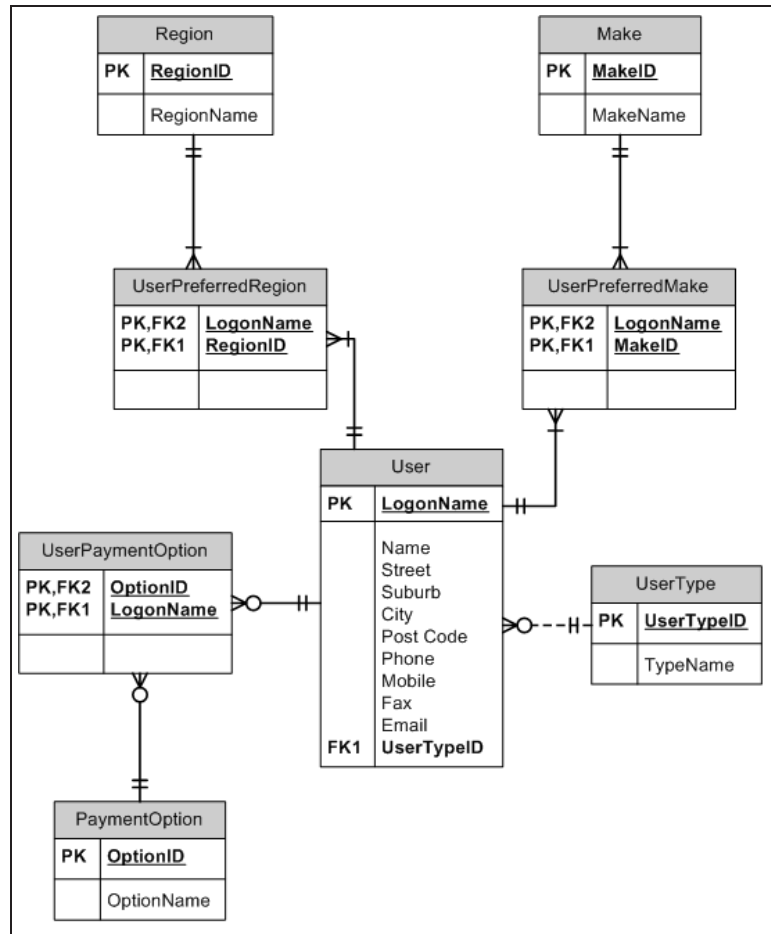


Figure 3.6: The database schema relevant to Registration module

buyer and a seller, two different accounts are then created for this same person and they are considered as totally unrelated users of the system. The User entity has many-to-many relationships with Region, Make and Payment. Therefore, intermediate entities are created to follow ERD standard. Among all these entities, Region, Make and PaymentOption have predefined values and they remain unchanged. When a ConfirmationPage is generated, a new record is inserted into User table, and possibly its associations are created in UserPreferredRegion, UserPreferredMake and UserPaymentOption tables. The data types of the attributes in each of these entities will be defined later in our Form-Oriented data model.

We carefully selected the registration module for many reasons. Firstly, the business rules and data set involved are fairly simple. Business roles can be understood without knowing domain background. Only five basic data entities are used in this module,

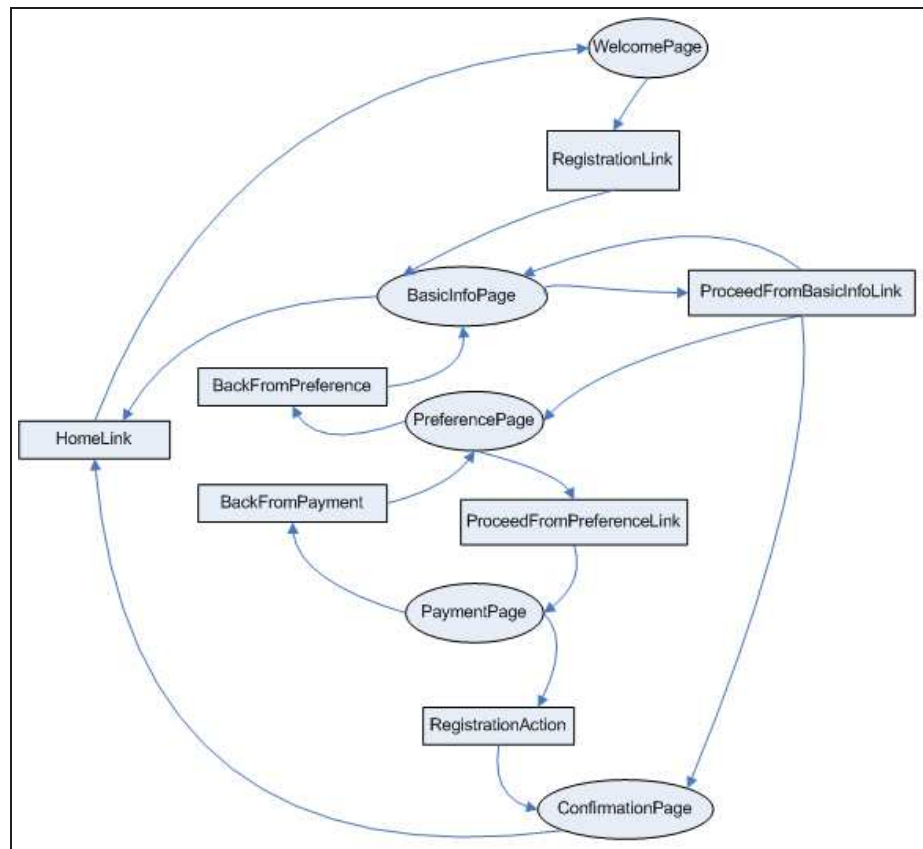


Figure 3.7: Dialog Model (Formchart) of WrectDirect registration module

and they are clearly associated. Secondly, the registration task is not specific to on-line trading system only. Similar functional units can be found in various applications, where registration is open. The study on this particular module may hopefully bring meaningful outcomes to other systems. Although this unit is simple in many aspects, it can still offer enough features in Form-Oriented security analysis. As one of the few hotspots that are exposed to public, the registration component likely becomes the target of many kinds of attacks. On the other hand, based on its implementation, the construction of a model can easily take the advantages of the strength in Form-Oriented technology.

3.3 Formchart

Our first step was to produce dialog model. The entire formchart is built upon web interfaces. (see figure 3.7). Web pages are indicated by ellipse and server actions by

rectangles. Each of the *ProceedFromBasicInfoLink*, *ProceedFromPreferenceLink* and *RegistrationAction* is triggered by the **next** link in its upstream web page. The **back** hyperlinks on *PreferencePage* and *PaymentPage* are represented by *BackFromPreference* and *BackFromPayment* respectively. From the screen shots of *BasicInfoPage* and *ConfirmationPage*, we can find **Home** links, they both point to the initial *WelcomePage* and indicated by a path through *HomeLink* server action.

From the formchart, we can clearly see the registration process is a closed loop with no dummy outgoing arcs left. The only connection with the rest of the system is a branch from initial *WelcomePage*, where a legitimate user can input their username and password to login and go down a separate path. With model decomposition, when the logon component is to be modeled, we can just simply reuse the *WelcomePage* with an arc to *LogonAction*. In this way, the registration model we have constructed can easily fit into a bigger picture when it is required.

3.4 Layered Data Model and Dialog Specifications

Before putting forward the data model, it is worthwhile to restate the most frequently used labeling mechanisms in Form-Oriented standards. Further explanations of the following notions can be found in [12].

ServerAction : Server actions of user message model

ClientPage : Client pages of user message model

Information : Types in information model (drawn from database here)

Signature : Types in the shared model used by both user message and information Model

We start from defining information model. Drawn from the database schema, five top level entities can be found.(see Figure 3.8) This description is incorrect for the reason of missing associations, and implementations details are undesirably included (e.g. *userTypeID* as a

```

Information > User
  userName: String 1..1
  name: String 1..1
  street: String 1..1
  suburb: String 1..1
  city: String 1..1
  postCode: Number 1..1
  phone: Number 1..1
  fax: Number 1..1
  email: String 1..1
  uerTypeID: Number 1..1

Information > UserType
  typeID: Number 1..1
  typeName: String 1..1

Information > Region
  regionID: Number 1..1
  regionName: String 1..1

Information > Make
  makeID: Number 1..1
  makeName: String 1..1

Information > PaymentOption
  optionID: Number 1..1
  optionName: String 1..1

```

Figure 3.8: Information Model From Database Schema

foreign key). However, it is still listed here to demonstrate the process of abstraction. A cleaner and better information model would look like the following.

```

Information > User
  name: String 1..1
  street: String 1..1
  suburb: String 1..1
  city: String 1..1
  postCode: Number 1..1
  phone: Number 1..1
  fax: Number 1..1
  email: String 1..1
  uerType: UserType 1..1
  preference: Preference 1..*
  paymentOption: PaymentOption 1..2

Signature > UserType
  typeID: Number 1..1
  typeName: String 1..1

Signature > Preference
  preferredRegion: String 1..1
  preferredMake: String 1..1

Signature > PaymentOption
  optionID: Number 1..1
  optionName: String 1..1

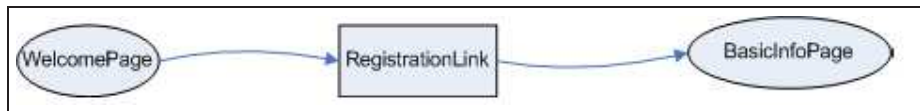
```

Apparently, several changes have been made here. A Signature namespace is added to create a shared model. In this shared model, we define three types, UserType, Preference, and PaymentOption. Theses three types are also valid in user message model, since they are exactly the objects submitted in BasicInfoPage, PreferencePage and PaymentPage respectively. Consequently, we merged the preferredRegion and preferredMake into Preference type to explicitly emphasis that they are submitted as pairs in PreferencePage. The only object left in information model then is user. With multiple associations to

userType, preference and paymentOption, our task in registration becomes solely putting a user object into persistent storage. Therefore, this conceptual simplicity makes the model not only tidy but sensible.

3.5 User Message Model and Dialog Specifications

User message model is described in two groups: ClientPage and ServerAction. The dialog specifications include seven categories that cover all aspects of system behavior. For a summary of these seven elements, please refer to book [12]. In this section, We decompose the formchart into subgraphs and discuss them individually. In this way, readability is optimized and details of the application can be digested.



```
ClientPage > WelcomePage
```

```
  defaultUserName: String 1..1
```

```
  defaultPassword: String 1..1
```

```
SeverAction > RegistrationLink
```

```
WelcomePage > RegistrationLink
```

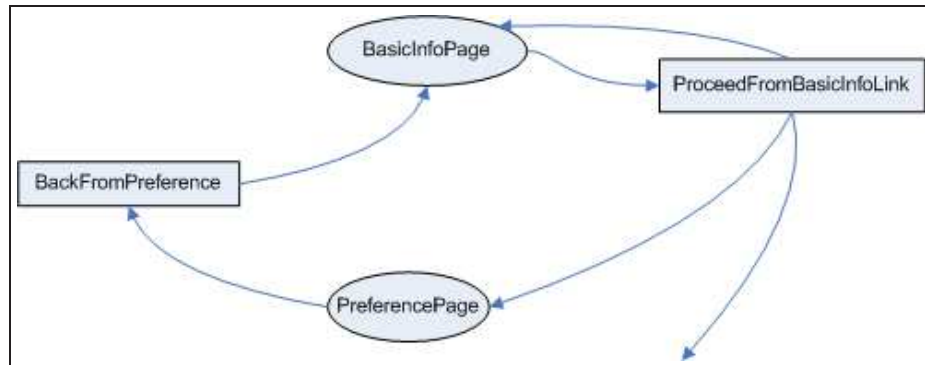
```
RegistrationLink TO BasicInfoPage
```

```
  serverOutput:
```

```
    target.defaultUserInfo.type = "buyer"
```

Warming up with the above submodel, we capture the initial transactions needed to enter the registration process. The WelcomePage is able to take two inputs: user name and password. These two fields are used for logging on, so they do not appear in our dialog specifications. The server action RegistrationLink does not require any processing, so a

simple static link like this is modeled without any specifications. In subsequent modeling such links will be omitted. On the other half of this transaction, is our BasicInfoPage. Based on the implementation the drop down list for user type, a server output constraint must be added to give a default selection when the page is rendered. The definition of BasicInfoPage is deferred to the next step to avoid repetition.



ClientPage > BasicInfoPage

```
defaultUserInfo: BasicInformation 1..1
```

Signature > BasicInformation:

```
logonName: String 1..1
```

```
name: String 1..1
```

```
street: String 1..1
```

```
suburb: String 1..1
```

```
city: String 1..1
```

```
postCode: Number 1..1
```

```
phone: Number 1..1
```

```
mobile: Number 1..1
```

```
fax: Number 1..1
```

```
email: String 1..1
```

```
type: UserType 1..1
```

ServerAction > ProceedFromBasicInfoLink

```
userInfo: BasicInformation 1..1
```

```
serverInput:User->allInstances ->
```

```
select(basicInformation.userName = source.logonName)-> isEmpty
```

BasicInfoPage TO ProceedFromBasicInfoLink

```

clientOutput: not target.userInfo.logonName -> isEmpty
              and not target.userInfo.name -> isEmpty
              and not target.userInfo.email -> isEmpty
              and (target.userInfo.type.typeName="buyer"
                  or target.userInfo.type.typeName="seller")

```

ProceedFromBasicInfoLink TO PreferencePage

```

flow: source.userInfo.type="buyer"
      and User->allInstances ->
          select(basicInformation.userName = source.logonName)
          -> isEmpty
serverOutput: target.carriedUserBasicInfo->isClone(source.userInfo)
              and (target.defaultPreference.make = "Toyota"
                  or target.defaultPreference.region = "Auckland")

```

ProceedFromBasicInfoLink TO ConfirmationPage

```

flow: source.userInfo.type="seller"
sideEffect:
    insert User x
    x.basicInformation.logonName=source.userBasicInfo.logonName
    x.basicInformation.name=source.userBasicInfo.name
    x.basicInformation.street=source.userBasicInfo.street
    x.basicInformation.suburb=source.userBasicInfo.suburb
    x.basicInformation.city=source.userBasicInfo.city
    x.basicInformation.postcode=source.userBasicInfo.postcode
    x.basicInformation.phone=source.userBasicInfo.phone
    x.basicInformation.fax=source.userBasicInfo.fax
    x.basicInformation.email=source.userBasicInfo.email

```

ServerAction > BackFromPreference

```

oldUserInfo: BasicInformation 1..1

```



```
BackFromPreference TO BasicInfoPage
```

```
serverOutput: target.defaultUserInfo->isClone(source.oldUserInfo)
```

For this part, we modeled BasicInfoPage, PreferencePage, ProceedFromBasicInfoLink, BackFromPreference and surrounding transitions. The mechanism used in this implementation requires a second level object created for BasicInfoPage. In BasicInfoPage, only a partial user object (in the absence of payment and Region/Make attributes) is created, but the following steps need to carry this incomplete information along, till the end of registration. Without this second level object, we need to copy each field from BasicInfoPage into PreferencePage, which is a tedious process. Thus was born the “BasicInformation” in our shared model. With the help of “BasicInformation”, both BasicInfoPage and ProceedFromBasicInfoLink server actions are simplified into with only one attribute of BasicInformation type. In the ProceedFromBasicInfoLink, we need to make sure the provided logon name is unique since it serves as an identifier of a user. This checking is represented by a server input constraint. The first-letter capitalized “User” means Information model is searched. On the transitions, from BasicInfoPage to ProceedFromBasicInfoLink, several fields are checked to be non-empty. This is done via client side scripts. And user type selection is limited by drop down list. From this server action to the next web page PreferencePage, whatever entered before is carried on as a BasicInformation object. Of course this object will not get displayed or modified in the PreferencePage. The BasicInfoPage can also be visited through a back link from PreferencePage. This server action is simply reloading the basic user information specified before. In case of a seller registration, the flow condition will lead the navigation to the last page, and seller information is written into database through side effect. Again, the PreferencePage is postponed to next step. With the addition of a BasicInformation type in shared model, we must update related types in the information model to accommodate this change (From Figure 3.8).

```
Information > User
```

```
basicInformation: BasicInformation 1..1
```

```
preference: Preference 1..*
```

```
paymentOption: PaymentOption 1..2
```

The above new definition replaces the older user type, and is used in ProceedFromBasicInfoLink, where logon names are checked for uniqueness. In real environment, the server input constraint may be violated, which means a user inputs an existing logon name. The dialog specification has pointed out this identifier must be unique by did not tell what would happen if client attempts otherwise. With model refinement in next chapter, we replace this server output constraints with two flow conditions as implementation suggested: one to the PreferencePage in case user name is ok, and one back to BasicInfoPage if it is not.

ProceedFromBasicInfoLink TO PreferencePage

```

flow: source.userInfo.type="buyer"
    User->allInstances ->
        select(basicInformation.userName = source.logonName)
        -> isEmpty
    serverOutput: target.carriedUserBasicInfo->isClone(source.userInfo)
        and (target.defaultPreference.make = "Toyota"
        or target.defaultPreference.region = "Auckland")

```

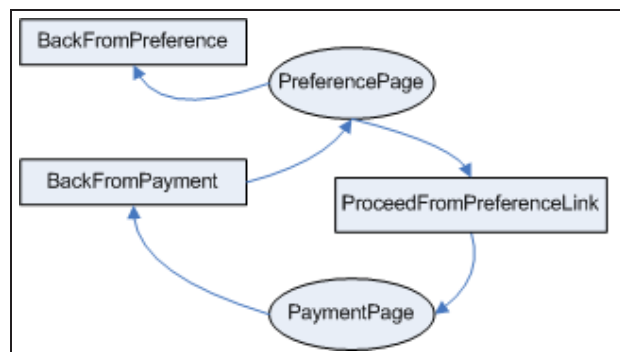
ProceedFromBasicInfoLink TO BasicInfoPage

```

flow: not User->allInstances ->
    select(basicInformation.userName = source.logonName)-> isEmpty
serverOutput: target.defaultUserInfo.type = "buyer"

```

The next part consists of the preference page and its transitions.



ClientPage>PreferencePage

```
    carriedUserBasicInfo: BasicUserInformation 1..1
    defaultPreference: Preference 1..1
```

ServerAction > ProceedFromPreferenceLink

```
    carriedUserBasicInfo: BasicUserInformation 1..1
    preferences: Preference 1..*
```

ClientPage > PaymentPage

```
    carriedUserBasicInfo: BasicUserInformation 1..1
    carriedPreferenceOption: Preference 1..*
    defaultPaymentOption: 0..2
```

PreferencePage TO ProceedFromPreferenceLink

```
    clientOutput: target.preferences.forAll(
        (region = "Auckland"
         or region="Waikato"
         or region = "Manukau")
        and ( make= "Toyota"
            or make = "Ford"
            or make = "BMW"
            )
        )
    and target.carriedUserBasicInfo
    -> isClone(source.carriedUserBasicInfo)
```

PreferencePage TO BackFromPreference

```
    clientOutput: target.oldUserInfo
    -> isClone(source.carriedUserBasicInfo)
```

```
ProceedFromPreferenceLink TO PaymentPage
```

```
serverOutput: target.defaultPayment -> set{0}
and target.carriedUserBasicInfo
-> isClone(source.carriedUserBasicInfo)
and target.carriedPreference
-> isClone(source.Preferences)
```

```
ServerAction > BackFromPayment
```

```
carriedPreference: Preference 1..*
```

This part has nothing special comparing to last step. The preferences will be selected from drop down list, so client output constraints are specified in PreferencePage \mapsto ProceedFromPreferenceLink transition to give a selection base. On the preference page, multiple selections are allowed, so multiplicity was used with the mark *. The basic user information entered in last step is still carried forward to payment page, or backward to basic user information page when the back link is clicked.



```
ServerAction>RegistrationAction
```

```
userBasicInfo: BasicUserInformation 1..1
preference: Preference 1..*
paymentOption: PaymentOption 0..2
```

```
ClientPage>PaymentPage
```

```
carriedUserBasicInfo: BasicUserInformation 1..1
carriedPreference: Preference 1..*
paymentOption: paymentOption 0..2
```

```
ClientPage>ConfirmationPage
```

PaymentPage TO RegistrationAction

```
clientOutput:
    target.userBasicInfo -> isClone(source.carriedUserBasicInfo)
    and target.preference -> isClone(source.carriedPreference)
    and ( target.paymentOption.optionName = "credit card"
        or target.paymentOption.optionName = "direct debit"
    )
```

RegistrationAction TO ConfirmationPage

```
sideEffect:
    insert User x
    x.basicInformation.logonName=source.userBasicInfo.logonName
    x.basicInformation.name=source.userBasicInfo.name
    x.basicInformation.street=source.userBasicInfo.street
    x.basicInformation.suburb=source.userBasicInfo.suburb
    x.basicInformation.city=source.userBasicInfo.city
    x.basicInformation.postcode=source.userBasicInfo.postcode
    x.basicInformation.phone=source.userBasicInfo.phone
    x.basicInformation.fax=source.userBasicInfo.fax
    x.basicInformation.email=source.userBasicInfo.email
    x.preference -> isClone(source.carriedPreference)
    x.paymentOption -> isClone (source.paymentOption)
```

This last part writes all the user information collected in previous steps into database. This database insertion is included in the side effect of transition from RegistrationAction to ConfirmationPage. Implementation details are hidden with the use of deep copying “isClone”. Individual fields are explicitly populated with user attributes because they are the actual attributes of user table in database. However, for creating new associations from a user to his preferred regions and makes and his payment options, we need to insert new records into intermediate tables. This is way too complicated and not included in application level modeling. We want the above side effect pseudo-code to have the following meaning: if the server action RegistrationAction is left via the transition to the page ConfirmationPage, a new information object x of type User is generated. This new

object has the data gathered by RegistrationAction as attribute values.

To wrap up our specifications, it may be better to include static hyper links for completeness reason.

```
ServerAction>HomeLink
```

3.6 Summary

The completed Form-Oriented Information System model is generated for registration module of WrecDirect. The walkthrough has revealed how the modeling went about from initial database rendering to conceptual objects, from server input constraints to refined flow conditions. Although small in size, this module has involved almost all aspects of Form-Oriented technology. On the other hand, implementation is complicated enough to produce fruitful results in security analysis later on. WrecDirect does not have a formal development document coming with it. The presented result is mainly based on oral communications between the modeler and developer.

4

Security Analysis

In the previous chapter, a Form-Oriented information system model was built for WrecDirect registration module. In this chapter, we will discuss some of the security concerns reflected from our model and how they can be possibly resolved.

4.1 Input Validation

The text fields in web pages allow input to hold anything. However, in real applications, business rules enforce constraints on these textual inputs, when a user is expected to enter data of a particular format, for instance, a number, a string of limited length, or an email address. Failures in input validations may leave security holes to various attacks. *SQL injection* [34] is one such example. It has been found that “this injection typically occurs through a web form and associated CGI script that does not perform appropriate input validation” [35]. In such attacks, user information can be extracted from a database, or even more seriously, data tables can be altered.

To fix such vulnerabilities, it is commonly accepted that input must be checked before actually being used. Client side validations can be implemented in many technologies, such as Javascript [36]. In Client side validation, the checking routines are executed locally

by browser before a form is submitted to the web server. Most of these programs validate input formats through techniques like regular expressions. Conducting validations on client side instead of server side has several benefits. First, it moves the heavy workload from server to client, as a server may handle multiple clients at the same time. Second, in case an invalid input is detected, client can get notified immediately, rather than wait for the error message to loop back from web servers. Network traffic is also saved in this situation. Finally, server side validation requires programming checking functions into application code, which can be a tedious task.

Client side validation also has significant downside. The use of this mechanism means developers implicitly put trust onto client softwares. These softwares, such as browsers, are trusted to enforce whatever input validation policies that come with each web page. Unfortunately, this is not always guaranteed. In the case described at the beginning of this writing, only a simple text editor was involved to tamper the constraint policy. As a consequence, text fields are no longer restricted as they are designed to be. To protect against this type of attacks, server side validation must exist. Server side validation means the validation policies are enforced in server processes, which a client has no way to influence. One may argue the problems can of course be avoided by employing sever side checking alone. In this way, all the inputs are transferred to web server and validation is carried out there. Considering the pre-listed benefits of client-side validation, however, we probably want to double check an input on both client and server. With this design, a genuine mistake is detected and handled in the client's browser and server does not have to deal with it. In case of malicious attempt, when an unexpected input can actually come out from client, the server invokes it error handling routines to avoid damages and optionally, this attempt may be logged with client information for security auditing purpose.

In Form-Oriented modeling, whether an error is actually handled by server-side or client-side processing can be considered as too low level from analysis point of view. However, the distinctions are accommodated through using client output constraints and server input constraints. For example, in the BasicInfoPage of client pages, the input for

logon name, name and email must be non-empty, and the use of drop down list implicitly restricts the user type field must be either buyer or seller. These rules appeared as client output constraints. In testing the actual implementation, we captured the HTTP packages and manually modified them to put arbitrary content into some of the restrained fields. We intentionally changed this HTTP POST message to have “userType=5” and left out name field. As result, a record is created in user table with empty name, and a database exception is thrown when a user type association was attempted in intermediate data table. Apparently, in this implementation, we rely on client side checking and give things a window to go wrong. Therefore, in addition to the client side validation, we propose a server side validation as following. (Client side validation is modeled in **BasicInfoPage TO ProceedFromBasicInfoLink**)

```
ServerAction > ProceedFromBasicInfoLink
    userInfo: BasicInformation 1..1
serverInput:User->allInstances ->
    select(basicInformation.userName = source.logonName)-> isEmpty
    and userInfo.logonName->notEmpty
    and userInfo.name->notEmpty
    and userInfo.email->notEmpty
    and (userInfo.userType.name="buyer"
        or userInfo.userType.name="seller"
    )
```

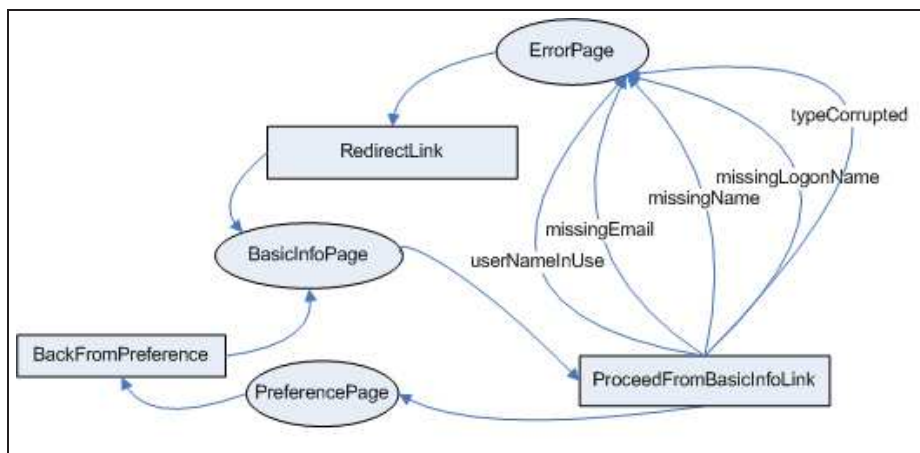
In practice, we suggest double checking input at both client side and server side. Some programming languages have already added facilities to achieve this goal. One way, for example, is to create corresponding server-side validation routine when a client validation is programmed by developers.(ASP 2.0 [37]) Being platform independent, our Form-Oriented modeling is able to present this feature no matter what it is built on. For the WrecDirect registration module, the same practice is suggested for preference page and payment option page.

4.2 Error Handling

In chapter two, we mentioned model refinement and argued server input constraints have to be replaced in later stages by branches from their server states. Failing to do so, may result in meaningless messages getting displayed to clients, depending on implementation platforms.

In testing the implementation, we have found violations to server input constraints lead clients to ASP exception page. For example, in the above DCL, user inputs are checked in server-side code, but how the errors are handled is missing. The implementation goes through each of these inputs, and when an unacceptable field is encountered, an exception is thrown. Since database is not written until the end, one may argue this is not so much a security issue. We believe otherwise, for: A. When a non-technical user sees confusing exception messages, he or she may take unpredictable actions such as trying out different URLs or reattempt the same registration process. This may cause performance penalty to our server. B. Stack trace may reveal sensitive implementation details that can lead to more serious attacks. Therefore, we need to at least hide the exception stack trace.

One obvious and delegant solution is to create error message/page in error handling branch. Based on this idea, we improve our formchart to be the following.



ClientPage>ErrorPage

errorMessage: String 1..1

```
ProceedFromBasicInfoLink > missingLogonName > ErrorPage
    flow: source.userInfo.logonName->isEmpty
    serverOutput:target.errorMessage="Missing logon name!"

ProceedFromBasicInfoLink > missingName > ErrorPage
    flow: source.userInfo.name->isEmpty
    serverOutput: target.errorMessage="Missing name!"

ProceedFromBasicInfoLink > missingEmail > ErrorPage
    flow: source.userInfo.email->isEmpty
    serverOutput: target.errorMessage="Missing email!"

ProceedFromBasicInfoLink > typeCorrupted > ErrorPage
    flow: userInfo.userType.name="buyer"
           or userInfo.userType.name="seller"
    serverOutput: target.errorMessage="Data corrupted, maybe malicious attempt!"

ProceedFromBasicInfoLink > missingEmail > ErrorPage
    flow: User->allInstances ->
           select(basicInformation.userName
                 = source.logonName)-> notEmpty
    serverOutput: target.errorMessage="Logon already existed!"

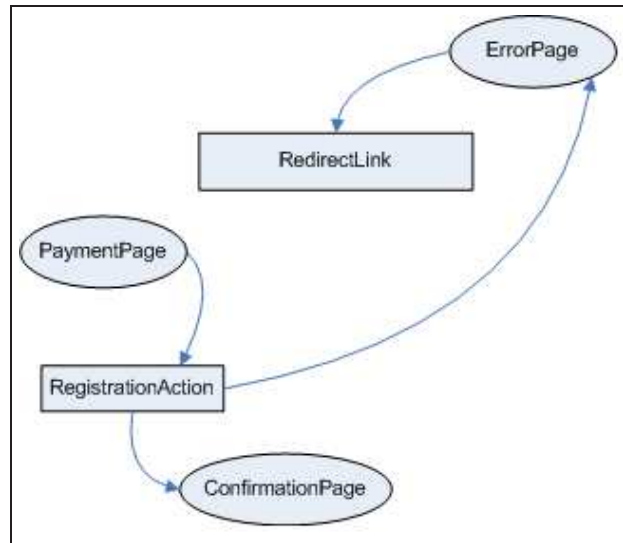
ServerAction > redirectLink
```

We have added five transitions to a newly created error page; each delivers an error message. This replacement has left `ProceedFromBasicInfoLink` with virtually no server input constraints. Whenever the validation fails, an informative error message is displayed to the user on the error page. Similarly, the same design should be applied to wherever there is a chance of error occurrence. Towards the end, we should ideally get a server input safe model without any server input constraints.

4.3 Multi-User Registration

One special and important fact that we have not touched much yet is that WrecDirect registration module operates in a multi-user environment. It is possible that more than one client is trying to go through the registration process at the same time. Therefore, we have to consider many clients acting on the same data model. Each client has its own instance of the server state machine, and these instances only interact via data model. These interactions are presented in our model as the registration action's side effect, where user data get written into database. Throughout the registration process, constraints imposed on data are pretty much unrelated between clients, except for the logon name. The logon name serves as a unique identifier for a user account. The creation of a logon name in database may invalidate another ongoing session. Imagine two users are trying to register at the same time. User A has managed to specify a logon name at BasicInfoPage and then get to PreferencePage. User B initializes its own registration process at this point of time and coincidentally specified the same logon name as user A did. Assume after a few steps, user B managed to finish the process before user A. Therefore, this logon name is eventually held by user B, and persisted into database. The problem is that user A is completely unaware of the existence of user B and the logon name clash. When user A reaches the actual registration action at last page, the logon name is not rechecked and will be inserted into user tabel. In practice, we have found in such an incident, a SQL exception would be thrown out.

The above demonstration is fairly easy to see from our model, because the logon name is checked for uniqueness only once at the BasicInfoPage. However, if two concurrent registrations with same logon name are attempted, the system should not run into an undefined state. Thus, we improve our model into multi-user safe model. Multi-user safe model requires new transitions from occurring multi-user exceptions. We direct this transition to the standard error page with appropriate message.



RegistrationAction TO ErrorPage

```

flow: User->allInstances ->
    select(basicInformation.userName = source.logonName)-> notEmpty
    serverOutput: target.errorMessage="Logon already existed!"
  
```

Here, we recheck the logon name at RegistrationAction, and redirect to error page in case the logon name already exists in user table.

4.4 Summary

In this chapter, we have exercised security analysis on WrecDirect registration Form-Oriented model from three aspects. We recognize server side validations are necessary. In our model, we ensure these validations are not forgotten, by matching up client output constraints and server input constraints. Even with server input constraints in place, we still want a proper, meaningful and safe error page, in case these constraints are violated. So we have refined our model into server input safe model. Since the registration process takes several steps and allows concurrent sessions, multi-user management is a security concern. We have made recommendations to refine our model into multi-user safe model.

5

Conclusion

The WrecDirect registration module is a small application unit easily described in our model, yet rich and complicated enough to raise security questions. We believe this functional unit gives a good coverage in representing today’s web applications. The registration process concerns input validation, page navigation, database access, and concurrent usage. Every one of these elements can become a target in an attack. The Form-Oriented model that we have delivered successfully captured these characteristics from an abstract viewpoint. Our work was greatly benefited by the new artifacts introduced by Form-Oriented analysis, namely formcharts, data models, and dialog specifications.

5.1 Contributions to Security Modeling

The aim of this project was to produce a working model for WrecDirect registration module on which to perform security analysis. We have found certain features in Form-Oriented modeling are of tremendous help.

Bipartite Structure. As we all understand, web applications are different from traditional softwares in the sense that they have two relatively decoupled components — the client and the server. Normally, servers can expect client behaviors but do not have

control over them. This fact makes client ends somewhat untrustworthy. In the bipartite structure, we clearly draw a line that separates out client pages. Then we can carefully examine all the client-server transitions and add all necessary constraints on the server side to repeat client side validations.

Architecture. Previously, researchers have tried to model GUI, hyper text, navigation and data individually. Imagine we have our user interface in page diagrams, hyper text in WebML, navigation in UML and data model in ER diagram. Establishing references and maintaining consistency between these models is doubtlessly a tough job, let alone doing a thorough analysis. In Form-Oriented modeling, we put all the elements of a web system into one architecture. They exist under one global name space and seamlessly integrate with each other.

Model Decomposition. Model decomposition has helped us twice in our modeling. Firstly, it allows us to separate the registration module from the rest of the system without damaging local completeness. Because the motivation was to test the safety of registration module, other parts of the system are more or less irrelevant. Having a small complete model has enabled us to focus on limited number of objects. For the second time, we created submodels each containing one or two major server actions. In this way, discussions became compact and informative.

Application Level. The Form-Oriented modeling and analysis work on an abstraction level that is decoupled from underlying implementation. Our models can be viewed as being executed on a virtual business machine. As a result, the vulnerabilities found in our models are related to the business logic of the system and are platform independent.

5.2 Limitations of Form-Oriented Security Analysis

As one can see, the security analysis done on the Form-Oriented model heavily depends on the application itself and its underlying data structure. Unlike some of the other work done in the same area [38], we do not claim a universal solution that can automatically be deployed on all applications, for the reason that most features are specific to their

host applications. Admittedly, the registration module was chosen to represent some commonalities, but the values of these commonalities can not be directly transferred. Therefore, given the task of doing Form-Oriented security analysis on distinct systems, one may have to model them one by one, although it is the same principles that should be followed.

The Form-Oriented modeling systematically addresses security issues mainly through model refinement. We do not explicitly work against particular attacking models, but defend them by improving our own system. We can of course avoid a large number of typical application design defects by running through the model, but it is no easy task to keep our application away from new dangers. For example, the Form-Oriented model refinement tackles two type of problems — missing error handling and concurrent use. With server input constraint, we enforce server side validation. Despite these well-known attacks, there is still risk of more customized and sophisticated attacks, that the model may not easily see. Therefore, Form-Oriented analysis achieves security confidence only to a certain level.

Security sometimes should be considered in the big picture. Once a single point of defense is compromised by an intrusion, the damage may propagate to a large area. Application level modeling apparently can limit our focus to certain types of high-level problems, but when talking about vulnerabilities, we must put them in context with other layers. For example, if the traffic is spoofed, or a cookie is stolen [19], what impact will it have on our application? How much damage can it cause? To include these considerations, we need to offer a way that our Form-Oriented model can be used to interact with other layers. How we can do this to let our application level model work with other levels on security analysis remains a question to be answered.

5.3 Future Work

There are tools in Form-Oriented modeling that can help development and analysis. **JSPick** [39] is a reverse engineering tool that can automatically recover web signatures

and form types from Java Server Pages. **Angie** [40] is a forward engineering tool for the type-safe specification of web presentation layers and the subsequent generation of an executable interface prototype. We note model refinement is a pretty well-defined process. Including model refinement into these tools would benefit security analyzers.

Furthermore, since the use of patterns in object-oriented programming has been widely accepted as a good practice, an introduction of patterns into Form-Oriented model may be of value as well. We support this argument because Form-Oriented model is strictly defined and does not contain ambiguity. Therefore, if patterns are helpful in object-oriented design, they may be also useful in Form-Oriented models. The client-server double field validations can be considered as one good pattern. Whether object-oriented patterns such as singleton can be migrated into Form-Oriented modeling demands serious investigation.

Bibliography

- [1] S. Goodley, *Security hole threatens british e-tailers*, The Daily Telegraph Newspaper (UK). 25th January (2001).
- [2] I. S. Systems, *Form tampering vulnerabilities in several web-based shopping cart applications*, 2000, At <http://xforce.iss.net/xforce/alerts/id/advice42.php>.
- [3] S. Bhasin, *Web Security Basics*, Premier Press, 2003.
- [4] A. Rubin and D. Geer Jr, *A survey of Web security*, Computer **31**, 34 (1998).
- [5] D. Scott and R. Sharp, *Specifying and enforcing application-level web security policies*, IEEE Transactions on Knowledge and Data Engineering **15**, 771 (2003).
- [6] O. T. T. Project, *The Top Ten Most Critical Web Application Security Vulnerabilities*, 2004.
- [7] D. M. R. Marc I. Kellner, Raymond J. Madachy, *Software Process Modeling and Simulation: Why? What? How?*, Journal of Systems and Software **46** (1999).
- [8] L. Bouillon, J. Vanderdonckt, and J. Eisenstein, *Model-Based Approaches to Reengineering Web Pages*, International Workshop on Task Model and Diagrams for user interface design, TAMODIA (2002).
- [9] D. Scott and R. Sharp, *Abstracting application-level web security*, Proceedings of the eleventh international conference on World Wide Web , 396 (2002).

-
- [10] R. Oppliger, *Internet security: firewalls and beyond*, Communications of the ACM **40**, 92 (1997).
- [11] E. Felten, D. Balfanz, D. Dean, and D. Wallach, *Web spoofing: An Internet con game*, Software World **28**, 6 (1997).
- [12] D. Draheim and G. Weber, *Form-Oriented Analysis-A New Methodology to Model Form-Based Applications*, Springer Verlag, 2004.
- [13] D. Draheim and G. Weber, *Modeling Submit/Response Style Systems with Form Charts and Dialogue Constraints*, Proceedings of the Workshop on Human Computer Interface for Semantic Web and Web Applications, LNCS **2889** (2003).
- [14] D. Draheim and G. Weber, *Modelling form-based interfaces with bipartite state machines*, Interacting with Computers **17**, 207 (2005).
- [15] D. Scott and R. Sharp, *Developing secure Web applications*, Internet Computing, IEEE **6**, 38 (2002).
- [16] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, and S.-Y. Kuo, *Securing web application code by static analysis and runtime protection*, in *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 40–52, New York, NY, USA, 2004, ACM Press.
- [17] S. GRITZALIS, *Developing secure Web-based medical applications*, Medical Informatics and the Internet in Medicine **24**, 75 (1999).
- [18] A. Bhimani, *Securing the commercial Internet*, Communications of the ACM **39**, 29 (1996).
- [19] J. Park, R. Sandhu, and G. Ahn, *Role-based access control on the web*, ACM Transactions on Information and System Security **4**, 37 (2001).
- [20] J. Joshi, W. Aref, A. Ghafoor, and E. Spafford, *Security models for web-based applications*, Communications of the ACM **44**, 38 (2001).

- [21] D. Schwabe, G. Rossi, and S. Barbosa, *Systematic hypermedia application design with OOADM*, Proceedings of the the seventh ACM conference on Hypertext , 116 (1996).
- [22] J. Gomez, C. Cachero, and O. Pastor, *Extending a Conceptual Modelling Approach to Web Application Design*, CAiSE00 , 79 (2000).
- [23] P. F. Stefano Ceri and A. Bongio, *Web modeling language (WebML): a modeling language for designing web sites*, in *9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netourking*, pages 137–157, North-Holland Publishing Co., 2000.
- [24] D. Schwabe, L. Esmeraldo, G. Rossi, and F. Lyardet, *Engineering Web applications for reuse*, Multimedia, IEEE **8**, 20 (2001).
- [25] R. Gronmo, D. Skogan, I. Solheim, and J. Oldevik, *Model-driven Web services development*, e-Technology, e-Commerce and e-Service, 2004. EEE'04. 2004 IEEE International Conference on , 42 (2004).
- [26] H. Gellersen and M. Gaedke, *Object-oriented Web application development*, Internet Computing, IEEE **3**, 60 (1999).
- [27] L. de Alfaro, *Model checking the world wide web*, Computer Aided Verification **2102**, 337 (2001).
- [28] L. Baresi, F. Garzotto, and P. Paolini, *From Web Sites to Web Applications: New Issues for Conceptual Modeling*, ER Workshops , 89 (2000).
- [29] J. Jurjens, *UMLsec: Extending UML for secure systems development*, UML , 412 (2002).
- [30] J. Jürjens and P. Shabalin, *Automated Verification of UMLsec Models for Security Requirements*, UML , 412 (2004).

-
- [31] J. Jurjens and S. Houmb, *RISK-DRIVEN DEVELOPMENT OF SECURITY-CRITICAL SYSTEMS USING UMLSEC*.
- [32] J. J. Houmb, S.H., *Developing secure networked Web-based systems using model-based risk assessment and UMLsec*, in *Software Engineering Conference, 2003*, pages 488–497, 2003.
- [33] D. Ferraiolo, J. Barkley, and D. Kuhn, *A role-based access control model and reference implementation within a corporate intranet*, *ACM Transactions on Information and System Security (TISSEC)* **2**, 34 (1999).
- [34] C. VU, *Vulnerability Note VU# 282403*, AdCycle does not adequately validate user input thereby allowing for SQL injection at <http://www.kb.cert.org/vuls/id/282403> (2002).
- [35] S. Boyd and A. Keromytis, *SQLrand: Preventing SQL Injection Attacks*, *Proceedings of the 2nd Applied Cryptography and Network Security (ACNS) Conference*, 292 (2004).
- [36] J. Sinclair, *Revisiting Java technology on the client*, *Develop Works* **3** (2001).
- [37] M. Volodarsky, *Are You Protected? Design and Deploy Secure Web Apps with ASP.NET 2.0 and IIS 6.0*, *MSDN Magazine* (2005).
- [38] D. Scott and R. Sharp, *Specifying and Enforcing Application-Level Web Security Policies*, *IEEE Transactions on Knowledge and Data Engineering* **15**, 771 (2003).
- [39] D. Draheim, E. Fehr, and G. Weber, *JSPick-a server pages design recovery tool*, *Software Maintenance and Reengineering, 2003. Proceedings. Seventh European Conference on*, 230 (2003).
- [40] D. Draheim, C. Lutteroth, and G. Weber, *A Source Code Independent Reverse Engineering Tool for Dynamic Web Sites*, *9th European Conference on Software Maintenance and Reengineering*.