UNIVERSITY OF SOUTHAMPTON

FACULTY OF ENGINEERING AND APPLIED SCIENCE

School of Electronics and Computer Science

Implementation and Validation of Model-Based Multi-threaded Java Applications and Web Services

by

Pengfei Xue

Thesis for the degree of Doctor of Philosophy

October 2008

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND APPLIED SCIENCE

SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

IMPLEMENTATION AND VALIDATION OF MODEL-BASED MULTI-THREADED JAVA APPLICATIONS AND WEB SERVICES

by Pengfei Xue

In the software engineering world, many modelling notations and languages have been developed to aid application development. The technologies, Java and Web services, play an increasingly important role in web applications. However, because of issues of complexity, it is difficult to build multi-threaded Java applications and Web Service applications, and even more difficult to model. Furthermore, it is difficult to reconcile the directly-coded application with the model-based application.

Based on the formal modelling system, RDT, the new work here covers: (*i*) a translator, RDTtoJava, used to automatically convert an RDT model into an executable multi-threaded Java application; (*ii*) the framework for developing an RDT model into a Java synchronous distributed application that is supported by the JAX-RPC Web Services; and, (*iii*) the framework for developing an RDT model into a Java asynchronous distributed application that is supported by the JMS Web services.

Experience was gained by building distributed computing models and client/server models and generation of the application based on such models. This work is helpful for the software developers and software researchers in formal software development.

Table of Contents

List of Figures

Declaration of Authorship

I, Pengfei Xue, declare that the thesis entitled “Implementation and Validation of Model-Based Multi-threaded Java Applications and Web Services” and work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Name: Pengfei Xue

Date: 21/03/2009

Acknowledgements

My main thanks and gratitude go to my supervisor Professor Peter Henderson. He stimulated my research interest in software engineering. He also made many helpful suggestions, and gave me important advice and constant encouragement during my study.

Special thanks should go to my supervisor Dr Robert J Walters. He gave me remarkable help when I was in difficulty. I could not finish my study without his valuable instructions and advice.

Furthermore I am deeply indebted to Eric Cooke for his kindness, friendship and help. I am honoured to show my respects to him.

I also would like to thank Dr Des Watson, the external examiner of my PhD dissertation committee, for the valuable advice he provided in completing this thesis.

Finally I want to thank my parents, Bingzhong Xue and Yuchun Liu, and my brother Pengyu Xue, for their encouragement and support throughout my PhD study in the UK. They are my powerful source of inspiration and energy. A special thought is devoted to my wife Zhiqiang Xu for her never-ending support. It is to them that I dedicate this thesis.

Southampton in Oct 2008

Pengfei Xue

Glossary

BDD	Binary Decision Diagram
EJB	Enterprise JavaBeans
FSP	Finite State Processes
JCAT	Java Coordination And Transactions
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
JPF	Java PathFinder
PRC	Remote Procedure Calling
RMI	Remote Method Invocation
SMV	Symbolic Model Verifier
SCR	Software Cost Reduction
TFG	Trace Flow Graph
WSDL	Web Service Definition Language

1 Introduction

1.1 Models and Formal Modelling

1.1.1 Models

It is difficult to study the real world, because of its complexity. The popular approach to studying the world is to study an object or an issue in its simplest situation, only considering those circumstances relevant to the problem and ignoring irrelevant aspects, and then go on to study more complex problems. This approach is thus through a model, where the model is a simplified representation of the real world or the problem.

In software engineering, models of software are often based on finite state machines or graphs with well-defined mathematics [Clarke, Grumberg, et al. 2000]. Both *models* and *modelling* in software engineering are extensive topics and have a long history.

Before systematic knowledge of modelling was available, researchers built models manually. This manual building process involved designing, building, analysis, checking and verification. The notations used for the development of models has attracted a lot of attention, and, having developed them, and researchers then used them to build models. The most popular modelling notation used now is Unified Modelling Language (UML) [Stevens and Pooley 2000]. UML modelling has been a remarkable success in software engineering.

Notations help the software developers build formal models, but the developers still need to put a lot of effort into learning the notation and building models in a specific notation. Researchers and IT companies build powerful code-generation tool, such as Eclipse [Eclipse 2007] and Microsoft Visual Studio [Microsoft 2005], to assist the developers to build models. With the help of such tools, developers take the code-only approach to develop software. The developers develop ideas of what model they want to build, and then finish some high level work in the tool's development environment. This is a good approach for small sized software development projects, but it could possibly handle larger projects.

An improved approach is to use visualization of the modelling process. The powerful software development tool, IBM WebSphere Studio, is a modelling environment, which allows the developers to build and analyse models through the graphical notation and editable text. The benefit of this approach is to show the developers the code view and the model view simultaneously, but this approach requires that the diagrams are tightly coupled representations of the code. This approach is a better choice for experienced developers who have deep understanding in modelling notation and use of the development tools.

At a higher level, the model-centric approach can be employed when the system model has sufficient details. The developers can then generate the full system implementation, or the framework only, from the models themselves with the help of the generation tools. The model, the tools and the generation rules must correspond. This approach requires that the developers create a model with rich information, and the implementation generation tool should be specified for these models. But the generation process is sometimes complex.

The model-only approach requires that as the size and the complexity of the software and application increases, the size of the developer team increases as well. The developers may come from the single organisation, or multiple organisations. The developers need an agreed model to keep the same understanding of the big picture. They will discuss, demonstrate and analyse such models. A good example can be found in the enterprise architecture.

As described above, the model and modelling are important for the developers and for software development. Based on the model and the development requirements, different approaches can be used. The developers will select the appropriate approach based on their requirements, their skill and experience, and the development tools available.

1.1.2 Business process model

Generally, a model can represent a set of components of a process, system or subject area. A business process can be seen as a set of components that shows a set of activities.

The definition of a business process first comes from the business world. It is "*A structured, measured set of activities designed to produce a specific output for a particular customer or market*" [Davenport 1992]. It has also been defined as "*a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer*" [Hammer and Champy 2001]. A business process model describes the tasks involved in the process, and the order in which those tasks have to be carried out. A task is usually the smallest unit of action. Each task is performed by a function within the model. A role is a group of entities that performs one or more tasks. A role may be assigned to carry out any number of tasks. An entity can act in any number of roles.

A large number of established techniques support business process modelling. These techniques include: Process Mapping [Damelio 1996], Role Activity Diagrams [Phalp, Henderson, et al. 1998], Integrated Definition for Function Modelling (IDEF) [Kalian and Watson 2003], and Flowcharting [HCI 2008]. Regardless of the technique, their common goal is to provide a representation of objects that perform some functions and implement the information system.

1.1.3 Formal modelling

Formal modelling means the development of a model using a formal language, based on a grammar, or formal notation. The formal modelling languages and self-describing mark-up languages such as XML (eXtensible Markup Language) [Ahmed, Ancha, et al. 2001; Ozu, Anderson, et al. 2001] are usually used to build software models formally, and are currently used especially for web applications.

BPEL4WS (or BPEL) [BEA, IBM, et al. 2003] defines a model and a grammar for describing a Web service, which is the behaviour of a business process based on interactions between the process and its partners.

JML, the Java Modelling Language [Cheon and Leavens 2002], is a formal behavioural interface specification language, for Java only, which specifies the behaviour and the detailed design of Java program modules, such as classes and interfaces. JML is more suitable than the other languages for documenting the detailed design of an existing Java program.

There are many other existing formal modelling languages, such as B [Leuschel and Butler 2003], apart from the languages we have mentioned. Each of them uses a different approach, and has distinct features and application area. However, all of them share the same criteria:

1. They should be easy for the designer or modeller to use.
2. The level should be acceptable to the user.
3. They should describe the model exactly.
4. It should be possible to check the implementation against the model.

1.1.4 Model checking

Model checking [Clarke, Grumberg, et al. 2000; Holzmann 1997; Visser, Havelund, et al. 2000] usually deals with establishing whether the design of a finite system satisfies some properties. It is automatic, fast and repeatable.

When the model has an error, model checking will produce a contradiction, which can be used to detect the errors in the design. Some powerful model checkers have a runtime analysis function to trace the execution history of a model and then to perform static analysis on this history. When doing static analysis of an individual execution history, the state space of the model is reduced and errors in the design can be checked. But, as some paths may not be executed in an individual execution, it is possible to miss checking the whole model. This approach is taken by the Eraser algorithm [Savage, Nelson, et al. 1997] for detecting potential data traces, and the LockTree [Visser, Havelund, et al. 2000] and GoodLock [Havelund 2000] algorithms for detecting potential deadlocks.

1.1.5 Model checkers

As we know, the model checkers are the assistance tools for model checking. So, a model checker should have the ability to detect whether a finite state satisfies some properties. Or we can say, a model checker is a procedure that decides whether a given structure is a model of a logical formula or not.

When the model checker analyses all the reachable states and finds no errors, or no violations, the checking is passed, and the model has been qualified, at least in this model checker environment. When the model checker detects a violation, it will generate a counterexample. The counterexample is a sequence of reachable states, beginning with an initial state and ending with the property violation. The model developer can determine what the error is, where the error is, and why there is an error through analysing the counterexample. A counterexample is always used as a test to compute the expected outputs.

1.1.5.1 The SPIN model checker

SPIN [Holzmann 1990; 1997; 2003; Holzmann and Joshi 2004] is a model checker that can help the user find and diagnose concurrency-related bugs, such as deadlock errors, race-conditions, and some problems related to improper synchronization, in multi-threaded and distributed software systems. SPIN also can be used to prove sophisticated temporal properties of models of asynchronous processes.

But SPIN is only supported by its own input language Promela [Holzmann 1997; 2003], which is similar in style to the C programming language. Developers must be familiar with Promela, build a model in this language for the target system, and then run the model through SPIN.

Running SPIN will identify sequences of system behaviour. When it finds counterexamples to a properties correctness claim, it displays the error trace information using the graphical interface XSPIN. Simulation and verification are tightly coupled in SPIN.

SPIN was selected by Walters as the target for the automated transformation of RDT models, and as the tool applied to verify RDT models [Walters 2002a; b]. The tool RDTtoSPIN, which generates Promela code for the SPIN model checker, developed by Walters, will be introduced in 2.

1.1.5.2 Other model checkers

Other model checkers are introduced and their features are discussed below.

1. SMV's description language is too low a level for widespread use. SMV represents the reachable states symbolically as a BDD formula [McMillan 2000]. It captures system behaviour as combinatorial and sequential logic, and captures systems requirements as statements in temporal logic.

2. FSP (Finite State Processes) is an algebraic notation used to describe process models [Magee and Kramer 1999]. FSP combines ideas from both Hoare's CSP [Hoare 1985] and Milner's CCS [Milner 1989], and is designed to be easily machine-readable. The tool LTSA checks FSP models for a variety of fundamental properties [Magee and Kramer 1999].

3. JPF, Java PathFinder, has been developed by the Automated Software Engineering group at NASA Ames Research Centre to make model checking technology part of the software process [Havelund 1999; 2000; Havelund and Pressburger 2000; Visser, Havelund, et al. 2000]. It has the advantage of combining model checking techniques with techniques for dealing with large or infinite state spaces. JPF uses state compression to deal with enormous numbers of states, and partial order reduction and runtime analysis techniques to reduce state space.

4. JPF and VeriSoft [Godefroid 1997] operate directly on Java programs, and systematically explore their state space to check correctness. Bandera [Corbett, Dwyer, et al. 2000] and JCAT [Demartini, Iosif, et al. 1999] translate Java programs into the input language of an existing model checker such as SPIN or SMV [McMillan 2000].

1.1.6 Model-based approach

The model-based paradigm uses formal models to enhance the maintainability and correctness of software and system development. It helps in the design phase of the system lifecycle, and plays an important role in testing and verifying the developed system.

Agarwal has presented a model-based approach for building web-based complex information systems [Agarwal, Bruno, et al. 2001]. This approach can integrate features of an information system at the model level.

A visual approach, based on the use of software models and graph transformations is presented by Hausmann [Hausmann, Heckel, et al. 2005]. This approach enables the seamless development of Web service descriptions in a standard model-based context.

1.1.7 Model driven architecture

Model Driven Architecture (MDA) is an approach to software development that is centred on the creation of a model itself, rather than program code [Frankel 2003; Kleppe, Warmer, et al. 2003; OMG 2007]. The MDA approach is to build an architecture that separates the specification of a system from its implementation. The issues of portability, interoperability, and reusability throughout this process are very important.

Schmit proposed a model-driven approach [Schmit and Dustdar 2005], which introduces transactions into the design without increasing the complexity of the basic UML diagram. This approach can assists designers reuse the model of the system to specify the properties of Web service.

Grønmo presented a framework which supports the model-driven development of Web services [Grønmo, Skogan, et al. 2004]. With the help of this framework, a Web service implementation template can be generated, which is based on the specification of a Web service. The web application then will be developed, based on this template.

1.2 Distributed Systems

Distributed systems [Ahmed, Ancha, et al. 2001; IEEE 1987; Kalian, Watson, et al. 2004; Long and Strooper 2001; Mukhar, Weaver, et al. 2003; Stevens and Pooley 2000] encompasses many areas of computer science, such as computer architecture, networking, operating systems, embedded devices and security. In recent years, the maturity of principal theories of distributed systems has led to great success in many application domains, such as e-Business and web technologies. A typical definition of a distributed system is "*one in which components located at networked computers communicate and coordinate their actions*" [Colouris, Dollimore, et al. 2001]. The meaning of the term of *computers* in this definition is comprehensive. Any device whose behaviour is totally or partially the same as the behaviour of a computer is included in this definition. For example, a mobile telephone is not normally regarded as a computer, but with the development of mobile technology, it can be used to browse web sites and receive e-mails across a wireless network, and has therefore become part of a distributed system.

1.3 Synchronisation

In distributed systems, communication between the sending process and receiving process must be either synchronous or asynchronous.

In synchronous communication, the sending and receiving processes synchronize at every message. Whenever a *send* is issued, the sending process is blocked until the corresponding *receive* is issued. Whenever a *receive* is issued, the receiving process suspends until a message arrives [Colouris, Dollimore, et al. 2001].

In the asynchronous form of communication, the use of the *send* is non-blocking in that the sending process is allowed to proceed as soon as the message has been copied to a local buffer. In distributed systems, most of applications are asynchronous rather than synchronous.

In a Java system environment, the multiple threads mechanism in a single process is efficient in handling both asynchronous and synchronous communication using queues.

1.4 Web Services

Java Web applications are important feature of the Java 2 Platform Enterprise Edition (J2EE). J2EE consists of application technologies for defining business logic and accessing enterprise resources such as databases, Enterprise Resource Planning (ERP) systems, messaging systems, e-mail servers, and so forth.

Web services is a new breed of web applications as a foundation for creating the next generation of distributed applications [Gottschalk 2000]. Web services can be developed and used by any language, using any component model, running on all operating systems. HTTP is employed as the underlying transport to pass requests through firewalls. XML is used to format the parameters of the request, and the parameters of the feedback, so the request and its feedback are independent and are not tied to any particular component technology or object calling convention.

1.4.1 JAX-RPC Web service

1.4.1.1 RPC

Distributed systems require that computations running in different address spaces, potentially on different hosts, are able to communicate [Gottschalk 2000]. The most popular programming abstraction for distributed computing is the remote procedure call (RPC), using a middleware package such as CORBA [OMG 1993], DCOM [Microsoft 1996], and Java RMI (Remote Method Invocation) [SUN 2003a; b]. Java supports remote objects through RMI. RMI essentially allows remote Java objects, that implement a remote interface, to be invoked by clients almost as though they were invoking a local method [Harold 1997]. RMI provides heterogeneity across operating systems and the Java vendor, but not across languages.

Web services are components, which reside on the Internet, that have been designed to be published, discovered, and invoked dynamically across various platforms. The methods that reside in a specific Web service, may use Simple Object Access Protocol (SOAP) to send or receive data in the form of XML [Ruggiero 2003].

The following are the major technical reasons for choosing Web service applications [Nagappan, Skoczylas, et al. 2003].

- Web services can be invoked through XML-based RPC mechanisms across firewalls.

- Web services provide a cross-platform, cross-language solution based on XML messaging.
- Web services facilitate ease of application integration using a lightweight infrastructure without affecting scalability.
- Web services enable interoperability among heterogeneous applications.

1.4.1.2 SOAP

The fundamentals of SOAP (Simple Object Access Protocol), and the role of SOAP in developing Web services architecture, and its implementation, will be briefly introduced here.

Using XML notation, SOAP defines a lightweight protocol and encoding format to represent data types, programming languages, and databases. SOAP can use a variety of Internet standard protocols (such as HTTP) as its message transport, and it provides conventions for communication models like RPCs and document-driven messaging. This enables synchronous communication and asynchronous communication over HTTP.

To enable SOAP messages to communicate with J2EE-based components and messaging applications, most vendors provide SOAP messaging over Java Messaging Service (JMS), with JMS-compliant MOM (Message-Oriented Middleware) providers. This allows SOAP-based asynchronous messaging, and enables the SOAP messages to achieve reliability and guaranteed message delivery [SUN 2003c].

1.4.1.3 JAX-RPC

JAX-RPC stands for Java API for XML-based RPC. JAX-RPC uses the remote procedure calls (RPC) and XML-based protocol, such as SOAP, to build Web services and clients. It can be used to develop applications, in a distributed client/server model, across platforms.

With JAX-RPC, clients and Web services have a big advantage: the platform independence of the Java programming language. JAX-RPC uses technologies defined by the World Wide Web Consortium (W3C): HTTP, SOAP, and the Web Service Description Language (WSDL). WSDL is an XML-based language for describing Web services and how to access them.

1.4.2 JMS

JMS (Java Message Service) is one important library in the J2EE, and includes a set of interfaces and associated semantics, which define how a JMS client accesses the facilities of an enterprise messaging product. JMS supplies an API for the Java application to create, send, receive and read messages, and support a framework for asynchronous messaging.

JMS provides two types of messaging models: point-to-point messaging and publish-and-subscribe messaging. The characteristics of each model are covered below.

1.4.2.1 Point-to-point messaging

The application, based on the point-to-point messaging model, is built around message queues, and has a one-to-one relationship between sender and receiver. Each sender posts the message to a queue, from where the receiver removes messages. There is no mechanism to send a message to a particular receiver. Many receivers can access the same queue, but only the first to pick up the message will receive it. The sender may set a timeout on a message, after which it will be deleted from the queue, but this is not mandatory.

JMS point-to-point messaging has the following characteristics:

- Each message is produced by the sender and consumed by one and only one receiver.
- Messages are either, consumed by the receiver, or they are timed out and are deleted by the JMS provider, if a timeout on this message has been set.

- Receivers can consume the message only after it has been sent.
- The instance of the receiver is dependent on whether the message is produced.
- The receiver cannot request a message.
- The receiver can acknowledge receipt of the message if required.

1.4.2.2 Publish/subscribe messaging

The JMS publish/subscribe messaging domain has a completely different mechanism. With the publish/subscribe model, senders post messages to a topic; many receivers can register interest in a topic by subscribing to it. Because of these features, this messaging approach is not suitable to communication within RDT models, so I selected the point-to-point model.

1.4.3 Synchronous Web service

The JAX-RPC runtime system runs on both the client side and the server side. It automatically takes care of marshalling/un-marshalling messages between the client and the server. These messages (basically SOAP messages) are sent using the HTTP protocol. JAX-RPC also provides both Java-to-WSDL and WSDL-to-Java mapping tools. The former generates a WSDL [W3C 2001] description of the service from the service's definition classes. However, this tool cannot handle overloaded methods. The tool automatically renames an overloaded method by appending some characters to it. For example, two methods, called "sendMessage" in Java, in the WSDL document might be called "sendMessage" and "sendMessage_1".

The WSDL-to-Java mapping tool works on the client side to generate references (stubs) to the service's methods from the WSDL document of the service. These stubs are used by the client program to call the service's methods. Unlike RMI (Java's version of RPC), the stubs are generated on the client side (and not on the server) and thus are not downloaded at run time.

1.4.4 Asynchronous Web service

Many Web service frameworks, such as Apache Axis, only allow for synchronous invocation. This is unacceptable, especially as the Internet has latency and unexpected errors cause unpredictable invocations. In such cases, we require the client to handle the invocation asynchronously. That means the client processes should resume their work while the invocation is handled, no matter how long the latency is. In addition, the time taken to process the Web service should be tolerated. As mentioned before, most Web service frameworks are initially designed for synchronous communication rather than for asynchronous communication, so we need to provide the asynchronous behaviour on top of the synchronous invocation layer, to handle all of cases. My work covers building a framework using patterns for asynchronous invocation of Web services, and can handle with both asynchronous communication and synchronous communication.

There are various approaches to integrating messaging protocols into Web services, such as the use of Java Message Service (JMS) [Monson-Haefel and Chappell 2001] in Axis and WSIF [Apache 2007], JAXM [SUN 2007], and Reliable HTTP (HTTPR) [Banks, Challenger, et al. 2002]. These protocols provide asynchrony at the protocol level. They are more sophisticated than simple asynchronous invocations and use a different communication paradigm than synchronous protocols.

1.4.5 Web application servers

Because of the success of the Java platform, the term *application server* refers to a J2EE application server. Many companies offer application servers; here the features of some of them are introduced.

1.4.5.1 Apache Tomcat

Apache Tomcat is an open source implementation of Sun's J2EE Web container [Apache 2002]. It is designed to run on J2SE 5.0 and later, and requires configuration to run on J2SE 1.4. Tomcat can be freely downloaded on any operating systems and used in any organization for academic or commercial purposes. Tomcat can function as a web server, and can also be integrated with other web servers for open source application development.

1.4.5.2 IBM WebSphere

WebSphere is the IBM software product designed to help deliver dynamic e-business quickly [BEA, IBM, et al. 2003]. The technology that powers WebSphere products is Java. It contains full J2EE 1.4 support, but its limitation is that only single-server environments are supported.

1.4.5.3 BEA WebLogic

BEA WebLogic server includes BEA WebLogic Express, which is a scalable platform that serves dynamic content and data to web and wireless applications [BEA 2003]. WebLogic offers many services and APIs, including JDBC, JSP, Java servlets, RMI, and Web server functionality. WebLogic Express is different from WebLogic Server in that the former does not provide EJB, JMS, or the two-phase commit protocol for transactions.

1.4.5.4 JBoss

JBoss application server is the most widely used Java application server on the market [JBoss 2003]. It is a J2EE certified platform for developing and deploying enterprise Java applications, Web applications, and Portals.

JBoss application server has the following advantages:

- Open standards and open source
- Simple to use
- Clustering and high availability
- Pure Java

I chose Tomcat 5.5 and JBoss 4.0.2 as application servers. This decision was based on the common features of both products as follows:

- Free software

- Open source
- Pure Java support
- JMS support
- Queue management

1.5 Software Testing and Verification

Most software has faults. It is complicated to develop provably error-free software, and there are no efficient testing approaches which can test all type of software.

It is important to make sure that the software can perform as expected, and its functional and non-functional specifications are satisfied. This is the task of software testing. As the size and the complexity of software increases, testing becomes more challenging. Testing will cost a lot during the software development life cycle. The cost of this assurance ranges between 50 and 75 percent of the total development cost [Patton 2000].

The following section briefly describes the verification and testing technologies in the different software development stages, and introduces some support tools as well.

1.5.1 Verification

Verification consists of checking that a specification satisfies a property which may be given by a temporal logic formula, algorithm or another more abstract definition. The process of verification presents a lot of features: partial verification, on-the-fly checking, reductions, etc., which are all relevant for the problem of test generation from a formal specification.

Researchers have attempted to build tools to support automated verification. Thompson presented a tool used to represent properties as deterministic finite state automata over the TFG (Trace Flow Graph) language [Thompson 2000]. Although this has so far not been a natural language template, clearly it will be helpful in analysing finite state software. Tufarolo describes the design and implementation of an RTI (Run Time Infrastructure) verification system (the Verifier) [Tufarolo, Ives, et al. 1999; Tufarolo, Nielsen, et al. 1998].

1.5.2 Testing

Testing is obligatory for software validation during the software development lifecycle. The principal purpose of testing is to detect the faults and errors in a software system. The developers use commercial tools and testing approaches to provide a solution to the problem of building fault-free systems.

The overall goal of testing is to provide confidence in the correctness of a program. The only way to guarantee a program's correctness is to execute it on all possible inputs, but this is usually impossible. The most feasible alternative then is to build a test set that has enough significance to reveal the maximum number of errors, so that a test can give confidence to the programmer that the program meets its specification as regards correctness.

The two most popular testing approaches are black box testing and white box testing. The black box testing method (called behavioural testing) is an approach to find errors in a program by validating its functionalities, without analyzing the details of its code, but using the specifications of the system. The goal is to answer the question "does a program satisfy its specification?" Black box testing is much simpler than white box testing because it ignores the details of the structure, thus testing at higher levels of abstraction.

White box testing (called structural testing) is to test the software from code level to the functions level. Each line of source code will be executed, and each single function will be tested. Parts of a program may be tested as well, but this is difficult to design.

In practice, a single test design method has not proven effective to use to test all of the software. A mixture of different methods should be used, so that we can expect to detect more faults and errors. Such an approach is called grey-box testing.

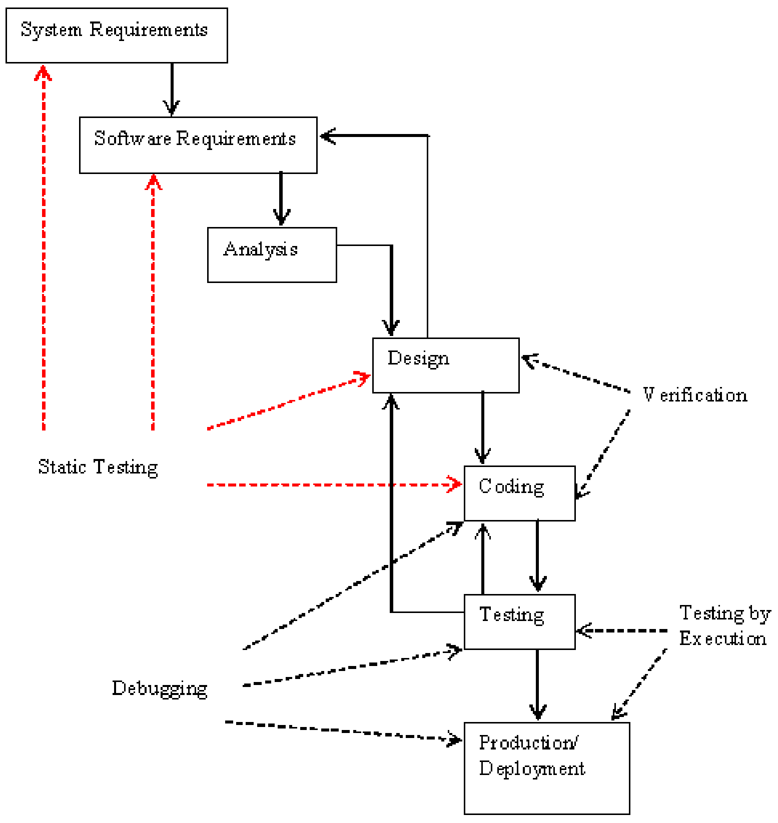


Figure 1: The activities in a software development life cycle

Regression testing refers to the testing approach where a modified version of a component or application is tested, in order to ensure that existing features are still intact. This testing approach and other testing methods have been used by Beydeda [Beydeba and Gruhn 2002] and Yamaura [Yamaura and Onoma 2002].

Integration testing for object-oriented and concurrent programs was introduced by Chen [Chen, Chen, et al. 2002]. The paper reviews the common techniques for program testing at four levels, namely the algorithms level, class level, cluster level, and system level. Program testing includes state-based testing, event-based testing, fault-based testing, deterministic and reachability techniques, and formal and semi-formal techniques, at the cluster level.

System testing is designed to reveal defects that are not caused by individual components, or that only happen during execution of the system. This approach focuses on the issues and the behaviours that can only be exposed by executing the whole system or a major part of the system. In practice, black-box testing is predominantly used for system testing. The obvious reason is that the number of possible paths that are required to structure test a system is far too large to handle.

When a partition testing approach is employed, first the testing criteria must be decided. The input domain is then divided into two or more separate sub-domains according to such criteria, and then test data are selected for each sub-domain. Chan designed a partition schema that manages how the input domain can be divided, and a test case allocation scheme that controls how to allocate test cases to the sub-domains [Chan, Chen, et al. 1997]. Chen also used the partition test and another testing technique, random testing [Chen, Tse, et al. 2000].

The lowest level of testing is called unit testing. Each function, module or class is individually tested. Unit testing has the highest chance of controlling the execution and observing unit faults, but it does not give any information about the correctness of the behaviour of the whole system.

Another noticeable issue is that there are many claims for automatic test assistant tools, such as JTest [Parasoft 2003]. Boyapati presents a novel framework, called Korat, for automated testing of Java programs [Boyapati, Khurshid, et al. 2002]. Korat takes a given formal specification for a method, and uses the model pre-conditions to generate test cases automatically. The method will be executed for each test case, and the method post-conditions are checked against the correctness of each expected output.

1.5.2.1 Model-based testing

Model-based testing is proposed as a technique to automatically verify that the implementation of a system is matched to its specification. Apfelbaum and Doyle introduced this technique generally [Apfelbaum and Doyle 1997]. Esser and Struss presented the case study of model-based testing for embedded software system [Esser and Struss 2006].

This technique is employed for my work, along with test case generation.

1.5.2.2 Test cases

The Test Case (see Figure 2) has been defined in several ways.

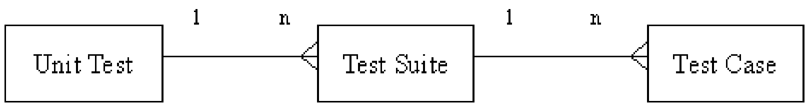


Figure 2: Relationships between Unit Test, Test Suite and Test Case

- Documentation specifying inputs, predicate results, and a set of execution conditions for a test item [IEEE 1998].
- A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement .
- The specific inputs to try and the procedures to follow when testing the software [Patton 2000].
- A sequence of one of or more subtests executed as a sequence because the outcome and/or final state of one subtest are the input and/or initial state of the next. The word ‘test’ is used to include subtests, the test proper, and test suites [Beizer 1995].

It is complicated to design suitable test cases, and the complexity comes from three sources:

- A specific type of test is efficient for some test targets. Increasing the number of tests will increase the efficiency.
- There are no specific test cases that are perfect for all tests. It is possible that a specific test case is perfect in one or more tests, but it is impossible that a specific test case is perfect for all of tests.

- Test cases are designed by the testers, so the experience, skill and style of the testers will impact the quality of the test cases.

Model checking is applied to test case generation and coverage evaluation as a popular formal verification technique for both software [Krichen and Tripakis 2004] and hardware [Lerda, Sinha, et al. 2003]. Model checking can be used to compute the test outputs. It also can be used to create counterexamples.

Kansomkeat and Rivepiboon proposed a transformation method from UML state chart diagrams, created by the Rational Rose tool, into intermediate diagrams, which are used to generate test sequences [Kansomkeat and Rivepiboon 2003]. The testing criterion used to guide the generation of test cases is the coverage of the state and transition of the TFG (Trace Flow Graph). The measure of effectiveness of test cases is their ability to detect faults. Simple test experiments show high effectiveness of the generated test cases. However, extensive experiments are needed for more confidence in the testing techniques and to compare them with other techniques in terms of cost and effectiveness. Kansomkeat and Rivepiboon evaluated the effectiveness of their test cases using a fault injection technique, called mutation analysis. Mutation analysis is a fault-based strategy that starts with a program to be tested and makes numerous small syntactic changes to the original program (or the specification).

The model checker SMV is used to obtain a test sequence from a system property and an SCR (Software Cost Reduction) requirement specification [Gargantini and Heitmeyer 1999].

Testing remains a labour-intensive activity, thus error-prone. My work includes automatically generating test cases from an RDT model. When testing the message flows of a system, the processes, or changing states, the following are some of the more frequently used testing techniques. The most important, and widely used, testing techniques are based on path testing. This kind of testing uses the flow graph, which is the Process View or Model View graphics that are shown on the

RDT. It compares the model executed behaviour with the desired behaviour. Transaction flow graphs specify the high-level behaviour of a whole system. These testing methods are the normal approaches for static model testing. This research also uses them in testing RDT models and in the actual distributed system, with automatic test case generation. Message passing in the distributed system is important. An invocation of a particular test case can lead to one of the following results [Goschl and Sneed 2002]:

- Passed
- Failed
- Crashed
- Obsolete

A test case fails if it does not fulfil all required post-conditions of the test. A test case crashes if an exception is propagated out of the test case implementation. A test case is obsolete if it is not used any longer [Sneed 1998]. The test cases in my research will cover dynamic message passing.

1.5.2.3 Testing and verification of distributed systems

There is much published on testing traditional systems. There are a number of proposals for concurrent and distributed systems, but it is not clear how they scale up or how widely applicable they are. There are very few case studies on this topic, and typically, they are not detailed enough to be of practical use [Long and Strooper 2001]. In addition, very few papers on distributed systems testing deal with the issue of concurrency. They assume that the middleware handles all concurrency issues [Xing, Lyu, et al. 2000].

Distributed systems testing covers acceptable performance (latency), fault tolerance (partial failure), concurrency, operating environment issues, and security.

Latency of responses between the server and its distributed components is an issue. In the case of partial failure, components need to decide how long to wait to be serviced before “giving up” and proceeding, or throwing an exception, or timing out. Servers often handle multiple clients by using concurrent programming techniques such as multiple threads. Hence, the existence and safety properties of these concurrent systems need to be addressed. Due to the heterogeneous nature of distributed system platforms and architectures, consistent behaviour across the application cannot be guaranteed. Errors can occur when objects are serialised in one environment and reconstructed in another environment. Security is a further issue that may need to be addressed, since messages may be sent across a public network.

Distributed applications (such as file sharing, web and mail service) are very difficult to implement. Traditional testing methods alone are not suitable for verifying the correctness of distributed applications, due to their significantly higher complexity over local applications [Kaveh and Emmerich 2001].

Testing distributed systems usually follows the steps of single component test, integration tests of components, and finally the system test.

Stoller described the multi-process approach to model checking and testing distributed programs [Stoller and Liu 2001]. Their approach is to combine multiple processes into a single process, to replace RMIs with local method invocations that simulate RMIs, and to replace cryptographic operations with symbolic counterparts. In my research, the local Java model is extended to the remote Java object model. The testing covered both models.

Tsai proposes a scenario-based and object-oriented test framework to test distributed systems [Tsai, Yu, et al. 2003].

Callahan and his colleagues show a formal testing approach based on model checking to assess the impact of a specification change in terms of the proportion of existing tests that are invalidated due to the change [Callahan, Schneider, et al. 1996].

My approach will provide automatic test case generation based on model checking to support and cooperate with other technologies to test distributed applications [Long and Strooper 2001; Tufarolo, Ives, et al. 1999; Tufarolo, Nielsen, et al. 1998].

1.6 My Contribution

The building of a Java multi-threaded application, and of Web services, and of a distributed system based on a formal model, are still tough problems. Here I offer a solution to both issues.

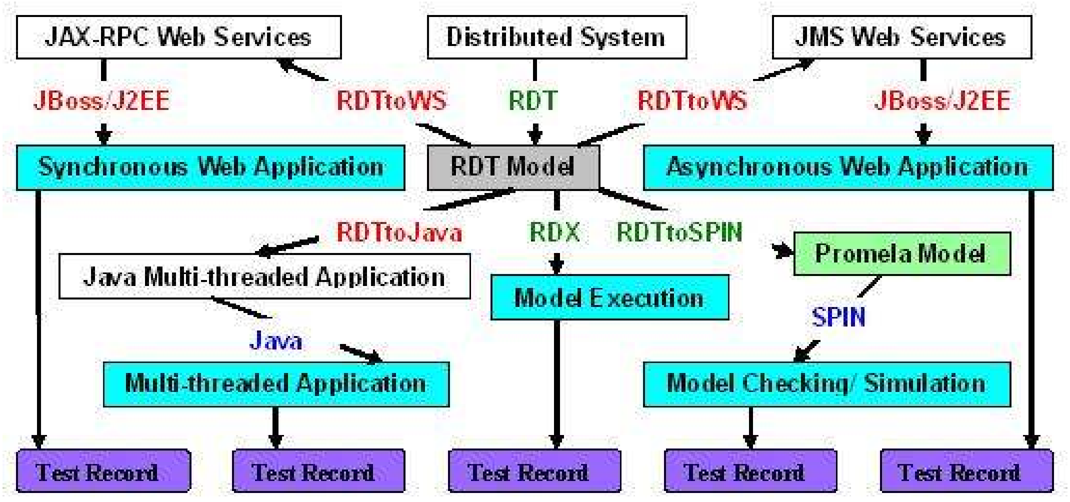


Figure 3: Activities in the approach to my current work

The RDT tool [Walters 2002b] has been accepted as a formal model developing tool. Based on RDT, I am trying to build a set of tools, elaborated in Figure 3. The RDT tool is used to build up a model of the distributed system. The tools RDX and RDTtoSPIN will help me analyse and evaluate the models I built, so that the quality of model can be guaranteed. The tool RDTtoJava, which will run in a Java environment, will translate the model into a Java threaded application automatically and directly. The framework RDTtoWS will help me develop RDT models into JAX-RPC Web service applications and JMS Web service applications. The J2EE technology, the web application JBoss, and Apache Tomcat, will run

JAX-RPC Web service as a synchronous Web application, and run JMS Web service as asynchronous Web service.

During the tool and the framework development, testing is an important issue.

The items in Figure 3 are specified thus.

- Distributed System (white box). The application or system will be built.
- RDT Model (grey box). A distributed system is described in a RDT model
- RDT, RDX and RDTtoSPIN (in green). Tools developed by Robert Walters.
- RDTtoJava and RDTtoWS (in red). Tools developed by me.
- Java and SPIN (in blue). Free public software tools used in my work.
- Promela Model (light green box). A Promela model is translated from RDT model by the tool RDTtoSPIN.
- JAX-RPC Web Service, JMS Web Service and Java Multi-threaded Application (white box). Applications generated by my tools.
- Model Execution, Model Checking/Simulation, Multi-threaded Application, Synchronous Web Application, Asynchronous Web Application (light blue box). The implementations of web application and multi-threaded application.
- Test Record (purple box). Test reports from the applications.

With the assistance of my tools, software developers can build applications directly and quickly. Software engineering research will benefit as well.

The classic distributed communication models have been selected for investigation. The RDT tool focuses on the business process model; this is the first time this tool has been used to build up models of distributed systems. Special attention has been paid to synchronous and asynchronous communication models. From my experiments, developers can get some idea of the architecture and structure of the system they can choose.

The automatic generation of test cases and model-based testing are also addressed. As my tools can generate both the code and the whole application, the software developer can undertake the simple code change work to design their test cases.

This is a development package, based on RDT, which starts from a model, through creating the application, and on to acceptance testing.

1.7 Overview

2 gives an overview of the RDT, RDX and RDTtoSPIN in the RDT toolset developed by Robert Walters. The language and the notation employed in RDT are presented. The features of RDX and RDTtoSPIN are also introduced.

3 introduces a new tool, RDTtoJava, which is a translator for the conversion of an RDT model into a Java multi-threaded application. In particular, the mapping from the RDT language to the Java language and the deployed system are described.

4 introduces the framework to transform an RDT model to a synchronous application using JAX-RPC Web service, and to an asynchronous application using JMS Web service.

5 presents sample RDT models to illustrate my approach. The model generation tool RDT builds the models and generates the XML files, which will be employed by the RDX and RDTtoJava tools. The experimental results for these models are given. RDTtoWS develops and deploys these RDT models into synchronous and asynchronous Web services. The testing issue will be introduced as well.

6 summarises my work and discusses future work.

2 Formal Modelling System RDT

This chapter introduces the formal modelling system RDT developed by Walters [Walters 2002a; b; 2005]. This system includes RDT modelling language and notation, and three tools in the RDT toolset: RDT, RDX and RDTtoSPIN.

RDT is a graphic language for the description of processes and systems built from communicating systems and instances of these processes [Walters 2002a]. As the foundation of my work, RDT language along with the RDT notation will be introduced in detail in this chapter. Generally speaking, RDT is a feasible tool with which to build a model of a system. RDX is a RDT model execution tool. RDTtoSPIN is a tool to translate an RDT model into a Promela model, "source code to source code", automatically; and then the model checker SPIN will run this Promela model to check the RDT model. These tools are used in my work to develop RDT models and model checking.

2.1 RDT Language and Notation

The tool RDT differs from traditional modelling languages in that models constructed using the language, are built by drawing diagrams in place of normal textual descriptions. It also generates a textual file in XML format. As a modelling system, RDT gives the modeller a friendly diagrammatic interface to start building a model, and it supplies a simple system for the modeller to build a model. The modeller can define the behaviour and structure of the channel-based communications system.

RDT language uses the pi-calculus as its foundation [Milner 1993]. The RDT model is made up of some Instances of some type of Process. The Process is defined by Before state, Event, Channel, Value and the After state. Communication between pairs of Instances of Process is done by the Connection function, which connects the Channels of one Instance of one Process to another. More details about RDT language and notation are covered below, along with some features of RDT tools.

2.1.1 Process

The basic component in the RDT system is a Process. Each Process is a type of object. It has a public identifier. All events in the Process can be created and identified by the modeller.

2.1.1.1 State

When a new process is created, the internal state is set to a value of "initial". Processes proceed from one state to another by taking part in events. For each event, there is a Before state and an After state (see Figure 4). Where a state is re-visited, its name is suffixed with an '=' character (see Figure 5) as a special case.

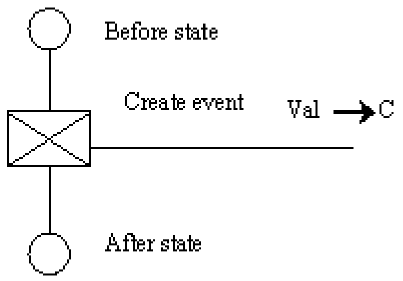


Figure 4: Before state and After state

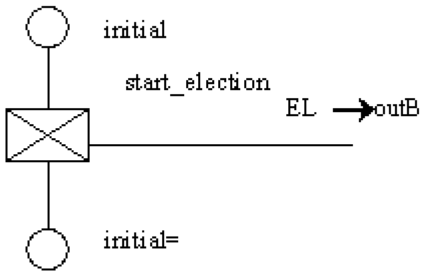


Figure 5: Notation where the state is revisited

2.1.1.2 Event

An RDT process may take part in three types of event, which are Read, Write and Create. A Read event is an action where the process reads a message from a channel; a Write event is an action where an object writes a message to a channel; a Create event is a special type of Write event, where the value to be written to the channel by the event is new and created as part of the event. A Write event is shown as a clear square. A Create event is a special case of a Write event distinguished in the diagram by a cross in its box. The Read event is drawn as a black square. The conditions for these events are discussed below.

Four conditions for the Write event (Figure 6) must be true:

1. The named state of the process must be the Before state of the event.
2. The specified local channel name must be associated with a channel.
3. The specified process value name must be associated with a value.
4. The channel must be prepared to accept a new value.

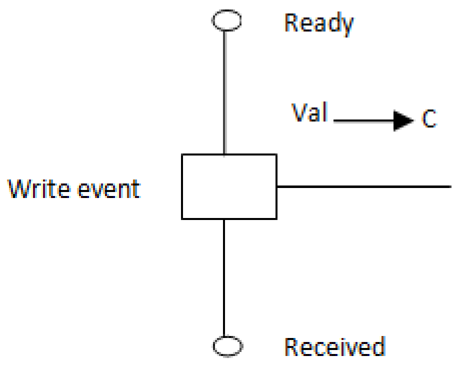


Figure 6: A Write event

Three conditions for a Create event (Figure 7) must be true:

1. The process must be in the stated Before state.
2. A new local channel is created.
3. The specified local channel name must be associated with a channel.
4. The channel must be prepared to accept a new value.

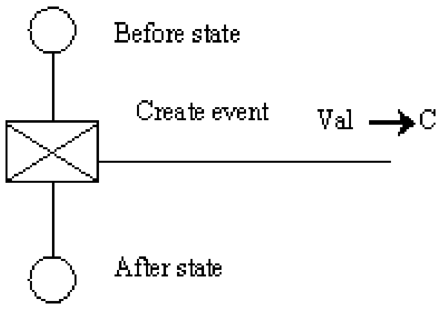


Figure 7: A Create event

Three conditions for the Read event (Figure 8) must be true:

1. The name state of the process must be the Before state of the event.
2. The specified local channel name must be associated with a channel.
3. The specified channel must have a value available for the process to read.

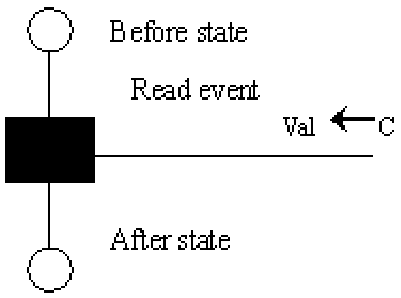


Figure 8: A Read event

2.1.1.3 Channel and value

The communication between instances of the processes must be done through channels. The instances of the same type of process can be connected with one or more channels by exchanging the value. The value can be viewed as a message which is passed from one process to another via a specified channel. A value is written to a channel, and a value is read from a channel as well.

Value ->Channel

Figure 9: Channel and Value

2.1.2 Model

After creating the processes, the modeller can create instances of them and can then set up connections between them to form the complete model.

2.1.2.1 Instance of Process

A single Instance is one of a type of Process that has been defined. For all instances of one type of Process, each one has an identification name and inherits all functions of this type of Process.

Jack: Barber

Figure 10: An instance of the Barber process named Jack

2.1.2.2 Connection

The Connection describes the associations between pairs of Instances of Processes. When the Connection is created, the Instances that have been connected can send and receive messages from each other. In the example shown in Figure 11, an Instance *a* of the Process *Process1* has a channel outbox, and an Instance *b* of the Process *Process2* has a channel *inbox*. When the Connection between *inbox* and *outbox* is done, it means *a* and *b* is connected, and the messages could be passed through this connection.

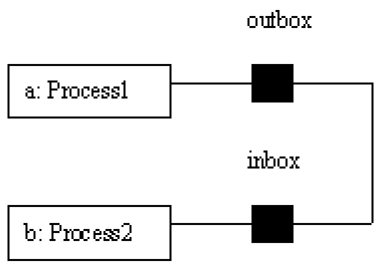


Figure 11: Connection notation

2.1.3 RDT tool features

The use of the RDT tool to build a model is covered in [Walters 2002b]. Here, I present three major functions of the RDT tool.

2.1.3.1 Channel length

RDT offers the modeller the option to specify the length of Channels used in their. When the length of a Channel is zero (the modeller just types 0 to make this selection), then the communication is synchronous. However, when the length of Channel is non-zero (an integer, which must be larger than 0), then the communication is asynchronous.

2.1.3.2 Process View

Along with the Model View, Process View allows the modeller to look at the Process during and after modelling. Process View will show all information for each Process, but does not cover the information about the connection with other Processes.

2.1.3.3 Model View

During the process of modelling, the modeller may view the current model from time to time to confirm that the modelling process is acceptable or not. The RDT tool supplies a Model View mechanism to look at interaction between the Instances of all type of Processes. Here, the information within the Process is invisible.

2.2 RDTtoSPIN

The SPIN model checker is one of the most widely used in the world, and it is selected as the model checker for the RDT model. The RDTtoSPIN tool takes the XML file of a RDT model and transforms this into Promela, the input language for the SPIN model checker. The modeller can then use SPIN to check this RDT model.

2.3 RDX

RDX is the model execution tool. It takes the XML file of a model generated by RDT and executes the model. The tool uses an interface inspired by that of the RolEnact execution tool [Henderson, Howard, et al. 2001; Henderson and Walters 1999; Phalp, Henderson, et al. 1998]. A successful outcome of this tool is that the modeller can see the dynamic asynchronous and synchronous communication between the processes, the status of the processes, the event that the process will execute next, and messages in the channels.

Each process instance in the model has its own window, in which the state of the process, the channels, and the event for next action, are shown. Each channel has its own window in which the values written to the channels and not yet read are shown.

3 RDTtoJava

RDTtoJava transforms a RDT model to a Java multi-threaded application. This chapter describes the approach of transforming RDT language into Java by defining transformation rules and methods, and discusses some important ways to enhance productivity and to reduce chances of making mistakes in system development.

We can learn some lessons from the Java2Promela translator [Basin, Friedrich, et al. 1999], which was designed to generate the Promela description of a Java multi-threaded application. The RDTtoJava is developed in Visual Basic 6.0 [Halvorson 1998]. This tool can be run on any operating system by Microsoft.

3.1 Conversion: RDT Model to Multi-threaded Java Application

My work focuses on the translation not only from RDT language to Java code, but also from RDT models to Java object models at the model level. As a completely object-oriented programming language, Java develops models or applications using analysis focusing on object classes and their relationships [Sommerville 2001]. All classes in Java extend the class Object, either implicitly or explicitly. An object class is an abstraction over a set of objects which identifies common attributes, and the service or operations provided by each object. Objects are executable entities with the attributes and services of the object class. Objects are instantiations of the object class and many different objects may be created from a class.

The executable RDT models focus on the synchronous and asynchronous communication among the instances of processes. Based on this, the Java thread technique is used to develop communication between the objects within the Java multi-threaded model. There are two basic points to guide this work:

1. At the model level, the translation should convert RDT models into Java models.
2. At the implementation level, the translation should make the implementation of the new model (Java threaded model) in Java as expected.

The following section describes the rules for mapping an RDT model to a local Java object model.

3.2 Mapping of RDT to Java

During the mapping, the XML file for the RDT model is not used directly, since the XML describes the RDT model at a low level. The DTD (Document Type Definition) defines the XML. I choose the DTD for the RDT XML file to build up the rules for translating XML to Java, and then map the XML file to the detailed Java code. This mapping principle is employed in both RDTtoJava and RDTtoWS.

3.2.1 Model

In RDT, the model definition consists of Instance and Process elements. Instance provides a name that can be used to distinguish the instance from all other process definitions within a model. An Instance definition requires this attribute to have a value. The Process provides a name that can be used to distinguish the process from all other process definitions. A process definition requires this attribute to have a value. The DTD (Data Type Definition) definition syntax for an RDT model definition is:

```
<!ELEMENT Model (Instance, Process+)>
<!ATTLIST Instance Name CDATA #REQUIRED>
<!ELEMENT Process (Event+)>
```

In a Java program, the definition consists of one public class definition and two class definitions. The public class contains a method named `main()`, which is the only entry point for the application, that is, the point at which the program execution starts. Each instance of a model in RDT will be translated into one Java program, whose name is the same as the name of the instance. The combination of all such Java programs will be the RDT model expected. Each process in RDT is one object class, which is the entity that extends Java thread's facility to communicate.

An example of translation from model and process in RDT into classes in Java is shown in Figure 12.

```
<Model>
    <Instance Name="cycle_election">
    </Instance>
    <Process Name="participant0">
    </Process>
</Model>
```

```
public class cycle_election {
public static void main(String args[]) {
. . .
static class Process extends Thread {
. . .
}
static class participant0 extends Process{
. . .
}
}}
```

Figure 12: Translation of the Model block in an RDT model into a Java object model

In this chapter and the next, I use DTD files to explain how the RDT is translated into applications. DTD is used to define the XML file, and all legal elements are defined in the structure. The DTD structure is easy to understand, so I use DTD rather than a specific XML example to explain my work.

3.2.2 Process

In RDT, a process definition consists of the Name attribute, which provides a name that can be used to distinguish this type of process from all other processes in an RDT model. A process definition requires this attribute to have a value. The DTD definition syntax for a process definition is:

```
<!ELEMENT Process (Event+)>
    <!ATTLIST Process Name (participant0 | participant1 | participant2 |
participant3) #REQUIRED>
```

Each process in the RDT model is one class in the Java program. The Java programs, like other object-oriented programs, use a separate class for each kind of object. A class defines a collection of state variables, as well as the functionality for working with those variables. Classes are like C *struct* or Pascal *record* definitions that allow functions within them. Each type of process in RDT is deployed to a class, which inherits from the `Thread` class in Java. Such a class defines one built-in constructor, the variables and the methods which are ready to support the future methods for the events within the process. The variable *state* is for the object's private state, and it is used to label the current state of the object. Like any state machine, an object within the system has a sole state. In particular, the class specifying the thread defines the name of the thread and the message queues. A message queue is required to communicate with other processes and is used for synchronous or asynchronous transactions. One example is shown below.

```
<Process Name="participant0">
</Process>
```

```
static class Process extends Thread{
MessageQueue inbox;
...
String name;
public String toString(){
return this.name;
}
}

static class participant0 extends Process{
public participant(String name){
inbox =new MessageQueue(10);
this.name =name;
this.start();
```

```
}
String state=" ";

public void run(){
}
. . . .
public String getname(){
return name;
}
public void transformState(String s){
state=s;
System.out.println(name +" : " + state);
}
}
```

Figure 13: Translation of system model specification in RDT into Java code

3.2.3 Event

An RDT Event is an atomic activity. It provides the context for performing an operation involving the exchange of messages with other processes. The Event is a composition of the following attributes:

Attribute	Description
Name	The event name
Type	The type of operation being performed
Before	The pre-state
After	The post-state
Channel	A sender outputs a message to a channel, or a receiver expects input from a channel
Value	The outgoing or incoming message

The DTD definition syntax for the Event definition is:

```
<!ELEMENT Event EMPTY>
      <!ATTLIST Event
Name (receive_election | receive_elected | receive_boss) #REQUIRED
            Type (Create | Read | Write) #REQUIRED
            Before (initial | election_start | send_election | election_send) #REQUIRED
            After CDATA #REQUIRED
            Channel (inbox | outbox) #REQUIRED
            Value (election | elected | boss) #REQUIRED
      >
```

The syntaxes for Create, Read and Write events are different and will be presented below. In Java, the name of the method for any type of event is the same as the name of the event in RDT.

3.2.3.1 Read event

The DTD definition syntax for the Read event definition is:

```
<!ELEMENT Event EMPTY>
      <!ATTLIST Event
Name (receive_election | receive_elected | receive_boss) #REQUIRED
         Type (Read) #REQUIRED
         Before (initial | election_start | send_election | election_send)
#REQUIRED
         After CDATA #REQUIRED
         Channel (inbox | outbox) #REQUIRED
         Value (election | elected | boss) #REQUIRED
      >
```

In Java, for the Read event, the translation is in two steps. The first step is to clarify the conditions for a Read event to occur. The second step is to complete the action for this Read event. The conditions are as follows:

1. the message queue that this event uses to receive the message is ready;
2. the Before state for this event should be satisfied, and
3. the message received should be the same as described in the RDT model.

The completion of the Read event is as follows:

1. call the method for this event, and then
2. change the object state from Before state to After state.

```
<Event Name="receive_election" Type="Read" Before="election_start"
After="election_receive" Channel="inbox" Value="election"/>
```

```
public void run(){
new Thread(){public void run(){
   try{for(;;){
       Message m=(Message)inbox.receive();
       if(m.type=="election" && state=="election_start")
         receive_election(m.sender,m.type,"election_start");
…
}}catch(exception e){System.out.println(name + ": demultiplex error");}}}.start();
}
public void receive_election(Process from, String message, String current_state){
System.out.println(name+"'s Event is: "+ current_state +" and read " + message +"
from "+ from.name);
transformState("election_receive");
       election_receive();
}
```

Figure 14: Translation of the Read event in RDT into Java code

An exception will be thrown when de-multiplex errors occur. The errors include:

1. no message received
2. unmatched receive channel
3. unmatched message received and the event state

3.2.3.2 Write event

The DTD definition syntax for the Write event definition is:

```
<!ELEMENT Event EMPTY>
      <!ATTLIST Event
Name (receive_election | receive_elected | receive_boss) #REQUIRED
            Type (Write) #REQUIRED
            Before (initial | election_start | send_election | election_send) #REQUIRED
            After CDATA #REQUIRED
            Channel (inbox | outbox) #REQUIRED
            Value (election | elected | boss) #REQUIRED
      >
```

In Java, The Write event writes a message to a channel, and then changes the state to After state. I will introduce how messages are transmitted in 3.2.4. Any exception will be caught during the sending of the message.

```
<Event Name="send_election" Type="Write" Before="election_receive"
After="election_send" Channel="outbox" Value="elected"/>
```

```
public void send_election(){
  System.out.println(name+": send_election");
  try{p1.inbox.send(new Message("elected",this,p1, "outbox"));
        transformState("election_send");
        }catch(Exception e){System.out.println(name + " : send_election- send error");}
}
```

Figure 15: Translation of the Write event in RDT into Java code

3.2.3.3 Create event

The Create event is a special kind of Write event. The DTD definition syntax for the Create event definition is:

```
<!ELEMENT Event EMPTY>
      <!ATTLIST Event
Name (receive_election | receive_elected | receive_boss) #REQUIRED
            Type (Create) #REQUIRED
            Before (initial | election_start | send_election | election_send) #REQUIRED
            After CDATA #REQUIRED
            Channel (inbox | outbox) #REQUIRED
            Value (election | elected | boss) #REQUIRED
      >
```

When the model is implemented in Java, each process in RDT is a class executing a thread with an initialisation state. The Before state is one condition which must be satisfied to start the event. The After state is another condition indicating the current state when the event is completed. For the Create event implemented in Java, the channel and the message must be specified.

```
 <Event Name="start_election" Type="Create" Before="initial" After="election_start"
 Channel="outbox" Value="election"/>
 public void start_election(){
    System.out.println(name+": start_election");
try{p1.inbox.send(new Message("election",this,p1,"outbox"));
        transformState("election_start");
        }
 catch(Exception e){System.out.println(name + " : start_election- send error");}
 }
```

Figure 16: Translation of the Create event in RDT into Java code

Each Channel in RDT shown in Figure 17 is a message queue object in the newly generated Java program. When the connection between a pair of channels is created, such as *channel_0* and *channel_A*, a message could be sent from one channel to another.

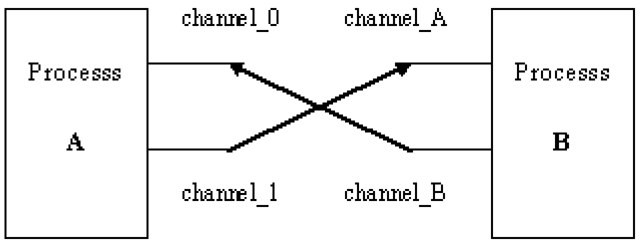


Figure 17: Channels and Connections

3.2.4 Value

The Value in RDT is translated into the information component in a Java message. A message is an information unit which is composed of the following four fields:

m.sender	The name of the process sending message *m*.
m.receiver	The name of the process receiving message *m*.
m.channel	The name of the channel through which the message *m* is passing.
m.type	The information component of message *m*.

3.2.5 Before state and After state

The Before state is the condition or part of the condition which must be satisfied to start the event. For a Create or Write event, the Before state is the condition which must be satisfied to invoke the event.

```
public void RED2(){
if(state=="RED2")
  S_ED_2();
}
public void S_ED_2(){
   try{p1.elected.send(new Message("V2",this,p1,"Ps_elected"));
   transformState("Boss2");
} catch(Exception e){System.out.println(name + " : S_ED_2- send
error");}
}
```

Figure 18: Java methods for a Create (or Write) event and its Before state

For a Read event, the Before state is a condition along with the other conditions.

```
public void run(){
new Thread(){public void run(){
try{for(;;){
Message m=(Message)elected.receive();
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V1" && state =="initial")
  R_Elected_1(m.writer,m.type,"R_Elected_1");
.....
}} catch(Exception e){System.out.println(name + ": demultiplex
error");}}}.start(); }
public void R_Elected_1(Process from, String message, String
current_state){
...
transformState("RED1");
RED1();
}
```

Figure 19: Java methods for a Read event and its Before state

After the successful completion of an event, the state will be translated from a Before state to an After state.

3.2.6 Process Instance

Each instance of a type of process has a unique instance identifier. The DTD definition syntax for a Process Instance definition is:

```
<!ELEMENT ProcInstance EMPTY>
    <!ATTLIST ProcInstance
            Name (p0 | p1 | p2 | p3) #REQUIRED
            Type (participant0 | participant1 | participant2 | participant3) #REQUIRED>
```

A new object of a specified process is created by defining both the object name, and its identifier value, the same as the instance name. This object is then ready to start.

```
<ProcInstance Name="p0" Type="participant" />
participant p0=new participant("p0");
```

Figure 20: Translation of an Instance of one Process in RDT into Java code

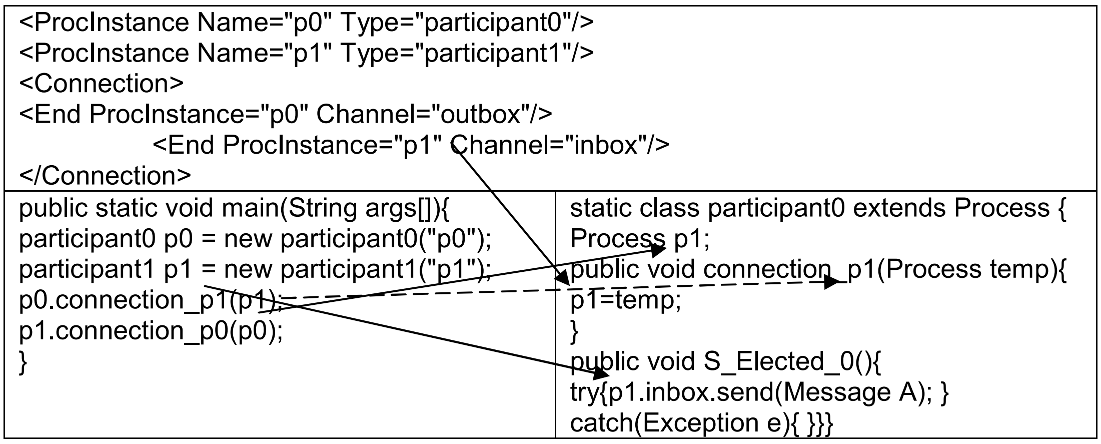
3.2.7 Connection

Two Channels build one Connection (see Figure 17) when they are connected. The DTD definition syntax for the Connection definition is:

```
<!ELEMENT Connection (End+)>
      <!ELEMENT End EMPTY>
      <!ATTLIST End
            ProcInstance (p0 | p1 | p2 | p3) #REQUIRED
            Channel (inbox | outbox) #REQUIRED
      >
```

The connection is built in two steps by making a one-way connection for one instance and then making another one-way connection for the other instance. So the two-way connection is set. An example is given in Figure 21. Observed from the RDT specification, we know that *p0* is an instance of the process *participant0*; and *p1* is an instance of the process *participant1*; *p0* has one channel named *outbox*, and *p1* has one channel named *inbox*; the channel *inbox* and the channel *outbox* are connected, so the connection between *p0* and *p1* is set up. This means that there will be possible communication between *p0* and *p1* via the connection set up by the channel *inbox* and the channel *outbox*. Experience from the development of RDT models suggests that two instances of the process can communicate via at least one connection set up by two channels, but for one specified channel of one process involved in this communication, it must be connected to a fixed channel, rather than more than one channel, of another process. It is helpful to analyse communication behaviour and also consider an intelligent mechanism which makes the translation from the RDT model to a Java object model efficient and accurate. In Java, a new object *p0* of the *participant0* class is created and a new object *p1* of the *participant1* class is created in the main class. *p0* calls up the *connection_p1()* method in the *participant0* class and refers to *p1* as the argument, knowing that the object *p1* in the *participant0* class is a referent of the instance *p1* of the process *participant0*. The connection from *p0* to *p1* is set up, and *p0* is ready to send messages to *p1*. The connection from *p1* to *p0* can also be set up in the same way. *p0* knows that (if successful) some messages will only be received by *p1*'s channel *inbox* and not other channel(s), if such messages are sent out to *p1* through the *outbox* channel successfully.

An example of implementation of Connection in Java is:



```
<ProcInstance Name="p0" Type="participant0"/>
<ProcInstance Name="p1" Type="participant1"/>
<Connection>
<End ProcInstance="p0" Channel="outbox"/>
            <End ProcInstance="p1" Channel="inbox"/>
</Connection>
```

```
public static void main(String args[]){
participant0 p0 = new participant0("p0");
participant1 p1 = new participant1("p1");
p0.connection_p1(p1);
p1.connection_p0(p0);
}
```

```
static class participant0 extends Process {
Process p1;
public void connection_p1(Process temp){
p1=temp;
}
public void S_Elected_0(){
try{p1.inbox.send(Message A); }
catch(Exception e){ }}}
```

Figure 21: Translation of the Connection in RDT into Java code

The above example provides further details of the differences in communication using the RDT channel and using the Java message queue. In Figure 22, for *p0*, the message will be send out by the channel inbox to *p1*'s *outbox* through the connection between *inbox* and *outbox*. For *p1*, the message will be sent out by the channel *outbox* to *p0*'s *inbox* through the connection between *inbox* and *outbox*.

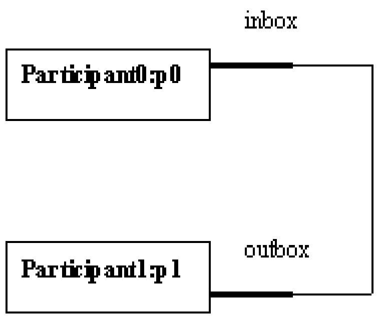


Figure 22: Connection in RDT

In the generated Java, the object *p0* calls the `send()` method of *p1*'s message queue *outbox* and makes it produce a new message. The message queue *outbox* holds this new message, and the object *p1* reads it when it calls the `receive()` method of the message queue *outbox*. The object *p1* calls the `send()` method of *p0*'s message queue *inbox* and makes it produce a new message. The message queue *inbox* holds this new message, and the object *p0* reads it when it calls the `receive()` method of the message queue *inbox*.

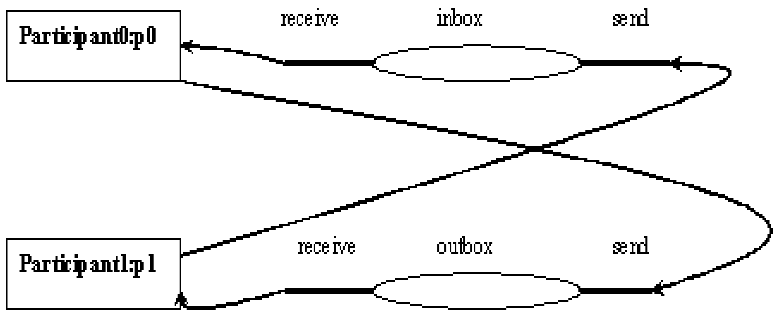


Figure 23: Connection in Java

3.3 Synchronization

In RDTtoJava, the modeller can make the choice between synchronous and asynchronous communication by setting different parameters for the length of the buffer. When the length is set to 0, it means that communication will be synchronous, the sender blocks until the message is sent, and the receiver is suspended until the message is received. When the length of the message queue is initialized to any positive value other than 0, the communication will be asynchronous. If the communication is asynchronous, the receiver explicitly fetches the messages from the destination by calling the receive method. The receive method can block until a message arrives or can time out if a message does not arrive within a specific time limit. RDTtoJava has two separate types of message queue that support asynchronous communication and synchronous communication. Asynchronous communication uses the produce-consume style message queue. Synchronous communication uses the acquire-release style message queue.

```
static class MessageQueue{
int entries;
int maxEntries;
String name;
Message[] elements;

public MessageQueue(String n, int m){
name=n;
maxEntries=m;
elements=new Message[maxEntries];
entries=0;
}

synchronized void send(Message x)
throws InterruptedException{
while(entries==maxEntries)wait();
elements[entries]=x;
entries=entries+1;
System.out.println("send("+x+")");
notify();
}

synchronized Message receive() throws
InterruptedException{
while(entries==0)wait();
Message x; x=elements[0];
for(int i=1; i<entries; i++){
elements[i-1]=elements[i];
}
entries=entries-1;
System.out.println("receive("+x+")");
notify();
return x;
}
}
```

Figure 24: Message queue for asynchronous communication

In asynchronous communication, the message queue (see Figure 24) is a buffer. Each queue has a specified name and a stack whose size is defined and limited, so it needs a counter to count how many messages are in the queue. The `send()` method is used to produce a new message, which should be kept in the message queue. Once one new message is produced and accepted by the queue, the counter will increase by one. If the queue is full, it will be blocked. The `receive()` method is used to consume messages. If one message is read from this queue, the counter will decrease by one. If the queue is empty, it is blocked until it is woken up. The message flow is in first-in-first-out (FIFO) order.

In synchronous communication, the message queue (see Figure 25) is controlled. Each queue has a specified name. The mechanism of the `send()` method is as follows. The new message is created first, and the message queue woken up to let this message in, before the queue is blocked again. The mechanism of the `receive()` method is as follows. The message queue is woken up and then reads that message. After that, the message queue is blocked again.

```
static class MessageQueue{
String name;
boolean sendFlag, receiveFlag;
Message share;

public MessageQueue(String n, int m){
name=n;
sendFlag=false;
receiveFlag=false;
}

synchronized void send(Message x)
throws InterruptedException{
sendFlag=true;share=x;
notifyAll();
System.out.println("send("+x+")");
    while(!receiveFlag) wait();
    receiveFlag=false;
}

synchronized Message
receive()throws InterruptedException{
receiveFlag=true;
notifyAll();
while(!sendFlag) wait();
Message x; x=share;
System.out.println("receive("+x+")");
sendFlag=false;
return x;
}
}
```

Figure 25: Message queue for synchronous communication

The Java programming language has special support for multi-threaded programming [Artho and Biere 2001]. Non-trivial multi-threaded programs require synchronization between threads. The classic cases are a semaphore [Dijkstra 1965], and a monitor [Hansen 1975]. RDTtoJava supports the case that one process has many threads that each receive and send messages. Figure 26 shows that a couple of (synchronous or asynchronous) message queues will be used in this system. For the process *participant0*, it has two message queues, which are message queue *election* and message queue *elected*.

```
static class Process extends Thread {
MessageQueue startElection;
MessageQueue Ps_election;
MessageQueue election;
MessageQueue Ps_elected;
MessageQueue elected;
}

static class participant0 extends Process
{
public participant0 (String name){
election=new
MessageQueue("election",10);
elected=new
MessageQueue("elected",10);
this.name =name;
    this.start();
    }
```

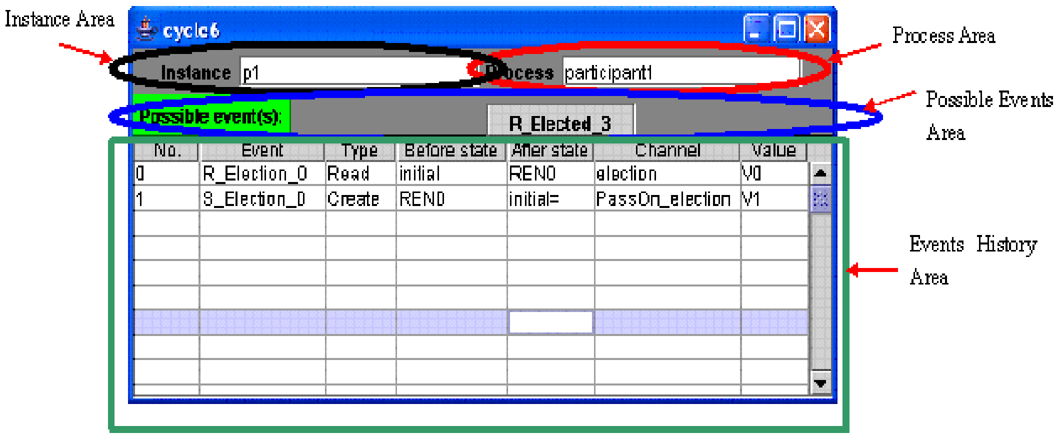
```
public void run(){
new Thread(){public void run(){
try{for(;;){
Message m=(Message)election.receive();
…
}}catch(Exception e){ }}}.start();

new Thread(){public void run(){
try{for(;;){
Message m=(Message)elected.receive();
.. .
}}catch(Exception e){ }}}.start();
}
}
```

Figure 26: Java code for the multi-threaded process

3.4 The Traceable GUIs

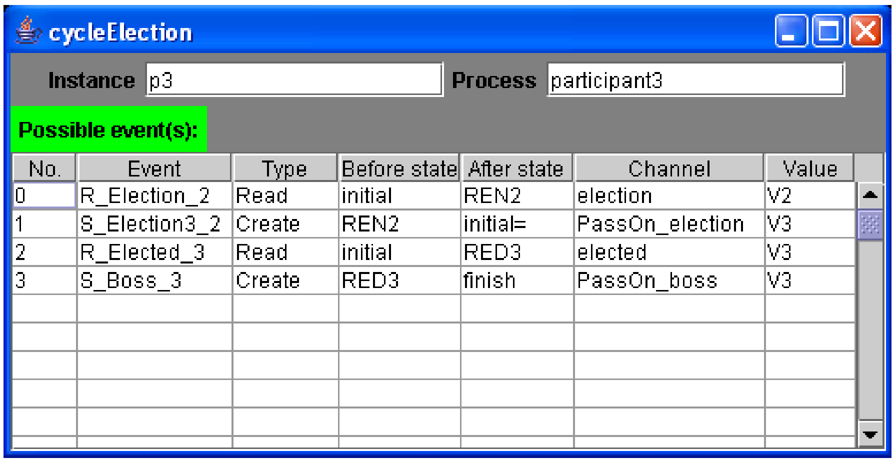
My design offers two ways to observe the behaviour of the model, one of which is a Java GUI (see Figure 27) for single instance of the processes. The other is the system output to the terminal window.



No.	Event	Type	Before state	After state	Channel	Value
0	R_Election_0	Read	initial	REN0	election	V0
1	S_Election_0	Create	REN0	initial=	PassOn_election	V1

Figure 27: The GUI for each instance of the process

In the GUI, the model name is shown as the title. The process type is shown in the Process area, and the instance name is shown in the Instance area. All possible events that will happen next are listed as buttons, whose names are the same as the event's name, in the Possible Events area. In the Event History area, the events that have been executed are listed in sequence. It is possible for the user to check the trace of the events and all corresponding information for each event. The After state listed in the last event is the current state for the instance. An example is given in Figure 28.


cycleElection

Instance p3 Process participant3

Possible event(s):

No.	Event	Type	Before state	After state	Channel	Value
0	R_Election_2	Read	initial	REN2	election	V2
1	S_Election3_2	Create	REN2	initial=	PassOn_election	V3
2	R_Elected_3	Read	initial	RED3	elected	V3
3	S_Boss_3	Create	RED3	finish	PassOn_boss	V3

Figure 28: An Example GUI

The system output to the terminal window is also supplied to show the model process. As the information for each instance of the process is shown in the single GUI, the window information shows all the information, such as the behaviour of the single instance of the process, the messages passing, and the sequence of events of the model, etc. An example is given below.

```
p2 : RED3
receive(V3 from p1 to p2 via Ps_elected)
p2's Event is: R_Elected_3 and read V3 from p1
p2 : S_ED_3
send(V3 from p2 to p3 via Ps_elected)
p2 : Boss3
receive(V3 from p2 to p3 via Ps_elected)
p3's Event is: R_Elected_3 and read V3 from p2
p3 : IMBoss
```

3.5 Branching Execution of Events

The possible flows of the events in RDT models are: sequential and branching (see Figure 29). Their behaviour and properties are:

1. Sequential events
    - Only one event shares the same Before state;
    - Execute all events in sequence, and
    - An activity cannot start until previous events in sequence are complete.
2. Branching events
    - Two or more events share the same Before state, and
    - Execute the next activity in the flow which satisfies given conditions.

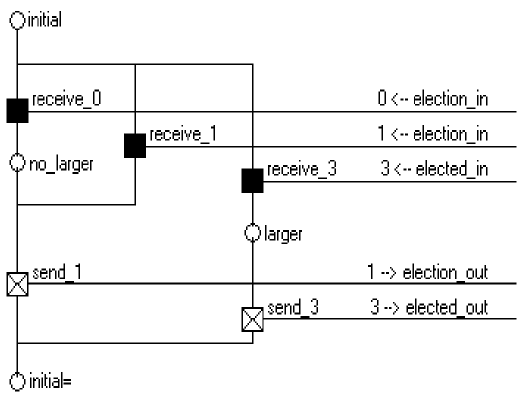


Figure 29: RDT model with branching events

The approach to supporting the sequential execution of activities is as follows. For a single Read event, the conditions that must be satisfied are: the Before state and the message the process receives. For a single Write event, the condition that must be satisfied is the Before state. The condition for the Create event is the same as the condition for the Write event.

The discussion on the branching execution of events within the process is based on three types of case. In Case 1, the type of all events sharing the same Before state is Read. In Case 2, the type of all events sharing the same Before state could be Write, Create or both. Case 3 is a synthesis of Case 1 and Case2. This means that at least one Read event and at least one Write or Create event share the same Before state.

The approach to Case 1 is to pick only one of all Read events according to the rules as follows:

1. Only the Read event(s), for which the conditions are true, are listed on the GUI;
2. The user can make the decision to implement only one event by pushing one button, whose name is the same as the name of the corresponding event.

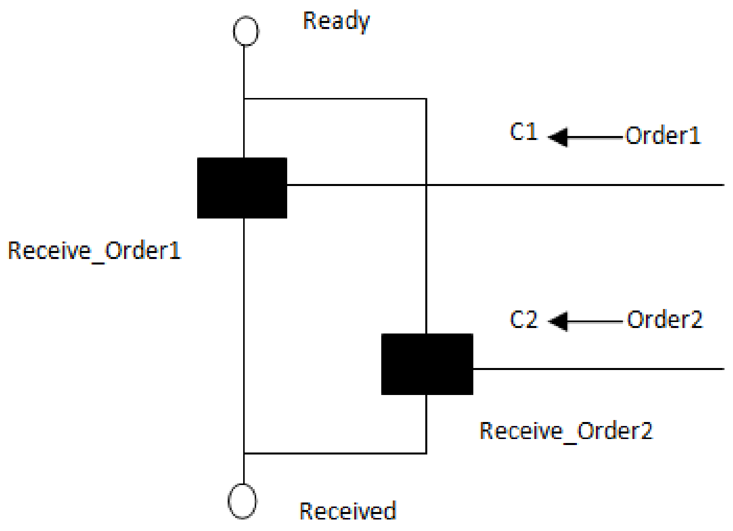
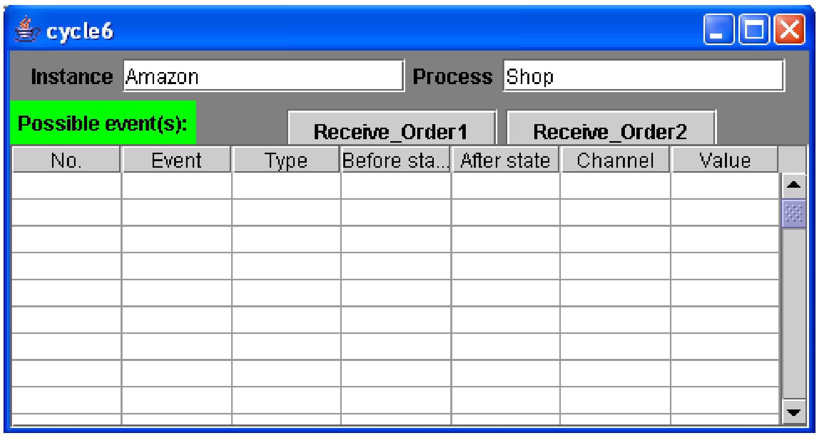


Figure 30: Path selection: Case 1

An example of Case 1 (see Figure 30) is given here. There is one possible Read event *Receive_Order1* and another possible Read event *Receive_Order2* both sharing the same Before state *initial*. For *Receive_Order1*, a message *order1* is expected to be received through channel *C1*. For *Receive_Order2*, a message *order2* is expected to be received through channel *C2*. The solution to this case is shown in Figure 31 and its GUI in Figure 32.

```
if (channel C1 ready & Before state=="Ready" && Message Order1)
goto Receive_Order1;
if (channel C2 ready & Before state=="Ready" && Message Order2)
goto Receive_Order2;
```

Figure 31: The solution to Case 1



No.	Event	Type	Before sta...	After state	Channel	Value

Figure 32: An example GUI for Case 1

The approach to Case 2 is to pick only one of all the Write and Create events according to the rules as follows:

1. Only the Write or Create event(s), for which the conditions are true, are listed on the GUI;
2. The user can make the decision to implement only one event by pushing one button, whose name is the same as the name of the corresponding event.

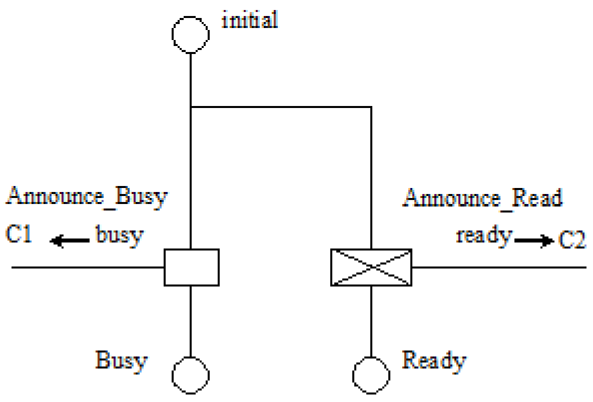
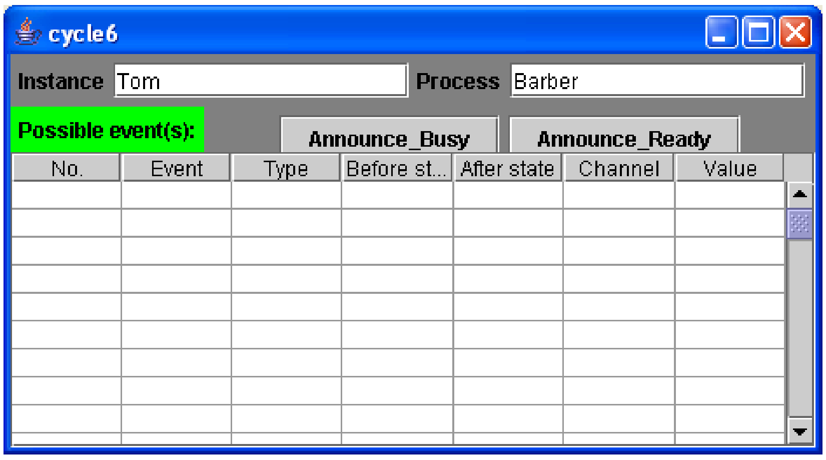


Figure 33: Path selection: Case 2

An example of Case 2, see Figure 33. There is one possible Write event *Announce_Busy* and another possible Create event *Announce_Ready*, both sharing the same Before state *initial*. For *Announce_Busy*, a message *busy* is expected to be sent out through channel *C1*. For *Announce_Ready*, a message *ready* is expected to be sent out through channel *C2*. The solution to this case is shown in Figure 34 and its GUI in Figure 35.

```
if (channel C1 ready & Before state=="initial")
goto Announce_Busy;
if (channel C2 ready & Before state=="initial")
goto Announce_Ready;
```

Figure 34: The solution to Case 2



Instance	Tom	Process	Barber

Possible event(s):

No.	Event	Type	Before st...	After state	Channel	Value

Figure 35: An example GUI for Case 2

Case 3 is one special and complex case of the branching execution of activities. In this case, the Read events, the Create events and Write events could share the same Before state. The rules to select one event among the events are as follows:

1. Only the Write, Create or Read event(s), for which the conditions are true, are listed on the GUI;
2. The user can make the decision to implement only one event by pushing one button, whose name is the same as the name of the corresponding event.

An example of Case 3, see Figure 36. There is one possible Write event *Take_Break* and another possible Read event *Continue* both sharing the same Before state *initial*. The solution to this case is shown in Figure 37 and its GUI in Figure 38.

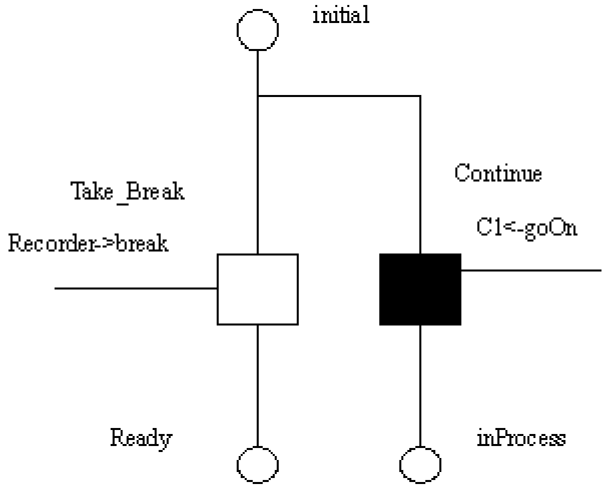
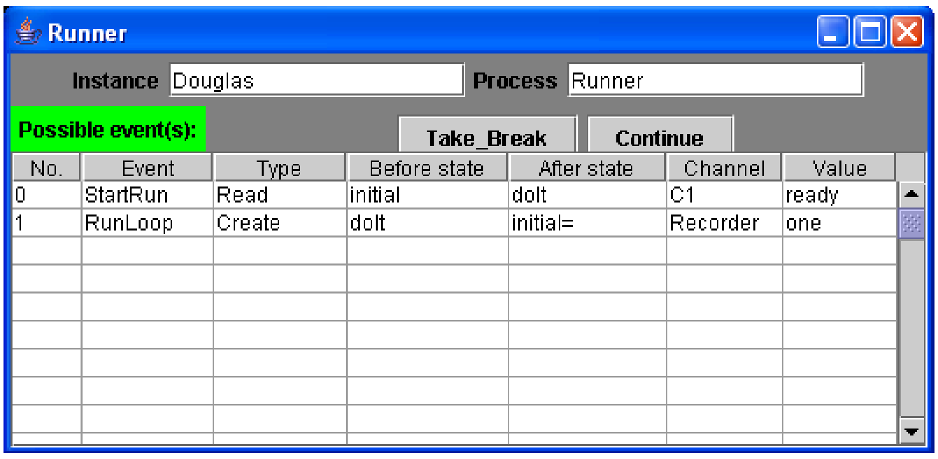


Figure 36: Path selection: Case 3

```
if (channel Recorder ready & Before state=="initial")
goto Take_Break;
if (channel C1 ready & Before state=="initial" & (Message) goOn)
goto Continue;
```

Figure 37: The solution to Case 3



Instance Douglas Process Runner

No.	Event	Type	Before state	After state	Channel	Value
0	StartRun	Read	initial	doIt	C1	ready
1	RunLoop	Create	doIt	initial=	Recorder	one

Figure 38: An example GUI for Case 3

3.6 Exception Handling

An exception will be raised when an error occurs during initiation or execution of a communication action. The actions taken in exception handling communicate the exception to the process.

For a specified Read event, the messages will be received through one channel, which is a thread in the Java program. If an unexpected message has been received through that channel, the process will refuse to accept it and will not start this Read event. The system will display an error message. Any exceptions that might be thrown will be caught.

```
new Thread(){public void run(){
try{for(;;){
Message m=(Message)elected.receive();
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V0" && state =="initial")
R_Elected_0(m.writer,m.type,"R_Elected_0");
. . .
}}catch(Exception e){System.out.println(name + ": demultiplex error");}}}.start();
```

Figure 39: Java code for the Read event exceptions

For a specified Write or Create event, the messages will be sent through one channel, which is a thread in the Java program, to an expected process. If the message is received by the receiver through a channel unsuccessfully, the system will display an error message.

```
public void Send_Event_A(){
try{p1.elected.send(new Message("V0",this,p1,"Ps_elected"));
transformState("initial");
}
  catch(Exception e){System.out.println(name + " : Send_Event_A - send error");}
}
```

Figure 40: Java code for the Write/Create event exceptions

Another point should be noticed. In RDT, an instance of Process can send a message to a channel, and it can receive this message through the same channel. However, as Java thread does not support this, my solution to this problem is to use another channel to receive this message. In the next part of this work, RDTtoWS, the same method is used as it also uses Java.

3.7 Conclusion

This chapter presented the rules employed by the RDTtoJava tool, mapping an RDT model to a Java threaded application. The RDTtoJava supports:

1. Java multi-threaded communication;
2. the complex branching paths of activities; and
3. asynchronous/synchronous communication.

The difference in behaviour between RDT and Java was discussed.

The user can complete the transformation after entering each required option (see Figure 41).

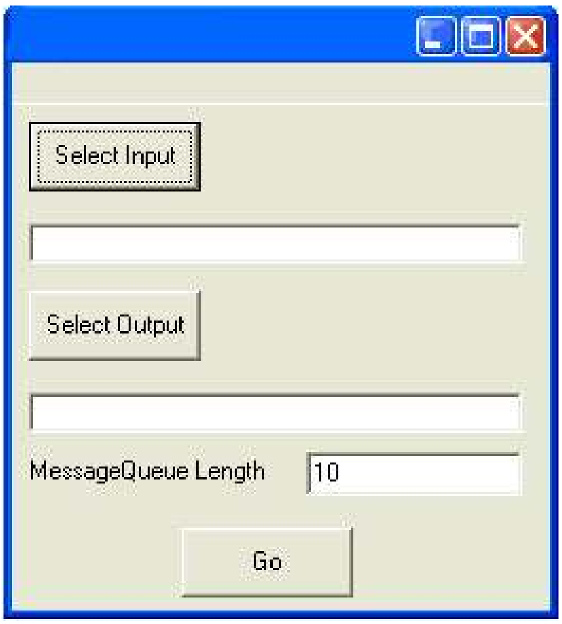


Figure 41: The RDTtoJava window

1. Select Input: The address of the XML file for the RDT model, which has been built.
2. Select Output: A new Java file should be created to store the Java multi-threaded application, which will be generated by the RDT model.
3. MessageQueue Length: The message queue length may be defined as any non-negative integer. There are two types of communication available:

synchronous and asynchronous. If the queue length is set to 0, the communication will be synchronous; if the queue length is set to a positive integer, the communication will be asynchronous, and the length of the all message queues used in the asynchronous communication will be that positive number.

4. Go button: After entering the above options and pushing the button, the user will successfully obtain the expected Java multi-threaded application, but if one or more options are missing, an error message will be displayed.

4 RDTtoWS

This chapter focuses on how to build a distributed system based on the RDT model using Web services technology. As we know, the RDT system can handle both synchronous and asynchronous communication, and the current Web service technologies, which can handle both communication, is limited as discussed in section 1.4.3. My approach here is to build the system separately: one is to build RPC Web services using JAX-RPC based on the RDT model; the other is to convert an RDT model to an asynchronous communication application using JMS Web service.

4.1 Build Web Services based on RDT Models with JAX-RPC

This section presents how to build a synchronous Web service based on the RDT model. Synchronous services are characterized by the client invoking a service, and then waiting for a response to the request. Web services that reply to synchronous communication are usually RPC-oriented. Generally, I consider using an RPC-oriented approach for synchronous Web services. I will introduce how to translate the RDT language into Java code for synchronised Web services, and then introduce the solution of some issues that occurred during system building.

4.1.1 Mapping of RDT language to JAX-RPC Web service source code

In RDT, a model includes at least one process, which sends messages to another process, or receives messages from another process, or both. Web service is based on the client/server model. So a process in the RDT model has the potential to be a server, or a client, or both. The application architecture I design to develop an RDT as a Web service is to make each process both a server-side and a client-side entity.

A typical JAX-RPC application architectural model consists of the server-side and the client-side. JAX-RPC service represents a business component that can be implemented in Java, or generated from existing Java classes, or from a WDSL document. In a J2EE environment, it can be implemented as a servlet, a stateless session bean, or a message-driven bean. During deployment, the JAX-RPC service is assigned to one or more service endpoints and then is configured with a transport protocol binding. For instance, a JAX-RPC can be bound to HTTP and all the messages are exchanged as HTTP-based requests and responses using its assigned endpoint. The JAX-RPC services do not dictate that it has to be accessed by a JAX-RPC client and thus a non-Java client running on a heterogeneous environment can access it.

JAX-RPC service client represents a JAX-RPC-based service client that can access a service. The service clients are independent of the target implementation on the service provider. This means that the accessed service can be a service implemented using a Java platform, or a SOAP compliant service running on a non-Java platform. To support these client scenarios, JAX-RPC defines a variety of client mechanisms, dynamic proxies, and dynamic invocation. The JAX-RPC service clients can import WSDL exposed by a service provider and can generate a Java-based client class to access the service.

The key steps for creating a JAX-RPC-based Web service based on RDT model using a Tomcat-based [Apache 2002] environment are as follows.

1. Develop the remote interface of the service
2. Create the implementation class of the remote interface
3. Configure the service
4. Set up the environment and compile the source code
5. Generate the server-side artefacts (ties) and the WSDL document

6. Package and deploy the service
7. Test the service deployment and the WSDL
8. Generate the client stubs and package as a client JAR.

The following section will describe how to get a JAX-RPC Web service from an RDT model. The events trace GUI employed here is the same as in the RDTtoJava tool, see Figure 38.

4.1.1.1 Interface and implementation

The programming model of JAX-RPC is like EJBs and Java RMI, in that the details of the underlying protocols are hidden behind Web service stubs. A stub implements the same interface as the Web service that exists remotely, and it communicates with a Web service tie on the server. The tie calls the methods of a Web service, and communicates the return value, and any exceptions encountered, back to the client through the stub.

```
package participant3;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface participant3_IF extends Remote{
public void createMessage(String type, String writer, String reader, String channel) throws
RemoteException;
}
```

```
package participant3;
. . .

public class participant3 extends JFrame implements participant3_IF
{ // the service method implementations
public void createMessage(String type, String writer, String reader, String channel) {
//messages sending
// exception handling and a warning window exposed
}
}
```

Figure 42: JAX-RPC-based Web Service Interface

As illustrated in Figure 42, the first Java source file that we need to create is the Web service interface for the model. The file is called participant3_IF.Java. In this file, the public attributes and the public methods are defined. The method `createmessage()` is defined here as the public method, and its function should be clarified in the implementation file. The next file that we need to create is the class that implements the Web service interface participant3_IF.Java. The functions of the public method `createMessage()` is given.

4.1.1.2 Process

The DTD definition syntax for the definition of Process in RDT is:

<Process Name="participant0"> </Process>
static class participant0 extends Process{…}

Figure 43: Development of system model specification in RDT into Java Web services code

A Process in RDT is a collection of clients and services. In a Process, an instance of such a collection defines its behaviours and some of its properties.

4.1.1.3 Event

There are three types of Event in the RDT model: Create, Read and Write. In the Web service, when the client calls the Web service method, the server side sends out return values (messages), and any exceptions encountered, back to the client through the stub. The client receives the messages. So the Create event and the Write event occur on the server side, and the Read event occurs on the client side. Since the RDT model is mostly a communication model, most Web services are both server side and client side, and any side could perform a service to the others.

It is important to distinguish the different thinking behind RDT and Web service. In RDT, for the Write event, an instance of a process creates a new channel and then sends a value through this channel. For the Write event, an instance of a process sends a value through an existing channel. For the Read event, an instance of a process receives a value through a channel that is connected with another channel through which the value came. In the Web service, the client should know which service is available through the HTTP, and which port on the service side is open for the client to call the Web service methods. This means that the service is available first, and ready for the client to call its Web service methods. The server side sends the return value, generated from the method call by the client, and the exceptions encountered, back to the client. Here the client does not need to expose a port to receive the return value and the exceptions.

4.1.1.3.1 Read event

In my Java Web service, for the Read event, the translation is completed in two steps. The first step is to clarify the conditions for a Read event to occur. The second step is to complete the action for this Read event. The conditions are as follows:

1. the message queue that this event uses to receive the message is ready;
2. the Before state for this event should be satisfied; and
3. the message received should be as expected.

The completion of the Read event is as follows:

1. call the method for this event; and then
2. change the object state from Before state to After state.

If an instance of a Process has at least one Read event, this instance will consume messages, and it is a client. This client must know where the message it will consume is from. The connection issue will be discussed later. The `getMessage()` method is called synchronously whenever a message arrives.

```
<Event Name="R_Election_0" Type="Read" Before="initial" After="REN0"
Channel="election" Value="V0"/>
```

```
package participant0_p0;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface participant0_IF extends Remote
{
public void getMessage(Message m) throws RemoteException;
}
```

```
package participant0_p0 ;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.JAXRPCException;
import javax.xml.namespace.QName;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.ParameterMode;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.awt.Font;
import java.awt.FontMetrics;
import javax.swing.*;
import java.awt.Graphics;
public class participant0_p0_Impl extends JFrame implements participant0_IF
{
static class Message {
. . .
Message (String t, Process p, Process r, String c){
. . .
        }

        }
static class participant0 extends Process{
public void createConnection_Participant_Tr(){
        try {
participant1_p1_factory = ServiceFactory.newInstance();
participant1_p1_service = (Service)participant1_p1_factory.createService(new
QName(participant1_p1_qnameService));
participant1_p1_port = new QName(participant1_p1_qnamePort);
participant1_p1_call = participant1_p1_service.createCall(participant1_p1_port);
participant1_p1_call.setTargetEndpointAddress("http://localhost:8080/participant1_p1-j
axrpc/participant_p1?wsdl");
participant1_p1_call.setProperty(Call.SOAPACTION_USE_PROPERTY, new
Boolean(true));
participant1_p1_call.setProperty(Call.SOAPACTION_URI_PROPERTY, "");
participant1_p1_call.setProperty(ENCODING_STYLE_PROPERTY, URI_ENCODING);
participant1_p1_call.setReturnType(null);
participant1_p1_call.setOperationName(new QName(BODY_NAMESPACE_VALUE,
"getMessage"));
participant1_p1_call.invokeOneWay(null);
} catch (Exception ex) {
ex.printStackTrace();
}
}

public void getMessage(Message m) {
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V0" && state =="initial")
R_Election_0(m.writer,m.type,"R_Election_0");
. . .
}

. . .
}

}
```

Figure 44: Translation of the Read event in RDT into Java Web services code

4.1.1.3.2 Write event

For the Write event, an instance of a Process will write a message over HTTP, and then change the state to After state. Considering the characteristics of Java Web service, a process sends a message over the HTTP. Here the definition of the message is different from the message in the RDT model. The message here includes the *value*, the *channel* in which the value is sent out, and other information. I will discuss it later. Transition of the messages is in 3.2.1.7. If an instance of a Process has a Write event, it supplies service that will send one or more messages.

```
<Event Name="S_Elected_0" Type="Write" Before="REN0" After="initial"
Channel="outbox" Value="elected"/>
```

```
public void setMessage(Message m) {
      message= m;
}

public void send_election(){
. . .
final Message m=new Message("outbox",this,p1,"elected");
ButtonS_send_election.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent e){
try{
transformState("initial");
}
catch(Exception f){System.out.println(name + " : send_election.error");}

displayTrace( "S_Elected_0", "Write", "REN0", "initial", "outbox", "elected");
ButtonS_send_election.setVisible(false);
}

});

}
```

Figure 45: Translation of the Write event in RDT into Java Web services code

4.1.1.3.3 Create event

```
<Event Name="start_election" Type="Create" Before="initial" After="election_start"
Channel="outbox" Value="election"/>
```

```
public void start_election(){
System.out.println(name+" : start_election");

traceTable.center.add(Buttonstart_election);
final Message m=new Message("election",this,p1,"outbox");
Buttonstart_election.addActionListener( new ActionListener(){
public void actionPerformed(ActionEvent e){
try{
transformState("election_start");
}
catch(Exception f){System.out.println(name + " : p1- send error");}
displayTrace("start_election", "Create", "initial", "election_start", "outbox", "election");
Buttonstart_election.setVisible(false);
}
});
}
```

Figure 46: Translation of the Create event in RDT into Java Web services code

The Create event will generate a message and a new channel in the RDT model. I have mentioned that the message is passed over HTTP in Java Web service and there is no thread-like approach employed here. So I only create a new message and this message will be delivered to the client when this service is called. The Before state is one condition which must be satisfied to start the event. The After state is transformed by another method when the event is completed.

4.1.1.4 Channel

I will discuss this issue together with Connection in 4.1.1.8.

4.1.1.5 Value

The same code employed as in RDTtoJava. See 3.2.4.

4.1.1.6 Before state and After state

The same code employed as in RDTtoJava. See 3.2.5.

4.1.1.7 Process Instance

Based on understanding of the RDT model, we know that each process's behaviour is defined, and any instance of such process will do the same thing.

Correspondingly, in our Web service, each instance of a process will have the same behaviour. If a Process has a Create event or a Write event, the instance of this Process is a service. If a Process has a Read event, the instance of this process is a service client. If a Process has a Read event and a Create or Write event, this instance of this process is a server/client. This is a combined side which sends messages and receives messages as well.

4.1.1.8 Connection

Two Channels build one Connection (see Figure 16) when they are connected. The DTD definition syntax for the Connection definition is:

```
<!ELEMENT Connection (End+)>
    <!ELEMENT End EMPTY>
    <!ATTLIST End
        ProcInstance (p0 | p1 | p2 | p3) #REQUIRED
        Channel (inbox | outbox) #REQUIRED
    >
```

As we know, JAX-RPC communicates over HTTP. A service offers messages, and a service client will consume the message transport over HTTP. The client should know the remote location of the service. The client will connect the service over HTTP to a specific port. For example, http://localhost:8080/ connects to http://www.google.co.uk/, and http://localhost:8000/ connects to http://www.ecs.soton.ac.uk/. When we use the RDT model to develop an application using Web services, the information of the Channel is unnecessary. We build the Channel information in the message, and the client will check this message when it receives the message as important identification information.

```
<ProcInstance Name="p0" Type="participant0"/>
<ProcInstance Name="p1" Type="participant1"/>
<Connection>
        <End ProcInstance="p0" Channel="Ps_election"/>
        <End ProcInstance="p1" Channel="election"/>
</Connection>
```

```
public void createConnection_participant0_p0(){
try {
participant0_p0_factory = ServiceFactory.newInstance();
participant0_p0_service = (Service)participant0_p0_factory.createService(new
QName(participant0_p0_qnameService));

participant0_p0_port = new QName(participant0_p0_qnamePort);

participant0_p0_call = participant0_p0_service.createCall(participant0_p0_port);

participant0_p0_call.setTargetEndpointAddress
("http://localhost:8080/participant0_p0-jaxrpc/participant0_p0?wsdl");
participant0_p0_call.setProperty(Call.SOAPACTION_USE_PROPERTY, new
Boolean(true));
participant0_p0_call.setProperty(Call.SOAPACTION_URI_PROPERTY, "");
participant0_p0_call.setProperty(ENCODING_STYLE_PROPERTY, URI_ENCODING);

. . .
} catch (Exception ex) {
ex.printStackTrace();
}
}
```

Figure 47: Channel and Connection in Web services code

For the Channel of an Instance of Process, the service client in Web service corresponds to a specific port. For example, the channel *outbox* corresponds to the port 8080, while the channel *Ps_election* corresponds to the port 8000.

The client throws an Exception when a JAX-RPC exception occurs. The exception details the reasons for the failure, which are related to JAX-RPC runtime-specific problems.

4.1.2 Deployment

4.1.2.1 Set up the environment and compile the source code

A CLASSPATH environment should be created that includes the JWSDP 1.0 class libraries for JAX-RPC and its supporting packages. Use Javac and compile the source code of the remote interface and the implementation.

4.1.2.2 Generate server-side artefacts (ties) and WSDL

Using the xrpcc tool, generate the service side artefacts and the WSDL document associated with the service. As a result, this generates the following:

- Client-side stubs and server-side tie class
- Serialization and deserialization classes representing the data-type mappings between Java primitives and XML data types
- A WSDL document
- Property files associated with the service.

4.1.2.3 Package and deploy the service

To pack a JAX-RPC service as a Web application, we need to create a WAR file that includes the following classes and other configuration files:

- Remote interface to the service
- Service implementation of the remote interface
- Serializer and deserializer classes
- Server-side classes created by *xrpcc*
- Property files created by *xrpcc*
- Other supporting classes required by the service implementation
- WSDL document classes
- Web application deployment description.

4.1.2.4 Test the service deployment and WSDL

To configure the service, a configuration file in XML, which provides information about the URL location of the WSDL, should be created. A sample code of a configuration file is below:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://Java.sun.com/xml/ns/jax-rpc/ri/config">
<service name="trigger_Tr_Service" targetNamespace="urn:Foo"
typeNamespace="urn:Foo" packageName="trigger_Tr">
<interface name="trigger_Tr.trigger_IF"/>
</service>
</configuration>
```

Figure 48: Service configuration using WSDL

4.1.2.5 Generate client-side artefacts (stubs)

Use the *xrpcc* tool to generate the stubs and tie classes, the WSDL document associated with this service, and the property files required by the JAX-RPC runtime environment. In the typical scenario, e.g. Windows, the xrpcc tool can be executed as a command line utility as follows:
xrpc -classpath %CLASSPATH% -keep -both -d build\classes serviceconfig.xml

4.2 Mapping of RDT Language to JMS-implemented Web Service Source Code

4.2.1 Develop from RDT to Java Message Service

This section will cover how asynchronous web applications are developed JMS-based Web service, based on the RDT model.

4.2.1.1 Interface

Here is the interface definition for the remote class. The queue and the behaviour of the queue are defined in the interface.

```
package participant3;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface participant3_IF extends MessageProducer{
Queue getQueue() throws JMSException;
public void send(Message message) throws JMSException, MessageFormatException,
InvalidDestinationException;
}
```

```
package participant3;
import java.util.StringTokenizer;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.jms.TopicConnectionFactory;
import javax.jms.QueueConnectionFactory;
import javax.jms.Topic;
import javax.jms.Queue;
import javax.jms.QueueReceiver;
import javax.jms.Sessions;
import javax.jms.TextMessage;
. . .

public class participant3 extends JFrame implements participant3_IF,
javax.jms.MessageListener
{ // the service method implementations
public void createMessage(javax.jms.Message message) {
//messages sending
// exception handling and a warning window exposed
try{
}catch (javax.jms.JMSException jmse){
jmse.printStackTrace();
}
}
}
```

Figure 49: JMS-based Web Services Interface

4.2.1.2 Process

The DTD definition syntax for the definition of Process in RDT is:

```
<Process Name="participant0">
</Process>
```

```
public class participant0 implements javax.jms.MessageListener{…}
```

Figure 50: Development of system model specification in RDT into JMS-based Web services code

For each Process, the MessageListener is implemented to send and invoke messages.

4.2.1.3 Event

For the three types of Event, mapping from RDT to JMS Web service is discussed below.

In JMS, the simplest type of message is the `Javax.jms.Message`, which serves as the base interface for the other message types. There are other types of message, which extends Message: `TextMessage`, `ObjectMessage`, `BytesMessage`, `StreamMessage`, and `MapMessage`. An `ObjectMessage` object is used to send a message that contains a serializable object in the Java programming language [Monson-Haefel and Chappell 2001]. In RDT, the content of messages passing between the instances of processes is defined in Value (see 2.1.1.3). In RDTtoJava, the messages communicating between the objects are defined in the self-defined object called `Message`. Instead of just the Value of RDT, this object contains the information on the message sender, the expected receiver of the message, the name of the channel through which the messages pass, and the Value. In asynch-RDTtoWS, the `ObjectMessage` is used to define a JMS message. An object of `ObjectMessage`, called `Message`, will be passed by the server to the queue, and the receivers will get this object from the queue.

4.2.1.3.1 Read event

When an instance of the Process completes one Read event, it will receive a message. In JMS, the receive (Read event) program performs the following steps:

1. Perform a JNDI API lookup of the QueueConnectionFactory and queue
2. Create a connection and a session
3. Create a QueueReceiver
4. Start the connection, message delivery begins
5. Receive the messages sent to the queue, until the end-of-message-stream control message is received
6. Close the connection in a final block, automatically closing the session and QueueReceiver.

The QueueSession is created by using the `createQueueSession()` method on the QueueConnection object. A receiver is created by using the `createReceiver()` method on the QueueSession object. When an instance of a Process has one or more channels, corresponding queues will be created in JMS.

```
<Event Name="R_Election_0" Type="Read" Before="initial" After="REN0"
Channel="election" Value="V0"/>
```

```
package participant3;
import java.util.StringTokenizer;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.jms.TopicConnectionFactory;
import javax.jms.QueueConnectionFactory;
import javax.jms.Topic;
import javax.jms.Queue;
import javax.jms.QueueReceiver;
import javax.jms.Sessions;
import javax.jms.TextMessage;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.awt.Font;
import java.awt.FontMetrics;
import javax.swing.*;
import java.awt.Graphics;

public class participant0_p0_Impl extends JFrame implements
javax.jms.MessageListener
{
static class Message {
. . .
Message (String t, Process p, Process r, String c){
. . .
     }

     }
static classs participant0 extends Process{

private javax.jms.TopicConnection tConnect = null;
private javax.jms.TopicSession tSession = null;
private javax.jms.TopicPublisher tPublish = null;

private javax.jms.QueueConnection qConnect = null;
private javax.jms.QueueSession qSession = null;
private javax.jms.Queue receiveQueue = null;

private javax.jms.Topic messageTopic = null;

public static void main (String args []){

try{
receiveMessage (message);
} catch (java.io.IOException i) {
i.printStacktrace ();
}
}

public void createConnection_Participant_Tr(){
        try {
TopicConnection tFactory = null;
QueueConnectionFactory qFactory = null;
InitialContext jndi = null;

Properties env = new Properties();
jndi = new InitialContext(env);

tFactory = (TopicConnectionFactory)jndi.lookup();
messageTopic= (Topic) jndi.lookup("inbox");
tPublish = tSession.createPublisher(messageTopic);

QueueReceiver qReceiver = qSession.createReceiver (receiveQueue);
qReceiver.setMessageListener (this);

//start the connection;
qConnect.start();
tConnect.start();
} catch (Javax.jms.JMSException jmse){
jmse.printStackTrace();
System.exit(1);
} catch (Javax.naming.NamingException jne){
jne.printStackTrace();
System.exit(1);
}

}

public void receiveMessage(Message m) {
try
{
javax.jms.Stream Message m=tSession.createStreamMessage();
m.setJMSReplyTo(receiveQueue);
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V0" && state =="initial")
R_Election_0(m.writer,m.type,"R_Election_0");
. . .
} catch (javax.jms.JMSException jmse){
jmse.printStackTrace();
}
}
. . .
}

}
```

Figure 51: Translation of the Read event in RDT into JMS-implemented Web services code

4.2.1.3.2 Create event

The sending program performs the following steps:

1. Perform a Java Naming and Directory Interface (JNDI) API lookup of the *QueueConnectionFactory* and queue
2. Create a connection and a session
3. Create a *QueueSender*
4. Create an ObjectMessage
5. Send one or more messages to the queue
6. Send a control message to indicate the end of the message stream
7. Close the connection in a final block, automatically closing the session and QueueSender.

```
<Event Name="start_election" Type="Create" Before="initial"
After="election_start" Channel="outbox" Value="election"/>
```

```
public void onMessage(javax.jms.Message m){
try{
ObjectMessage objectMessage= (ObjectMessage)m;
Message m=(Message)objectMessage.getObject();
if (state=="initial")
start_election(m);
else if (state=="elected")
. . .

}catch (java.lang.RuntimeException e){
e.printStackTrace();
}
}

private void start_election(javax.jms.Message message){
try{
    System.out.println(name+" : start_election");

    traceTable.center.add(Buttonstart_election);
    Message m=new Message("election",this,p1,"outbox");

    ObjectMessage objectMessage= session.createMessage();
    ObjectMessage.set(m);

    Queue outbox=(Queue)message.getJMSReplyto();
    outbox.send(objectMessage);

    outbox_qSendder =outbox_qSession.createSender(outbox);

    outbox_qSender.send(objectMessage,
    javax.jms.DeliveryMode.PERSISTENT,
    javax.jms.Message.DEFAULT_PRIORITY,
    1800000);

ButtonS_ start_election.addActionListener( new ActionListener(){
public void actionPerformed(ActionEvent e){
try{
transformState("election_send");
}
catch(javax.jms.JMSException e){System.out.println(name + " :
start_election.error");}

displayTrace( " start_election", "Create", "initial", " election_start",
"outbox", "election");
ButtonS_ send_election.setVisible(false);
}

});

}catch(javax.jms.JMSException e){
}
}
```

Figure 52: Translation of the Create event in RDT into JMS-implemented Java Web services code

4.2.1.3.3 Write event

Create event will create a new channel and then use this channel, and the Write event will use the created channel. In JMS, we know the channel in RDT is the queue. So, a Create event in JMS will create a new queue and use it until closed, and a Write event in JMS will use an opened queue.

4.2.1.4 Channel

A channel in RDT, for both Receive and Write/Create, is a queue, and I will now describe how to open, use and close a queue.

1. Declare a queue

```
<ProcInstance Name="p0" Type="participant0"/>
<ProcInstance Name="p1" Type="participant1"/>
<Connection>
        <End ProcInstance="p0" Channel="Ps_election"/>
        <End ProcInstance="p1" Channel="election"/>
</Connection>
```

```
public class participant0 implements javax.jms.MessageListener{

private javax.jms.QueueConnection Ps_election_qConnect = null;
private javax.jms.QueueSession Ps_election_qSession = null;
private javax.jms.QueueSender Ps_election_qSender = null;

private javax.jms.TopicConnection Ps_election_tConnect = null;
private javax.jms.TopicSession Ps_election_tSession = null;

private javax.jms.Topic messageTopic=null;
private javax.jms.TopicSubscriber Ps_election_tSubscriber = null

public participant0(){
try{
}
catch(javax.jms.JMSException jmse){
jmse.printStackTrace();
System.exit(1);
}catch(javx.jms.NamingException jne){
jne.printStackTrace();
System.exit(1);
}

}
}
```

As multiple queues could be used, so I name the queue corresponding to the Channel, rules are followed. An example is shown here.

```
<End ProcInstance="p0" Channel="Ps_election"/>
```

```
javax.jms.QueueConnection Ps_election_qConnect;
javax.jms.QueueSession Ps_election_qSession;
javax.jms.QueueSender Ps_election_qSender;

javax.jms.TopicConnection Ps_election_tConnect;
javax.jms.TopicSession Ps_election_tSession;

ToicConnectionFactory Ps_election_tFactory;
QueueConnectionFactory Ps_election_qFactory;
```

2. Use the current queue

When a queue is ready to use, it means that connection is possible to other receiver or sender. When a message is created, it will be ready to be sent through the queue.

```
<Event Name="start_election" Type="Create" Before="initial" After="election_start"
Channel="outbox" Value="election"/>
```

```
public void onMessage(javax.jms.Message m){
try{
ObjectMessage objectMessage= (ObjectMessage)m;
Message m=(Message)objectMessage.getObject();
if (state=="initial")
start_election(m);
else if (state=="elected")
. . .

}catch (java.lang.RuntimeException e){
e.printStackTrace();
}
}

private void start_election(javax.jms.Message message){
try{
    System.out.println(name+" : start_election");

    traceTable.center.add(Buttonstart_election);
    Message m=new Message("election",this,p1,"outbox");

    ObjectMessage objectMessage= session.createMessage();
    ObjectMessage.set(m);

    Queue outbox=(Queue)message.getJMSReplyto();
    outbox.send(objectMessage);

    outbox_qSender =outbox_qSession.createSender(outbox);

    outbox_qSender.send(objectMessage,
    Javax.jms.DeliveryMode.PERSISTENT,
    Javax.jms.Message.DEFAULT_PRIORITY,
    1800000);

ButtonS_start_election.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent e){
try{
transformState("election_send");
}
catch(javax.jms.JMSException e){System.out.println(name + " :
start_election.error");}

displayTrace("start_election", "Create", "initial", "election_start", "outbox", "election");
ButtonS_send_election.setVisible(false);
}

});

}catch(javax.jms.JMSException e){
}
}
```

3. Close a queue

When all sessions on one queue is finished, this queue should be closed.

<End ProcInstance="p0" Channel="Ps_election"/>

private void exit(Strings){
try {if (s!=null && s.equalsIgorecase("unsubscribe")){
Ps_election_tSubscriber.close();
Ps_election_tSession.unsubscribe("Ps_election Subscription");
}
Ps_election_tConnect.close();
Ps_election_qConnect.close();
}catch(Javax.jms.JMSException jmse){
jmse.printStackTrace();
}
System.exit(0);
}

4.2.1.5 Value

The same code is employed as in RDTtoJava (See 3.2.4). The Value is still part of the Message.

4.2.1.6 Before state and After state

The same code is employed as in RDTtoJava. See 3.2.5.

4.2.1.7 Process Instance

As in RDTtoJava, one instance of Process is one application. In RDTtoWS, one instance of Process is developed into one application as well. The identification or the name of the instance is packaged in Message. When the message is passed to others, the receiver will recognise the sender.

4.2.1.8 Connection

On the sender side, JNDI is first created, and then the connection factory is looked up and the queue is created. If either of these does not exist, then exit. Then create the connection, and the session from the connection. Now the connection is ready, the sender can create a message, and send it. Finally, close the connection.

The receiver side creates a JNDI InitialContext object, and then looks up the connection factory and queue. It then creates a connection and session from the connection. Message delivery is now started, and receive all messages from the queue until a non-object message is receive indicating end of message stream. Finally, close the connection.

The code is found in 4.2.1.3.1 and 4.2.1.3.2.

4.3 Conclusion

In this chapter, the framework of RDTtoWS is introduced in two parts: how to transform an RDT model into a JAX-PRC Web service, synchronous Web service, and one is how to transform a RDT model into JMS Web service, asynchronous Web service. It includes the transformation of the RDT language and the system configuration.

5 Experimental Results

5.1 Introduction

This chapter presents the experimental results observed using the RDTtoJava and RDTtoWS in the target environments. Test and analysis reports are given.

In order to validate my tools, my framework and my approach, many experimental models and system have been researched. Five models of system are presented:

1. The cycle election algorithm
2. The Bully algorithm
3. The Probe/Echo algorithm
4. An agent model
5. Online flight ticket booking system

All experimental systems have been modelled with the RDT tool. All of the models executed in RDX have been simulated in SPIN/XSPIN, using the model checking technology with the help of the RDTtoSPIN tool, which translates the model into Promela code. The RDTtoJava tool generated Java multi-threaded applications from the RDT models. These Java programs are run in JEDPLUS, as this tool requires us to do the configuration work. The web application, based on Web services technology and JMS, was built using RDTtoWS, and is deployed and implemented.

5.2 The Experimental Models

Current research in distributed computing encompasses two major and often separate trends: distributed algorithms [Lynch 1996], and distributed programming models. In the following sections, three well-studied distributed algorithms in the theory of distributed computing will be specified, namely the ring-based election algorithm [Chang and Roberts 1979], the Bully algorithm [Garcia-Molina 1982] and the Probe/Echo algorithm [Andrews 1991], along with an example of the client/server model. I describe these models and then make these models into applications. In addition, I build two client/server examples.

5.2.1.1 The Cycle Election Algorithm

A ring-based election algorithm [Colouris, Dollimore, et al. 2001], called the cycle election algorithm, requires exactly one processor in the ring to be chosen as leader. A participant receives a message from his previous neighbour, and this participant deals with the message, distinguishing the different conditions. Finally, the *boss* for each participant is elected. When all of the participants know their own *boss*, the program will finish. This occurs in such a way that

1. each participant eventually knows his boss
2. no message is left in the message queue, and
3. no matter which participant initiates, the result for all will be the same.

Only one processor performs an action at one time.

The communication of the cycle election algorithm is in the same direction as the synchronous (or asynchronous) message-passing, unlike the bi-directional communication with asynchronous message-passing applied in the Peterson Leader-Election algorithm [Lynch 1996]. The Promela model of a cycle election algorithm is given in Appendix A, and the Java program of this cycle election is given in Appendix A. The correspondence between the Java program and the Promela model in the activity mapping of sending and receiving messages is shown in Figure 53.

Java specification	**Promela model**
if(m.type=="election")	::in?election(value)->

if(m.candidate.value>value)
next.send(m);
	::n>value->
	   out!election(n)

else if(m.candidate.value<value)
next.send(new Message("election",this));
	::n<value->
	out!election(value)

else
next.send(new Message("elected",this));
	::n==value->
	out!elected(n)

if (m.type=="elected"){ }	::in?elected(value)->

if(m.candidate.value==value)
return;
	if
	::n==value->break

else
next.send(m);
	::else->
	out!elected(value)

Figure 53: Java specification and the Promela model for the cycle election algorithm

5.2.1.2 The Bully Algorithm

The Bully algorithm survives processes which crash during an election; although it assumes that message delivery between processes is reliable. To begin an election, each participant sends a message to every participant in the system to announce the start. After the election begins, each participant receives messages from all, or some (if crashes occur), of the other participants in the system. Each then compares the participants' received identifiers and records the highest one as the boss. Then the process stops. In the ideal scenario, every participant should know the boss in the system, and there should be one and only one boss.

The Promela model of the bully algorithm is given in Appendix A/B, and the Java program of the bully algorithm is given in Appendix A/B. The correspondence mapping the Java program to Promela model is given in Figure 54.

The experience of mapping from Promela to Java and from Java to Promela is valuable for deploying tools and building models of distributed systems.

Java specification	**Promela model**
`for(int i=0;i<neighbour.length;i++)try{ neighbour[i].send(new Message(me)); }catch(Exception e){ }`	`do ::i<N-> queue[i]!mynumber; i=i+1 ::i>=N->break od ;`
`try{ while(true){ Message m=(Message)inbox.receive(); if(m.candidate.value>boss.value) boss = m.candidate; }}catch(Exception e){ }`	`do ::in?value-> if ::max<value->max=value; ::else->skip fi od;`

Figure 54: Java specification and the Promela model of the Bully algorithm

5.2.1.3 The Probe/Echo Algorithm

This section discusses the probe/echo algorithm for distributed computations on trees. A *probe* is a message sent by one node to its successor; an *echo* is the subsequent reply. The probe paradigm is first illustrated by showing how information is broadcast to all nodes in a network. The echo paradigm was then added by developing an algorithm for discovering the topology of a network.

In a network, to get information about the utilization of the nodes, a Probe-Echo algorithm can be used. The general idea is simple. The net is treated as a tree, where the root asks all its subtrees for information about their trees; these continue in a similar way and ask their children, etc. After having received the required information from its children, the node sends the complete subtree information to its parent.

5.2.1.4 An Agent Model

Distributed applications are often implemented using some kind of 2-tier or 3-tier client/server model. For some servers, it may be satisfactory to accept one request at a time, and to process each request to completion before accepting the next. However, it is often necessary to process a number of requests in parallel. Multi-threaded servers are commonly used in practice to achieve this. Parallelism may be possible, because a set of clients can concurrently use different objects in the same server process, or because some of the objects in the server process can be concurrently used by a number of clients.

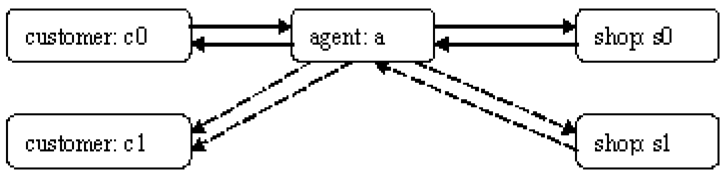


Figure 55: An agent model architecture

The agent model (see Figure 55), designed during this research, describes a scenario where each customer try to buy an item from a couple of shops with the help of an agent. First, each customer presents a request to the agent, and then waits for a reply from the agent. After receiving the request from the customer, the agent will intelligently make a decision, based on experience, to select one shop to pass on the request from the customer, and then wait for a response from the shop. The shop deals with the request passed by the agent, sends back the response to the agent, and then stops. The agent passes the response to the customer who sent the request, and then stops. The customer receives the response from the agent.

5.2.1.5 Online flight ticket booking system

This example most closely reflects the real world of the travel agent. Imagine there are three online flight ticket booking websites, Expedia, Alitalia, and Omega, and one travel agent. The customer will send a query to the travel agent. The travel agent offers a service to consumers for planning flights. Each website will handle the jobs sent from the travel agent, and generate responses back. The travel agent is an intermediary acting on behalf of the consumer, and the consumer never interacts directly with the booking websites.

Booking a flight in one business transaction includes a price comparison for the flight portion through the websites Expedia.co.uk [Expedia 2008], Alitalia.co.uk

[Alitalia 2008], and Omegatravel.net [Omega 2008], subsequently confirming one of the flight options, and informing the others of cancellation of bookings.

The whole process is discussed here in detail. First, the Consumer (Initiator) creates a business transaction for the job (request) it wants to accomplish. It does this through its Coordinator. The Initiator then makes a Service Requests to the Travel Agent with the transaction details. The Travel Agent's Coordinator receives the request and then undertakes the task of creating and managing the sub-transactions that make up the overall business transaction.

The Travel Agent's Coordinator makes its Service Requests to Expedia.co.uk, Alitalia.co.uk, and Omega.co.uk based on the Initiator's request. These recipients (Participants) all agree to participate in the transaction and confirm with the Travel Agent's Coordinator if each can meet the request, for example, the flight tickets are still available on their sites. Once all the parties have agreed to participate in the transaction, the Travel Agent's Coordinator can agree to be part of the transaction initiated by the Initiator. All parties also make a commitment to the Travel Agent's Coordinator with regard to the transaction (Prepared). The Travel Agent's Coordinator could also make a commitment (Prepared) to the Initiator when it agrees to participate in the transaction (Enrol). In this case, however, the Travel Agent's Coordinator simply replies to the Initiator and agrees to participate in the transaction (Enrol). Now the whole booking system is ready.

The Initiator now can decide to make the booking, or to cancel, depending upon the information returned by the Travel Agent's Coordinator. One of two actions will be taken.

1. When the Initiator decides to not purchase the flight ticket offered (Cancel). The Initiator's Coordinator now asks Travel Agent to cancel the booking. The Travel Agent, who has already received commitments from the Participants, must now cancel with Expedia.co.uk, Alitalia.co.uk, and Omega.co.uk. Once the Travel Agent has received confirmation of the requests to cancel from all Participants, it can confirm the cancel operation

with the Initiator's Coordinator, and that Coordinator in turn can confirm the cancellation with the Initiator.

2. When the Initiator decides to purchase the flight ticket offered (Confirm). The Initiator's Coordinator now asks Travel Agent to confirm the booking. The Travel Agent's Coordinator, who has already received commitments from the Participants, now confirms the booking, for example, with Expedia.co.uk, and cancels the booking with Alitalia.co.uk, and Omega.co.uk. Once the Travel Agent's Coordinator has received confirmation of the request to confirm one booking, and the requests to cancel from other Participants, it can confirm the operation with the Initiator's Coordinator, and that Coordinator in turn can confirm the booking with the Initiator.

When all transactions are accomplished, the coordination between the Customer, travel agent and websites is finished as well.

Furthermore, I build other models to expand this system:

1. System 1: Customer 1, Customer 2, Travel Agent, Expedia, Alitalia, and Omega
2. System 2: Customer, Travel Agent, Expedia, Alitalia, Omega, and BAA
3. System 3: Customer 1, Customer 2, Travel Agent, Expedia, Alitalia, Omega, and BAA

After checking the experimental results, I found that our framework and the system generated by it have guaranteed the performance of this online flight booking system. All such applications are working well, so I will only introduce the basic system: Customer, Travel Agent, Expedia, Alitalia, and Omega.

5.3 The Cycle Election Model

Communication within the cycle election model is relatively simple. For instance, the participant *participant0* only receives messages from participant *participant3* and only sends messages to participant *panticipant1***.** Two models were developed based on ***cycle election*** algorithm. In the first (see Figure 56), every participant in the cycle election is awake, and one process, which is not a participant in the election, will send out the first message to one participant in the election to start the election.

In the second model (see 5.3.2), every participant sends out the message to its neighbour to start the election.

5.3.1 The Cycle Election Model (Model 1)

This is the first model (see Figure 56) for the ***cycle election*** algorithm. In this model, the election starts from the process *participant1* after it receives a message from the process *trigger*. The message passing order is: *participant1 ->participant2 ->participant3 ->participant0 ->participant1*, so that a cycle election is created.

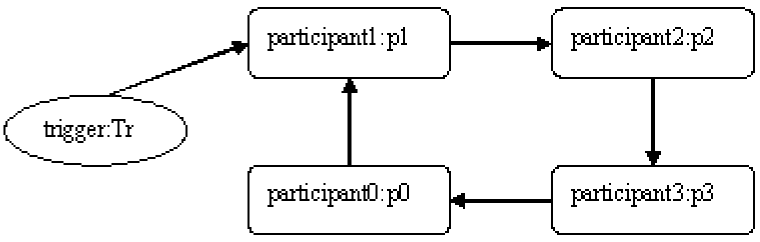


Figure 56: The cycle election model architecture (Model 1)

This model is executed in the RDX tool in both asynchronous and synchronous communication (see Figure 57). All the participants know the correct boss in the system.

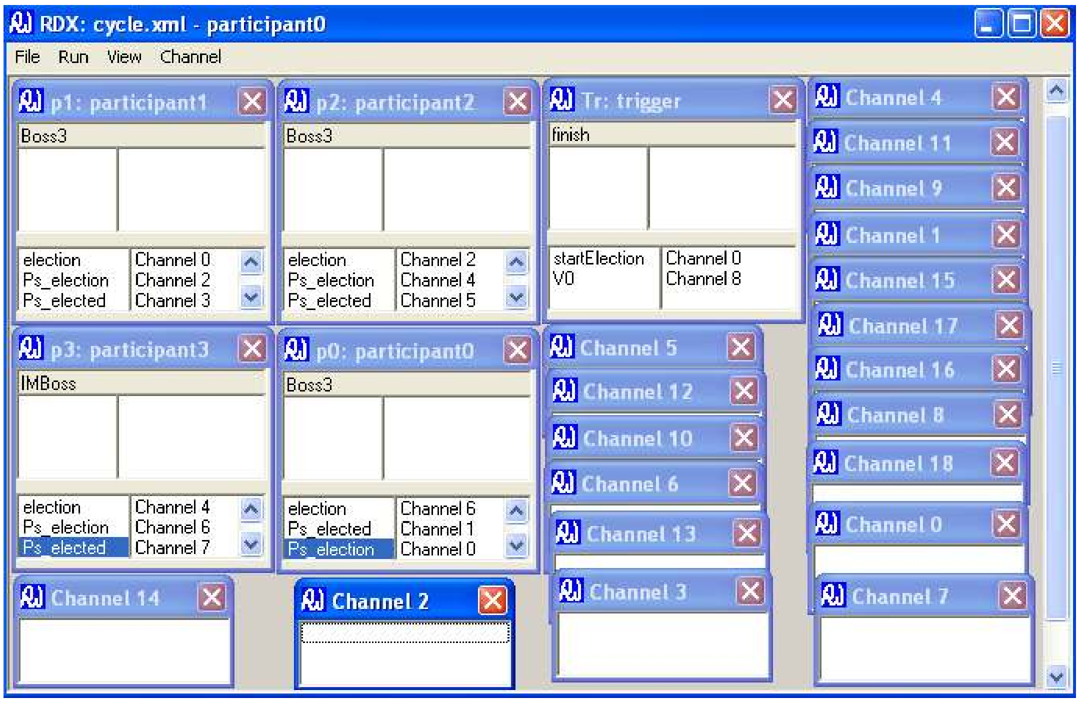


Figure 57: The cycle model during execution (Model 1)

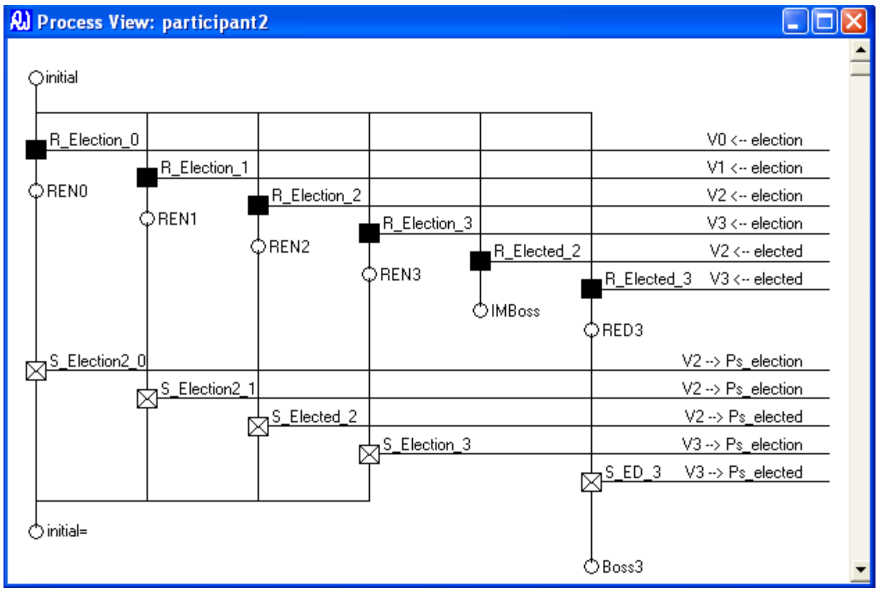


Figure 58: Process view of participant 2 in the cycle election model

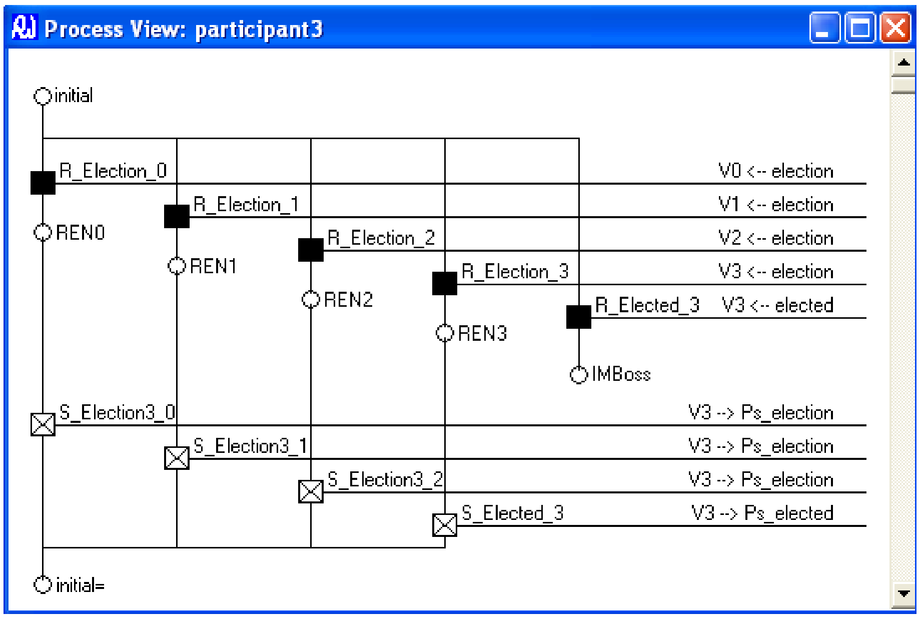


Figure 59: Process view of participant 3 in the cycle election model

Part of the asynchronous communication behaviour of this model in SPIN is as follows (see also Figure 60) when the buffer length is 3;

```
155:  proc 3 (participant2) line 165 "pan_in" (state 35)      [Ps_elected!V3]
156:  proc 3 (participant2) line 165 "pan_in" (state 36)      [i = (i+1)]
157:  proc 3 (participant2) line 166 "pan_in" (state 38)      [goto Boss3]
158:  proc 3 (participant2) line 170 "pan_in" (state 42)      [(1)]
159:  proc 4 (participant3) line 189 "pan_in" (state -)  [values: 6?32]
159:  proc 4 (participant3) line 185 "pan_in" (state 15)      [elected?V3]
160:  proc 4 (participant3) line 189 "pan_in" (state 14)      [goto IMBoss]
161:  proc 4 (participant3) line 214 "pan_in" (state 38)      [(1)]
161:  proc 4 (participant3) terminates
161:  proc 3 (participant2) terminates
161:  proc 2 (participant1) terminates
161:  proc 1 (participant0) terminates
161:  proc 0 (:init:) terminates
5 processes created
```

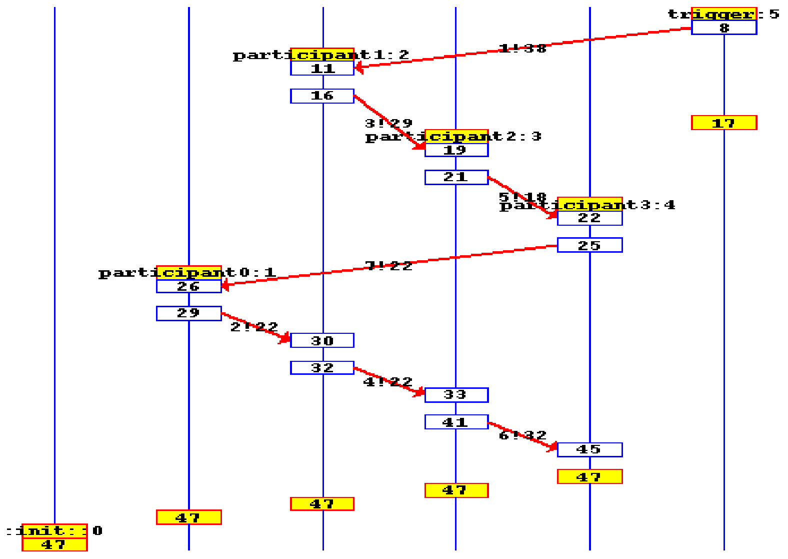


Figure 60: Message sequence chart in XSPIN of asynchronous communication of the cycle election model (Model 2)

There is no deadlock for asynchronous communication.

After *participant2* had sent out the message *V3* successfully, its state was changed to the state *Boss3* and it was still alive at that moment. As expected, *participant3* received the message *V3* successfully from *participant2*, and then its state was changed to the state *IMBoss* and it was still alive at that moment. The state *Boss3* is the termination state for the process *participant2*, and the state *IMBoss* is the termination state for the process *participant3*. Now every participant in the election knows the boss, and then the election stops. The trace history also shows that every process terminates after all of them know the boss.

Part of the asynchronous communication behaviour of this model in Java is as follows.

```
p2 : RED3
receive(V3 from p1 to p2 via Ps_elected)
p2's Event is: R_Elected_3 and read V3 from p1
p2 : S_ED_3
send(V3 from p2 to p3 via Ps_elected)
p2 : Boss3
receive(V3 from p2 to p3 via Ps_elected)
p3's Event is: R_Elected_3 and read V3 from p2
p3 : IMBoss
```

An instance *p2* of the process *participant2* received the message *V3* from an instance *p1* of the process *participant1* through the channel *Ps_elected*. The event *S_ED_3* occurred to send out the message *V3* to *p3*, which is an instance of the process *participant3*, when its conditions have been satisfied. The state of *p2* was then changed to the state *Boss3* which is the terminal state for the process *participant2*. For *p3*, the state was changed to the state *IMBoss* after it received the message *V3* successfully through the channel *Ps_elected*. By now, all processes have reached final states and know the boss in the system, and there are no messages that have not been received.

The synchronous communication of this model in Java was found to be almost the same as asynchronous communication.

```
p2's Event is: R_Elected_3 and read V3 from p1
p2 : RED3
p2 : S_ED_3
send(V3 from p2 to p3 via PassOn_elected)
p2 : 3_Boss
receive(V3 from p2 to p3 via PassOn_elected)
p3's Event is: R_Elected_3 and read V3 from p2
p3 : IMBoss
```

The synchronous communication of this model in SPIN is different from asynchronous communication. Only the process *trigger* terminated, other processes did not terminate, because of the timeout.

```
18:   proc 5 (trigger) terminates
30:   proc 2 (participant1) line 129 "pan_in" (state 51)       [(1)]
timeout
#processes: 5
30:   proc 4 (participant3) line 184 "pan_in" (state 11)
30:   proc 3 (participant2) line 138 "pan_in" (state 13)
30:   proc 2 (participant1) line 130 "pan_in" (state 52)
30:   proc 1 (participant0) line 12 "pan_in" (state 17)
30:   proc 0 (:init:) line 266 "pan_in" (state 7)
6 processes created
```

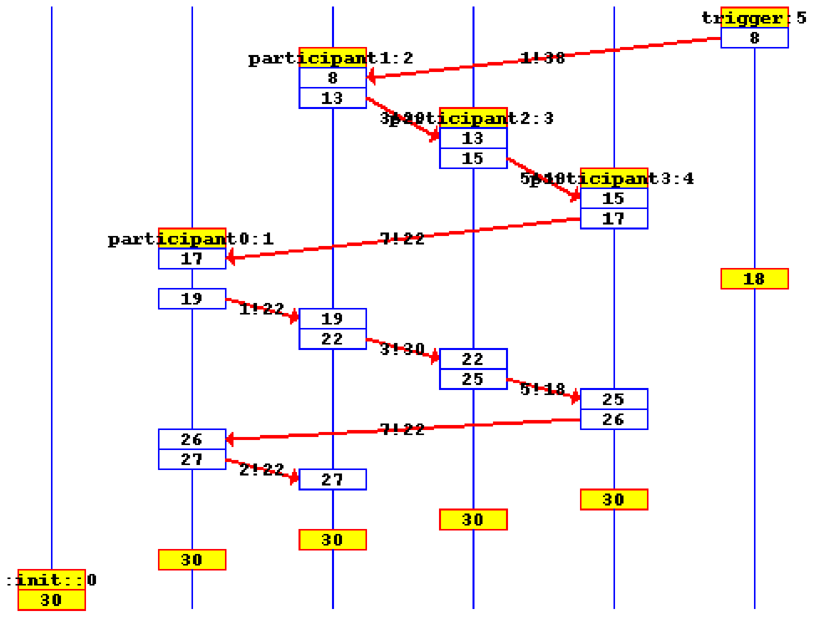


Figure 61: Message sequence chart in XSPIN of synchronous communication of the cycle election model (Model 1)

In RPC Web service, the system is started from the command line. All processes or clients and services should first be ready. This is different to a local Java multi-threaded application. When the all of system is ready, the job is send out to the customer. Controlled by the event selection on the GUI, the system works well. Two scenarios:

1. when one participant goes offline before the system finishes the job, it is impossible for any other participant to send the message to others, and the system is idle.
2. when more than one participant goes offline before the system finishes the job, the system is idle.

In JMS Web service, the system is started from the command line and all queues are created. Controlled on the GUI, the system works well. Some scenarios are:

1. when one queue of one participant is shut down before expected, a warning message will pop up.
2. when one queue of one participant is shut down, and will not be used any more, it does not affect the system.
3. when one queue of one participant is not ready to use, and is expected to receive a message which is already sent out by other participant, the system is waiting.
4. after the *trigger* sends out the message and switches off, it will not affect the whole system.

5.3.2 The Cycle Election Model (Model 2)

In this model, the election starts when each process sends a message out to announce an election to its neighbour. In other words, the process *participant1* receives messages from the process *participant3* and sends out messages to the process *participant2*; the process *participant2* receives messages from the process *participant2* and sends out messages to the process *participant0*, and so on. Then a cycle election is created.

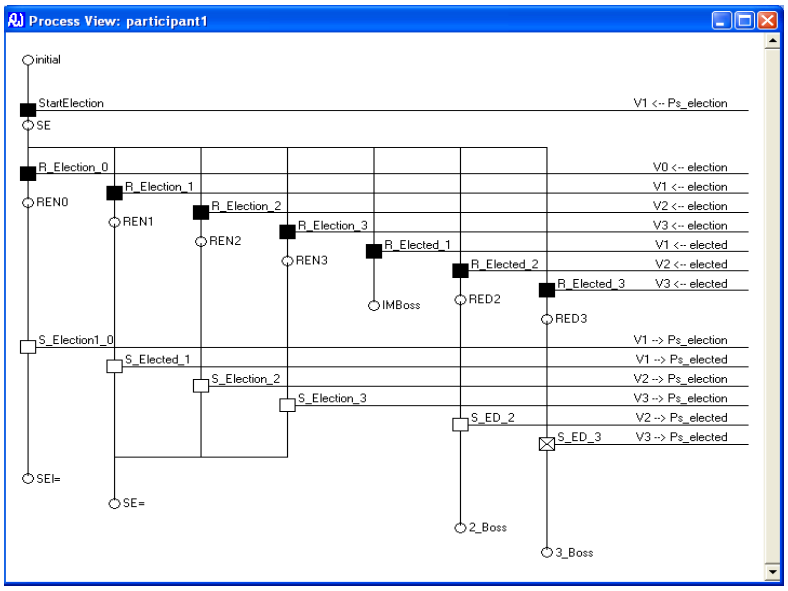
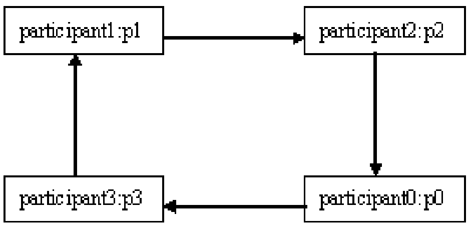
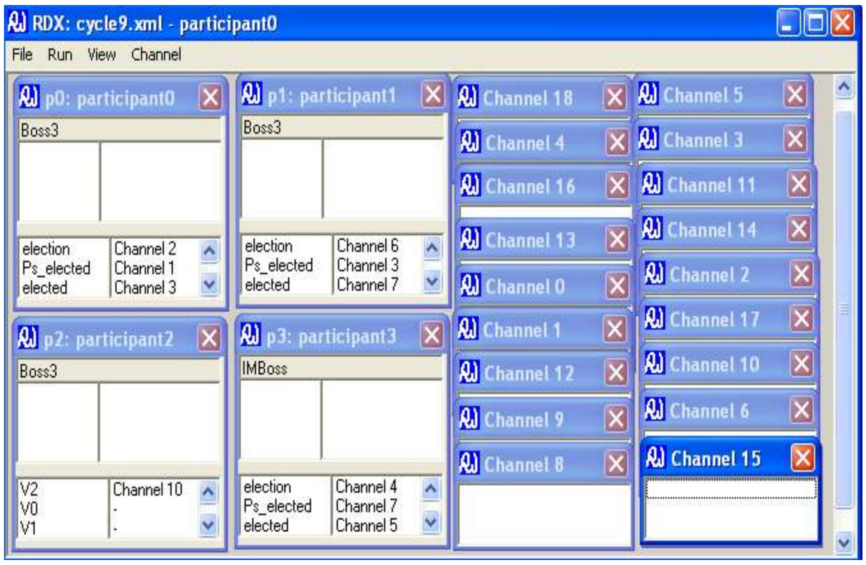


Figure 62: Process view of participant 1 in the cycle election model (Model 2)



$$\begin{tikzcd}
\text{participant1:p1} \arrow[r] & \text{participant2:p2} \arrow[d] \\
\text{participant3:p3} \arrow[u] & \text{participant0:p0} \arrow[l]
\end{tikzcd}$$

Figure 63: The cycle election model architecture (Model 2)



p0: participant0 — Boss3

election	Channel 2
Ps_elected	Channel 1
elected	Channel 3

p1: participant1 — Boss3

election	Channel 6
Ps_elected	Channel 3
elected	Channel 7

p2: participant2 — Boss3

V2	Channel 10
V0	-
V1	-

p3: participant3 — IMBoss

election	Channel 4
Ps_elected	Channel 7
elected	Channel 5

Figure 64: The cycle model during execution (Model 2)

When the communication of this model is asynchronous in SPIN, every process terminates. Part of the simulation in SPIN is shown below:

```
155: proc 3 (participant2) line 165 "pan_in" (state 35)       [Ps_elected!V3]
156: proc 3 (participant2) line 165 "pan_in" (state 36)       [i = (i+1)]
157: proc 3 (participant2) line 166 "pan_in" (state 38)       [goto Boss3]
158: proc 3 (participant2) line 170 "pan_in" (state 42)       [(1)]
159: proc 4 (participant3) line 189 "pan_in" (state -)  [values: 6?32]
159: proc 4 (participant3) line 185 "pan_in" (state 15)       [elected?V3]
160: proc 4 (participant3) line 189 "pan_in" (state 14)       [goto IMBoss]
161: proc 4 (participant3) line 214 "pan_in" (state 38)       [(1)]
161: proc 4 (participant3) terminates
161: proc 3 (participant2) terminates
161: proc 2 (participant1) terminates
161: proc 1 (participant0) terminates
161: proc 0 (:init:) terminates
5 processes created
```

When the communication of this model is asynchronous in Java, every process terminates. Part of the implementation in Java is shown below:

```
receive(V3 from p0 to p2 via election)
p2's Event is: R_Election_3 and read V3 from p0
p2 : REN3
p2 : S_Election_3
send(V3 from p2 to p3 via election)
p2 : SE
receive(V3 from p0 to p2 via election)
p2's Event is: R_Election_3 and read V3 from p0
p2 : REN3
p2 : S_Election_3
send(V3 from p2 to p3 via election)
p2 : SE
p3 : S_Elected_3
```

In RPC Web service, the system is started from the command line. All the processes or clients and services should first be ready. This is different to a local Java multi-threaded application. When the all of system is ready, the job is sent out by the customer. Controlled by the event selection on the GUI, the system works well. Two scenarios:

1. when one participant goes offline before the system finishes the job, it is impossible for any other participant to send the message to others, and the system is idle.
2. when more than one participant goes offline before the system finishes the job, the system is idle.

In JMS Web service, the system is started from the command line and all queues are created. Controlled on the GUI, the system works well. Some scenarios are:

1. when one queue of one participant is shut down before expected, a warning message will pop up.
2. when one queue of one participant is shut down, and will not be used any more, it does not affect the system.
3. when one queue of one participant is not ready to use, and is expected to receive a message which is already sent out by other participant, the system is waiting.

5.4 The Probe/Echo Model

In this model (see Figure 65), *p0* sends the message *probe* to *p1* and *p3*, and then waits for the replies. After it receives the message *probe* from *p0*, *p1* passes the message *probe* to *p2*, and then waits for the reply from *p2*. After it receives the message *probe* from *p0*, *p3* detects no child and then sends back the message *echo* to *p0*. After it receives the message *probe* from *p1*, *p2* detects no child and then sends back the message *echo* to *p1*. After it receives the message *echo* from *p2*, *p1* passes the message *echo* to *p0*. *p0* receives the message *echo* from *p1* and *p3* as well. All of them then stop.

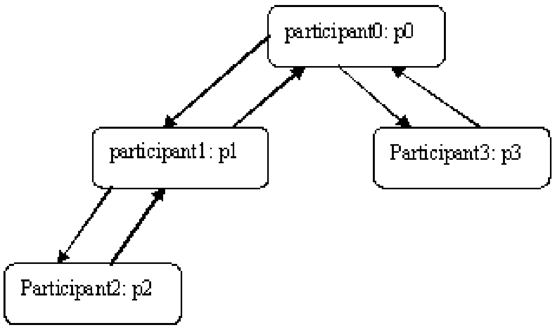


Figure 65: The probe/echo model architecture

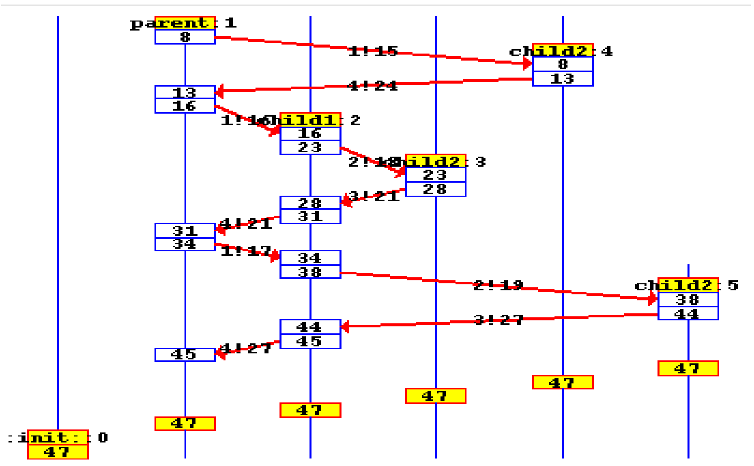


Figure 66: Message sequence chart in XSPIN of synchronous communication in a probe/echo model

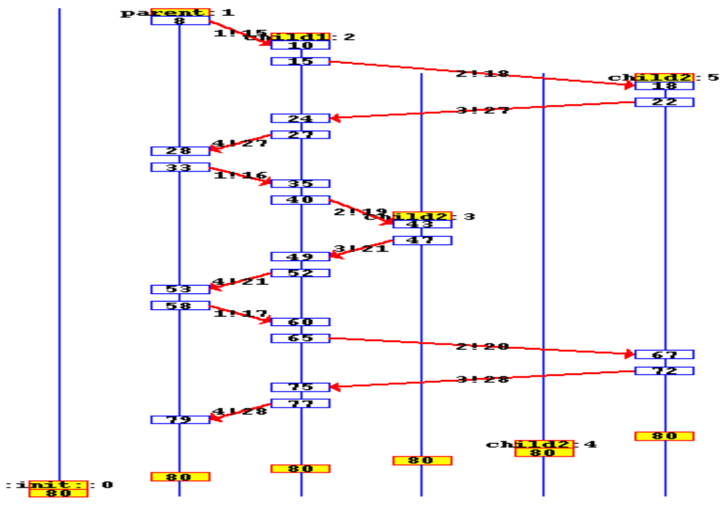


Figure 67: Message sequence chart in XSPIN of asynchronous communication in a probe/echo model

The synchronous communication (see Figure 66) and the asynchronous communication (see Figure 67) of this model in SPIN are the same. Every process is terminated.

p2's Event is: receive_probe and read probe from p1 p0 : probe_send send(probe from p0 to p3 via outbox) p2 : probe_receive p2 : send_echo send(echo from p2 to p1 via outbox_1) receive(echo from p2 to p1 via outbox_1) p1's Event is: receive_echo and read echo from p2 p1 : echo_receive p1 : send_echo send(echo from p1 to p0 via inbox) p2 : initial	p2's Event is: receive_probe and read probe from p1 p2 : probe_receive p2 : send_echo send(echo from p2 to p1 via outbox_1) p2 : initial receive(echo from p3 to p0 via outbox_0) p0's Event is: receive_echo and read echo from p3 p0 : initial p0 : send_probe send(probe from p0 to p1 via outbox) p0 : probe_send send(echo from p1 to p0 via inbox) p1 : initial receive(echo from p2 to p1 via outbox_1) p1's Event is: receive_echo and read echo from p2 p1 : echo_receive p1 : send_echo receive(probe from p0 to p1 via outbox)

The synchronous communication and the asynchronous communication of this model in Java have the same feature of repetitive behaviour. In the asynchronous communication, it is possible that some messages are still left in the message queues when the process terminates.

In RPC Web service, the system is started from the command line. All processes or clients and services should first be ready. This is different to a local Java multi-threaded application. When the system is ready, the job is sent out by the customer. Controlled by the event selection on the GUI, the system works well. Some scenarios:

1. when *participant0* goes offline before the system finishes the job, the system is idle.
2. when *participant1* goes offline before the system finishes the job, the system sometimes continues the communication between *particant0* and *participant3*, and pops up a warning message and sometimes breaks down.
3. when *participant2* goes offline before the system finishes the job, the system sometimes continues the communication between *participant0* and *participant3*, and between *participant0* and *participant3*, or pops up a warning message and sometimes breaks down.
4. when *participant3* goes offline before the system finishes the job, the system sometimes continues the communication between *participant0*, *participant1* and *participant2*, and pops up a warning message and sometimes breaks down.

In JMS Web service, the system is started from the command line and all queues are created. Controlled on the GUI, the system works well. Some scenarios are:

1. when one queue of one participant is shut down before expected, a warning message will pop up.
2. when one queue of one participant is shut down and will not be used any more, if does not affect the system.
3. when one queue of one participant is not ready to use, and is expected to receive a message which is already sent out by other participant, the system is waiting.

From these experiments, we can see that the JMS Web service can guarantee a more stable performance for the communication than the RPC Web service.

5.5 An Agent Model

This model (see Figure 68) is an example of the classic client/server model. In this model, there are two *customer* processes, two *shop* processes and one *agent* process. The *customer* sends a request to the agent and then waits for a reply from the *agent*. After it receives the request from the *customer*, the *agent* passes the request to the *shop*, and then waits for a reply from the *shop*. After it receives the request from the *agent*, the *shop* sends a reply to the *agent*, and then it terminates. After it receives the reply from the *shop*, the *agent* passes the reply to the *customer*, and then it terminates. The *customer* receives the reply from the *agent* and then terminates.

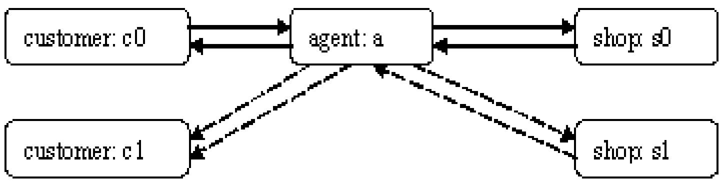


Figure 68: An agent model architecture

The model checker SPIN only implements the *customer ->agent ->shop ->agent ->customer* behaviour and does not repeat this behaviour to another *customer* and another *shop*. After checking the simulation output of the synchronous communication of this model in SPIN, it was found that the processes that have not terminated have ended communication, but have just not reached terminal states. As we know (see Figure 69), *c0* has a relationship with *a*, and *a* has a relationship with *s0* and *s1*. The expected relationship between the instances, between the process *customer* and the process *shop*, is not the relationship between *c0* and *s1*, but the relationship between *c0* and *s0*.

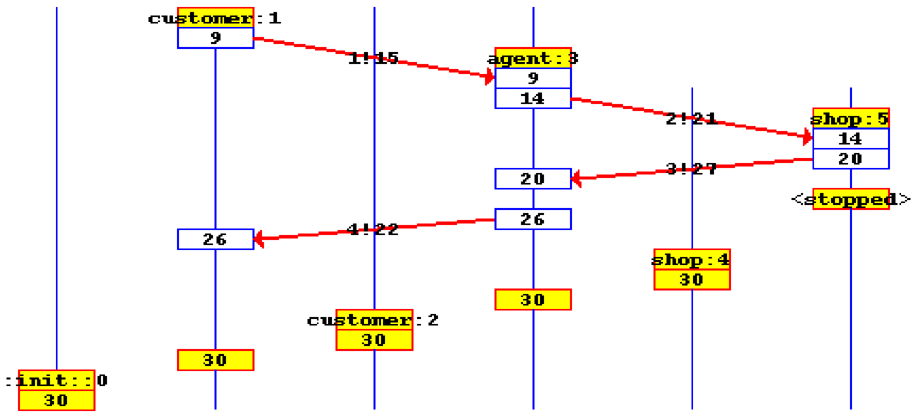


Figure 69: Message sequence chart in XSPIN of synchronous communication in an agent model

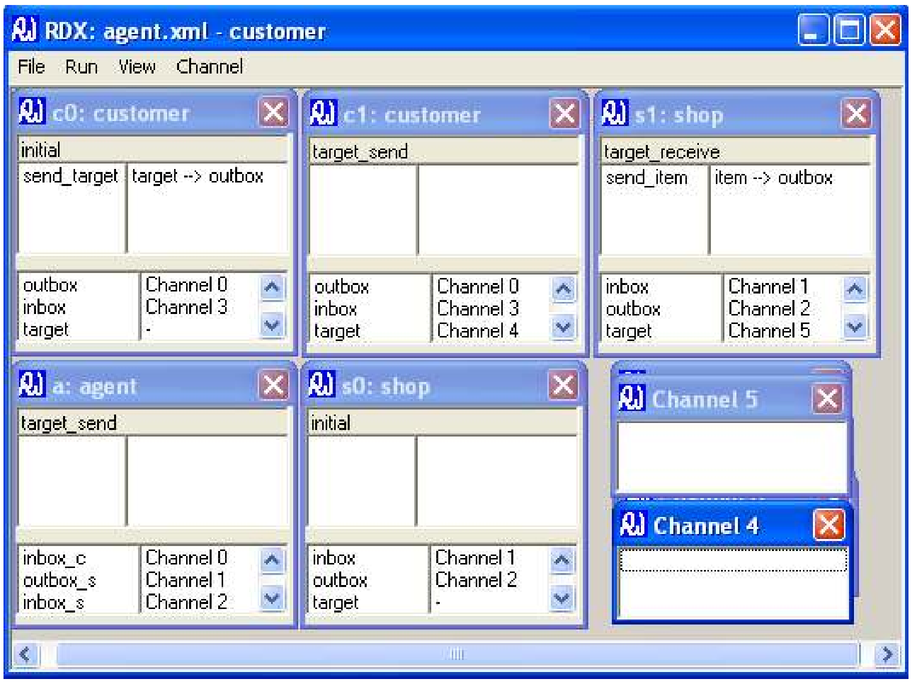


Figure 70: An agent model during execution

```
23:  proc 5 (shop) terminates
24:  proc 3 (agent) line 47 "pan_in" (state 23)   [((i<3))]
25:  proc 3 (agent) line 48 "pan_in" (state 21)   [item = supp[i]]
26:  proc 3 (agent) line 48 "pan_in" (state 19)   [outbox_c!item]
26:  proc 1 (customer) line 19 "pan_in" (state 9)      [inbox?item]
26:  proc 3 (agent) line 48 "pan_in" (state -)     [values: 4!22]
26:  proc 1 (customer) line 19 "pan_in" (state -)[values: 4?22]
27:  proc 3 (agent) line 48 "pan_in" (state 20)   [i = (i+1)]
28:  proc 3 (agent) line 49 "pan_in" (state 22)   [goto finish]
29:  proc 1 (customer) line 22 "pan_in" (state 13)     [(1)]
30:  proc 3 (agent) line 52 "pan_in" (state 25)   [(1)]
timeout
#processes: 5
30:  proc 4 (shop) line 61 "pan_in" (state 3)
30:  proc 3 (agent) line 53 "pan_in" (state 26)
30:  proc 2 (customer) line 13 "pan_in" (state 3)
30:  proc 1 (customer) line 23 "pan_in" (state 14)
30:  proc 0 (:init:) line 98 "pan_in" (state 7)
6 processes created
```

The synchronous communication in this Java model is complete. Every process has reached its terminal state. The implementation in Java reveals one problem which also occurred in RDX (see Figure 70). When three or more processes work together, it is possible to miss the relationships between their instances. In the following Java implementation output, it was found that the message *target* had been sent from *a* to *s0* (see the statement marked $$ below), as expected, and also from *a* to *s1* (see the statement marked ££ below), which was unexpected. The reason for that is that the process *agent* always sends a message to the process *shop* through the channel *outbox_s*. In other words, the process *shop* receives the messages from the channel *outbox_s* of the process *agent*. So in this model, both instances of the process *shop* inherit this feature. Once the *agent* sends a message to *s0*, this message is also delivered to *s1*. The solution to this problem is to use different channels to make the connections between the two processes, if one or both of them has a complex connection between their instances. For example, for the instance of the process agent *a*, it sends messages to *s0*, which is an instance of the process *agent*, through the channel *outbox_s0*, and in the meantime it sends the messages to *s1*, which is another instance of the process *agent*, through the channel *outbox_s1*. Part of the implementation history in Java is given below:

```
c0 : initial
c0 : send_target
send(target from c0 to a via outbox)
a : initial
s0 : initial
a : target_receive
a : send_target
send(target from a to s0 via outbox_s) $$
a : target_send
send(target from a to s1 via outbox_s) ££
a : target_send
receive(target from a to s0 via outbox_s)
s0's Event is: receive_target and read target from a
s0 : target_receive
s0 : send_item
send(item from s0 to a via outbox)
a : item_receive
a : send_item
send(item from a to c0 via outbox_c)
s0 : finish
send(item from s0 to a via outbox)
```

In RPC Web service, the system will start from the command line. All processes or clients and services should first be ready. This is different to a local Java multi-threaded application. When the system is ready, the job is sent from *customer*. Controlled by the event selection on the GUI, the system works well. Some scenarios are:

1. when *customer1* goes offline before the system finishes the job, the system sometimes blocks, and sometimes continues the process between *customer2*, *agent*, *shop1*, and *shop2* depending on the transaction.
2. when *customer2* goes offline before the system finishes the job, the system sometimes blocks, and sometimes continues the process between *customer2*, *agent*, *shop1*, and *shop2* depending on the transaction.
3. when *shop1* goes offline before the system finishes the job, the system sometimes blocks, and sometimes continues the process between *customer1*, *customer2*, *agent*, and *shop2* depending on the transaction.
4. when *shop2* goes offline before the system finishes the job, the system sometimes blocks, and sometimes continues the process between *customer1*, *customer2*, *agent*, and *shop1* depending on the transaction.
5. when *agent* goes offline before the system finishes the job, the system is blocked.

In JMS Web service, the system is started from the command line and all queues are created. Controlled on the GUI, the system works well. Some scenarios are:

1. when *customer1* goes offline before the system finishes the job, the system sometimes blocks, and sometimes continues the process between *customer2*, *agent*, *shop1*, and *shop2* depending on the transaction.
2. when *customer2* goes offline before the system finishes the job, the system sometimes blocks, and sometimes continues the process between *customer2*, *agent*, *shop1*, and *shop2* depending on the transaction.
3. when *shop1* goes offline before the system finishes the job, the system sometimes blocks, and sometimes continues the process between *customer1*, *customer2*, *agent*, and *shop2* depending on the transaction.
4. when *shop2* goes offline before the system finishes the job, the system sometimes blocks, and sometimes continues the process between *customer1*, *customer2*, *agent*, and *shop1* depending on the transaction.
5. when *agent* goes offline before the system finishes the job, the system is waiting, and no warning message is given.

From these experiments, we can see that the JMS Web service can supply tolerable performance for the communication that RPC Web service.

5.6 Online Flight Ticket Booking System

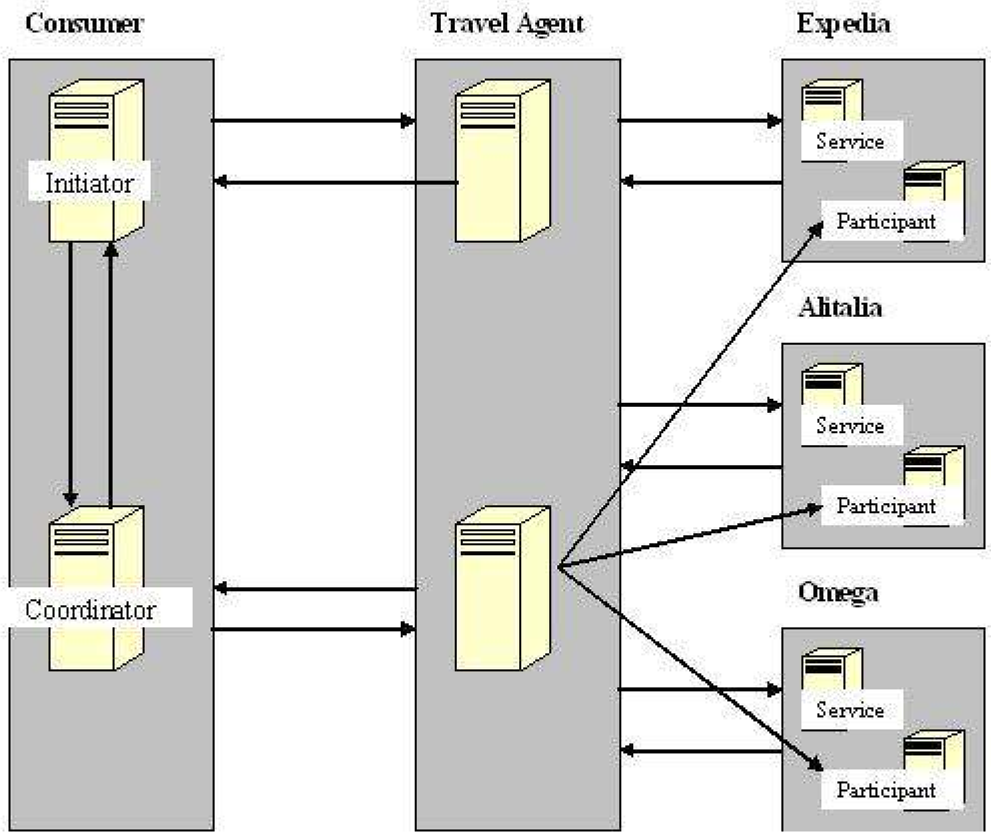


Figure 71: Online flight ticket booking system

When modelling this system initially, three Processes are considered: Customer, Travel Agent, and the Website. One instance of Customer, one instance of Travel Agent, and three instances of Website, named as Expedia, Alitalia and Omega, were modelled. In the testing, problems happened, so the design was changed and the system was modelled by five Processes: *Customer*, *Travel Agent*, *Expedia*, *Alitalia*, and *Omega*.

In SPIN, the system works well. All possible business processes were verified, and there was no block, whether the communication was asynchronous or synchronous.

In Java, when communication is synchronous, the transaction between the *Website* and *Travel Agent*, and the transaction between the *Customer* and *Travel Agent* are implemented well. The system will start with a message sent by the *Customer*, and the *Customer* will wait for the message from the *Travel Agent*. When the message from the *Travel Agent* is received by the *Customer*, the system has accomplished all its work, and the status of all processes are standby.

In RPC Web service, the system will start from the command line. All processes or clients and services should first be ready. This is different to a local Java multi-threaded application. When the system is ready, the job is sent by the *Customer*. Controlled by the event selection on the GUI, the system works well. Some scenarios are:

1. the system works well, when all clients and services are running well.
2. when *Customer* goes offline before the system finishes the job, the system finishes also.
3. when *Travel Agent* goes offline before the system finishes the job, the system sometime finishes. The *Customer* still is available, and can start a new transaction.
4. when *Expedia* goes offline before the system finishes the job, the system still works. The *Travel Agent* will delete the link with *Expedia* whatever the transaction. The system is reduced to *Customer*, *Travel Agent*, *Alitalia*, and *Omega*.
5. when *Expedia* goes offline before the system finishes the job, the system still works. The *Travel Agent* will delete the link with *Expedia* whatever the transaction. The system is reduced to *Customer*, *Travel Agent*, *Alitalia*, and *Omega*.
6. when *Alitalia* goes offline before the system finishes the job, the system still works. The *Travel Agent* will delete the link with *Alitalia* whatever the transaction. The system is reduced to *Customer*, *Travel Agent*, *Expedia*, and *Omega*.
7. when *Omeg*a goes offline before the system finishes the job, the system still works. The *Travel Agent* will delete the link with *Expedia* whatever

the transaction. The system is reduced to *Customer*, *Travel Agent*, *Alitalia*, and *Expedia*.

8. when *Expedia* and *Alitalia* are offline before the system finishes the job, the system still works. The *Travel Agent* will delete the link with *Expedia* and *Alitalia* whatever the transaction. The system is reduced to *Customer*, *Travel Agent*, and *Omega*.
9. when *Expedia*, *Alitalia* and *Omega* are offline before the system finishes the job, the system still works. The *Travel Agent* will delete the link with *Expedia*, *Alitalia* and *Omega* whatever the transaction. The system will send out a warning message.

In JMS Web service, the system is started from the command line and all queues are created. Controlled on the GUI, the system works well. Some scenarios are:

1. the system works well, when all clients and services are running well.
2. when *Customer* goes offline before the system finishes the job, the system finishes also.
3. when *Travel Agent* goes offline before the system finishes the job, the system sometimes finishes. The *Customer* still is available, and can start a new transaction.
4. when *Expedia* goes offline before the system finishes the job, the system still works. The *Travel Agent* will delete the link with *Expedia* whatever the transaction. The system is reduced to *Customer*, *Travel Agent*, *Alitalia*, and *Omega*.
5. when *Expedia* goes offline before the system finishes the job, the system still works. The *Travel Agent* will delete the link with *Expedia* whatever the transaction. The system is reduced to *Customer*, *Travel Agent*, *Alitalia*, and *Omega*.
6. when *Alitalia* goes offline before the system finishes the job, the system still works. The *Travel Agent* will delete the link with *Alitalia* whatever the transaction. The system is reduced to *Customer*, *Travel Agent*, *Expedia*, and *Omega*.

7. when *Omega* goes offline before the system finishes the job, the system still works. The *Travel Agent* will delete the link with *Expedia* whatever the transaction. The system is reduced to *Customer*, *Travel Agent*, *Alitalia*, and *Expedia*.
8. when *Expedia* and *Alitalia* are offline before the system finishes the job, the system still works. The *Travel Agent* will delete the link with *Expedia* and *Alitalia* whatever the transaction. The system is reduced to *Customer*, *Travel Agent*, and *Omega*.
9. when *Expedia*, *Alitalia* and *Omega* are offline before the system finishes the job, the system still works. The *Travel Agent* will delete the link with *Expedia*, *Alitalia* and *Omega* whatever the transaction. The system will wait. If any one or more of *Expedia*, *Alitalia* and *Omega* comes back online, the system can re-start.

5.7 Model-based Testing

Here I discuss how the test cases are generated. The RDT language includes: Process, Instance, Write event, Read event, Create event, Before state, After state, Channel, Value, and Connection. The test cases generated are based on such attributes.

As there is a separate trace for every participant in the system, it is possible to check individual behaviour and the system behaviour also. When the system is working stably, and has been validated against the RDT model, a record of individual and system behaviour is made, and the code generation process generates test cases to further test and validate our approach.

5.7.1 State-based testing

Before state is one of the conditions of the Event, and the Before state of *Event A* is the After state for the next event, *Event B*, except the last event. When we change the value of After state of *Event A* only, we should see that *Event A*'s After state is different from *Event B*'s Before state, otherwise it is wrong. If the Before state of

Event A changed, the condition of *Event A* should not be satisfied, and *Event A* could be executed.

5.7.2 Event-based testing

This kind of test can be executed by changing the name of the Event during the code and application generation. An additional testing method is by changing the type of Event during the code and application generation.

5.7.3 Message-based testing

When the code and application from RDT model is generated into Java application and Web service, the Message object is important for communication and the expanded Value in RDT. It includes more information than Value alone. The Message includes: the name of sender (Name of Instance), the channel, the name of receiver (Name of Instance), the information (Value). Any attribute of Message that changes will involve a test case. Noticeably, the channel in Java application is a queue of an object, and is a queue of a JMS receiver or sender, and is part of message in PRC Web service.

5.7.4 Connection-based testing

As Java application and Web service are different at the implementation level, the connection is translated differently as well. In a Java multi-threaded application, the connection of two instances of processes is made through a queue. The message sender will know (connect to) the message receiver. The message is sent by the sender, not from the queue owned by this sender, to the receiver. And the receiver will store the message in a specific queue. The message having been stored in the correct (expected) queue, means that the connection is completed and the communication is completed.

In RPC Web service, the communication is over HTTP. The name of the channel is part of the message. When the message is passed over HTTP, it is received by the client. The client will check the message details against those expected.

In JMS Web service, the communication is done by SOAP over JMS, and the channel is the queue. The server will send a message through the queue, and the client will receive a message from the queue.

As described above, when the connection changes, a test case will be generated. This testing will involve complicated issues, to which more attention needs to be addressed.

5.8 Conclusion

From the experiments, I discovered:

1. When modelling a system, the attention to the instance of process should be put. In most cases, when more than one instance of process is involved in the system, and could take different and complicated actions, it is better to build another one or more separate processes to re-design the system.
2. The message should be an object containing useful information.
3. Loosely-coupled system design is better.
4. The model-based testing validates the application efficiently.

I demonstrated that the RDTtoJava tool has the following benefits:

1. synchronous communication and asynchronous communication;
2. state transition;
3. the condition of events checking;
4. event implementation;
5. exception handling; and
6. the integrated interaction of processes at the model level.

6 Conclusions and Further Work

6.1 My Work

In this part, I draw conclusions from the work that has been done. The goals of my research are:

1. model-based generation of Java multi-threaded applications
2. synchronisation in the Java multi-threaded applications
3. model-based generation of Web service applications
4. synchronisation in the Web services
5. model-based generation of testing

A toolkit has been developed to extend the RDT tools that were developed by Walters. Two tools have been developed. One is RDTtoJava, another one is RDTtoWS. The synchronisation issue is resolved for both tools. The model based test case generation has been applied.

The tool RDTtoJava has been fully developed to translate a RDT model directly into a Java threaded application. The tool was developed in Visual Basic 6. Using this tool, I built a synchronous local threaded application, (length of the queues employed is zero), and also built an asynchronous local threaded application, (length of the queues employed is not zero). After checking the test record and the application behaviour against the model execution behaviour, I am confident that the development mechanism employed is correct. The trace table I used is a good method to trace the event and is a good testing method for test case generation and error-finding.

The RDTtoJava can help developers build multi-threaded Java application in minutes and guarantee the quality of the applications. The traceable process table can help developers and testers validate the application.

The RDTtoWS was developed to transform the RDT model into a Web service. For the synchronisation issue, two parts were considered. One was to develop RDT model into a JAX-RPC Web service, the other as to develop RDT model into a JMS Web service. It is difficult and complicated for the developer to build a Web service, both synchronous and asynchronous. It is difficult to handle the SOAP over JMS to build a Web service. Currently, most JAX-PRC Web services only handle the pure HTTP communication; the queues within the application have only been touched. The JMS developers only handle one queue for each application most of the time, and do not give much attention to multiple queues. RDTtoWS works, while more research effort is needed to extend this topic. It can help developers build complicated Web services. This research contributes to academic` and industrial web application development.

This is valuable work for the software developer and researchers. It can help developers build models and generate applications quickly in hours rather than months. The quality of applications generated is guaranteed by my tools. It is also helpful to research on software engineering.

6.2 Further Work

6.2.1 Improvement

There are some ideas to improve the performance of tools and transformation.

6.2.1.1 Application test tool

As the complex Java threaded application and Web service can be developed with these tools, the next target is to build bigger and more complicated applications. Subsequently, testing will be more important to sort out the problems occurring. One idea is to build testing and validation tools for Java threaded applications and Web services.

The testing tool will read the source code of the application directly, and generate the test cases. So the testing function will be separated from the current tools, and the current tools only generate the pure application.

6.2.1.2 Configuration and deployment tool

After the framework assists the user develop the Web service, a deployment and configuration tool is needed to complete the Web service deployment automatically. Cross-platform and different versions of the operating system will be considered.

6.2.2 Enhancement

In the future, I will focus on the new techniques, and the new products, offered by the vendors and .NET platform.

6.2.2.1 .NET Web service

The .NET Framework 2.0 and 3.8 are Microsoft's managed code programming model and runtime for building applications on the Windows platform [Microsoft 2008]. Web services are an evolutionary step in software development, and have formed the foundation of Microsoft's inter-operability efforts. A new feature in the .NET Framework 3.0 used for Web Service is WF. WF is the programming model engine, and includes tools for quickly building workflow-enabled applications under Windows. I can use the new technology supported by WF to build multi-tier applications. But, Java is not supported by the development environment in Window Studio, leaving C# as the main language to use.

6.2.2.2 AJAX

Asynchronous JavaScript technology and XML (AJAX) [SUN 2007] is a new technology to build asynchronous Web services in Java. These techniques have been available to developers targeting Internet Explorer on the Windows platform for many years. This technology can be used to develop Web applications based on the RDT model.

Appendix A Source Code

A.1 Promela Model for a Cycle Election Algorithm (Asynchronous Communication)

```
/*
*cycle.Spin
*/

    #define N 8
    mtype={election, elected};

    chan queue[N]=[0] of {mtype,int};

    proctype participant(chan in ,out;int n)
        {
        int value=n;
        xr in;
        xs out;
        out!election(n);
        end:   do
        ::in?election(value)->
            if
              ::n>value->
                out!election(n)
              ::n==value->
                out!elected(n)
              ::n<value->
                out!election(value)
            fi
        ::in?elected(value)->
            if
              ::n==value->
                printf("%d's boss is:%d\n",n,value);
                break /*flow out of do-od loop*/
              ::else->
                out!elected(value)
            fi
            od
        }

        init{
        int nr_participant;
        atomic{
      nr_participant=0;
        do
          ::nr_participant<N->
        run participant(queue[nr_participant],queue[(nr_participant+1)%N],nr_participant);
                nr_participant++
          ::nr_participant>=N->
                break
            od
          }
        }
```

A.2 A Cycle Election Algorithm in Java (Synchronous Communication)

```
import java.io.*;
public class Cycle
{
static class Message{
String type;
Participant candidate;
Message (String t, Participant p){
type=t;
candidate=p;
}
}

static class Participant extends Thread{
MessageQueue previous;
MessageQueue next;
int value;
Participant boss;
public void run(){
try{while(true){
Message m=(Message)previous.receive();
System.out.println(value + " receives " + m.type+ " "
+ m.candidate.value);
if(m.type=="election"){
if(m.candidate.value>value)
next.send(m);
else if(m.candidate.value<value)
next.send(new Message("election",this));
else
next.send(new Message("elected",this));
}
if (m.type=="elected"){
boss=m.candidate;
if(m.candidate.value==value)
return;
else
next.send(m);
}
}}catch(Exception e){ }
}
}

public static void main(String[] args) throws
IOException{
final int n = 9;
final int [] value = {5,12,31,47,53,72,85,90,35};
Participant[] part = new Participant[n];
MessageQueue[] q = new MessageQueue[n];
for(int i=0;i<n;i++){
part[i]=new Participant();
part[i].value=value[i];
q[i]=new MessageQueue(0);
}
for(int i=0;i<n;i++){
part[i].previous=q[i];
part[i].next=q[(i+1)%n];
}
for(int i=0;i<n;i++){
System.out.println(part[i].value + " next " +
part[(i+1)%n].value);
if(part[i].next!=part[(i+1)%n].previous)
System.out.println("Connection Error");
}
for(int i=0;i<n;i++)
part[i].start();
Participant p=part[0];
try{
p.next.send(new Message("election",p));
Thread.sleep(1000);
}catch(Exception e){ }
for(int i=0;i<n;i++)
part[i].interrupt();
for(int i=0;i<n;i++)
System.out.println(part[i].value + " boss is " +
part[i].boss.value);
}

static class MessageQueue{
boolean sendDone, receiveFlag;
Object share;

public MessageQueue(int i){
      sendDone=false;
      receiveFlag=false;
}

synchronized void send(Object x)throws
InterruptedException{
     sendDone=true;
     share=x;
     notifyAll();

     while(!receiveFlag)
         wait();
     receiveFlag=false;
}

synchronized Object receive()throws
InterruptedException{
     receiveFlag=true; notifyAll();
     while(!sendDone)wait();
     Object x; x=share;
     System.out.println(x.toString());
sendDone=false;
return x;
}
} }
```

A.3 Promela Model for the Bully Algorithm (Asynchronous Communication)

```
/*
* bully.Spin
*/



    #define N 4
    chan queue[N]=[N] of {int};

    proctype participant(chan in;int mynumber)
    {
        int value,max;
        xr in;
        end:
        max=mynumber;
        int i=0;
        do
        ::i<N->
```

```
            queue[i]!mynumber;    /*messages in
queue*/
         i=i+1                    /*counter*/
        ::i>=N->break                         /*if i=>N
then stop*/
         od ;
        do
            ::in?value->
                if
                    ::max<value->
                        max=value;
                    ::else->skip
                fi
        od;
        printf("%d's boss
is:%d\n\n",mynumber,max)
        }

        init
        {
        int nr_participant;
        atomic{
            nr_participant=0;
        do
             ::nr_participant<N->    run
participant(queue[nr_participant],nr_participant);
        nr_participant=nr_participant+1

             ::nr_participant>=N-> break
        od
         }
        }
```

A.4 The Bully Algorithm in Java (Asynchronous Communication)

```
import java.io.*;
public class Bully
{
        static class Message{
            Participant candidate;
            Message(Participant p){
                candidate=p;
            }
        }

        static class MessageQueue{
            int entries,maxEntries;
            Object[] elements;
            public MessageQueue(int m){
                maxEntries=m;
                elements=new Object[maxEntries];
                entries=0;
        }
        synchronized void send(Object x)throws
InterruptedException{
            while(entries==maxEntries)
                wait();
            elements[entries]=x;
            entries=entries+1;
            notifyAll();
        }
            synchronized Object receive()throws
InterruptedException{
            while(entries==0)
                wait();
            Object x;
            x=elements[0];
            for(int i=1; i<entries; i++)
                elements[i-1]=elements[i];
            entries=entries-1;
            notifyAll();
            return x;
        }
}

static class Participant extends Thread{
        MessageQueue inbox;
        MessageQueue[] neighbour;
        int value;
        Participant boss;
        Participant me;
        public void run(){
            boss=this;
            me=this;
            for(int i=0;i<neighbour.length;i++)
                try{neighbour[i].send(new
Message(me));
                }catch(Exception e){ }
            try{while(true){
            Message m=(Message)inbox.receive();
                System.out.println(value + " receives
"+ m.candidate.value);
            if(m.candidate.value>boss.value)
                    boss=m.candidate;
            }}catch(Exception e){ }
            }
        }

        public static void main(String[] args) throws
IOException{
            final int n = 9;
            final int [] value =
{43,51,47,89,9,28,49,58,3};
            Participant[] part = new Participant[n];
            MessageQueue[] q = new
MessageQueue[n];
            for(int i=0;i<n;i++){
                part[i]=new Participant();
                part[i].value=value[i];
                q[i]=new MessageQueue(4);
            }
            for(int i=0;i<n;i++){
                part[i].inbox=q[i];
                part[i].neighbour=new
MessageQueue[n];
            }
            for(int i=0;i<n;i++){
                for(int j=0;j<n;j++)

        part[i].neighbour[j]=part[j].in
box;
              }
              for(int i=0;i<n;i++)
```

```
                part[i].start();

        try{Thread.sleep(500);}catc
h(Exception e){ }
                for(int i=0;i<n;i++)

        part[i].interrupt();
                for(int i=0;i<n;i++){
                        if(part[i].boss!=null)
                        System.out.println(part[i].value + "
boss is " + part[i].boss.value);
                }
        }
}
```

A.5 RDTtoJava

I started this work from studying the work, RDTtoPromela, of my supervisor Dr Robert Walters. I used his code for the following functions:

1. Open a file
2. Save into a file
3. Collect all information about RDT Model

```
Option Explicit
'global params
Public Channel_Length As Integer
Public Number_of_Channel As Integer
Public doc As DOMDocument

Private fileSysObject As Object
''''''''''
'params for the Process
''''''''''
Private Type process_type
Name As String
ports() As String
End Type
Private ptypes() As process_type
''''''''''
'params for the Instance
''''''''''
Private Type instance_type
Name As String 'Name of the process instance
type As String
ports() As String
channels() As String 'Global name of the channel
End Type
Private pinsts() As instance_type
'select a XML file for a model and then Open this file
Private Sub Command1_Click()
CommonDialog1.Filter = "All Files (*.*)|*.*|XML
Files(*.xml)|*.xml"
CommonDialog1.FilterIndex = 2
CommonDialog1.ShowOpen
If CommonDialog1.FileName <> "" Then Text1.Text
= CommonDialog1.FileName
End Sub
Private Sub Command2_Click()
''''''''''
'save the model into a Java file
''''''''''
CommonDialog2.Filter = "All Files (*.*)|*.*|Java
Source(*.Java)|*.Java"
CommonDialog2.FilterIndex = 2
CommonDialog2.ShowOpen
If CommonDialog2.FileName <> "" Then Text2.Text
= CommonDialog2.FileName
End Sub
Private Sub Command3_Click()
''''''''''
'params- private and global
''''''''''
Dim txtStream As Object
Dim el As IXMLDOMElement
Dim el0 As IXMLDOMElement
Dim el1 As IXMLDOMElement
Dim el2 As IXMLDOMElement
Dim tmpel As IXMLDOMElement
Dim nodes0 As IXMLDOMNodeList
Dim nodes1 As IXMLDOMNodeList
Dim nodes2 As IXMLDOMNodeList
Dim nodes3 As IXMLDOMNodeList
Dim nodes4 As IXMLDOMNodeList
Dim nodes5 As IXMLDOMNodeList
Dim nodes6 As IXMLDOMNodeList
Dim nodes7 As IXMLDOMNodeList
Dim nodes8 As IXMLDOMNodeList
Dim i As Integer
Dim j As Integer
Dim k As Integer
Dim l As Integer
Dim m As Integer
Dim n As Integer
Dim s As String
Dim s0 As String
Dim s1 As String
Dim s2 As String
Dim s3 As String
Dim s4 As String
Dim s5 As String
Dim s6 As String
Dim s7 As String
Dim s8 As String
Dim s9 As String
Dim s10 As String
Dim s11 As String
Dim s12 As String
Dim s13 As String
Dim s14 As String
Dim found As Boolean
Dim found1 As Boolean
Dim end1proc As Integer
Dim end1port As Integer
Dim end2proc As Integer
Dim end2port As Integer
Dim chs As Integer
Dim s_list As New State_thing
Call s_list.Reset
chs = 0
'select a process file
If Text1.Text = "" Then
Call MsgBox("Select a file to process")
Exit Sub
End If
```

```
' select output file
If Text2.Text = "" Then
Call MsgBox("Select output file name")
Exit Sub
End If
'Read the input file of a model
Set txtStream =
fileSysObject.OpenTextFile(CommonDialog1.FileN
ame, 1)
Set doc = New DOMDocument
doc.loadXML (txtStream.ReadAll)
txtStream.Close
'If there is more than one model in the file, the user
will pick one.
PickModelFrm.Show (vbModal)

If PickModelFrm.Selection = "" Then Exit Sub

ReDim ptypes(10)

Set nodes1 =
doc.getElementsByTagName("Process")
i = 0
While i < nodes1.length
If i + 1 > UBound(ptypes) Then ReDim Preserve
ptypes(UBound(ptypes) + 10)
Set el = nodes1.Item(i)
Set nodes2 = el.getElementsByTagName("Event")
ReDim ptypes(i).ports(10)

ptypes(i).Name = el.getAttribute("Name")

j = 0
While j < nodes2.length 'for each Event of the
Process
If j + 3 > UBound(ptypes(i).ports) Then
ReDim Preserve
ptypes(i).ports(UBound(ptypes(i).ports) + 10)
End If
k = 0
Set tmpel = nodes2.Item(j)
Call s_list.AddState(tmpel.getAttribute("Before"),
ptypes(i).Name)
Call s_list.AddState(tmpel.getAttribute("After"),
ptypes(i).Name)

s = tmpel.getAttribute("Channel")
While ptypes(i).ports(k) <> s And ptypes(i).ports(k)
<> "" 'k < UBound(ptypes(i).ports)
Set tmpel = nodes2.Item(j)
s = tmpel.getAttribute("Channel")
k = k + 1
Wend
ptypes(i).ports(k) = s

k = 0
Set tmpel = nodes2.Item(j)
s = tmpel.getAttribute("Value")
While ptypes(i).ports(k) <> s And ptypes(i).ports(k)
<> "" 'k < UBound(ptypes(i).ports)
Set tmpel = nodes2.Item(j)
s = tmpel.getAttribute("Value")
k = k + 1
Wend
ptypes(i).ports(k) = s 'Doesn't matter if we found it -
just overwrite with the same string

j = j + 1
Wend
i = i + 1
Wend
Set nodes1 =
doc.getElementsByTagName("Instance")
i = 0
Set el = nodes1(i)
s = el.getAttribute("Name")
While s <> PickModelFrm.Selection And i <
nodes1.length
i = i + 1
Set el = nodes1(i)
s = el.getAttribute("Name")
Wend

Set el = nodes1(i)

Set nodes1 =
el.getElementsByTagName("ProcInstance")
ReDim pinsts(nodes1.length + 2)
i = 0
While i < nodes1.length
Set el2 = nodes1.Item(i)
pinsts(i).Name = el2.getAttribute("Name")
pinsts(i).type = el2.getAttribute("Type")

j = 0
While j < UBound(ptypes) And ptypes(j).Name <>
pinsts(i).type
j = j + 1
Wend
If ptypes(j).Name <> pinsts(i).type Then
Call MsgBox("Error finding process type, "" " &
pinsts(i).type)
Exit Sub 'No point in proceeding further
End If

ReDim pinsts(i).ports(UBound(ptypes(j).ports))
ReDim pinsts(i).channels(UBound(ptypes(j).ports))
k = 0
While k < UBound(ptypes(j).ports)
pinsts(i).ports(k) = ptypes(j).ports(k)
k = k + 1
Wend
i = i + 1
Wend

Set nodes1 =
el.getElementsByTagName("Connection")
i = 0
While i < nodes1.length
Set el2 = nodes1.Item(i)
Set nodes2 = el2.getElementsByTagName("End")
If nodes2.length <> 2 Then
Call MsgBox("Connection with wrong number of
ends!")
Exit Sub
End If

Set tmpel = nodes2.Item(0)
s = tmpel.getAttribute("ProcInstance")
end1proc = 0
While pinsts(end1proc).Name <> s
end1proc = end1proc + 1
Wend

s = tmpel.getAttribute("Channel")
end1port = 0
While pinsts(end1proc).ports(end1port) <> s
end1port = end1port + 1
Wend

Set tmpel = nodes2.Item(1)
s = tmpel.getAttribute("ProcInstance")
end2proc = 0
While pinsts(end2proc).Name <> s
```

```
end2proc = end2proc + 1
Wend

s = tmpel.getAttribute("Channel")
end2port = 0
While pinsts(end2proc).ports(end2port) <> s
end2port = end2port + 1
Wend

If pinsts(end1proc).channels(end1port) <> "" And
pinsts(end2proc).channels(end2port) <> "" Then
Call MsgBox("Not able to connect channels
properly")
End If

If pinsts(end1proc).channels(end1port) = "" And
pinsts(end2proc).channels(end2port) = "" Then
pinsts(end1proc).channels(end1port) = "ch" & chs
pinsts(end2proc).channels(end2port) = "ch" & chs
chs = chs + 1
End If

If pinsts(end1proc).channels(end1port) <> "" And
pinsts(end2proc).channels(end2port) = "" Then
pinsts(end2proc).channels(end2port) =
pinsts(end1proc).channels(end1port)
End If

If pinsts(end1proc).channels(end1port) = "" And
pinsts(end2proc).channels(end2port) <> "" Then
pinsts(end1proc).channels(end1port) =
pinsts(end2proc).channels(end2port)
End If
i = i + 1
Wend

'Now write out to a Java file...
Set txtStream =
fileSysObject.OpenTextFile(Text2.Text, 2, True)
txtStream.write "/* Generated from file " &
Text1.Text & " */" & vbNewLine & vbNewLine

txtStream.write "import Java.io.*;" & vbNewLine
txtStream.write "import Java.awt.*;" & vbNewLine
txtStream.write "import Java.math.*;" & vbNewLine
txtStream.write "import Java.util.*;" & vbNewLine
txtStream.write "import Javax.swing.*;" &
vbNewLine
txtStream.write "import Java.awt.event.*;" &
vbNewLine
txtStream.write "import Javax.swing.text.*;" &
vbNewLine
txtStream.write "import Javax.swing.table.*;" &
vbNewLine & vbNewLine

Set nodes0 =
doc.getElementsByTagName("Model")

'First the Model type
'
'Set nodes0 =
doc.getElementsByTagName("Model")
txtStream.write ("public class " &
el.getAttribute("Name") & "{" & vbNewLine &
vbNewLine)

'txtStream.write (" static myGUI traceTable;" &
vbNewLine)

'Message class
txtStream.write ("static class Message {" &
vbNewLine)
txtStream.write (" String type; Process writer;
Process reader; String channel; " & vbNewLine &
vbNewLine)

txtStream.write (" Message (String t, Process p,
Process r, String c){" & vbNewLine)
txtStream.write (" type=t; writer=p; reader=r;
channel=c; " & vbNewLine)
txtStream.write (" }" & vbNewLine & vbNewLine)

txtStream.write (" public String toString(){" &
vbNewLine)
txtStream.write (" return type + "" from "" +
writer.toString() + "" to "" + reader.toString() + "" via
"" +channel; " & vbNewLine)
txtStream.write (" }" & vbNewLine)
txtStream.write ("}" & vbNewLine & vbNewLine)

'MyGUI class
txtStream.write ("static class myGUI extends
JFrame{" & vbNewLine)
txtStream.write (" String[] headerStr =
{""No."",""Event"", ""Type"", ""Before state"", ""After
state"", ""Channel"", ""Value""};" & vbNewLine)
txtStream.write (" DefaultTableModel dm = new
DefaultTableModel(headerStr, 60);" & vbNewLine)
txtStream.write (" JTable table = new JTable(dm);"
& vbNewLine)
txtStream.write (" JPanel center=new JPanel();" &
vbNewLine & vbNewLine)

txtStream.write (" JLabel instanceLabel;" &
vbNewLine)
txtStream.write (" JTextField instanceField;" &
vbNewLine & vbNewLine)

txtStream.write (" JLabel eventsLabel;" &
vbNewLine)
txtStream.write (" JLabel processLabel;" &
vbNewLine)
txtStream.write (" JTextField processField;" &
vbNewLine & vbNewLine)

txtStream.write (" myGUI(String a, String b){" &
vbNewLine)
txtStream.write (" setTitle(""" &
el.getAttribute("Name") & """);" & vbNewLine)
txtStream.write (" setLocation(200,200);" &
vbNewLine)
txtStream.write (" setSize(30,30);" & vbNewLine &
vbNewLine)


'NORTH Panel
txtStream.write (" JPanel top =new JPanel();" &
vbNewLine)
txtStream.write (" top.setBackground(Color.gray);"
& vbNewLine)
txtStream.write (" instanceLabel= new
JLabel(""Instance"");" & vbNewLine)
txtStream.write (" top.add(instanceLabel);" &
vbNewLine & vbNewLine)

txtStream.write (" instanceField=new
JTextField(a,15);" & vbNewLine)
txtStream.write (" Font g =new
Font(""Roman"",Font.PLAIN,12);" & vbNewLine)
txtStream.write (" top.setFont(g);" & vbNewLine)
txtStream.write (" top.add(instanceField);" &
vbNewLine & vbNewLine)

txtStream.write (" processLabel= new
JLabel(""Process"");" & vbNewLine)
txtStream.write (" top.add(processLabel);" &
vbNewLine & vbNewLine)

txtStream.write (" processField=new
JTextField(b,15);" & vbNewLine)
txtStream.write (" Font h =new
Font(""Roman"",Font.ITALIC,12);" & vbNewLine)
txtStream.write (" top.setFont(h);" & vbNewLine)
txtStream.write (" top.add(processField);" &
vbNewLine & vbNewLine)
txtStream.write (" getContentPane().add(top,
BorderLayout.NORTH);" & vbNewLine &
vbNewLine)

'WEST Panel
txtStream.write (" JPanel middle =new JPanel();" &
vbNewLine)
txtStream.write ("
middle.setBackground(Color.green);" & vbNewLine)
txtStream.write (" eventsLabel= new
JLabel(""Possible event(s):"");" & vbNewLine)
txtStream.write (" middle.add(eventsLabel);" &
vbNewLine)
txtStream.write (" getContentPane().add(middle,
BorderLayout.WEST);" & vbNewLine & vbNewLine)

'CETER Panel
txtStream.write ("
center.setBackground(Color.gray);" & vbNewLine)
txtStream.write (" getContentPane().add(center,
BorderLayout.CENTER);" & vbNewLine &
vbNewLine)

'SOUTH Panel
txtStream.write (" JPanel record =new JPanel();" &
vbNewLine)
txtStream.write ("
table.setPreferredScrollableViewportSize(new
Dimension(200, 150));" & vbNewLine)
txtStream.write (" getContentPane().add(new
JScrollPane(table), BorderLayout.SOUTH);" &
vbNewLine & vbNewLine)
txtStream.write (" pack(); " & vbNewLine)
txtStream.write (" setVisible(true);" & vbNewLine &
vbNewLine)
txtStream.write (" }" & vbNewLine & vbNewLine)
txtStream.write (" }" & vbNewLine & vbNewLine)

'MessageQueue class
If (Channel_Length > 0) Then
txtStream.write ("static class MessageQueue{ " &
vbNewLine)
txtStream.write (" int entries;" & vbNewLine)
txtStream.write (" int maxEntries; " & vbNewLine)
txtStream.write (" String name; " & vbNewLine)
txtStream.write (" Message[] elements; " &
vbNewLine & vbNewLine)

txtStream.write (" public MessageQueue(String n,
int m){" & vbNewLine)
txtStream.write (" name=n; " & vbNewLine)
txtStream.write (" maxEntries=m; " & vbNewLine)
txtStream.write (" elements=new
Message[maxEntries]; " & vbNewLine)
txtStream.write (" entries=0; " & vbNewLine)
txtStream.write (" }" & vbNewLine & vbNewLine)

txtStream.write (" synchronized void send(Message
x) throws InterruptedException{ " & vbNewLine)
txtStream.write (" while(entries==maxEntries)wait();
" & vbNewLine)
txtStream.write (" elements[entries]=x; " &
vbNewLine)
txtStream.write (" entries=entries+1; " & vbNewLine)
txtStream.write ("
System.out.println(""send(""+x+"")""); " &
vbNewLine)
txtStream.write (" notify(); " & vbNewLine)
txtStream.write (" } " & vbNewLine & vbNewLine)

txtStream.write (" synchronized Message receive()
throws InterruptedException{ " & vbNewLine)
txtStream.write (" while(entries==0)wait();" &
vbNewLine)
txtStream.write (" Message x; x=elements[0]; " &
vbNewLine)
txtStream.write (" for(int i=1; i<entries; i++) {" &
vbNewLine)
txtStream.write (" elements[i-1]=elements[i]; " &
vbNewLine)
txtStream.write (" } " & vbNewLine)
txtStream.write (" entries=entries-1; " & vbNewLine)
txtStream.write ("
System.out.println(""receive(""+x+"")""); " &
vbNewLine)
txtStream.write (" notify(); " & vbNewLine)
txtStream.write (" return x; " & vbNewLine)
txtStream.write (" } " & vbNewLine)
txtStream.write ("} " & vbNewLine & vbNewLine)

End If

If Channel_Length = 0 Then
txtStream.write ("static class MessageQueue{ " &
vbNewLine)
txtStream.write (" String name; " & vbNewLine)
txtStream.write (" boolean sendFlag, receiveFlag; "
& vbNewLine)
txtStream.write (" Message share; " & vbNewLine &
vbNewLine)

txtStream.write (" public MessageQueue(String n,
int m){" & vbNewLine)
txtStream.write (" name=n; " & vbNewLine)
txtStream.write (" sendFlag=false; " & vbNewLine)
txtStream.write (" receiveFlag=false; " &
vbNewLine)
txtStream.write (" }" & vbNewLine & vbNewLine)

txtStream.write (" synchronized void send(Message
x) throws InterruptedException{ " & vbNewLine)
txtStream.write (" sendFlag=true; " & vbNewLine)
txtStream.write (" share=x; " & vbNewLine)
txtStream.write (" notifyAll(); " & vbNewLine)
txtStream.write ("
System.out.println(""send(""+x+"")""); " &
vbNewLine)
txtStream.write (" while(!receiveFlag) wait(); " &
vbNewLine)
txtStream.write (" receiveFlag=false;" & vbNewLine)
txtStream.write (" } " & vbNewLine & vbNewLine)

txtStream.write (" synchronized Message receive()
throws InterruptedException{ " & vbNewLine)
txtStream.write (" receiveFlag=true;" & vbNewLine)
txtStream.write (" notifyAll(); " & vbNewLine)
txtStream.write (" while(!sendFlag) wait();" &
vbNewLine)
txtStream.write (" Message x; x=share; " &
vbNewLine)
txtStream.write ("
System.out.println(""receive(""+x+"")""); " &
vbNewLine)
txtStream.write (" sendFlag=false; " & vbNewLine)
txtStream.write (" return x; " & vbNewLine)
txtStream.write (" } " & vbNewLine)
txtStream.write ("} " & vbNewLine & vbNewLine)

End If


'Process class
txtStream.write ("static class Process extends
Thread { " & vbNewLine)

Set nodes2 = doc.getElementsByTagName("End")
j = 0
While j < nodes2.length
Set el2 = nodes2.Item(j)
s2 = el2.getAttribute("Channel")
found = False

'search the same one in the END blocks

l = j + 1
While l < nodes2.length

Set el2 = nodes2.Item(l)
If el2.getAttribute("Channel") = s2 Then
found = True
End If

l = l + 1
Wend

If found = False Then
txtStream.write (" MessageQueue " & s2 & ";" &
vbNewLine)
Else
End If
j = j + 1
Wend

'i = i + 1
'Wend


txtStream.write (" String name; " & vbNewLine)
txtStream.write (" public String toString(){ " &
vbNewLine)
txtStream.write (" return this.name; " & vbNewLine)
txtStream.write (" } " & vbNewLine)

txtStream.write ("} " & vbNewLine & vbNewLine)

'Second the process types
Set nodes1 =
doc.getElementsByTagName("Process")
i = 0
While i < nodes1.length 'for each process: write the
start line, write each event, and write the end
Set el = nodes1(i)
s4 = el.getAttribute("Name")

txtStream.write ("static class " &
el.getAttribute("Name") & " extends Process {" &
vbNewLine)
txtStream.write (" static myGUI traceTable;" &
vbNewLine)
txtStream.write (" public " & el.getAttribute("Name")
& " (String name){" & vbNewLine)
txtStream.write (" this.name =name;" & vbNewLine)
txtStream.write (" traceTable=new myGUI(name,"""
& el.getAttribute("Name") & """);" & vbNewLine)

'all channels for each process
Set nodes2 = el.getElementsByTagName("Event")
j = 0
While j < nodes2.length
Set el2 = nodes2.Item(j)
found = False
If el2.getAttribute("Type") = "Read" Then

s2 = el2.getAttribute("Channel")
s3 = el2.getAttribute("Value")

k = j + 1
While k < nodes2.length
Set el = nodes2.Item(k)

If el.getAttribute("Channel") = s2 And
el2.getAttribute("Type") = "Read" Then
found = True
End If

k = k + 1
Wend


If found = False Then
txtStream.write (" " & s2 & "=new
MessageQueue(""" & s2 & """," & Channel_Length
& ");" & vbNewLine)
Else
End If

End If
j = j + 1
Wend

txtStream.write (" this.start();" & vbNewLine)
txtStream.write (" }" & vbNewLine & vbNewLine)

j = 0
While j < nodes2.length
Set el2 = nodes2.Item(j)
'Buttons for the events
txtStream.write (" JButton Button" &
el2.getAttribute("Name") & " = new JButton(""")
txtStream.write (el2.getAttribute("Name") & """ );" &
vbNewLine)
j = j + 1
Wend

txtStream.write (vbNewLine & " public String
state="" "";" & vbNewLine)
txtStream.write (" int noOfevents=0;" & vbNewLine
& vbNewLine)

txtStream.write (" public void run(){ " & vbNewLine)
txtStream.write (" transformState(""initial""); " &
vbNewLine)

'check the type of the event labelled with the initial
state
j = 0
While j < nodes2.length
Set el2 = nodes2.Item(j)
found = False
If el2.getAttribute("Type") <> "Read" And
el2.getAttribute("Before") = "initial" Then
txtStream.write (" initial(); " & vbNewLine)
End If
j = j + 1
Wend

'find the Read Event
j = 0
While j < nodes2.length
Set el2 = nodes2.Item(j)
found = False
If el2.getAttribute("Type") = "Read" Then
s2 = el2.getAttribute("Channel")
s3 = el2.getAttribute("Value")

k = j + 1
While k < nodes2.length
Set el = nodes2.Item(k)

If el.getAttribute("Channel") = s2 And
el2.getAttribute("Type") = "Read" Then
found = True
End If

k = k + 1
Wend

If found = False Then
txtStream.write (" new Thread(){public void run(){" &
vbNewLine)
txtStream.write (" try{for(;;){" & vbNewLine)
txtStream.write (" Message m=(Message)" & s2 &
".receive();" & vbNewLine)
txtStream.write (" if(m.type=="" XXXX_XXXX "" &&
state==null){ }" & vbNewLine)

l = 0
While l < nodes2.length
Set el2 = nodes2.Item(l)
found1 = False

If el2.getAttribute("Channel") = s2 Then
s1 = el2.getAttribute("Name")
s3 = el2.getAttribute("Value")

k = l + 1
While k < nodes2.length
Set el = nodes2.Item(k)

If el.getAttribute("Name") = s1 And
el.getAttribute("Value") = s3 Then
found1 = True
End If

k = k + 1
Wend

If found1 = False Then
txtStream.write (" else if(m.type==""" &
el2.getAttribute("Value") & """")
txtStream.write (" && state ==""" &
el2.getAttribute("Before") & """" & ") " & vbNewLine)
txtStream.write (" " & el2.getAttribute("Name") &
"(m.writer,m.type,""" & el2.getAttribute("Name") &
""");" & vbNewLine)
Else
End If
End If

l = l + 1
Wend

txtStream.write (" }}catch(Exception
e){System.out.println(name + "": demultiplex
error"");}}}.start();" & vbNewLine & vbNewLine)

Else
End If

End If
j = j + 1
Wend
txtStream.write (" }" & vbNewLine & vbNewLine)
''''''''''''''''
'all Before states used of the Write and Create
events
''''''''''''''''
j = 0
While j < nodes2.length
Set el2 = nodes2.Item(j)
found = False
If el2.getAttribute("Type") = "Write" Then
s2 = el2.getAttribute("Name")
s3 = el2.getAttribute("Before")

k = j + 1
While k < nodes2.length
Set el = nodes2.Item(k)

If el.getAttribute("Name") = s2 And
el.getAttribute("Before") = s3 And s3 = "initial" Then
found = True
End If

k = k + 1
Wend

If found = False Then
txtStream.write (" public void " & s3 & "(){ " &
vbNewLine)
txtStream.write (" if(state==""" & s3 & """) " &
vbNewLine)
txtStream.write (" " & s2 & "(); " & vbNewLine)
txtStream.write (" } " & vbNewLine & vbNewLine)
Else
End If

End If

found = False
If el2.getAttribute("Type") = "Create" Then
s2 = el2.getAttribute("Name")
s3 = el2.getAttribute("Before")

k = j + 1
While k < nodes2.length
Set el = nodes2.Item(k)

If el.getAttribute("Name") = s2 And
el.getAttribute("Before") = s3 And s3 = "initial" Then
found = True
End If

k = k + 1
Wend

If found = False Then
txtStream.write (" public void " & s3 & "(){ " &
vbNewLine)
txtStream.write (" if(state==""" & s3 & """) " &
vbNewLine)
txtStream.write (" " & s2 & "(); " & vbNewLine)
txtStream.write (" } " & vbNewLine & vbNewLine)
Else
End If

End If
j = j + 1
Wend

s2 = el2.getAttribute("Name")

k = j
While k < nodes2.length
Set el2 = nodes2.Item(k)
If el2.getAttribute("Name") = s2 Then
```

```
Else

End If

k = k + 1
Wend

j = 0
Do While j < nodes2.length
Set el2 = nodes2.Item(j)
If el2.getAttribute("Type") = "Read" And
el2.getAttribute("Channel") =
el2.getAttribute("Value") Then
'txtStream.write ("chan tmp;" & vbNewLine)
Exit Do
End If
j = j + 1
Loop
'''''


''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''for each event''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''

'Read event
j = 0
While j < nodes2.length
Set el2 = nodes2.Item(j)

found = False
If el2.getAttribute("Name") <> "" And
el2.getAttribute("Type") = "Read" Then
s2 = el2.getAttribute("Before")
s3 = el2.getAttribute("After")

'New code
s1 = el2.getAttribute("Name")
l = InStr(s3, "=")
If l > 0 Then s3 = Left(s3, l - 1) Else s3 = s3
s8 = el2.getAttribute("Name")

k = j + 1
While k < nodes2.length
Set el = nodes2.Item(k)

If el.getAttribute("Type") = "Read" And
el.getAttribute("Before") = s2 And
el.getAttribute("After") = s3 And
el.getAttribute("Name") = s3 Then
found = True
End If

k = k + 1
Wend

If found = False Then
txtStream.write (" public void " & s1)
txtStream.write ("(Process from, String message,
String current_state){" & vbNewLine)
'txtStream.write (" System.out.println(name+""'s
Event is: """)
'txtStream.write ("+ current_state + "" and read """)
'txtStream.write ("+ message + "" from """)
'txtStream.write ("+ from.name);" & vbNewLine)

txtStream.write (" traceTable.center.add(Button" &
el2.getAttribute("Name") & ");" & vbNewLine)
txtStream.write (" Button" & s1 &
".addActionListener( new ActionListener(){" &
vbNewLine)
txtStream.write (" public void
actionPerformed(ActionEvent e){" & vbNewLine)
txtStream.write (" transformState(""")
txtStream.write (s3 & """); " & vbNewLine)

txtStream.write (" displayTrace( """)
txtStream.write (el2.getAttribute("Name") & """, """)
txtStream.write (el2.getAttribute("Type") & """, """)
txtStream.write (s2 & """, """)
txtStream.write (el2.getAttribute("After") & """, """)
txtStream.write (el2.getAttribute("Channel") & """,
""")
txtStream.write (el2.getAttribute("Value") & """); " &
vbNewLine)

k = 0
found = False
While k < nodes2.length
Set el2 = nodes2.Item(k)

'found = False

If el2.getAttribute("Type") <> "Read" And
el2.getAttribute("Before") = s3 Then
found = True
End If

k = k + 1
Wend

If found = True Then
txtStream.write (" " & s3 & "();" & vbNewLine)


End If

txtStream.write (" Button" & s1 & ".setVisible(false);"
& vbNewLine)
txtStream.write (" }" & vbNewLine)
txtStream.write (" });" & vbNewLine)
txtStream.write (" }" & vbNewLine & vbNewLine)
End If
End If
j = j + 1
Wend


'Create event
j = 0
While j < nodes2.length
Set el2 = nodes2.Item(j)

found = False
If el2.getAttribute("Name") <> "" And
el2.getAttribute("Type") = "Create" Then
s1 = el2.getAttribute("Name")
s2 = el2.getAttribute("Before")
s3 = el2.getAttribute("After")
s14 = el2.getAttribute("After")
l = InStr(s3, "=")
If l > 0 Then s3 = Left(s3, l - 1) Else s3 = s3
k = j + 1
While k < nodes2.length
Set el = nodes2.Item(k)

If el.getAttribute("Type") = "Create" And
el.getAttribute("Before") = s2 And
el.getAttribute("After") = s3 Then
found = True
End If

k = k + 1
Wend
```

```
If found = False Then
s5 = el2.getAttribute("Channel")
s6 = el2.getAttribute("Value")

'txtStream.write (" public void " &
el2.getAttribute("Name") & "(){")
txtStream.write (" public void " & s1 & "(){")
' txtStream.write (vbNewLine & "
System.out.println(name+"" : " &
el2.getAttribute("Name") & """);" & vbNewLine)

Set nodes6 =
doc.getElementsByTagName("Instance")
k = 0
While k < nodes6.length
Set el = nodes6(k)

Set nodes7 =
el.getElementsByTagName("ProcInstance")
l = 0
While l < nodes7.length
Set el2 = nodes7.Item(l)
s7 = el2.getAttribute("Name")
s8 = el2.getAttribute("Type")

If s4 = s8 Then

Set nodes8 = el.getElementsByTagName("End")
m = 0
While m < nodes8.length
Set el0 = nodes8.Item(m)
s9 = el0.getAttribute("ProcInstance")
s10 = el0.getAttribute("Channel")

n = m + 1
Set el1 = nodes8.Item(n)
'txtStream.write (el2.getAttribute("ProcInstance") &
"); " & vbNewLine)
s11 = el1.getAttribute("ProcInstance")
s12 = el1.getAttribute("Channel")

'txtStream.write (s11 & " " & s12 & vbNewLine)

If s7 = s9 And s5 = s10 Then
txtStream.write (vbNewLine & "
traceTable.center.add(Button" & s1 & ");" &
vbNewLine)
txtStream.write (" final Message m=new
Message("""")
txtStream.write (s6 & """",this," & s11 & ","""" & s5 &
"""");" & vbNewLine)

txtStream.write (" Button" & s1 &
".addActionListener( new ActionListener(){" &
vbNewLine)
txtStream.write (" public void
actionPerformed(ActionEvent e){" & vbNewLine)

txtStream.write (" try{" & s11 & "." & s12 &
".send(m);" & vbNewLine)
txtStream.write (" transformState("""")
txtStream.write (s3 & """"); " & vbNewLine & " }")
txtStream.write (vbNewLine & " catch(Exception
f){System.out.println(name + "" : " & s1 & "- send
error"");}" & vbNewLine)

End If

If s7 = s11 And s5 = s12 Then
txtStream.write (vbNewLine & "
traceTable.center.add(Button" & s1 & ");" &
vbNewLine)
txtStream.write (" final Message m=new
Message("""")
txtStream.write (s6 & """",this," & s9 & ","""" & s5 &
"""");" & vbNewLine)
txtStream.write (" Button" & s1 &
".addActionListener( new ActionListener(){" &
vbNewLine)
txtStream.write (" public void
actionPerformed(ActionEvent e){" & vbNewLine)

txtStream.write (" try{" & s9 & "." & s10 &
".send(new Message(m);" & vbNewLine)
txtStream.write (" transformState("""")
txtStream.write (s3 & """"); " & vbNewLine & " }")
txtStream.write (vbNewLine & " catch(Exception
f){System.out.println(name + "" : " & s1 & "- send
error"");}" & vbNewLine)

End If

m = m + 2
Wend

End If
l = l + 1

Wend
k = k + 1
Wend

k = 0
found = False
While k < nodes2.length
Set el2 = nodes2.Item(k)

'found = False

If el2.getAttribute("Type") <> "Read" And
el2.getAttribute("Before") = s3 Then
found = True
End If

k = k + 1
Wend

If found = True Then

txtStream.write (" " & s3 & "();" & vbNewLine)
End If

txtStream.write (vbNewLine & " displayTrace( """")
txtStream.write (s1 & """", """")
txtStream.write ("Create""" & ", """")
txtStream.write (s2 & """", """")
txtStream.write (s14 & """", """")
txtStream.write (s5 & """", """")
txtStream.write (s6 & """"); " & vbNewLine)

txtStream.write (" Button" & s1 & ".setVisible(false);"
& vbNewLine)
txtStream.write (" }" & vbNewLine & vbNewLine)
txtStream.write (" });" & vbNewLine & vbNewLine)
txtStream.write (" }" & vbNewLine & vbNewLine)
End If
End If
j = j + 1
Wend


''''''''''''''''''''''''''''''
'Write event''''''''
''''''''''''''''''''''''''''''
j = 0
```

```
While j < nodes2.length
Set el2 = nodes2.Item(j)

found = False
If el2.getAttribute("Name") <> "" And
el2.getAttribute("Type") = "Write" Then
s2 = el2.getAttribute("Before")
s3 = el2.getAttribute("After")
s14 = el2.getAttribute("After")
s1 = el2.getAttribute("Name")
k = j + 1
While k < nodes2.length
Set el = nodes2.Item(k)

If el.getAttribute("Type") = "Write" And
el.getAttribute("Before") = s2 And
el.getAttribute("After") = s3 Then
found = True
End If

k = k + 1
Wend

If found = False Then
txtStream.write (" public void " &
el2.getAttribute("Name") & "(){")
txtStream.write (vbNewLine & "
System.out.println(name+"" : " &
el2.getAttribute("Name") & """);" & vbNewLine)

s5 = el2.getAttribute("Channel")
s6 = el2.getAttribute("Value")

l = InStr(s3, "=")
If l > 0 Then s3 = Left(s3, l - 1) Else s3 = s3

Set nodes6 =
doc.getElementsByTagName("Instance")
k = 0
While k < nodes6.length
Set el = nodes6(k)

Set nodes7 =
el.getElementsByTagName("ProcInstance")
l = 0
While l < nodes7.length
Set el2 = nodes7.Item(l)
s7 = el2.getAttribute("Name")
s8 = el2.getAttribute("Type")

If s4 = s8 Then

Set nodes8 = el.getElementsByTagName("End")
m = 0
While m < nodes8.length
Set el0 = nodes8.Item(m)
s9 = el0.getAttribute("ProcInstance")

s10 = el0.getAttribute("Channel")

n = m + 1
Set el1 = nodes8.Item(n)

s11 = el1.getAttribute("ProcInstance")
s12 = el1.getAttribute("Channel")

If s7 = s9 And s10 = s5 Then

txtStream.write (vbNewLine & "
traceTable.center.add(Button" & s1 & ");" &
vbNewLine)
txtStream.write (" final Message m=new
Message(""")
txtStream.write (s6 & """,this," & s11 & ",""" & s12 &
""");" & vbNewLine)
txtStream.write (" Button" & s1 &
".addActionListener( new ActionListener(){" &
vbNewLine)
txtStream.write (" public void
actionPerformed(ActionEvent e){" & vbNewLine)



txtStream.write (" try{" & s11 & "." & s12 &
".send(m);" & vbNewLine)
'txtStream.write (s6 & """,this," & s11 & ",""" & s12 &
"""));" & vbNewLine)
txtStream.write (" transformState(""")
txtStream.write (s3 & """); " & vbNewLine & " }")
txtStream.write (vbNewLine & " catch(Exception
f){System.out.println(name + "" : " &
el2.getAttribute("Name") & "- send error"");}")


End If

If s7 = s11 And s12 = s5 Then

txtStream.write (vbNewLine & "
traceTable.center.add(Button" & s1 & ");" &
vbNewLine)
txtStream.write (" final Message m=new
Message(""")
txtStream.write (s6 & """,this," & s9 & ",""" & s10 &
""");" & vbNewLine)
txtStream.write (" Button" & s1 &
".addActionListener( new ActionListener(){" &
vbNewLine)
txtStream.write (" public void
actionPerformed(ActionEvent e){" & vbNewLine)

txtStream.write (" try{" & s9 & "." & s10 & ".send(m);"
& vbNewLine)
' txtStream.write (s6 & """,this," & s9 & ",""" & s10 &
"""));" & vbNewLine)
txtStream.write (" transformState(""")
txtStream.write (s3 & """); " & vbNewLine & " }")
txtStream.write (vbNewLine & " catch(Exception
f){System.out.println(name + "" : " &
el2.getAttribute("Name") & "- send error"");}")


End If

m = m + 2
Wend

End If
l = l + 1

Wend
k = k + 1
Wend

k = 0
found = False
While k < nodes2.length
Set el2 = nodes2.Item(k)


If el2.getAttribute("Type") <> "Read" And
el2.getAttribute("Before") = s3 Then
found = True
End If

k = k + 1
Wend

If found = True Then

txtStream.write (" " & s3 & "();")
End If


txtStream.write (vbNewLine & " displayTrace( """)
txtStream.write (s1 & """, """)
txtStream.write ("Write""" & ", """)
txtStream.write (s2 & """, """)
txtStream.write (s14 & """, """)
txtStream.write (s5 & """, """)
txtStream.write (s6 & """); " & vbNewLine)

txtStream.write (" Button" & s1 & ".setVisible(false);"
& vbNewLine)
txtStream.write (" }" & vbNewLine & vbNewLine)
txtStream.write (" });" & vbNewLine & vbNewLine)

txtStream.write (vbNewLine & " }" & vbNewLine &
vbNewLine)
End If

End If
j = j + 1
Wend


j = 0
While j < nodes2.length
Set el2 = nodes2.Item(j)


s2 = el2.getAttribute("Before")

k = j
While k < nodes2.length
Set el2 = nodes2.Item(k)

If el2.getAttribute("Before") = s2 Then
If el2.getAttribute("Before") = "initial" Then

Call s_list.MarkUsed(el2.getAttribute("Type"),
el.getAttribute("Name"))

End If

'Remove any trailing "=" from the new state name...
l = InStr(el2.getAttribute("After"), "=")
If l > 0 Then s = Left(el2.getAttribute("After"), l - 1)
Else s = el2.getAttribute("After")
'txtStream.write (vbNewLine & " " & s & "();" &
vbNewLine)
Call el2.setAttribute("Name", "")
End If
k = k + 1
Wend
j = j + 1
Wend


'add connection to the potential processes
Set nodes3 =
doc.getElementsByTagName("Instance")
k = 0
While k < nodes3.length
Set el = nodes3(k)

Set nodes4 =
el.getElementsByTagName("ProcInstance")
l = 0
While l < nodes4.length
Set el2 = nodes4.Item(l)
s2 = el2.getAttribute("Name")

txtStream.write (" Process " & s2 & ";" & vbNewLine)
txtStream.write (" public void connection_" & s2 &
"(Process temp){" & vbNewLine)
txtStream.write (" " & s2 & "=temp;" & vbNewLine)
txtStream.write (" }" & vbNewLine & vbNewLine)
l = l + 1
Wend
k = k + 1
Wend

txtStream.write (" public String getname(){" &
vbNewLine)
txtStream.write (" return name;" & vbNewLine)
txtStream.write (" }" & vbNewLine & vbNewLine)

txtStream.write (" public void displayTrace(String
ev, String ty, String be, String af, String ch, String
va){" & vbNewLine)
txtStream.write (" traceTable.table.setValueAt((new
Integer(noOfevents)).toString(), noOfevents,0); " &
vbNewLine)
txtStream.write (" traceTable.table.setValueAt(ev,
noOfevents,1); " & vbNewLine)
txtStream.write (" traceTable.table.setValueAt(ty,
noOfevents,2); " & vbNewLine)
txtStream.write (" traceTable.table.setValueAt(be,
noOfevents,3);" & vbNewLine)
txtStream.write (" traceTable.table.setValueAt(af,
noOfevents,4);" & vbNewLine)
txtStream.write (" traceTable.table.setValueAt(ch,
noOfevents,5);" & vbNewLine)
txtStream.write (" traceTable.table.setValueAt(va,
noOfevents,6);" & vbNewLine)
txtStream.write (" noOfevents++;" & vbNewLine)
txtStream.write (" }" & vbNewLine & vbNewLine)

txtStream.write (" public void transformState(String
s){" & vbNewLine)
txtStream.write (" state=s;" & vbNewLine)
txtStream.write (" System.out.println(name +"" : "" +
state);" & vbNewLine)
txtStream.write (" }" & vbNewLine)


txtStream.write ("}" & vbNewLine & vbNewLine)


i = i + 1
Wend


'here it is the main method

txtStream.write ("public static void main(String
args[]) { " & vbNewLine)


Set nodes1 =
doc.getElementsByTagName("Instance")
k = 0
While k < nodes1.length
Set el = nodes1(k)
Set nodes1 =
el.getElementsByTagName("ProcInstance")

i = 0
While i < nodes1.length
Set el2 = nodes1.Item(i)
```

```
txtStream.write (" " & el2.getAttribute("Type") & " " &
el2.getAttribute("Name") & " = new ")
txtStream.write (el2.getAttribute("Type") & "(""" &
el2.getAttribute("Name") & """); " & vbNewLine)
i = i + 1

Wend

txtStream.write (vbNewLine)


'one.connection(another)
'
Set nodes1 = el.getElementsByTagName("End")
i = 0
While i < nodes1.length
Set el2 = nodes1.Item(i)
s2 = el2.getAttribute("ProcInstance")

k = i + 1
Set el2 = nodes1.Item(k)
s3 = el2.getAttribute("ProcInstance")
txtStream.write (" " & s2 & ".connection_" & s3 & "("
& s3 & ");")
txtStream.write (" " & s3 & ".connection_" & s2 & "("
& s2 & ");" & vbNewLine)
i = i + 2
Wend

k = k + 1
Wend


txtStream.write ("} " & vbNewLine & vbNewLine &
"}")

txtStream.Close
Call MsgBox("Done!")
End Sub

Private Sub Form_Load()
Set fileSysObject =
CreateObject("Scripting.FileSystemObject")

Channel_Length = 10
Text3.Text = Channel_Length

End Sub

Private Sub MnuChannel_Length_Click()
Dim str As String

str = InputBox("Enter the required length of
MessageQueue" & vbNewLine & "Present length: "
& Channel_Length, "MessageQueue Length", 10)

If str = "" Then Exit Sub 'Assume cancel has been
used

If Val(str) < 0 Or (Val(str) = 0 And str <> "10") Then
Call MsgBox(str & " is not a valid value, Message
queue length will be set to 10", vbOKOnly, "Error")
Channel_Length = 10
Else
Channel_Length = Val(str)
Text3.Text = str
End If
End Sub


Private Sub Text3_Change()
Channel_Length = Val(Text3.Text)
End Sub
```

Appendix B Example Models in XML

B.1 The XML Generated by the RDT Tool for a Cycle Election Model (Model 1)

```
<Model>
    <Instance Name="cycle">
        <ProcInstance Name="p0"
Type="participant0"/>
        <ProcInstance Name="p1"
Type="participant1"/>
        <ProcInstance Name="p2"
Type="participant2"/>
        <ProcInstance Name="p3"
Type="participant3"/>
        <ProcInstance Name="Tr" Type="trigger"/>
        <Connection>
            <End ProcInstance="Tr"
Channel="startElection"/>
            <End ProcInstance="p3"
Channel="election"/>
        </Connection>
        <Connection>
            <End ProcInstance="p0"
Channel="Ps_election"/>
            <End ProcInstance="p2"
Channel="election"/>
        </Connection>
        <Connection>
            <End ProcInstance="p0"
Channel="Ps_elected"/>
            <End ProcInstance="p2"
Channel="elected"/>
        </Connection>
        <Connection>
            <End ProcInstance="p1"
Channel="Ps_election"/>
            <End ProcInstance="p0"
Channel="election"/>
        </Connection>
        <Connection>
            <End ProcInstance="p1"
Channel="Ps_elected"/>
            <End ProcInstance="p0"
Channel="elected"/>
        </Connection>
        <Connection>
            <End ProcInstance="p2"
Channel="Ps_election"/>
            <End ProcInstance="p3"
Channel="election"/>
        </Connection>
        <Connection>
            <End ProcInstance="p2"
Channel="Ps_elected"/>
            <End ProcInstance="p3"
Channel="elected"/>
        </Connection>
        <Connection>
            <End ProcInstance="p3"
Channel="Ps_election"/>
            <End ProcInstance="p1"
Channel="election"/>
        </Connection>
        <Connection>
            <End ProcInstance="p3"
Channel="Ps_elected"/>
            <End ProcInstance="p1"
Channel="elected"/>
        </Connection>
    </Instance>
    <Process Name="participant0">
        <Event Name="R_Election_0"
Type="Read" Before="initial" After="REN0"
Channel="election" Value="V0"/>
        <Event Name="S_Elected_0" Type="Write"
Before="REN0" After="initial="
Channel="Ps_elected" Value="V0"/>
        <Event Name="R_Election_1"
Type="Read" Before="initial" After="REN1"
Channel="election" Value="V1"/>
        <Event Name="S_Election_1"
Type="Write" Before="REN1" After="initial="
Channel="Ps_election" Value="V1"/>
        <Event Name="R_Election_2"
Type="Read" Before="initial" After="REN2"
Channel="election" Value="V2"/>
```

```
            <Event Name="S_Election_2"
Type="Write" Before="REN2" After="initial="
Channel="Ps_election" Value="V2"/>
            <Event Name="R_Election_3"
Type="Read" Before="initial" After="REN3"
Channel="election" Value="V3"/>
            <Event Name="S_Election_3"
Type="Create" Before="REN3" After="initial="
Channel="Ps_election" Value="V3"/>
            <Event Name="R_Elected_0"
Type="Read" Before="initial" After="IMBoss"
Channel="elected" Value="V0"/>
            <Event Name="R_Elected_1"
Type="Read" Before="initial" After="RED1"
Channel="elected" Value="V1"/>
            <Event Name="S_ED_1" Type="Write"
Before="RED1" After="Boss1"
Channel="Ps_elected" Value="V1"/>
            <Event Name="R_Elected_2"
Type="Read" Before="initial" After="RED2"
Channel="elected" Value="V2"/>
            <Event Name="S_ED_2" Type="Write"
Before="RED2" After="Boss2"
Channel="Ps_elected" Value="V2"/>
            <Event Name="R_Elected_3"
Type="Read" Before="initial" After="RED3"
Channel="elected" Value="V3"/>
            <Event Name="S_ED_3" Type="Create"
Before="RED3" After="Boss3"
Channel="Ps_elected" Value="V3"/>
      </Process>
      <Process Name="participant1">
            <Event Name="R_Election_0"
Type="Read" Before="initial" After="REN0"
Channel="election" Value="V0"/>
            <Event Name="S_Election1_0"
Type="Create" Before="REN0" After="initial="
Channel="Ps_election" Value="V1"/>
            <Event Name="R_Election_1"
Type="Read" Before="initial" After="REN1"
Channel="election" Value="V1"/>
            <Event Name="S_Elected_1" Type="Write"
Before="REN1" After="initial="
Channel="Ps_elected" Value="V1"/>
            <Event Name="R_Election_2"
Type="Read" Before="initial" After="REN2"
Channel="election" Value="V2"/>
            <Event Name="S_Election_2"
Type="Write" Before="REN2" After="initial="
Channel="Ps_election" Value="V2"/>
            <Event Name="R_Election_3"
Type="Read" Before="initial" After="REN3"
Channel="election" Value="V3"/>
            <Event Name="S_Election_3"
Type="Write" Before="REN3" After="initial="
Channel="Ps_election" Value="V3"/>
            <Event Name="R_Elected_1"
Type="Read" Before="initial" After="IMBoss"
Channel="elected" Value="V1"/>
            <Event Name="R_Elected_2"
Type="Read" Before="initial" After="RED2"
Channel="elected" Value="V2"/>
            <Event Name="S_ED_2" Type="Write"
Before="RED2" After="Boss2"
Channel="Ps_elected" Value="V2"/>
            <Event Name="R_Elected_3"
Type="Read" Before="initial" After="RED3"
Channel="elected" Value="V3"/>
            <Event Name="S_ED_3" Type="Create"
Before="RED3" After="Boss3"
Channel="Ps_elected" Value="V3"/>
      </Process>
      <Process Name="participant2">
            <Event Name="R_Election_0"
Type="Read" Before="initial" After="REN0"
Channel="election" Value="V0"/>
            <Event Name="S_Election2_0"
Type="Write" Before="REN0" After="initial="
Channel="Ps_election" Value="V2"/>
            <Event Name="R_Election_1"
Type="Read" Before="initial" After="REN1"
Channel="election" Value="V1"/>
            <Event Name="S_Election2_1"
Type="Create" Before="REN1" After="initial="
Channel="Ps_election" Value="V2"/>
            <Event Name="R_Election_2"
Type="Read" Before="initial" After="REN2"
Channel="election" Value="V2"/>
            <Event Name="S_Elected_2" Type="Write"
Before="REN2" After="initial="
Channel="Ps_elected" Value="V2"/>
            <Event Name="R_Election_3"
Type="Read" Before="initial" After="REN3"
Channel="election" Value="V3"/>
            <Event Name="S_Election_3"
Type="Write" Before="REN3" After="initial="
Channel="Ps_election" Value="V3"/>
            <Event Name="R_Elected_2"
Type="Read" Before="initial" After="IMBoss"
Channel="elected" Value="V2"/>
            <Event Name="R_Elected_3"
Type="Read" Before="initial" After="RED3"
Channel="elected" Value="V3"/>
            <Event Name="S_ED_3" Type="Create"
Before="RED3" After="Boss3"
Channel="Ps_elected" Value="V3"/>
      </Process>
      <Process Name="participant3">
            <Event Name="R_Election_0"
Type="Read" Before="initial" After="REN0"
Channel="election" Value="V0"/>
            <Event Name="S_Election3_0"
Type="Create" Before="REN0" After="initial="
Channel="Ps_election" Value="V3"/>
            <Event Name="R_Election_1"
Type="Read" Before="initial" After="REN1"
Channel="election" Value="V1"/>
            <Event Name="S_Election3_1"
Type="Write" Before="REN1" After="initial="
Channel="Ps_election" Value="V3"/>
            <Event Name="R_Election_2"
Type="Read" Before="initial" After="REN2"
Channel="election" Value="V2"/>
            <Event Name="S_Election3_2"
Type="Write" Before="REN2" After="initial="
Channel="Ps_election" Value="V3"/>
            <Event Name="R_Election_3"
Type="Read" Before="initial" After="REN3"
Channel="election" Value="V3"/>
            <Event Name="S_Elected_3"
Type="Create" Before="REN3" After="initial="
Channel="Ps_elected" Value="V3"/>
            <Event Name="R_Elected_3"
Type="Read" Before="initial" After="IMBoss"
Channel="elected" Value="V3"/>
      </Process>
      <Process Name="trigger">
            <Event Name="start_election"
Type="Create" Before="initial" After="finish"
Channel="startElection" Value="V0"/>
      </Process>
</Model>
```

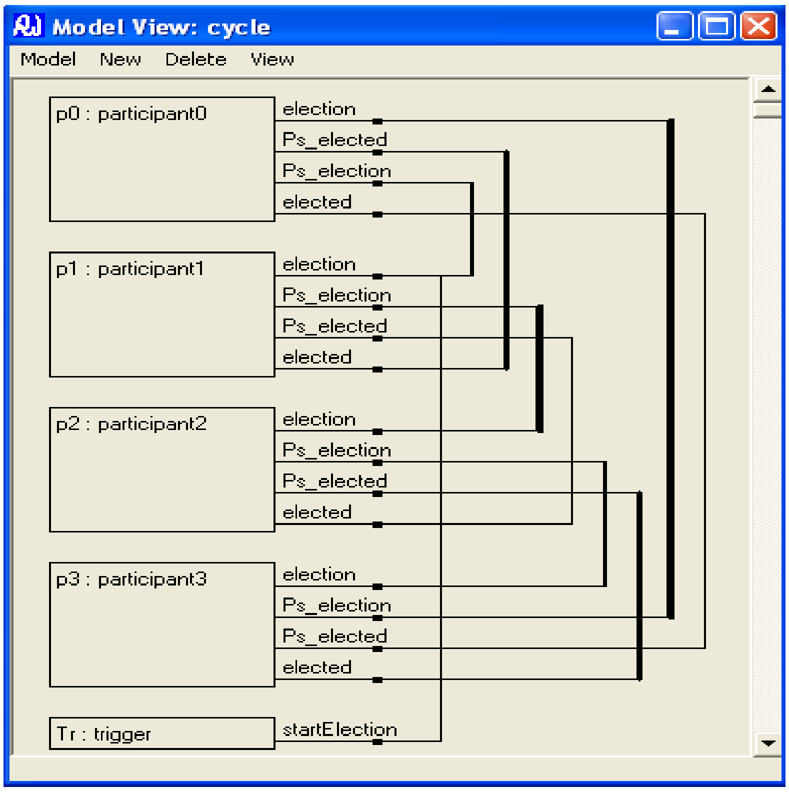


Figure 72: Model view of a cycle election model

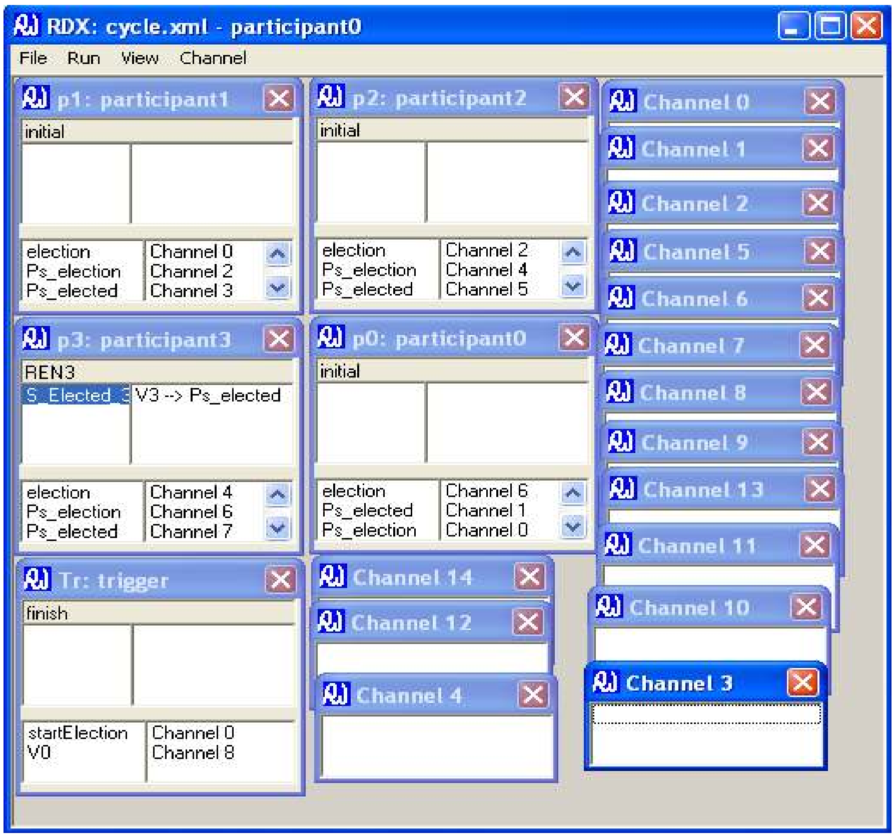


Figure 73: The cycle election model during execution (Model 1)

B.2 The XML Generated by the RDT Tool for a Cycle Election Model (Model 2)

```
<Model>
    <Instance Name="cycle">
        <ProcInstance Name="p0"
Type="participant0"/>
        <ProcInstance Name="p1"
Type="participant1"/>
        <ProcInstance Name="p2"
Type="participant2"/>
        <ProcInstance Name="p3"
Type="participant3"/>
        <Connection>
            <End ProcInstance="p0"
Channel="Ps_election"/>
            <End ProcInstance="p2"
Channel="election"/>
        </Connection>
        <Connection>
            <End ProcInstance="p0"
Channel="Ps_elected"/>
            <End ProcInstance="p2"
Channel="elected"/>
        </Connection>
        <Connection>
            <End ProcInstance="p1"
Channel="Ps_election"/>
            <End ProcInstance="p0"
Channel="election"/>
        </Connection>
        <Connection>
            <End ProcInstance="p1"
Channel="Ps_elected"/>
            <End ProcInstance="p0"
Channel="elected"/>
        </Connection>
        <Connection>
            <End ProcInstance="p2"
Channel="Ps_election"/>
            <End ProcInstance="p3"
Channel="election"/>
        </Connection>
        <Connection>
            <End ProcInstance="p2"
Channel="Ps_elected"/>
            <End ProcInstance="p3"
Channel="elected"/>
        </Connection>
        <Connection>
            <End ProcInstance="p3"
Channel="Ps_election"/>
            <End ProcInstance="p1"
```

```
Channel="election"/>
            </Connection>
            <Connection>
                <End ProcInstance="p3"
Channel="Ps_elected"/>
                <End ProcInstance="p1"
Channel="elected"/>
            </Connection>
      </Instance>
      <Process Name="participant0">
            <Event Name="SE_0" Type="Create"
Before="initial" After="SE" Channel="Ps_election"
Value="V0"/>
            <Event Name="R_Election_0"
Type="Read" Before="SE" After="REN0"
Channel="election" Value="V0"/>
            <Event Name="S_Elected_0" Type="Write"
Before="REN0" After="SE=" Channel="Ps_elected"
Value="V0"/>
            <Event Name="R_Election_1"
Type="Read" Before="SE" After="REN1"
Channel="election" Value="V1"/>
            <Event Name="S_Election_1"
Type="Write" Before="REN1" After="SE="
Channel="Ps_election" Value="V1"/>
            <Event Name="R_Election_2"
Type="Read" Before="SE" After="REN2"
Channel="election" Value="V2"/>
            <Event Name="S_Election_2"
Type="Write" Before="REN2" After="SE="
Channel="Ps_election" Value="V2"/>
            <Event Name="R_Election_3"
Type="Read" Before="SE" After="REN3"
Channel="election" Value="V3"/>
            <Event Name="S_Election_3"
Type="Write" Before="REN3" After="SE="
Channel="Ps_election" Value="V3"/>
            <Event Name="R_Elected_0"
Type="Read" Before="SE" After="IMBoss"
Channel="elected" Value="V0"/>
            <Event Name="R_Elected_1"
Type="Read" Before="SE" After="RED1"
Channel="elected" Value="V1"/>
            <Event Name="S_ED_1" Type="Write"
Before="RED1" After="Boss1"
Channel="Ps_elected" Value="V1"/>
            <Event Name="R_Elected_2"
Type="Read" Before="SE" After="RED2"
Channel="elected" Value="V2"/>
            <Event Name="S_ED_2" Type="Write"
Before="RED2" After="Boss2"
Channel="Ps_elected" Value="V2"/>
            <Event Name="R_Elected_3"
Type="Read" Before="SE" After="RED3"
Channel="elected" Value="V3"/>
            <Event Name="S_ED_3" Type="Create"
Before="RED3" After="Boss3"
Channel="Ps_elected" Value="V3"/>
      </Process>
      <Process Name="participant1">
            <Event Name="SE_1" Type="Create"
Before="initial" After="SE" Channel="Ps_election"
Value="V1"/>
            <Event Name="R_Election_0"
Type="Read" Before="SE" After="REN0"
Channel="election" Value="V0"/>
            <Event Name="S_Election1_0"
Type="Write" Before="REN0" After="SE="
Channel="Ps_election" Value="V1"/>
            <Event Name="R_Election_1"
Type="Read" Before="SE" After="REN1"
Channel="election" Value="V1"/>
            <Event Name="S_Elected_1" Type="Write"
Before="REN1" After="SE=" Channel="Ps_elected"
Value="V1"/>
            <Event Name="R_Election_2"
Type="Read" Before="SE" After="REN2"
Channel="election" Value="V2"/>
            <Event Name="S_Election_2"
Type="Write" Before="REN2" After="SE="
Channel="Ps_election" Value="V2"/>
            <Event Name="R_Election_3"
Type="Read" Before="SE" After="REN3"
Channel="election" Value="V3"/>
            <Event Name="S_Election_3"
Type="Write" Before="REN3" After="SE="
Channel="Ps_election" Value="V3"/>
            <Event Name="R_Elected_1"
Type="Read" Before="SE" After="IMBoss"
Channel="elected" Value="V1"/>
            <Event Name="R_Elected_2"
Type="Read" Before="SE" After="RED2"
Channel="elected" Value="V2"/>
            <Event Name="S_ED_2" Type="Write"
Before="RED2" After="Boss2"
Channel="Ps_elected" Value="V2"/>
            <Event Name="R_Elected_3"
Type="Read" Before="SE" After="RED3"
Channel="elected" Value="V3"/>
            <Event Name="S_ED_3" Type="Create"
Before="RED3" After="Boss3"
Channel="Ps_elected" Value="V3"/>
      </Process>
      <Process Name="participant2">
            <Event Name="SE_2" Type="Create"
Before="initial" After="SE" Channel="Ps_election"
Value="V2"/>
            <Event Name="R_Election_0"
Type="Read" Before="SE" After="REN0"
Channel="election" Value="V0"/>
            <Event Name="S_Election2_0"
Type="Write" Before="REN0" After="SE="
Channel="Ps_election" Value="V2"/>
            <Event Name="R_Election_1"
Type="Read" Before="SE" After="REN1"
Channel="election" Value="V1"/>
            <Event Name="S_Election2_1"
Type="Write" Before="REN1" After="SE="
Channel="Ps_election" Value="V2"/>
            <Event Name="R_Election_2"
Type="Read" Before="SE" After="REN2"
Channel="election" Value="V2"/>
            <Event Name="S_Elected_2" Type="Write"
Before="REN2" After="SE=" Channel="Ps_elected"
Value="V2"/>
            <Event Name="R_Election_3"
Type="Read" Before="SE" After="REN3"
Channel="election" Value="V3"/>
            <Event Name="S_Election_3"
Type="Write" Before="REN3" After="SE="
Channel="Ps_election" Value="V3"/>
            <Event Name="R_Elected_2"
Type="Read" Before="SE" After="IMBoss"
Channel="elected" Value="V2"/>
            <Event Name="R_Elected_3"
Type="Read" Before="SE" After="RED3"
Channel="elected" Value="V3"/>
            <Event Name="S_ED_3" Type="Create"
Before="RED3" After="Boss3"
Channel="Ps_elected" Value="V3"/>
      </Process>
      <Process Name="participant3">
            <Event Name="SE_3" Type="Create"
Before="initial" After="SE" Channel="Ps_election"
Value="V3"/>
            <Event Name="R_Election_0"
```

```
Type="Read" Before="SE" After="REN0"
Channel="election" Value="V0"/>
        <Event Name="S_Election3_0"
Type="Write" Before="REN0" After="SEI="
Channel="Ps_election" Value="V3"/>
        <Event Name="R_Election_1"
Type="Read" Before="SE" After="REN1"
Channel="election" Value="V1"/>
        <Event Name="S_Election3_1"
Type="Write" Before="REN1" After="SE="
Channel="Ps_election" Value="V3"/>
        <Event Name="R_Election_2"
Type="Read" Before="SE" After="REN2"
Channel="election" Value="V2"/>
        <Event Name="S_Election3_2"
Type="Write" Before="REN2" After="SE="
Channel="Ps_election" Value="V3"/>
        <Event Name="R_Election_3"
Type="Read" Before="SE" After="REN3"
Channel="election" Value="V3"/>
        <Event Name="S_Elected_3"
Type="Create" Before="REN3" After="SE="
Channel="Ps_elected" Value="V3"/>
        <Event Name="R_Elected_3"
Type="Read" Before="SE" After="IMBoss"
Channel="elected" Value="V3"/>
    </Process>
</Model>
```

B.3 A Bully Model

```
<Model>
    <Instance Name="bully">
        <ProcInstance Name="p0"
Type="participant0"/>
        <ProcInstance Name="p1"
Type="participant1"/>
        <ProcInstance Name="p2"
Type="participant2"/>
        <ProcInstance Name="p3"
Type="participant3"/>
        <ProcInstance Name="sink" Type="Sink"/>
        <Connection>
            <End ProcInstance="p0"
Channel="info"/>
            <End ProcInstance="sink"
Channel="inbox"/>
        </Connection>
        <Connection>
            <End ProcInstance="p1"
Channel="info"/>
            <End ProcInstance="sink"
Channel="inbox"/>
        </Connection>
        <Connection>
            <End ProcInstance="p2"
Channel="info"/>
            <End ProcInstance="sink"
Channel="inbox"/>
        </Connection>
        <Connection>
            <End ProcInstance="p3"
Channel="info"/>
            <End ProcInstance="sink"
Channel="inbox"/>
        </Connection>
        <Connection>
            <End ProcInstance="p0"
Channel="election"/>
            <End ProcInstance="p1"
Channel="inbox_0"/>
        </Connection>
        <Connection>
            <End ProcInstance="p0"
Channel="election"/>
            <End ProcInstance="p2"
Channel="inbox_0"/>
        </Connection>
        <Connection>
            <End ProcInstance="p0"
Channel="election"/>
            <End ProcInstance="p3"
Channel="inbox_0"/>
        </Connection>
        <Connection>
            <End ProcInstance="p1"
Channel="election"/>
            <End ProcInstance="p0"
Channel="inbox_1"/>
        </Connection>
        <Connection>
            <End ProcInstance="p1"
Channel="election"/>
            <End ProcInstance="p2"
Channel="inbox_1"/>
        </Connection>
        <Connection>
            <End ProcInstance="p1"
Channel="election"/>
            <End ProcInstance="p3"
Channel="inbox_1"/>
        </Connection>
        <Connection>
            <End ProcInstance="p2"
Channel="election"/>
            <End ProcInstance="p0"
Channel="inbox_2"/>
        </Connection>
        <Connection>
            <End ProcInstance="p2"
Channel="election"/>
            <End ProcInstance="p1"
Channel="inbox_2"/>
        </Connection>
        <Connection>
            <End ProcInstance="p2"
Channel="election"/>
            <End ProcInstance="p3"
Channel="inbox_2"/>
        </Connection>
        <Connection>
            <End ProcInstance="p3"
Channel="election"/>
            <End ProcInstance="p0"
Channel="inbox_3"/>
        </Connection>
        <Connection>
            <End ProcInstance="p3"
Channel="election"/>
```

```
            <End ProcInstance="p1"
Channel="inbox_3"/>
        </Connection>
        <Connection>
            <End ProcInstance="p3"
Channel="election"/>
            <End ProcInstance="p2"
Channel="inbox_3"/>
        </Connection>
    </Instance>
    <Process Name="participant0">
        <Event Name="send_0" Type="Create"
Before="initial" After="wait_receive"
Channel="election" Value="V0"/>
        <Event Name="receive_0" Type="Read"
Before="wait_receive" After="wait_receive="
Channel="inbox_0" Value="V0"/>
        <Event Name="receive_1" Type="Read"
Before="wait_receive" After="wait_receive="
Channel="inbox_1" Value="V1"/>
        <Event Name="receive_2" Type="Read"
Before="wait_receive" After="wait_receive="
Channel="inbox_2" Value="V2"/>
        <Event Name="receive_3" Type="Read"
Before="wait_receive" After="received3"
Channel="inbox_3" Value="V3"/>
        <Event Name="get_boss" Type="Create"
Before="received3" After="finish" Channel="info"
Value="boss"/>
    </Process>
    <Process Name="participant1">
        <Event Name="send_1" Type="Create"
Before="initial" After="wait_receive"
Channel="election" Value="V1"/>
        <Event Name="receive_0" Type="Read"
Before="wait_receive" After="wait_receive="
Channel="inbox_0" Value="V0"/>
        <Event Name="receive_1" Type="Read"
Before="wait_receive" After="wait_receive="
Channel="inbox_1" Value="V1"/>
        <Event Name="receive_2" Type="Read"
Before="wait_receive" After="wait_receive="
Channel="inbox_2" Value="V2"/>
        <Event Name="receive_3" Type="Read"
Before="wait_receive" After="received3"
Channel="inbox_3" Value="V3"/>
        <Event Name="get_boss" Type="Create"
Before="received3" After="finish" Channel="info"
Value="boss"/>
    </Process>
    <Process Name="participant2">
        <Event Name="send_2" Type="Create"
Before="initial" After="wait_receive"
Channel="election" Value="V2"/>
        <Event Name="receive_0" Type="Read"
Before="wait_receive" After="wait_receive="
Channel="inbox_0" Value="V0"/>
        <Event Name="receive_1" Type="Read"
Before="wait_receive" After="wait_receive="
Channel="inbox_1" Value="V1"/>
        <Event Name="receive_2" Type="Read"
Before="wait_receive" After="wait_receive="
Channel="inbox_2" Value="V2"/>
        <Event Name="receive_3" Type="Read"
Before="wait_receive" After="received3"
Channel="inbox_3" Value="V3"/>
        <Event Name="get_boss" Type="Create"
Before="received3" After="finish" Channel="info"
Value="boss"/>
    </Process>
    <Process Name="participant3">
        <Event Name="send_3" Type="Create"
Before="initial" After="wait_receive"
Channel="election" Value="V3"/>
        <Event Name="receive_0" Type="Read"
Before="wait_receive" After="wait_receive="
Channel="inbox_0" Value="V0"/>
        <Event Name="receive_1" Type="Read"
Before="wait_receive" After="wait_receive="
Channel="inbox_1" Value="V1"/>
        <Event Name="receive_2" Type="Read"
Before="wait_receive" After="wait_receive="
Channel="inbox_2" Value="V2"/>
        <Event Name="receive_3" Type="Read"
Before="wait_receive" After="received3"
Channel="inbox_3" Value="V3"/>
        <Event Name="get_boss" Type="Create"
Before="received3" After="finish" Channel="info"
Value="boss"/>
    </Process>
    <Process Name="Sink">
        <Event Name="know_boss" Type="Read"
Before="initial" After="initial=" Channel="inbox"
Value="boss"/>
    </Process>
</Model>
```

B.4 A Probe/Echo Model

```
<Model>
    <Instance Name="probeEcho">
        <ProcInstance Name="grandF"
Type="parent"/>
        <ProcInstance Name="father"
Type="child1"/>
        <ProcInstance Name="brother"
Type="child2"/>
        <ProcInstance Name="uncle"
Type="child2"/>
        <ProcInstance Name="sister"
Type="child2"/>
        <Connection>
            <End ProcInstance="grandF"
Channel="down"/>
            <End ProcInstance="father"
Channel="inbox_parent"/>
        </Connection>
        <Connection>
            <End ProcInstance="grandF"
Channel="down"/>
            <End ProcInstance="uncle"
Channel="inbox"/>
        </Connection>
        <Connection>
            <End ProcInstance="father"
Channel="down"/>
            <End ProcInstance="brother"
Channel="inbox"/>
        </Connection>
        <Connection>
            <End ProcInstance="father"
```

```
Channel="down"/>
            <End ProcInstance="sister"
Channel="inbox"/>
        </Connection>
        <Connection>
            <End ProcInstance="sister"
Channel="up"/>
            <End ProcInstance="father"
Channel="inbox_child"/>
        </Connection>
        <Connection>
            <End ProcInstance="brother"
Channel="up"/>
            <End ProcInstance="father"
Channel="inbox_child"/>
        </Connection>
        <Connection>
            <End ProcInstance="father"
Channel="up"/>
            <End ProcInstance="grandF"
Channel="inbox"/>
        </Connection>
        <Connection>
            <End ProcInstance="uncle"
Channel="up"/>
            <End ProcInstance="grandF"
Channel="inbox"/>
        </Connection>
    </Instance>
    <Process Name="parent">
        <Event Name="send_probe"
Type="Create" Before="initial" After="probe_send"
Channel="down" Value="probe"/>
        <Event Name="receive_echo"
Type="Read" Before="probe_send" After="initial="
Channel="inbox" Value="echo"/>
    </Process>
    <Process Name="child1">
        <Event Name="receive_probe"
Type="Read" Before="initial" After="probe_receive"
Channel="inbox_parent" Value="probe"/>
        <Event Name="send_probe"
Type="Create" Before="probe_receive"
After="initial=" Channel="down" Value="probe"/>
        <Event Name="receive_echo"
Type="Read" Before="initial" After="echo_receive"
Channel="inbox_child" Value="echo"/>
        <Event Name="send_echo" Type="Write"
Before="echo_receive" After="initial="
Channel="up" Value="echo"/>
    </Process>
    <Process Name="child2">
        <Event Name="receive_probe"
Type="Read" Before="initial" After="probe_receive"
Channel="inbox" Value="probe"/>
        <Event Name="send_echo" Type="Create"
Before="probe_receive" After="initial="
Channel="up" Value="echo"/>
    </Process>
</Model>
```

B.5 An Agent Model

```
<Model>
    <Instance Name="Agent">
        <ProcInstance Name="c0"
Type="customer"/>
        <ProcInstance Name="c1"
Type="customer"/>
        <ProcInstance Name="a" Type="agent"/>
        <ProcInstance Name="s0" Type="shop"/>
        <ProcInstance Name="s1" Type="shop"/>
        <Connection>
            <End ProcInstance="c0"
Channel="outbox"/>
            <End ProcInstance="a"
Channel="inbox_c"/>
        </Connection>
        <Connection>
            <End ProcInstance="c1"
Channel="outbox"/>
            <End ProcInstance="a"
Channel="inbox_c"/>
        </Connection>
        <Connection>
            <End ProcInstance="a"
Channel="outbox_s"/>
            <End ProcInstance="s0"
Channel="inbox"/>
        </Connection>
        <Connection>
            <End ProcInstance="a"
Channel="outbox_s"/>
            <End ProcInstance="s1"
Channel="inbox"/>
        </Connection>
        <Connection>
            <End ProcInstance="s0"
Channel="outbox"/>
            <End ProcInstance="a"
Channel="inbox_s"/>
        </Connection>
        <Connection>
            <End ProcInstance="s1"
Channel="outbox"/>
            <End ProcInstance="a"
Channel="inbox_s"/>
        </Connection>
        <Connection>
            <End ProcInstance="a"
Channel="outbox_c"/>
            <End ProcInstance="c0"
Channel="inbox"/>
        </Connection>
        <Connection>
            <End ProcInstance="a"
Channel="outbox_c"/>
            <End ProcInstance="c1"
Channel="inbox"/>
        </Connection>
    </Instance>
    <Process Name="customer">
        <Event Name="send_target"
Type="Create" Before="initial" After="target_send"
Channel="outbox" Value="target"/>
        <Event Name="receive_item" Type="Read"
Before="target_send" After="finish"
Channel="inbox" Value="item"/>
    </Process>
    <Process Name="agent">
        <Event Name="receive_target"
```

```
Type="Read" Before="initial" After="target_receive"
Channel="inbox_c" Value="target"/>
            <Event Name="send_target"
Type="Create" Before="target_receive"
After="target_send" Channel="outbox_s"
Value="target"/>
            <Event Name="receive_item" Type="Read"
Before="target_send" After="item_receive"
Channel="inbox_s" Value="item"/>
            <Event Name="send_item" Type="Create"
Before="item_receive" After="finish"
Channel="outbox_c" Value="item"/>
      </Process>
      <Process Name="shop">
            <Event Name="receive_target"
Type="Read" Before="initial" After="target_receive"
Channel="inbox" Value="target"/>
            <Event Name="send_item" Type="Create"
Before="target_receive" After="finish"
Channel="outbox" Value="item"/>
      </Process>
</Model>
```

Appendix C Example Models in Promela

C.1 A Cycle Election Model (Model 1 with Synchronous Communication)

```
/* Generated from file C:\Documents and
Settings\pfx01r\My
Documents\Report\cycle8\1\cycle.xml */

#define CHLEN 0
#define CHNO 3

proctype participant0(chan election, V0,
Ps_elected, V1, Ps_election, V2, V3, elected)
{
int i = 0;
chan supp[CHNO] = [CHLEN] of {chan};

initial:
if
:: election?V0; goto REN0;
:: election?V1; goto REN1;
:: election?V2; goto REN2;
:: election?V3; goto REN3;
:: elected?V0; goto IMBoss;
:: elected?V1; goto RED1;
:: elected?V2; goto RED2;
:: elected?V3; goto RED3;
fi;

REN0:
if
:: Ps_elected!V0; goto initial;
fi;

REN1:
if
:: Ps_election!V1; goto initial;
fi;

REN2:
if
:: Ps_election!V2; goto initial;
fi;

REN3:
if
::i < CHNO; atomic { V3 = supp[i]; Ps_election!V3; i
= i +1 }
goto initial;
fi;

RED1:
if
:: Ps_elected!V1; goto Boss1;
fi;

RED2:
if
:: Ps_elected!V2; goto Boss2;
fi;


RED3:
if
::i < CHNO; atomic { V3 = supp[i]; Ps_elected!V3; i =
i +1 }
goto Boss3;
fi;

IMBoss: skip;

Boss1: skip;


Boss2: skip;


Boss3: skip
}
```

```
proctype participant1(chan election, V0,
Ps_election, V1, Ps_elected, V2, V3, elected)
{
int i = 0;
chan supp[CHNO] = [CHLEN] of {chan};

initial:
if
:: election?V0; goto REN0;
:: election?V1; goto REN1;
:: election?V2; goto REN2;
:: election?V3; goto REN3;
:: elected?V1; goto IMBoss;
:: elected?V2; goto RED2;
:: elected?V3; goto RED3;
fi;

REN0:
if
::i < CHNO; atomic { V1 = supp[i]; Ps_election!V1; i
= i +1 }
goto initial;
fi;

REN1:
if
:: Ps_elected!V1; goto initial;
fi;

REN2:
if
:: Ps_election!V2; goto initial;
fi;

REN3:
if
:: Ps_election!V3; goto initial;
fi;

RED2:
if
:: Ps_elected!V2; goto Boss2;
fi;

RED3:
if
::i < CHNO; atomic { V3 = supp[i]; Ps_elected!V3; i =
i +1 }
goto Boss3;
fi;

IMBoss: skip;
Boss2: skip;
Boss3: skip
}

proctype participant2(chan election, V0,
Ps_election, V2, V1, Ps_elected, V3, elected)
{
int i = 0;
chan supp[CHNO] = [CHLEN] of {chan};

initial:
if
:: election?V0; goto REN0;
:: election?V1; goto REN1;
:: election?V2; goto REN2;
:: election?V3; goto REN3;
:: elected?V2; goto IMBoss;
:: elected?V3; goto RED3;
fi;

REN0:
if
:: Ps_election!V2; goto initial;
fi;

REN1:
if
::i < CHNO; atomic { V2 = supp[i]; Ps_election!V2; i
= i +1 }
goto initial;
fi;

REN2:
if
:: Ps_elected!V2; goto initial;
fi;

REN3:
if
:: Ps_election!V3; goto initial;
fi;

RED3:
if
::i < CHNO; atomic { V3 = supp[i]; Ps_elected!V3; i =
i +1 }
goto Boss3;
fi;

IMBoss: skip;
Boss3: skip
}

proctype participant3(chan election, V0,
Ps_election, V3, V1, V2, Ps_elected, elected)
{
int i = 0;
chan supp[CHNO] = [CHLEN] of {chan};

initial:
if
:: election?V0; goto REN0;
:: election?V1; goto REN1;
:: election?V2; goto REN2;
:: election?V3; goto REN3;
:: elected?V3; goto IMBoss;
fi;

REN0:
if
:: Ps_election!V3; goto initial;
fi;

REN1:
if
:: Ps_election!V3; goto initial;
fi;

REN2:
if
::i < CHNO; atomic { V3 = supp[i]; Ps_election!V3; i
= i +1 }
goto initial;
fi;

REN3:
if
::i < CHNO; atomic { V3 = supp[i]; Ps_elected!V3; i =
i +1 }
goto initial;
fi;

IMBoss: skip
}
```

```
proctype trigger(chan startElection, V0)
{
int i = 0;
chan supp[CHNO] = [CHLEN] of {chan};

initial:
if
::i < CHNO; atomic { V0 = supp[i]; startElection!V0; i
= i +1 }
goto finish;
fi;

finish: skip
}

init
{ atomic {
chan ch0 = [CHLEN] of {chan};
chan ch1 = [CHLEN] of {chan};
chan ch2 = [CHLEN] of {chan};
chan ch3 = [CHLEN] of {chan};
chan ch4 = [CHLEN] of {chan};
chan ch5 = [CHLEN] of {chan};
chan ch6 = [CHLEN] of {chan};
chan ch7 = [CHLEN] of {chan};
chan nch0 = [0] of {chan};
chan nch1 = [0] of {chan};
chan nch2 = [0] of {chan};
chan nch3 = [0] of {chan};
chan nch4 = [0] of {chan};
chan nch5 = [0] of {chan};
chan nch6 = [0] of {chan};
chan nch7 = [0] of {chan};
chan nch8 = [0] of {chan};
chan nch9 = [0] of {chan};
chan nch10 = [0] of {chan};
chan nch11 = [0] of {chan};
chan nch12 = [0] of {chan};
chan nch13 = [0] of {chan};
chan nch14 = [0] of {chan};
chan nch15 = [0] of {chan};
chan nch16 = [0] of {chan};

run participant0(ch6, nch0, ch1, nch1, ch0, nch2,
nch3, ch7);
run participant1(ch0, nch4, ch2, nch5, ch3, nch6,
nch7, ch1);
run participant2(ch2, nch8, ch4, nch9, nch10, ch5,
nch11, ch3);
run participant3(ch4, nch12, ch6, nch13, nch14,
nch15, ch7, ch5);
run trigger(ch0, nch16);
} };
```

C.2 A Cycle Election Model (Model 2 with Asynchronous Communication)

```
/* Generated from file C:\Documents and
Settings\pfx01r\My
Documents\Report\cyle9\cycle9.xml */

#define CHLEN 10
#define CHNO 3

proctype participant0(chan Ps_election, V0,
election, Ps_elected, V1, V2, V3, elected)
{
int i = 0;
chan supp[CHNO] = [CHLEN] of {chan};

initial:
if
::i < CHNO; atomic { V0 = supp[i]; Ps_election!V0; i
= i +1 }
goto SE;
fi;

SE:
if

:: election?V3; goto REN3;

:: elected?V3; goto RED3;
fi;

REN0:
if
:: Ps_elected!V0; goto SE;
fi;

REN1:
if
:: Ps_election!V1; goto SE;
fi;

REN2:
if
:: Ps_election!V2; goto SE;
fi;

REN3:
if
:: Ps_election!V3; goto SE;
fi;

RED1:
if
:: Ps_elected!V1; goto Boss1;
fi;

RED2:
if
:: Ps_elected!V2; goto Boss2;
fi;

RED3:
if
::i < CHNO; atomic { V3 = supp[i]; Ps_elected!V3; i =
i +1 }
goto Boss3;
fi;
```

```
IMBoss: skip;
Boss1: skip;
Boss2: skip;
Boss3: skip
}

proctype participant1(chan Ps_election, V1,
election, V0, Ps_elected, V2, V3, elected)
{
int i = 0;
chan supp[CHNO] = [CHLEN] of {chan};

initial:
if
::i < CHNO; atomic { V1 = supp[i]; Ps_election!V1; i
= i +1 }
goto SE;
fi;

SE:
if
:: election?V0; goto REN0;
:: election?V3; goto REN3;

:: elected?V3; goto RED3;
fi;

REN0:
if
:: Ps_election!V1; goto SE;
fi;

REN1:
if
:: Ps_elected!V1; goto SE;
fi;

REN2:
if
:: Ps_election!V2; goto SE;
fi;

REN3:
if
:: Ps_election!V3; goto SE;
fi;

RED2:
if
:: Ps_elected!V2; goto Boss2;
fi;

RED3:
if
::i < CHNO; atomic { V3 = supp[i]; Ps_elected!V3; i =
i +1 }
goto Boss3;
fi;

IMBoss: skip;
Boss2: skip;
Boss3: skip
}

proctype participant2(chan Ps_election, V2,
election, V0, V1, Ps_elected, V3, elected)
{
int i = 0;
chan supp[CHNO] = [CHLEN] of {chan};

initial:
if
::i < CHNO; atomic { V2 = supp[i]; Ps_election!V2; i
= i +1 }
goto SE;
fi;

SE:
if

:: election?V1; goto REN1;

:: election?V3; goto REN3;

:: elected?V3; goto RED3;
fi;

REN0:
if
:: Ps_election!V2; goto SE;
fi;

REN1:
if
:: Ps_election!V2; goto SE;
fi;

REN2:
if
:: Ps_elected!V2; goto SE;
fi;

REN3:
if
:: Ps_election!V3; goto SE;
fi;

RED3:
if
::i < CHNO; atomic { V3 = supp[i]; Ps_elected!V3; i =
i +1 }
goto Boss3;
fi;

IMBoss: skip;
Boss3: skip
}

proctype participant3(chan Ps_election, V3,
election, V0, V1, V2, Ps_elected, elected)
{
int i = 0;
chan supp[CHNO] = [CHLEN] of {chan};

initial:
if
::i < CHNO; atomic { V3 = supp[i]; Ps_election!V3; i
= i +1 }
goto SE;
fi;

SE:
if

:: election?V2; goto REN2;
:: election?V3; goto REN3;
:: elected?V3; goto IMBoss;
fi;

REN0:
if
:: Ps_election!V3; goto SEl;
fi;

REN1:
```

```
if
:: Ps_election!V3; goto SE;
fi;

REN2:
if
:: Ps_election!V3; goto SE;
fi;

REN3:
if
::i < CHNO; atomic { V3 = supp[i]; Ps_elected!V3; i =
i +1 }
goto SE;
fi;

SEl: skip;
IMBoss: skip
}

init
{ atomic {
chan ch0 = [CHLEN] of {chan};
chan ch1 = [CHLEN] of {chan};
chan ch2 = [CHLEN] of {chan};
chan ch3 = [CHLEN] of {chan};
chan ch4 = [CHLEN] of {chan};
chan ch5 = [CHLEN] of {chan};
chan ch6 = [CHLEN] of {chan};
chan ch7 = [CHLEN] of {chan};
chan nch0 = [0] of {chan};
chan nch1 = [0] of {chan};
chan nch2 = [0] of {chan};
chan nch3 = [0] of {chan};
chan nch4 = [0] of {chan};
chan nch5 = [0] of {chan};
chan nch6 = [0] of {chan};
chan nch7 = [0] of {chan};
chan nch8 = [0] of {chan};
chan nch9 = [0] of {chan};
chan nch10 = [0] of {chan};
chan nch11 = [0] of {chan};
chan nch12 = [0] of {chan};
chan nch13 = [0] of {chan};
chan nch14 = [0] of {chan};
chan nch15 = [0] of {chan};

run participant0(ch0, nch0, ch2, ch1, nch1, nch2,
nch3, ch3);
run participant1(ch2, nch4, ch6, nch5, ch3, nch6,
nch7, ch7);
run participant2(ch4, nch8, ch0, nch9, nch10, ch5,
nch11, ch1);
run participant3(ch6, nch12, ch4, nch13, nch14,
nch15, ch7, ch5);
} };
```

Appendix D Implementation in Java

D.1 A Cycle Election Model (Model 1 with Synchronous Communication)

```
/* Generated from file C:\Documents and
Settings\pfx01r\My Documents\Report\cycle.xml */

import java.io.*;
import java.awt.*;
import java.math.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.text.*;
import javax.swing.table.*;

public class cycle{

static class Message {
String type; Process writer; Process reader; String
channel;

Message (String t, Process p, Process r, String c){
type=t; writer=p; reader=r; channel=c;
}

public String toString(){
return type + " from " + writer.toString() + " to " +
reader.toString() + " via " +channel;
}
}

static class myGUI extends JFrame{
String[] headerStr = {"No.","Event", "Type", "Before
state", "After state", "Channel", "Value"};
DefaultTableModel dm = new
DefaultTableModel(headerStr, 60);
JTable table = new JTable(dm);
JPanel center=new JPanel();

JLabel instanceLabel;
JTextField instanceField;
JLabel eventsLabel;
JLabel processLabel;
JTextField processField;

myGUI(String a, String b){
setTitle("cycle");
setLocation(200,200);
setSize(30,30);

JPanel top =new JPanel();
top.setBackground(Color.gray);
instanceLabel= new JLabel("Instance");
top.add(instanceLabel);

instanceField=new JTextField(a,15);
Font g =new Font("Roman",Font.PLAIN,12);
top.setFont(g);
top.add(instanceField);

processLabel= new JLabel("Process");
top.add(processLabel);

processField=new JTextField(b,15);
Font h =new Font("Roman",Font.ITALIC,12);
top.setFont(h);
top.add(processField);

getContentPane().add(top, BorderLayout.NORTH);

JPanel middle =new JPanel();
middle.setBackground(Color.green);
eventsLabel= new JLabel("Possible event(s):");
middle.add(eventsLabel);
getContentPane().add(middle,
BorderLayout.WEST);

center.setBackground(Color.gray);
getContentPane().add(center,
BorderLayout.CENTER);

JPanel record =new JPanel();
table.setPreferredScrollableViewportSize(new
Dimension(200, 150));
getContentPane().add(new JScrollPane(table),
BorderLayout.SOUTH);

pack();
setVisible(true);

}

}

static class MessageQueue{
String name;
boolean sendFlag, receiveFlag;
Message share;

public MessageQueue(String n, int m){
name=n;
sendFlag=false;
receiveFlag=false;
}

synchronized void send(Message x) throws
InterruptedException{
sendFlag=true;
share=x;
notifyAll();
System.out.println("send("+x+")");
while(!receiveFlag) wait();
receiveFlag=false;
}

synchronized Message receive() throws
InterruptedException{
receiveFlag=true;
notifyAll();
while(!sendFlag) wait();
Message x; x=share;
System.out.println("receive("+x+")");
sendFlag=false;
return x;
}
}

static class Process extends Thread {
MessageQueue startElection;
MessageQueue PassOn_election;
MessageQueue election;
MessageQueue PassOn_elected;
MessageQueue elected;
String name;
public String toString(){
return this.name;
}
}

static class participant0 extends Process {
static myGUI traceTable;
public participant0 (String name){
this.name =name;
traceTable=new myGUI(name,"participant0");
election=new MessageQueue("election",0);
elected=new MessageQueue("elected",0);
this.start();
}

JButton ButtonR_Election_0 = new
JButton("R_Election_0" );
JButton ButtonS_Elected_0 = new
JButton("S_Elected_0" );
JButton ButtonR_Election_1 = new
JButton("R_Election_1" );
JButton ButtonS_Election_1 = new
JButton("S_Election_1" );
JButton ButtonR_Election_2 = new
JButton("R_Election_2" );
JButton ButtonS_Election_2 = new
JButton("S_Election_2" );
JButton ButtonR_Election_3 = new
JButton("R_Election_3" );
JButton ButtonS_Election_3 = new
JButton("S_Election_3" );
JButton ButtonR_Elected_0 = new
JButton("R_Elected_0" );
JButton ButtonR_Elected_1 = new
JButton("R_Elected_1" );
JButton ButtonS_ED_1 = new JButton("S_ED_1" );
JButton ButtonR_Elected_2 = new
JButton("R_Elected_2" );
JButton ButtonS_ED_2 = new JButton("S_ED_2" );
JButton ButtonR_Elected_3 = new
JButton("R_Elected_3" );
JButton ButtonS_ED_3 = new JButton("S_ED_3" );

public String state=" ";
int noOfevents=0;

public void run(){
transformState("initial");
new Thread(){public void run(){
try{for(;;){
Message m=(Message)election.receive();
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V0" && state =="initial")
R_Election_0(m.writer,m.type,"R_Election_0");
else if(m.type=="V1" && state =="initial")
R_Election_1(m.writer,m.type,"R_Election_1");
else if(m.type=="V2" && state =="initial")
R_Election_2(m.writer,m.type,"R_Election_2");
else if(m.type=="V3" && state =="initial")
R_Election_3(m.writer,m.type,"R_Election_3");
}}catch(Exception e){System.out.println(name + ":
demultiplex error");}}}.start();

new Thread(){public void run(){
try{for(;;){
Message m=(Message)elected.receive();
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V0" && state =="initial")
R_Elected_0(m.writer,m.type,"R_Elected_0");
else if(m.type=="V1" && state =="initial")
R_Elected_1(m.writer,m.type,"R_Elected_1");
else if(m.type=="V2" && state =="initial")
R_Elected_2(m.writer,m.type,"R_Elected_2");
else if(m.type=="V3" && state =="initial")
R_Elected_3(m.writer,m.type,"R_Elected_3");
}}catch(Exception e){System.out.println(name + ":
demultiplex error");}}}.start();

}

public void REN0(){
if(state=="REN0")
S_Elected_0();
}

public void REN1(){
if(state=="REN1")
S_Election_1();
}
```

```
public void REN2(){
if(state=="REN2")
S_Election_2();
}

public void REN3(){
if(state=="REN3")
S_Election_3();
}

public void RED1(){
if(state=="RED1")
S_ED_1();
}

public void RED2(){
if(state=="RED2")
S_ED_2();
}

public void RED3(){
if(state=="RED3")
S_ED_3();
}

public void R_Election_0(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_0);
ButtonR_Election_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN0");
displayTrace( "R_Election_0", "Read", "initial",
"REN0", "election", "V0");
REN0();
ButtonR_Election_0.setVisible(false);
}
});
}

public void R_Election_1(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_1);
ButtonR_Election_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN1");
displayTrace( "R_Election_1", "Read", "initial",
"REN1", "election", "V1");
REN1();
ButtonR_Election_1.setVisible(false);
}
});
}

public void R_Election_2(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_2);
ButtonR_Election_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN2");
displayTrace( "R_Election_2", "Read", "initial",
"REN2", "election", "V2");
REN2();
ButtonR_Election_2.setVisible(false);
}
});
}

public void R_Election_3(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_3);
ButtonR_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN3");
displayTrace( "R_Election_3", "Read", "initial",
"REN3", "election", "V3");
REN3();
ButtonR_Election_3.setVisible(false);
}
});
}

public void R_Elected_0(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_0);
ButtonR_Elected_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("IMBoss");
displayTrace( "R_Elected_0", "Read", "initial",
"IMBoss", "elected", "V0");
ButtonR_Elected_0.setVisible(false);
}
});
}

public void R_Elected_1(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_1);
ButtonR_Elected_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("RED1");
displayTrace( "R_Elected_1", "Read", "initial",
"RED1", "elected", "V1");
RED1();
ButtonR_Elected_1.setVisible(false);
}
});
}

public void R_Elected_2(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_2);
ButtonR_Elected_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("RED2");
displayTrace( "R_Elected_2", "Read", "initial",
"RED2", "elected", "V2");
RED2();
ButtonR_Elected_2.setVisible(false);
}
});
}

public void R_Elected_3(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_3);
ButtonR_Elected_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("RED3");
displayTrace( "R_Elected_3", "Read", "initial",
"RED3", "elected", "V3");
RED3();
ButtonR_Elected_3.setVisible(false);
}
});
}

public void S_Election_3(){
traceTable.center.add(ButtonS_Election_3);
```

```
final Message m=new
Message("V3",this,p1,"PassOn_election");
ButtonS_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p1.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " :
S_Election_3- send error");}

displayTrace( "S_Election_3", "Create", "REN3",
"initial=", "PassOn_election", "V3");
ButtonS_Election_3.setVisible(false);
}

});

}

public void S_ED_3(){
traceTable.center.add(ButtonS_ED_3);
final Message m=new
Message("V3",this,p1,"PassOn_elected");
ButtonS_ED_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p1.elected.send(m);
transformState("3_Boss");
}
catch(Exception f){System.out.println(name + " :
S_ED_3- send error");}

displayTrace( "S_ED_3", "Create", "RED3",
"3_Boss", "PassOn_elected", "V3");
ButtonS_ED_3.setVisible(false);
}

});

}

public void S_Elected_0(){
System.out.println(name+" : S_Elected_0");

traceTable.center.add(ButtonS_Elected_0);
final Message m=new
Message("V0",this,p1,"elected");
ButtonS_Elected_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p1.elected.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p0-
send error");}
displayTrace( "S_Elected_0", "Write", "REN0",
"initial=", "PassOn_elected", "V0");
ButtonS_Elected_0.setVisible(false);
}

});


}

public void S_Election_1(){
System.out.println(name+" : S_Election_1");

traceTable.center.add(ButtonS_Election_1);
final Message m=new
Message("V1",this,p1,"election");
ButtonS_Election_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p1.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p0-
send error");}
displayTrace( "S_Election_1", "Write", "REN1",
"initial=", "PassOn_election", "V1");
ButtonS_Election_1.setVisible(false);
}

});


}

public void S_Election_2(){
System.out.println(name+" : S_Election_2");

traceTable.center.add(ButtonS_Election_2);
final Message m=new
Message("V2",this,p1,"election");
ButtonS_Election_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p1.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p0-
send error");}
displayTrace( "S_Election_2", "Write", "REN2",
"initial=", "PassOn_election", "V2");
ButtonS_Election_2.setVisible(false);
}

});


}

public void S_ED_1(){
System.out.println(name+" : S_ED_1");

traceTable.center.add(ButtonS_ED_1);
final Message m=new
Message("V1",this,p1,"elected");
ButtonS_ED_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p1.elected.send(m);
transformState("1_Boss");
}
catch(Exception f){System.out.println(name + " : p0-
send error");}
displayTrace( "S_ED_1", "Write", "RED1",
"1_Boss", "PassOn_elected", "V1");
ButtonS_ED_1.setVisible(false);
}

});


}

public void S_ED_2(){
System.out.println(name+" : S_ED_2");

traceTable.center.add(ButtonS_ED_2);
final Message m=new
Message("V2",this,p1,"elected");
```

```
ButtonS_ED_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p1.elected.send(m);
transformState("2_Boss");
}
catch(Exception f){System.out.println(name + " : p0-
send error");}
displayTrace( "S_ED_2", "Write", "RED2",
"2_Boss", "PassOn_elected", "V2");
ButtonS_ED_2.setVisible(false);
}

});


}

Process p0;
public void connection_p0(Process temp){
p0=temp;
}

Process p1;
public void connection_p1(Process temp){
p1=temp;
}

Process p2;
public void connection_p2(Process temp){
p2=temp;
}

Process p3;
public void connection_p3(Process temp){
p3=temp;
}

Process Tr;
public void connection_Tr(Process temp){
Tr=temp;
}

public String getname(){
return name;
}

public void displayTrace(String ev, String ty, String
be, String af, String ch, String va){
traceTable.table.setValueAt((new
Integer(noOfevents)).toString(), noOfevents,0);
traceTable.table.setValueAt(ev, noOfevents,1);
traceTable.table.setValueAt(ty, noOfevents,2);
traceTable.table.setValueAt(be, noOfevents,3);
traceTable.table.setValueAt(af, noOfevents,4);
traceTable.table.setValueAt(ch, noOfevents,5);
traceTable.table.setValueAt(va, noOfevents,6);
noOfevents++;
}

public void transformState(String s){
state=s;
System.out.println(name +" : " + state);
}
}

static class participant1 extends Process {
static myGUI traceTable;
public participant1 (String name){
this.name =name;
traceTable=new myGUI(name,"participant1");
election=new MessageQueue("election",0);
elected=new MessageQueue("elected",0);
this.start();
}

JButton ButtonR_Election_0 = new
JButton("R_Election_0" );
JButton ButtonS_Election1_0 = new
JButton("S_Election1_0" );
JButton ButtonR_Election_1 = new
JButton("R_Election_1" );
JButton ButtonS_Elected_1 = new
JButton("S_Elected_1" );
JButton ButtonR_Election_2 = new
JButton("R_Election_2" );
JButton ButtonS_Election_2 = new
JButton("S_Election_2" );
JButton ButtonR_Election_3 = new
JButton("R_Election_3" );
JButton ButtonS_Election_3 = new
JButton("S_Election_3" );
JButton ButtonR_Elected_1 = new
JButton("R_Elected_1" );
JButton ButtonR_Elected_2 = new
JButton("R_Elected_2" );
JButton ButtonS_ED_2 = new JButton("S_ED_2" );
JButton ButtonR_Elected_3 = new
JButton("R_Elected_3" );
JButton ButtonS_ED_3 = new JButton("S_ED_3" );

public String state=" ";
int noOfevents=0;

public void run(){
transformState("initial");
new Thread(){public void run(){
try{for(;;){
Message m=(Message)election.receive();
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V0" && state =="initial")
R_Election_0(m.writer,m.type,"R_Election_0");
else if(m.type=="V1" && state =="initial")
R_Election_1(m.writer,m.type,"R_Election_1");
else if(m.type=="V2" && state =="initial")
R_Election_2(m.writer,m.type,"R_Election_2");
else if(m.type=="V3" && state =="initial")
R_Election_3(m.writer,m.type,"R_Election_3");
}}catch(Exception e){System.out.println(name + ":
demultiplex error");}}}.start();

new Thread(){public void run(){
try{for(;;){
Message m=(Message)elected.receive();
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V1" && state =="initial")
R_Elected_1(m.writer,m.type,"R_Elected_1");
else if(m.type=="V2" && state =="initial")
R_Elected_2(m.writer,m.type,"R_Elected_2");
else if(m.type=="V3" && state =="initial")
R_Elected_3(m.writer,m.type,"R_Elected_3");
}}catch(Exception e){System.out.println(name + ":
demultiplex error");}}}.start();

}

public void REN0(){
if(state=="REN0")
S_Election1_0();
}

public void REN1(){
if(state=="REN1")
S_Elected_1();
}
```

```
public void REN2(){
if(state=="REN2")
S_Election_2();
}

public void REN3(){
if(state=="REN3")
S_Election_3();
}

public void RED2(){
if(state=="RED2")
S_ED_2();
}

public void RED3(){
if(state=="RED3")
S_ED_3();
}

public void R_Election_0(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_0);
ButtonR_Election_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN0");
displayTrace( "R_Election_0", "Read", "initial",
"REN0", "election", "V0");
REN0();
ButtonR_Election_0.setVisible(false);
}
});
}

public void R_Election_1(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_1);
ButtonR_Election_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN1");
displayTrace( "R_Election_1", "Read", "initial",
"REN1", "election", "V1");
REN1();
ButtonR_Election_1.setVisible(false);
}
});
}

public void R_Election_2(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_2);
ButtonR_Election_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN2");
displayTrace( "R_Election_2", "Read", "initial",
"REN2", "election", "V2");
REN2();
ButtonR_Election_2.setVisible(false);
}
});
}

public void R_Election_3(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_3);
ButtonR_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN3");
displayTrace( "R_Election_3", "Read", "initial",
"REN3", "election", "V3");
REN3();
ButtonR_Election_3.setVisible(false);
}
});
}

public void R_Elected_1(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_1);
ButtonR_Elected_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("IMBoss");
displayTrace( "R_Elected_1", "Read", "initial",
"IMBoss", "elected", "V1");
ButtonR_Elected_1.setVisible(false);
}
});
}

public void R_Elected_2(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_2);
ButtonR_Elected_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("RED2");
displayTrace( "R_Elected_2", "Read", "initial",
"RED2", "elected", "V2");
RED2();
ButtonR_Elected_2.setVisible(false);
}
});
}

public void R_Elected_3(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_3);
ButtonR_Elected_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("RED3");
displayTrace( "R_Elected_3", "Read", "initial",
"RED3", "elected", "V3");
RED3();
ButtonR_Elected_3.setVisible(false);
}
});
}

public void S_Election1_0(){
traceTable.center.add(ButtonS_Election1_0);
final Message m=new
Message("V1",this,p2,"PassOn_election");
ButtonS_Election1_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p2.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " :
S_Election1_0- send error");}

displayTrace( "S_Election1_0", "Create", "REN0",
"initial=", "PassOn_election", "V1");
ButtonS_Election1_0.setVisible(false);
}

});

}
```

```
public void S_ED_3(){
traceTable.center.add(ButtonS_ED_3);
final Message m=new
Message("V3",this,p2,"PassOn_elected");
ButtonS_ED_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p2.elected.send(m);
transformState("3_Boss");
}
catch(Exception f){System.out.println(name + " :
S_ED_3- send error");}

displayTrace( "S_ED_3", "Create", "RED3",
"3_Boss", "PassOn_elected", "V3");
ButtonS_ED_3.setVisible(false);
}

});

}

public void S_Elected_1(){
System.out.println(name+" : S_Elected_1");

traceTable.center.add(ButtonS_Elected_1);
final Message m=new
Message("V1",this,p2,"elected");
ButtonS_Elected_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p2.elected.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p1-
send error");}
displayTrace( "S_Elected_1", "Write", "REN1",
"initial=", "PassOn_elected", "V1");
ButtonS_Elected_1.setVisible(false);
}

});


}

public void S_Election_2(){
System.out.println(name+" : S_Election_2");

traceTable.center.add(ButtonS_Election_2);
final Message m=new
Message("V2",this,p2,"election");
ButtonS_Election_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p2.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p1-
send error");}
displayTrace( "S_Election_2", "Write", "REN2",
"initial=", "PassOn_election", "V2");
ButtonS_Election_2.setVisible(false);
}

});


}

public void S_Election_3(){
System.out.println(name+" : S_Election_3");
traceTable.center.add(ButtonS_Election_3);
final Message m=new
Message("V3",this,p2,"election");
ButtonS_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p2.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p1-
send error");}
displayTrace( "S_Election_3", "Write", "REN3",
"initial=", "PassOn_election", "V3");
ButtonS_Election_3.setVisible(false);
}

});


}

public void S_ED_2(){
System.out.println(name+" : S_ED_2");

traceTable.center.add(ButtonS_ED_2);
final Message m=new
Message("V2",this,p2,"elected");
ButtonS_ED_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p2.elected.send(m);
transformState("2_Boss");
}
catch(Exception f){System.out.println(name + " : p1-
send error");}
displayTrace( "S_ED_2", "Write", "RED2",
"2_Boss", "PassOn_elected", "V2");
ButtonS_ED_2.setVisible(false);
}

});


}

Process p0;
public void connection_p0(Process temp){
p0=temp;
}

Process p1;
public void connection_p1(Process temp){
p1=temp;
}

Process p2;
public void connection_p2(Process temp){
p2=temp;
}

Process p3;
public void connection_p3(Process temp){
p3=temp;
}

Process Tr;
public void connection_Tr(Process temp){
Tr=temp;
}

public String getname(){
return name;
```

```
}

public void displayTrace(String ev, String ty, String
be, String af, String ch, String va){
traceTable.table.setValueAt((new
Integer(noOfevents)).toString(), noOfevents,0);
traceTable.table.setValueAt(ev, noOfevents,1);
traceTable.table.setValueAt(ty, noOfevents,2);
traceTable.table.setValueAt(be, noOfevents,3);
traceTable.table.setValueAt(af, noOfevents,4);
traceTable.table.setValueAt(ch, noOfevents,5);
traceTable.table.setValueAt(va, noOfevents,6);
noOfevents++;
}

public void transformState(String s){
state=s;
System.out.println(name +" : " + state);
}
}

static class participant2 extends Process {
static myGUI traceTable;
public participant2 (String name){
this.name =name;
traceTable=new myGUI(name,"participant2");
election=new MessageQueue("election",0);
elected=new MessageQueue("elected",0);
this.start();
}

JButton ButtonR_Election_0 = new
JButton("R_Election_0" );
JButton ButtonS_Election2_0 = new
JButton("S_Election2_0" );
JButton ButtonR_Election_1 = new
JButton("R_Election_1" );
JButton ButtonS_Election2_1 = new
JButton("S_Election2_1" );
JButton ButtonR_Election_2 = new
JButton("R_Election_2" );
JButton ButtonS_Elected_2 = new
JButton("S_Elected_2" );
JButton ButtonR_Election_3 = new
JButton("R_Election_3" );
JButton ButtonS_Election_3 = new
JButton("S_Election_3" );
JButton ButtonR_Elected_2 = new
JButton("R_Elected_2" );
JButton ButtonR_Elected_3 = new
JButton("R_Elected_3" );
JButton ButtonS_ED_3 = new JButton("S_ED_3" );

public String state=" ";
int noOfevents=0;

public void run(){
transformState("initial");
new Thread(){public void run(){
try{for(;;){
Message m=(Message)election.receive();
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V0" && state =="initial")
R_Election_0(m.writer,m.type,"R_Election_0");
else if(m.type=="V1" && state =="initial")
R_Election_1(m.writer,m.type,"R_Election_1");
else if(m.type=="V2" && state =="initial")
R_Election_2(m.writer,m.type,"R_Election_2");
else if(m.type=="V3" && state =="initial")
R_Election_3(m.writer,m.type,"R_Election_3");
}}catch(Exception e){System.out.println(name + ":
demultiplex error");}}}.start();
new Thread(){public void run(){
try{for(;;){
Message m=(Message)elected.receive();
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V2" && state =="initial")
R_Elected_2(m.writer,m.type,"R_Elected_2");
else if(m.type=="V3" && state =="initial")
R_Elected_3(m.writer,m.type,"R_Elected_3");
}}catch(Exception e){System.out.println(name + ":
demultiplex error");}}}.start();

}

public void REN0(){
if(state=="REN0")
S_Election2_0();
}

public void REN1(){
if(state=="REN1")
S_Election2_1();
}

public void REN2(){
if(state=="REN2")
S_Elected_2();
}

public void REN3(){
if(state=="REN3")
S_Election_3();
}

public void RED3(){
if(state=="RED3")
S_ED_3();
}

public void R_Election_0(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_0);
ButtonR_Election_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN0");
displayTrace( "R_Election_0", "Read", "initial",
"REN0", "election", "V0");
REN0();
ButtonR_Election_0.setVisible(false);
}
});
}

public void R_Election_1(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_1);
ButtonR_Election_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN1");
displayTrace( "R_Election_1", "Read", "initial",
"REN1", "election", "V1");
REN1();
ButtonR_Election_1.setVisible(false);
}
});
}

public void R_Election_2(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_2);
ButtonR_Election_2.addActionListener( new
ActionListener(){
```

```
public void actionPerformed(ActionEvent e){
transformState("REN2");
displayTrace( "R_Election_2", "Read", "initial",
"REN2", "election", "V2");
REN2();
ButtonR_Election_2.setVisible(false);
}
});
}

public void R_Election_3(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_3);
ButtonR_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN3");
displayTrace( "R_Election_3", "Read", "initial",
"REN3", "election", "V3");
REN3();
ButtonR_Election_3.setVisible(false);
}
});
}

public void R_Elected_2(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_2);
ButtonR_Elected_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("IMBoss");
displayTrace( "R_Elected_2", "Read", "initial",
"IMBoss", "elected", "V2");
ButtonR_Elected_2.setVisible(false);
}
});
}

public void R_Elected_3(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_3);
ButtonR_Elected_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("RED3");
displayTrace( "R_Elected_3", "Read", "initial",
"RED3", "elected", "V3");
RED3();
ButtonR_Elected_3.setVisible(false);
}
});
}

public void S_Election2_1(){
traceTable.center.add(ButtonS_Election2_1);
final Message m=new
Message("V2",this,p3,"PassOn_election");
ButtonS_Election2_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p3.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " :
S_Election2_1- send error");}

displayTrace( "S_Election2_1", "Create", "REN1",
"initial=", "PassOn_election", "V2");
ButtonS_Election2_1.setVisible(false);
}

});
}

public void S_ED_3(){
traceTable.center.add(ButtonS_ED_3);
final Message m=new
Message("V3",this,p3,"PassOn_elected");
ButtonS_ED_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p3.elected.send(m);
transformState("3_Boss");
}
catch(Exception f){System.out.println(name + " :
S_ED_3- send error");}

displayTrace( "S_ED_3", "Create", "RED3",
"3_Boss", "PassOn_elected", "V3");
ButtonS_ED_3.setVisible(false);
}

});

}

public void S_Election2_0(){
System.out.println(name+" : S_Election2_0");

traceTable.center.add(ButtonS_Election2_0);
final Message m=new
Message("V2",this,p3,"election");
ButtonS_Election2_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p3.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p2-
send error");}
displayTrace( "S_Election2_0", "Write", "REN0",
"initial=", "PassOn_election", "V2");
ButtonS_Election2_0.setVisible(false);
}

});


}

public void S_Elected_2(){
System.out.println(name+" : S_Elected_2");

traceTable.center.add(ButtonS_Elected_2);
final Message m=new
Message("V2",this,p3,"elected");
ButtonS_Elected_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p3.elected.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p2-
send error");}
displayTrace( "S_Elected_2", "Write", "REN2",
"initial=", "PassOn_elected", "V2");
ButtonS_Elected_2.setVisible(false);
}

});


}
```

```
public void S_Election_3(){
System.out.println(name+" : S_Election_3");

traceTable.center.add(ButtonS_Election_3);
final Message m=new
Message("V3",this,p3,"election");
ButtonS_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p3.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p2-
send error");}
displayTrace( "S_Election_3", "Write", "REN3",
"initial=", "PassOn_election", "V3");
ButtonS_Election_3.setVisible(false);
}

});


}

Process p0;
public void connection_p0(Process temp){
p0=temp;
}

Process p1;
public void connection_p1(Process temp){
p1=temp;
}

Process p2;
public void connection_p2(Process temp){
p2=temp;
}

Process p3;
public void connection_p3(Process temp){
p3=temp;
}

Process Tr;
public void connection_Tr(Process temp){
Tr=temp;
}

public String getname(){
return name;
}

public void displayTrace(String ev, String ty, String
be, String af, String ch, String va){
traceTable.table.setValueAt((new
Integer(noOfevents)).toString(), noOfevents,0);
traceTable.table.setValueAt(ev, noOfevents,1);
traceTable.table.setValueAt(ty, noOfevents,2);
traceTable.table.setValueAt(be, noOfevents,3);
traceTable.table.setValueAt(af, noOfevents,4);
traceTable.table.setValueAt(ch, noOfevents,5);
traceTable.table.setValueAt(va, noOfevents,6);
noOfevents++;
}

public void transformState(String s){
state=s;
System.out.println(name +" : " + state);
}
}

static class participant3 extends Process {
static myGUI traceTable;
public participant3 (String name){
this.name =name;
traceTable=new myGUI(name,"participant3");
election=new MessageQueue("election",0);
elected=new MessageQueue("elected",0);
this.start();
}

JButton ButtonR_Election_0 = new
JButton("R_Election_0" );
JButton ButtonS_Election3_0 = new
JButton("S_Election3_0" );
JButton ButtonR_Election_1 = new
JButton("R_Election_1" );
JButton ButtonS_Election3_1 = new
JButton("S_Election3_1" );
JButton ButtonR_Election_2 = new
JButton("R_Election_2" );
JButton ButtonS_Election3_2 = new
JButton("S_Election3_2" );
JButton ButtonR_Election_3 = new
JButton("R_Election_3" );
JButton ButtonS_Elected_3 = new
JButton("S_Elected_3" );
JButton ButtonR_Elected_3 = new
JButton("R_Elected_3" );

public String state=" ";
int noOfevents=0;

public void run(){
transformState("initial");
new Thread(){public void run(){
try{for(;;){
Message m=(Message)election.receive();
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V0" && state =="initial")
R_Election_0(m.writer,m.type,"R_Election_0");
else if(m.type=="V1" && state =="initial")
R_Election_1(m.writer,m.type,"R_Election_1");
else if(m.type=="V2" && state =="initial")
R_Election_2(m.writer,m.type,"R_Election_2");
else if(m.type=="V3" && state =="initial")
R_Election_3(m.writer,m.type,"R_Election_3");
}}catch(Exception e){System.out.println(name + ":
demultiplex error");}}}.start();

new Thread(){public void run(){
try{for(;;){
Message m=(Message)elected.receive();
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V3" && state =="initial")
R_Elected_3(m.writer,m.type,"R_Elected_3");
}}catch(Exception e){System.out.println(name + ":
demultiplex error");}}}.start();

}

public void REN0(){
if(state=="REN0")
S_Election3_0();
}

public void REN1(){
if(state=="REN1")
S_Election3_1();
}

public void REN2(){
if(state=="REN2")
S_Election3_2();
}
```

```
public void REN3(){
if(state=="REN3")
S_Elected_3();
}

public void R_Election_0(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_0);
ButtonR_Election_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN0");
displayTrace( "R_Election_0", "Read", "initial",
"REN0", "election", "V0");
REN0();
ButtonR_Election_0.setVisible(false);
}
});
}

public void R_Election_1(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_1);
ButtonR_Election_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN1");
displayTrace( "R_Election_1", "Read", "initial",
"REN1", "election", "V1");
REN1();
ButtonR_Election_1.setVisible(false);
}
});
}

public void R_Election_2(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_2);
ButtonR_Election_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN2");
displayTrace( "R_Election_2", "Read", "initial",
"REN2", "election", "V2");
REN2();
ButtonR_Election_2.setVisible(false);
}
});
}

public void R_Election_3(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_3);
ButtonR_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN3");
displayTrace( "R_Election_3", "Read", "initial",
"REN3", "election", "V3");
REN3();
ButtonR_Election_3.setVisible(false);
}
});
}

public void R_Elected_3(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_3);
ButtonR_Elected_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("IMBoss");
displayTrace( "R_Elected_3", "Read", "initial",
"IMBoss", "elected", "V3");
ButtonR_Elected_3.setVisible(false);
}
});
}

public void S_Election3_2(){
traceTable.center.add(ButtonS_Election3_2);
final Message m=new
Message("V3",this,p0,"PassOn_election");
ButtonS_Election3_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p0.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " :
S_Election3_2- send error");}

displayTrace( "S_Election3_2", "Create", "REN2",
"initial=", "PassOn_election", "V3");
ButtonS_Election3_2.setVisible(false);
}

});

}

public void S_Elected_3(){
traceTable.center.add(ButtonS_Elected_3);
final Message m=new
Message("V3",this,p0,"PassOn_elected");
ButtonS_Elected_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p0.elected.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " :
S_Elected_3- send error");}

displayTrace( "S_Elected_3", "Create", "REN3",
"initial=", "PassOn_elected", "V3");
ButtonS_Elected_3.setVisible(false);
}

});

}

public void S_Election3_0(){
System.out.println(name+" : S_Election3_0");

traceTable.center.add(ButtonS_Election3_0);
final Message m=new
Message("V3",this,p0,"election");
ButtonS_Election3_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p0.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p3-
send error");}
displayTrace( "S_Election3_0", "Write", "REN0",
"initial=", "PassOn_election", "V3");
ButtonS_Election3_0.setVisible(false);
}

});
```

```
}

public void S_Election3_1(){
System.out.println(name+" : S_Election3_1");

traceTable.center.add(ButtonS_Election3_1);
final Message m=new
Message("V3",this,p0,"election");
ButtonS_Election3_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p0.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p3-
send error");}
displayTrace( "S_Election3_1", "Write", "REN1",
"initial=", "PassOn_election", "V3");
ButtonS_Election3_1.setVisible(false);
}

});


}

Process p0;
public void connection_p0(Process temp){
p0=temp;
}

Process p1;
public void connection_p1(Process temp){
p1=temp;
}

Process p2;
public void connection_p2(Process temp){
p2=temp;
}

Process p3;
public void connection_p3(Process temp){
p3=temp;
}

Process Tr;
public void connection_Tr(Process temp){
Tr=temp;
}

public String getname(){
return name;
}

public void displayTrace(String ev, String ty, String
be, String af, String ch, String va){
traceTable.table.setValueAt((new
Integer(noOfevents)).toString(), noOfevents,0);
traceTable.table.setValueAt(ev, noOfevents,1);
traceTable.table.setValueAt(ty, noOfevents,2);
traceTable.table.setValueAt(be, noOfevents,3);
traceTable.table.setValueAt(af, noOfevents,4);
traceTable.table.setValueAt(ch, noOfevents,5);
traceTable.table.setValueAt(va, noOfevents,6);
noOfevents++;
}

public void transformState(String s){
state=s;
System.out.println(name +" : " + state);
}
}

static class trigger extends Process {
static myGUI traceTable;
public trigger (String name){
this.name =name;
traceTable=new myGUI(name,"trigger");
this.start();
}

JButton Buttonstart_election = new
JButton("start_election" );

public String state=" ";
int noOfevents=0;

public void run(){
transformState("initial");
initial();
}

public void initial(){
if(state=="initial")
start_election();
}

public void start_election(){
traceTable.center.add(Buttonstart_election);
final Message m=new
Message("V0",this,p1,"startElection");
Buttonstart_election.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p1.election.send(m);
transformState("finish");
}
catch(Exception f){System.out.println(name + " :
start_election- send error");}

displayTrace( "start_election", "Create", "initial",
"finish", "startElection", "V0");
Buttonstart_election.setVisible(false);
}

});

}

Process p0;
public void connection_p0(Process temp){
p0=temp;
}

Process p1;
public void connection_p1(Process temp){
p1=temp;
}

Process p2;
public void connection_p2(Process temp){
p2=temp;
}

Process p3;
public void connection_p3(Process temp){
p3=temp;
}

Process Tr;
public void connection_Tr(Process temp){
Tr=temp;
}

public String getname(){
```

```
return name;
}

public void displayTrace(String ev, String ty, String
be, String af, String ch, String va){
traceTable.table.setValueAt((new
Integer(noOfevents)).toString(), noOfevents,0);
traceTable.table.setValueAt(ev, noOfevents,1);
traceTable.table.setValueAt(ty, noOfevents,2);
traceTable.table.setValueAt(be, noOfevents,3);
traceTable.table.setValueAt(af, noOfevents,4);
traceTable.table.setValueAt(ch, noOfevents,5);
traceTable.table.setValueAt(va, noOfevents,6);
noOfevents++;
}

public void transformState(String s){
state=s;
System.out.println(name +" : " + state);
}
}

public static void main(String args[]) {
participant0 p0 = new participant0("p0");
participant1 p1 = new participant1("p1");
participant2 p2 = new participant2("p2");
participant3 p3 = new participant3("p3");
trigger Tr = new trigger("Tr");

Tr.connection_p1(p1); p1.connection_Tr(Tr);
p0.connection_p1(p1); p1.connection_p0(p0);
p0.connection_p1(p1); p1.connection_p0(p0);
p1.connection_p2(p2); p2.connection_p1(p1);
p1.connection_p2(p2); p2.connection_p1(p1);
p2.connection_p3(p3); p3.connection_p2(p2);
p2.connection_p3(p3); p3.connection_p2(p2);
p3.connection_p0(p0); p0.connection_p3(p3);
p3.connection_p0(p0); p0.connection_p3(p3);
}

}
```

D.2 A Cycle Election Model (Model 1 with Asynchronous Communication)

```
/* Generated from file C:\Documents and
Settings\pfx01r\My Documents\Report\cycle.xml */

import java.io.*;
import java.awt.*;
import java.math.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.text.*;
import javax.swing.table.*;

public class cycle{

static class Message {
String type; Process writer; Process reader; String
channel;

Message (String t, Process p, Process r, String c){
type=t; writer=p; reader=r; channel=c;
}

public String toString(){
return type + " from " + writer.toString() + " to " +
reader.toString() + " via " +channel;
}
}

static class myGUI extends JFrame{
String[] headerStr = {"No.","Event", "Type", "Before
state", "After state", "Channel", "Value"};
DefaultTableModel dm = new
DefaultTableModel(headerStr, 60);
JTable table = new JTable(dm);
JPanel center=new JPanel();

JLabel instanceLabel;
JTextField instanceField;

JLabel eventsLabel;
JLabel processLabel;
JTextField processField;

myGUI(String a, String b){
setTitle("cycle");
setLocation(200,200);
setSize(30,30);

JPanel top =new JPanel();
top.setBackground(Color.gray);
instanceLabel= new JLabel("Instance");
top.add(instanceLabel);

instanceField=new JTextField(a,15);
Font g =new Font("Roman",Font.PLAIN,12);
top.setFont(g);
top.add(instanceField);

processLabel= new JLabel("Process");
top.add(processLabel);

processField=new JTextField(b,15);
Font h =new Font("Roman",Font.ITALIC,12);
top.setFont(h);
top.add(processField);

getContentPane().add(top, BorderLayout.NORTH);

JPanel middle =new JPanel();
middle.setBackground(Color.green);
eventsLabel= new JLabel("Possible event(s):");
middle.add(eventsLabel);
getContentPane().add(middle,
BorderLayout.WEST);

center.setBackground(Color.gray);
getContentPane().add(center,
BorderLayout.CENTER);

JPanel record =new JPanel();
table.setPreferredScrollableViewportSize(new
Dimension(200, 150));
```

```
getContentPane().add(new JScrollPane(table),
BorderLayout.SOUTH);

pack();
setVisible(true);

}

}

static class MessageQueue{
int entries;
int maxEntries;
String name;
Message[] elements;

public MessageQueue(String n, int m){
name=n;
maxEntries=m;
elements=new Message[maxEntries];
entries=0;
}

synchronized void send(Message x) throws
InterruptedException{
while(entries==maxEntries)wait();
elements[entries]=x;
entries=entries+1;
System.out.println("send("+x+")");
notify();
}

synchronized Message receive() throws
InterruptedException{
while(entries==0)wait();
Message x; x=elements[0];
for(int i=1; i<entries; i++) {
elements[i-1]=elements[i];
}
entries=entries-1;
System.out.println("receive("+x+")");
notify();
return x;
}
}

static class Process extends Thread {
MessageQueue startElection;
MessageQueue PassOn_election;
MessageQueue election;
MessageQueue PassOn_elected;
MessageQueue elected;
String name;
public String toString(){
return this.name;
}
}

static class participant0 extends Process {
static myGUI traceTable;
public participant0 (String name){
this.name =name;
traceTable=new myGUI(name,"participant0");
election=new MessageQueue("election",3);
elected=new MessageQueue("elected",3);
this.start();
}

JButton ButtonR_Election_0 = new
JButton("R_Election_0" );
JButton ButtonS_Elected_0 = new
JButton("S_Elected_0" );
JButton ButtonR_Election_1 = new
JButton("R_Election_1" );
JButton ButtonS_Election_1 = new
JButton("S_Election_1" );
JButton ButtonR_Election_2 = new
JButton("R_Election_2" );
JButton ButtonS_Election_2 = new
JButton("S_Election_2" );
JButton ButtonR_Election_3 = new
JButton("R_Election_3" );
JButton ButtonS_Election_3 = new
JButton("S_Election_3" );
JButton ButtonR_Elected_0 = new
JButton("R_Elected_0" );
JButton ButtonR_Elected_1 = new
JButton("R_Elected_1" );
JButton ButtonS_ED_1 = new JButton("S_ED_1" );
JButton ButtonR_Elected_2 = new
JButton("R_Elected_2" );
JButton ButtonS_ED_2 = new JButton("S_ED_2" );
JButton ButtonR_Elected_3 = new
JButton("R_Elected_3" );
JButton ButtonS_ED_3 = new JButton("S_ED_3" );

public String state=" ";
int noOfevents=0;

public void run(){
transformState("initial");
new Thread(){public void run(){
try{for(;;){
Message m=(Message)election.receive();
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V0" && state =="initial")
R_Election_0(m.writer,m.type,"R_Election_0");
else if(m.type=="V1" && state =="initial")
R_Election_1(m.writer,m.type,"R_Election_1");
else if(m.type=="V2" && state =="initial")
R_Election_2(m.writer,m.type,"R_Election_2");
else if(m.type=="V3" && state =="initial")
R_Election_3(m.writer,m.type,"R_Election_3");
}}catch(Exception e){System.out.println(name + ":
demultiplex error");}}}.start();

new Thread(){public void run(){
try{for(;;){
Message m=(Message)elected.receive();
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V0" && state =="initial")
R_Elected_0(m.writer,m.type,"R_Elected_0");
else if(m.type=="V1" && state =="initial")
R_Elected_1(m.writer,m.type,"R_Elected_1");
else if(m.type=="V2" && state =="initial")
R_Elected_2(m.writer,m.type,"R_Elected_2");
else if(m.type=="V3" && state =="initial")
R_Elected_3(m.writer,m.type,"R_Elected_3");
}}catch(Exception e){System.out.println(name + ":
demultiplex error");}}}.start();

}

public void REN0(){
if(state=="REN0")
S_Elected_0();
}

public void REN1(){
if(state=="REN1")
S_Election_1();
}

public void REN2(){
if(state=="REN2")
```

```
S_Election_2();
}

public void REN3(){
if(state=="REN3")
S_Election_3();
}

public void RED1(){
if(state=="RED1")
S_ED_1();
}

public void RED2(){
if(state=="RED2")
S_ED_2();
}

public void RED3(){
if(state=="RED3")
S_ED_3();
}

public void R_Election_0(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_0);
ButtonR_Election_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN0");
displayTrace( "R_Election_0", "Read", "initial",
"REN0", "election", "V0");
REN0();
ButtonR_Election_0.setVisible(false);
}
});
}

public void R_Election_1(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_1);
ButtonR_Election_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN1");
displayTrace( "R_Election_1", "Read", "initial",
"REN1", "election", "V1");
REN1();
ButtonR_Election_1.setVisible(false);
}
});
}

public void R_Election_2(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_2);
ButtonR_Election_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN2");
displayTrace( "R_Election_2", "Read", "initial",
"REN2", "election", "V2");
REN2();
ButtonR_Election_2.setVisible(false);
}
});
}

public void R_Election_3(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_3);
ButtonR_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN3");
displayTrace( "R_Election_3", "Read", "initial",
"REN3", "election", "V3");
REN3();
ButtonR_Election_3.setVisible(false);
}
});
}

public void R_Elected_0(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_0);
ButtonR_Elected_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("IMBoss");
displayTrace( "R_Elected_0", "Read", "initial",
"IMBoss", "elected", "V0");
ButtonR_Elected_0.setVisible(false);
}
});
}

public void R_Elected_1(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_1);
ButtonR_Elected_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("RED1");
displayTrace( "R_Elected_1", "Read", "initial",
"RED1", "elected", "V1");
RED1();
ButtonR_Elected_1.setVisible(false);
}
});
}

public void R_Elected_2(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_2);
ButtonR_Elected_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("RED2");
displayTrace( "R_Elected_2", "Read", "initial",
"RED2", "elected", "V2");
RED2();
ButtonR_Elected_2.setVisible(false);
}
});
}

public void R_Elected_3(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_3);
ButtonR_Elected_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("RED3");
displayTrace( "R_Elected_3", "Read", "initial",
"RED3", "elected", "V3");
RED3();
ButtonR_Elected_3.setVisible(false);
}
});
}

public void S_Election_3(){
traceTable.center.add(ButtonS_Election_3);
final Message m=new
Message("V3",this,p1,"PassOn_election");
```

```
ButtonS_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p1.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " :
S_Election_3- send error");}

displayTrace( "S_Election_3", "Create", "REN3",
"initial=", "PassOn_election", "V3");
ButtonS_Election_3.setVisible(false);
}

});

}

public void S_ED_3(){
traceTable.center.add(ButtonS_ED_3);
final Message m=new
Message("V3",this,p1,"PassOn_elected");
ButtonS_ED_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p1.elected.send(m);
transformState("3_Boss");
}
catch(Exception f){System.out.println(name + " :
S_ED_3- send error");}

displayTrace( "S_ED_3", "Create", "RED3",
"3_Boss", "PassOn_elected", "V3");
ButtonS_ED_3.setVisible(false);
}

});

}

public void S_Elected_0(){
System.out.println(name+" : S_Elected_0");

traceTable.center.add(ButtonS_Elected_0);
final Message m=new
Message("V0",this,p1,"elected");
ButtonS_Elected_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p1.elected.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p0-
send error");}
displayTrace( "S_Elected_0", "Write", "REN0",
"initial=", "PassOn_elected", "V0");
ButtonS_Elected_0.setVisible(false);
}

});


}

public void S_Election_1(){
System.out.println(name+" : S_Election_1");

traceTable.center.add(ButtonS_Election_1);
final Message m=new
Message("V1",this,p1,"election");
ButtonS_Election_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p1.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p0-
send error");}
displayTrace( "S_Election_1", "Write", "REN1",
"initial=", "PassOn_election", "V1");
ButtonS_Election_1.setVisible(false);
}

});


}

public void S_Election_2(){
System.out.println(name+" : S_Election_2");

traceTable.center.add(ButtonS_Election_2);
final Message m=new
Message("V2",this,p1,"election");
ButtonS_Election_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p1.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p0-
send error");}
displayTrace( "S_Election_2", "Write", "REN2",
"initial=", "PassOn_election", "V2");
ButtonS_Election_2.setVisible(false);
}

});


}

public void S_ED_1(){
System.out.println(name+" : S_ED_1");

traceTable.center.add(ButtonS_ED_1);
final Message m=new
Message("V1",this,p1,"elected");
ButtonS_ED_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p1.elected.send(m);
transformState("1_Boss");
}
catch(Exception f){System.out.println(name + " : p0-
send error");}
displayTrace( "S_ED_1", "Write", "RED1",
"1_Boss", "PassOn_elected", "V1");
ButtonS_ED_1.setVisible(false);
}

});


}

public void S_ED_2(){
System.out.println(name+" : S_ED_2");

traceTable.center.add(ButtonS_ED_2);
final Message m=new
Message("V2",this,p1,"elected");
ButtonS_ED_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p1.elected.send(m);
transformState("2_Boss");
}
catch(Exception f){System.out.println(name + " : p0-
send error");}
displayTrace( "S_ED_2", "Write", "RED2",
"2_Boss", "PassOn_elected", "V2");
ButtonS_ED_2.setVisible(false);
}

});


}

Process p0;
public void connection_p0(Process temp){
p0=temp;
}

Process p1;
public void connection_p1(Process temp){
p1=temp;
}

Process p2;
public void connection_p2(Process temp){
p2=temp;
}

Process p3;
public void connection_p3(Process temp){
p3=temp;
}

Process Tr;
public void connection_Tr(Process temp){
Tr=temp;
}

public String getname(){
return name;
}

public void displayTrace(String ev, String ty, String
be, String af, String ch, String va){
traceTable.table.setValueAt((new
Integer(noOfevents)).toString(), noOfevents,0);
traceTable.table.setValueAt(ev, noOfevents,1);
traceTable.table.setValueAt(ty, noOfevents,2);
traceTable.table.setValueAt(be, noOfevents,3);
traceTable.table.setValueAt(af, noOfevents,4);
traceTable.table.setValueAt(ch, noOfevents,5);
traceTable.table.setValueAt(va, noOfevents,6);
noOfevents++;
}

public void transformState(String s){
state=s;
System.out.println(name +" : " + state);
}
}

static class participant1 extends Process {
static myGUI traceTable;
public participant1 (String name){
this.name =name;
traceTable=new myGUI(name,"participant1");
election=new MessageQueue("election",3);
elected=new MessageQueue("elected",3);
this.start();
}

JButton ButtonR_Election_0 = new
JButton("R_Election_0" );
JButton ButtonS_Election1_0 = new
JButton("S_Election1_0" );
JButton ButtonR_Election_1 = new
JButton("R_Election_1" );
JButton ButtonS_Elected_1 = new
JButton("S_Elected_1" );
JButton ButtonR_Election_2 = new
JButton("R_Election_2" );
JButton ButtonS_Election_2 = new
JButton("S_Election_2" );
JButton ButtonR_Election_3 = new
JButton("R_Election_3" );
JButton ButtonS_Election_3 = new
JButton("S_Election_3" );
JButton ButtonR_Elected_1 = new
JButton("R_Elected_1" );
JButton ButtonR_Elected_2 = new
JButton("R_Elected_2" );
JButton ButtonS_ED_2 = new JButton("S_ED_2" );
JButton ButtonR_Elected_3 = new
JButton("R_Elected_3" );
JButton ButtonS_ED_3 = new JButton("S_ED_3" );

public String state=" ";
int noOfevents=0;

public void run(){
transformState("initial");
new Thread(){public void run(){
try{for(;;){
Message m=(Message)election.receive();
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V0" && state =="initial")
R_Election_0(m.writer,m.type,"R_Election_0");
else if(m.type=="V1" && state =="initial")
R_Election_1(m.writer,m.type,"R_Election_1");
else if(m.type=="V2" && state =="initial")
R_Election_2(m.writer,m.type,"R_Election_2");
else if(m.type=="V3" && state =="initial")
R_Election_3(m.writer,m.type,"R_Election_3");
}}catch(Exception e){System.out.println(name + ":
demultiplex error");}}}.start();

new Thread(){public void run(){
try{for(;;){
Message m=(Message)elected.receive();
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V1" && state =="initial")
R_Elected_1(m.writer,m.type,"R_Elected_1");
else if(m.type=="V2" && state =="initial")
R_Elected_2(m.writer,m.type,"R_Elected_2");
else if(m.type=="V3" && state =="initial")
R_Elected_3(m.writer,m.type,"R_Elected_3");
}}catch(Exception e){System.out.println(name + ":
demultiplex error");}}}.start();

}

public void REN0(){
if(state=="REN0")
S_Election1_0();
}

public void REN1(){
if(state=="REN1")
S_Elected_1();
}

public void REN2(){
if(state=="REN2")
S_Election_2();
```

```
}

public void REN3(){
if(state=="REN3")
S_Election_3();
}

public void RED2(){
if(state=="RED2")
S_ED_2();
}

public void RED3(){
if(state=="RED3")
S_ED_3();
}

public void R_Election_0(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_0);
ButtonR_Election_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN0");
displayTrace( "R_Election_0", "Read", "initial",
"REN0", "election", "V0");
REN0();
ButtonR_Election_0.setVisible(false);
}
});
}

public void R_Election_1(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_1);
ButtonR_Election_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN1");
displayTrace( "R_Election_1", "Read", "initial",
"REN1", "election", "V1");
REN1();
ButtonR_Election_1.setVisible(false);
}
});
}

public void R_Election_2(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_2);
ButtonR_Election_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN2");
displayTrace( "R_Election_2", "Read", "initial",
"REN2", "election", "V2");
REN2();
ButtonR_Election_2.setVisible(false);
}
});
}

public void R_Election_3(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_3);
ButtonR_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN3");
displayTrace( "R_Election_3", "Read", "initial",
"REN3", "election", "V3");
REN3();
ButtonR_Election_3.setVisible(false);
}
});
}

public void R_Elected_1(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_1);
ButtonR_Elected_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("IMBoss");
displayTrace( "R_Elected_1", "Read", "initial",
"IMBoss", "elected", "V1");
ButtonR_Elected_1.setVisible(false);
}
});
}

public void R_Elected_2(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_2);
ButtonR_Elected_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("RED2");
displayTrace( "R_Elected_2", "Read", "initial",
"RED2", "elected", "V2");
RED2();
ButtonR_Elected_2.setVisible(false);
}
});
}

public void R_Elected_3(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_3);
ButtonR_Elected_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("RED3");
displayTrace( "R_Elected_3", "Read", "initial",
"RED3", "elected", "V3");
RED3();
ButtonR_Elected_3.setVisible(false);
}
});
}

public void S_Election1_0(){
traceTable.center.add(ButtonS_Election1_0);
final Message m=new
Message("V1",this,p2,"PassOn_election");
ButtonS_Election1_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p2.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " :
S_Election1_0- send error");}

displayTrace( "S_Election1_0", "Create", "REN0",
"initial=", "PassOn_election", "V1");
ButtonS_Election1_0.setVisible(false);
}

});

}

public void S_ED_3(){
traceTable.center.add(ButtonS_ED_3);
```

```
final Message m=new
Message("V3",this,p2,"PassOn_elected");
ButtonS_ED_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p2.elected.send(m);
transformState("3_Boss");
}
catch(Exception f){System.out.println(name + " :
S_ED_3- send error");}

displayTrace( "S_ED_3", "Create", "RED3",
"3_Boss", "PassOn_elected", "V3");
ButtonS_ED_3.setVisible(false);
}

});

}

public void S_Elected_1(){
System.out.println(name+" : S_Elected_1");

traceTable.center.add(ButtonS_Elected_1);
final Message m=new
Message("V1",this,p2,"elected");
ButtonS_Elected_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p2.elected.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p1-
send error");}
displayTrace( "S_Elected_1", "Write", "REN1",
"initial=", "PassOn_elected", "V1");
ButtonS_Elected_1.setVisible(false);
}

});


}

public void S_Election_2(){
System.out.println(name+" : S_Election_2");

traceTable.center.add(ButtonS_Election_2);
final Message m=new
Message("V2",this,p2,"election");
ButtonS_Election_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p2.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p1-
send error");}
displayTrace( "S_Election_2", "Write", "REN2",
"initial=", "PassOn_election", "V2");
ButtonS_Election_2.setVisible(false);
}

});


}

public void S_Election_3(){
System.out.println(name+" : S_Election_3");

traceTable.center.add(ButtonS_Election_3);
final Message m=new
Message("V3",this,p2,"election");
ButtonS_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p2.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p1-
send error");}
displayTrace( "S_Election_3", "Write", "REN3",
"initial=", "PassOn_election", "V3");
ButtonS_Election_3.setVisible(false);
}

});


}

public void S_ED_2(){
System.out.println(name+" : S_ED_2");

traceTable.center.add(ButtonS_ED_2);
final Message m=new
Message("V2",this,p2,"elected");
ButtonS_ED_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p2.elected.send(m);
transformState("2_Boss");
}
catch(Exception f){System.out.println(name + " : p1-
send error");}
displayTrace( "S_ED_2", "Write", "RED2",
"2_Boss", "PassOn_elected", "V2");
ButtonS_ED_2.setVisible(false);
}

});


}

Process p0;
public void connection_p0(Process temp){
p0=temp;
}

Process p1;
public void connection_p1(Process temp){
p1=temp;
}

Process p2;
public void connection_p2(Process temp){
p2=temp;
}

Process p3;
public void connection_p3(Process temp){
p3=temp;
}

Process Tr;
public void connection_Tr(Process temp){
Tr=temp;
}

public String getname(){
return name;
}
```

```
public void displayTrace(String ev, String ty, String
be, String af, String ch, String va){
traceTable.table.setValueAt((new
Integer(noOfevents)).toString(), noOfevents,0);
traceTable.table.setValueAt(ev, noOfevents,1);
traceTable.table.setValueAt(ty, noOfevents,2);
traceTable.table.setValueAt(be, noOfevents,3);
traceTable.table.setValueAt(af, noOfevents,4);
traceTable.table.setValueAt(ch, noOfevents,5);
traceTable.table.setValueAt(va, noOfevents,6);
noOfevents++;
}

public void transformState(String s){
state=s;
System.out.println(name +" : " + state);
}
}

static class participant2 extends Process {
static myGUI traceTable;
public participant2 (String name){
this.name =name;
traceTable=new myGUI(name,"participant2");
election=new MessageQueue("election",3);
elected=new MessageQueue("elected",3);
this.start();
}

JButton ButtonR_Election_0 = new
JButton("R_Election_0" );
JButton ButtonS_Election2_0 = new
JButton("S_Election2_0" );
JButton ButtonR_Election_1 = new
JButton("R_Election_1" );
JButton ButtonS_Election2_1 = new
JButton("S_Election2_1" );
JButton ButtonR_Election_2 = new
JButton("R_Election_2" );
JButton ButtonS_Elected_2 = new
JButton("S_Elected_2" );
JButton ButtonR_Election_3 = new
JButton("R_Election_3" );
JButton ButtonS_Election_3 = new
JButton("S_Election_3" );
JButton ButtonR_Elected_2 = new
JButton("R_Elected_2" );
JButton ButtonR_Elected_3 = new
JButton("R_Elected_3" );
JButton ButtonS_ED_3 = new JButton("S_ED_3" );

public String state=" ";
int noOfevents=0;

public void run(){
transformState("initial");
new Thread(){public void run(){
try{for(;;){
Message m=(Message)election.receive();
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V0" && state =="initial")
R_Election_0(m.writer,m.type,"R_Election_0");
else if(m.type=="V1" && state =="initial")
R_Election_1(m.writer,m.type,"R_Election_1");
else if(m.type=="V2" && state =="initial")
R_Election_2(m.writer,m.type,"R_Election_2");
else if(m.type=="V3" && state =="initial")
R_Election_3(m.writer,m.type,"R_Election_3");
}}catch(Exception e){System.out.println(name + ":
demultiplex error");}}}.start();

new Thread(){public void run(){
try{for(;;){
Message m=(Message)elected.receive();
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V2" && state =="initial")
R_Elected_2(m.writer,m.type,"R_Elected_2");
else if(m.type=="V3" && state =="initial")
R_Elected_3(m.writer,m.type,"R_Elected_3");
}}catch(Exception e){System.out.println(name + ":
demultiplex error");}}}.start();

}

public void REN0(){
if(state=="REN0")
S_Election2_0();
}

public void REN1(){
if(state=="REN1")
S_Election2_1();
}

public void REN2(){
if(state=="REN2")
S_Elected_2();
}

public void REN3(){
if(state=="REN3")
S_Election_3();
}

public void RED3(){
if(state=="RED3")
S_ED_3();
}

public void R_Election_0(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_0);
ButtonR_Election_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN0");
displayTrace( "R_Election_0", "Read", "initial",
"REN0", "election", "V0");
REN0();
ButtonR_Election_0.setVisible(false);
}
});
}

public void R_Election_1(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_1);
ButtonR_Election_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN1");
displayTrace( "R_Election_1", "Read", "initial",
"REN1", "election", "V1");
REN1();
ButtonR_Election_1.setVisible(false);
}
});
}

public void R_Election_2(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_2);
ButtonR_Election_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN2");
displayTrace( "R_Election_2", "Read", "initial",
"REN2", "election", "V2");
REN2();
ButtonR_Election_2.setVisible(false);
}
});
}

public void R_Election_3(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_3);
ButtonR_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN3");
displayTrace( "R_Election_3", "Read", "initial",
"REN3", "election", "V3");
REN3();
ButtonR_Election_3.setVisible(false);
}
});
}

public void R_Elected_2(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_2);
ButtonR_Elected_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("IMBoss");
displayTrace( "R_Elected_2", "Read", "initial",
"IMBoss", "elected", "V2");
ButtonR_Elected_2.setVisible(false);
}
});
}

public void R_Elected_3(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_3);
ButtonR_Elected_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("RED3");
displayTrace( "R_Elected_3", "Read", "initial",
"RED3", "elected", "V3");
RED3();
ButtonR_Elected_3.setVisible(false);
}
});
}

public void S_Election2_1(){
traceTable.center.add(ButtonS_Election2_1);
final Message m=new
Message("V2",this,p3,"PassOn_election");
ButtonS_Election2_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p3.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " :
S_Election2_1- send error");}

displayTrace( "S_Election2_1", "Create", "REN1",
"initial=", "PassOn_election", "V2");
ButtonS_Election2_1.setVisible(false);
}

});

}

public void S_ED_3(){
traceTable.center.add(ButtonS_ED_3);
final Message m=new
Message("V3",this,p3,"PassOn_elected");
ButtonS_ED_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p3.elected.send(m);
transformState("3_Boss");
}
catch(Exception f){System.out.println(name + " :
S_ED_3- send error");}

displayTrace( "S_ED_3", "Create", "RED3",
"3_Boss", "PassOn_elected", "V3");
ButtonS_ED_3.setVisible(false);
}

});

}

public void S_Election2_0(){
System.out.println(name+" : S_Election2_0");

traceTable.center.add(ButtonS_Election2_0);
final Message m=new
Message("V2",this,p3,"election");
ButtonS_Election2_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p3.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p2-
send error");}
displayTrace( "S_Election2_0", "Write", "REN0",
"initial=", "PassOn_election", "V2");
ButtonS_Election2_0.setVisible(false);
}

});


}

public void S_Elected_2(){
System.out.println(name+" : S_Elected_2");

traceTable.center.add(ButtonS_Elected_2);
final Message m=new
Message("V2",this,p3,"elected");
ButtonS_Elected_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p3.elected.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p2-
send error");}
displayTrace( "S_Elected_2", "Write", "REN2",
"initial=", "PassOn_elected", "V2");
ButtonS_Elected_2.setVisible(false);
}

});


}

public void S_Election_3(){
System.out.println(name+" : S_Election_3");
```

```
traceTable.center.add(ButtonS_Election_3);
final Message m=new
Message("V3",this,p3,"election");
ButtonS_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p3.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p2-
send error");}
displayTrace( "S_Election_3", "Write", "REN3",
"initial=", "PassOn_election", "V3");
ButtonS_Election_3.setVisible(false);
}

});


}

Process p0;
public void connection_p0(Process temp){
p0=temp;
}

Process p1;
public void connection_p1(Process temp){
p1=temp;
}

Process p2;
public void connection_p2(Process temp){
p2=temp;
}

Process p3;
public void connection_p3(Process temp){
p3=temp;
}

Process Tr;
public void connection_Tr(Process temp){
Tr=temp;
}

public String getname(){
return name;
}

public void displayTrace(String ev, String ty, String
be, String af, String ch, String va){
traceTable.table.setValueAt((new
Integer(noOfevents)).toString(), noOfevents,0);
traceTable.table.setValueAt(ev, noOfevents,1);
traceTable.table.setValueAt(ty, noOfevents,2);
traceTable.table.setValueAt(be, noOfevents,3);
traceTable.table.setValueAt(af, noOfevents,4);
traceTable.table.setValueAt(ch, noOfevents,5);
traceTable.table.setValueAt(va, noOfevents,6);
noOfevents++;
}

public void transformState(String s){
state=s;
System.out.println(name +" : " + state);
}
}

static class participant3 extends Process {
static myGUI traceTable;
public participant3 (String name){
this.name =name;
traceTable=new myGUI(name,"participant3");
election=new MessageQueue("election",3);
elected=new MessageQueue("elected",3);
this.start();
}

JButton ButtonR_Election_0 = new
JButton("R_Election_0" );
JButton ButtonS_Election3_0 = new
JButton("S_Election3_0" );
JButton ButtonR_Election_1 = new
JButton("R_Election_1" );
JButton ButtonS_Election3_1 = new
JButton("S_Election3_1" );
JButton ButtonR_Election_2 = new
JButton("R_Election_2" );
JButton ButtonS_Election3_2 = new
JButton("S_Election3_2" );
JButton ButtonR_Election_3 = new
JButton("R_Election_3" );
JButton ButtonS_Elected_3 = new
JButton("S_Elected_3" );
JButton ButtonR_Elected_3 = new
JButton("R_Elected_3" );

public String state=" ";
int noOfevents=0;

public void run(){
transformState("initial");
new Thread(){public void run(){
try{for(;;){
Message m=(Message)election.receive();
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V0" && state =="initial")
R_Election_0(m.writer,m.type,"R_Election_0");
else if(m.type=="V1" && state =="initial")
R_Election_1(m.writer,m.type,"R_Election_1");
else if(m.type=="V2" && state =="initial")
R_Election_2(m.writer,m.type,"R_Election_2");
else if(m.type=="V3" && state =="initial")
R_Election_3(m.writer,m.type,"R_Election_3");
}}catch(Exception e){System.out.println(name + ":
demultiplex error");}}}.start();

new Thread(){public void run(){
try{for(;;){
Message m=(Message)elected.receive();
if(m.type==" XXXX_XXXX " && state==null){ }
else if(m.type=="V3" && state =="initial")
R_Elected_3(m.writer,m.type,"R_Elected_3");
}}catch(Exception e){System.out.println(name + ":
demultiplex error");}}}.start();

}

public void REN0(){
if(state=="REN0")
S_Election3_0();
}

public void REN1(){
if(state=="REN1")
S_Election3_1();
}

public void REN2(){
if(state=="REN2")
S_Election3_2();
}

public void REN3(){
```

```
if(state=="REN3")
S_Elected_3();
}

public void R_Election_0(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_0);
ButtonR_Election_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN0");
displayTrace( "R_Election_0", "Read", "initial",
"REN0", "election", "V0");
REN0();
ButtonR_Election_0.setVisible(false);
}
});
}

public void R_Election_1(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_1);
ButtonR_Election_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN1");
displayTrace( "R_Election_1", "Read", "initial",
"REN1", "election", "V1");
REN1();
ButtonR_Election_1.setVisible(false);
}
});
}

public void R_Election_2(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_2);
ButtonR_Election_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN2");
displayTrace( "R_Election_2", "Read", "initial",
"REN2", "election", "V2");
REN2();
ButtonR_Election_2.setVisible(false);
}
});
}

public void R_Election_3(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_3);
ButtonR_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN3");
displayTrace( "R_Election_3", "Read", "initial",
"REN3", "election", "V3");
REN3();
ButtonR_Election_3.setVisible(false);
}
});
}

public void R_Elected_3(Process from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_3);
ButtonR_Elected_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("IMBoss");
displayTrace( "R_Elected_3", "Read", "initial",
"IMBoss", "elected", "V3");
ButtonR_Elected_3.setVisible(false);
}
});
}

public void S_Election3_2(){
traceTable.center.add(ButtonS_Election3_2);
final Message m=new
Message("V3",this,p0,"PassOn_election");
ButtonS_Election3_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p0.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " :
S_Election3_2- send error");}

displayTrace( "S_Election3_2", "Create", "REN2",
"initial=", "PassOn_election", "V3");
ButtonS_Election3_2.setVisible(false);
}

});

}

public void S_Elected_3(){
traceTable.center.add(ButtonS_Elected_3);
final Message m=new
Message("V3",this,p0,"PassOn_elected");
ButtonS_Elected_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p0.elected.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " :
S_Elected_3- send error");}

displayTrace( "S_Elected_3", "Create", "REN3",
"initial=", "PassOn_elected", "V3");
ButtonS_Elected_3.setVisible(false);
}

});

}

public void S_Election3_0(){
System.out.println(name+" : S_Election3_0");

traceTable.center.add(ButtonS_Election3_0);
final Message m=new
Message("V3",this,p0,"election");
ButtonS_Election3_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p0.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p3-
send error");}
displayTrace( "S_Election3_0", "Write", "REN0",
"initial=", "PassOn_election", "V3");
ButtonS_Election3_0.setVisible(false);
}

});


}
```

```
public void S_Election3_1(){
System.out.println(name+" : S_Election3_1");

traceTable.center.add(ButtonS_Election3_1);
final Message m=new
Message("V3",this,p0,"election");
ButtonS_Election3_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p0.election.send(m);
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p3-
send error");}
displayTrace( "S_Election3_1", "Write", "REN1",
"initial=", "PassOn_election", "V3");
ButtonS_Election3_1.setVisible(false);
}

});


}

Process p0;
public void connection_p0(Process temp){
p0=temp;
}

Process p1;
public void connection_p1(Process temp){
p1=temp;
}

Process p2;
public void connection_p2(Process temp){
p2=temp;
}

Process p3;
public void connection_p3(Process temp){
p3=temp;
}

Process Tr;
public void connection_Tr(Process temp){
Tr=temp;
}

public String getname(){
return name;
}

public void displayTrace(String ev, String ty, String
be, String af, String ch, String va){
traceTable.table.setValueAt((new
Integer(noOfevents)).toString(), noOfevents,0);
traceTable.table.setValueAt(ev, noOfevents,1);
traceTable.table.setValueAt(ty, noOfevents,2);
traceTable.table.setValueAt(be, noOfevents,3);
traceTable.table.setValueAt(af, noOfevents,4);
traceTable.table.setValueAt(ch, noOfevents,5);
traceTable.table.setValueAt(va, noOfevents,6);
noOfevents++;
}

public void transformState(String s){
state=s;
System.out.println(name +" : " + state);
}
}

static class trigger extends Process {
```

```
static myGUI traceTable;
public trigger (String name){
this.name =name;
traceTable=new myGUI(name,"trigger");
this.start();
}

JButton Buttonstart_election = new
JButton("start_election" );

public String state=" ";
int noOfevents=0;

public void run(){
transformState("initial");
initial();
}

public void initial(){
if(state=="initial")
start_election();
}

public void start_election(){
traceTable.center.add(Buttonstart_election);
final Message m=new
Message("V0",this,p1,"startElection");
Buttonstart_election.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{p1.election.send(m);
transformState("finish");
}
catch(Exception f){System.out.println(name + " :
start_election- send error");}

displayTrace( "start_election", "Create", "initial",
"finish", "startElection", "V0");
Buttonstart_election.setVisible(false);
}

});

}

Process p0;
public void connection_p0(Process temp){
p0=temp;
}

Process p1;
public void connection_p1(Process temp){
p1=temp;
}

Process p2;
public void connection_p2(Process temp){
p2=temp;
}

Process p3;
public void connection_p3(Process temp){
p3=temp;
}

Process Tr;
public void connection_Tr(Process temp){
Tr=temp;
}

public String getname(){
return name;
}
```

```
public void displayTrace(String ev, String ty, String
be, String af, String ch, String va){
traceTable.table.setValueAt((new
Integer(noOfevents)).toString(), noOfevents,0);
traceTable.table.setValueAt(ev, noOfevents,1);
traceTable.table.setValueAt(ty, noOfevents,2);
traceTable.table.setValueAt(be, noOfevents,3);
traceTable.table.setValueAt(af, noOfevents,4);
traceTable.table.setValueAt(ch, noOfevents,5);
traceTable.table.setValueAt(va, noOfevents,6);
noOfevents++;
}

public void transformState(String s){
state=s;
System.out.println(name +" : " + state);
}
}

public static void main(String args[]) {
participant0 p0 = new participant0("p0");
participant1 p1 = new participant1("p1");
participant2 p2 = new participant2("p2");
participant3 p3 = new participant3("p3");
trigger Tr = new trigger("Tr");

Tr.connection_p1(p1); p1.connection_Tr(Tr);
p0.connection_p1(p1); p1.connection_p0(p0);
p0.connection_p1(p1); p1.connection_p0(p0);
p1.connection_p2(p2); p2.connection_p1(p1);
p1.connection_p2(p2); p2.connection_p1(p1);
p2.connection_p3(p3); p3.connection_p2(p2);
p2.connection_p3(p3); p3.connection_p2(p2);
p3.connection_p0(p0); p0.connection_p3(p3);
p3.connection_p0(p0); p0.connection_p3(p3);
}

}
```

Appendix E Distributed System Based on Cycle Election Model using RPC-based Web Service

E.1 Trigger Application Implementation

```
/*
The Cycle Election Service

the participating Trigger
*/
package trigger;

import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.JAXRPCException;
import javax.xml.namespace.QName;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.ParameterMode;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.awt.Font;
import java.awt.FontMetrics;
import javax.swing.*;
import java.awt.Graphics;
import javax.swing.text.*;
import javax.swing.table.*;

public class trigger extends JFrame
{
public String name = "trigger";
public static String BODY_NAMESPACE_VALUE =
"urn:Foo";
public static String
ENCODING_STYLE_PROPERTY =
"Javax.xml.rpc.encodingstyle.namespace.uri";
public static String NS_XSD =
"http://www.w3.org/2001/XMLSchema";
public static String URI_ENCODING =
"http://schemas.xmlsoap.org/soap/encoding/";

public static String participant1_p1_qnameService =
"participant1_p1_Service";
public static String participant1_p1_qnamePort =
"participant1_IF";
public Call participant1_p1_call;
public ServiceFactory participant1_p1_factory;
public Service participant1_p1_service;
public QName participant1_p1_port;


        static myGUI traceTable;
public String state=" ";
private int noOfevents=0;
private Message message;
JButton Buttonstart_election = new
JButton("start_election" );

static class myGUI extends JFrame{
String[] headerStr = {"No.","Event", "Type", "Before
state", "After state", "Channel", "Value"};
DefaultTableModel dm = new
DefaultTableModel(headerStr, 60);
JTable table = new JTable(dm);
JPanel center=new JPanel();

JLabel instanceLabel;
JTextField instanceField;

JLabel eventsLabel;
JLabel processLabel;
JTextField processField;

myGUI(String a, String b){
setTitle("cycle");
setLocation(200,200);
setSize(30,30);

JPanel top =new JPanel();
top.setBackground(Color.gray);
instanceLabel= new JLabel("Instance");
top.add(instanceLabel);

instanceField=new JTextField(a,15);
Font g =new Font("Roman",Font.PLAIN,12);
top.setFont(g);
top.add(instanceField);

processLabel= new JLabel("Process");
top.add(processLabel);

processField=new JTextField(b,15);
Font h =new Font("Roman",Font.ITALIC,12);
top.setFont(h);
top.add(processField);

getContentPane().add(top, BorderLayout.NORTH);

JPanel middle =new JPanel();
middle.setBackground(Color.green);
eventsLabel= new JLabel("Possible event(s):");
middle.add(eventsLabel);
getContentPane().add(middle,
BorderLayout.WEST);

center.setBackground(Color.gray);
getContentPane().add(center,
BorderLayout.CENTER);

JPanel record =new JPanel();
table.setPreferredScrollableViewportSize(new
Dimension(200, 150));
getContentPane().add(new JScrollPane(table),
BorderLayout.SOUTH);

pack();
setVisible(true);

}

}


static class Message {
        String type; String writer; String reader; String
channel;

        Message (String t, String p, String r, String c){
        type=t; writer=p; reader=r; channel=c;
        }

        public String toString(){
        return type + " from " + writer.toString() + " to "
+ reader.toString() + " via " +channel;
        }
        }


public String name;

public String toString(){
return this.name;
}
}


public trigger(String n) {

this.name =name;
          traceTable=new myGUI(name,"trigger");
          state="initial";
          initial();
}

private void send(String messageName,String
currentName,String participantName,String state)
{
```

```
try {
          currentName = name;
participant1_p1_factory =
ServiceFactory.newInstance();
participant1_p1_service =
(Service)participant1_p1_factory.createService(ne
w QName(participant1_p1_qnameService));
participant1_p1_port = new
QName(participant1_p1_qnamePort);

participant1_p1_call =
participant1_p1_service.createCall(participant1_p1
_port);
participant1_p1_call.setTargetEndpointAddress("htt
p://localhost:8080/cycle1-jaxrpc/participant1?wsdl");
participant1_p1_call.setProperty(Call.SOAPACTIO
N_USE_PROPERTY, new Boolean(true));
participant1_p1_call.setProperty(Call.SOAPACTIO
N_URI_PROPERTY, "");
participant1_p1_call.setProperty(ENCODING_STY
LE_PROPERTY, URI_ENCODING);
participant1_p1_call.setOperationName(new
QName(BODY_NAMESPACE_VALUE,
"createMessage"));
participant1_p1_call.addParameter("String_1", new
QName(NS_XSD, "string"), ParameterMode.IN);
participant1_p1_call.addParameter("String_2", new
QName(NS_XSD, "string"), ParameterMode.IN);
participant1_p1_call.addParameter("String_3", new
QName(NS_XSD, "string"), ParameterMode.IN);
participant1_p1_call.addParameter("String_4", new
QName(NS_XSD, "string"), ParameterMode.IN);

participant1_p1_call.setReturnType(null);



Object [] params =
{ messageName,currentName,participantName,stat
e };
participant1_p1_call.invokeOneWay(params);



} catch (Exception ex) {
ex.printStackTrace();
}
}


public void createMessage(String type, String writer,
String reader, String channel) {
if(type==" XXXX_XXXX " && state==null){ }
else if(type=="V0" && state =="initial"){
          }

else
/*it will be a warnning window exposed*/
          System.out.println("sdsdfsadfas");
}

public void initial(){
if(state.equals("initial"))
start_election();
}

public void start_election(){
traceTable.center.add(Buttonstart_election);
Buttonstart_election.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
send("V0",name,"p1","startElection");
transformState("finish");
}
catch(Exception f){System.out.println(name + " :
start_election- send error");}

displayTrace( "start_election", "Create", "initial",
"finish", "startElection", "V0");
Buttonstart_election.setVisible(false);
}

});

}
```

```
traceTable.table.setValueAt(ev, noOfevents,1);
traceTable.table.setValueAt(ty, noOfevents,2);
traceTable.table.setValueAt(be, noOfevents,3);
traceTable.table.setValueAt(af, noOfevents,4);
traceTable.table.setValueAt(ch, noOfevents,5);
traceTable.table.setValueAt(va, noOfevents,6);
noOfevents++;
}

public void transformState(String s){
        state=s;
        System.out.println(name +" : " + state);
      }
```

```
public String getname(){
return name;
}

public void displayTrace(String ev, String ty, String
be, String af, String ch, String va){
traceTable.table.setValueAt((new
Integer(noOfevents)).toString(), noOfevents,0);
```

```
public static void main(String[] args) throws
Exception
{
trigger Tr= new trigger ("Tr");
}
}
```

E.2 Participant0 Application

E.2.1 Interface

```
package participant0;

import java.rmi.Remote;
import java.rmi.RemoteException;
```

```
public interface participant0_IF extends Remote
{
public void createMessage(String type, String writer,
String reader, String channel) throws
RemoteException;
}
```

E.2.2 Implementation

```
package participant0;

import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.JAXRPCException;
import javax.xml.namespace.QName;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.ParameterMode;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import javax.swing.*;
import java.awt.Font;
import java.awt.FontMetrics;
import javax.swing.text.*;
import javax.swing.table.*;

public class participant0 extends JFrame
implements participant0_IF
{
public String name = "p0";
public static String BODY_NAMESPACE_VALUE =
"urn:Foo";
```

```
public static String
ENCODING_STYLE_PROPERTY =
"Javax.xml.rpc.encodingstyle.namespace.uri";
public static String NS_XSD =
"http://www.w3.org/2001/XMLSchema";
public static String URI_ENCODING =
"http://schemas.xmlsoap.org/soap/encoding/";

/*this is the first participant*/
public static String participant_p1_qnameService =
"participant1_p1_Service";
public static String participant1_p1_qnamePort =
"p1articipant1_IF";

public Call participant1_p1_call;
public ServiceFactory participant1_p1_factory;
public Service participant1_p1_service;
public QName participant1_p1_port;

JButton ButtonR_Election_0 = new
JButton("R_Election_0" );
JButton ButtonS_Elected_0 = new
JButton("S_Elected_0" );
JButton ButtonR_Election_1 = new
JButton("R_Election_1" );
```

```
JButton ButtonS_Election_1 = new
JButton("S_Election_1" );
JButton ButtonR_Election_2 = new
JButton("R_Election_2" );
JButton ButtonS_Election_2 = new
JButton("S_Election_2" );
JButton ButtonR_Election_3 = new
JButton("R_Election_3" );
JButton ButtonS_Election_3 = new
JButton("S_Election_3" );
JButton ButtonR_Elected_0 = new
JButton("R_Elected_0" );
JButton ButtonR_Elected_1 = new
JButton("R_Elected_1" );
JButton ButtonS_ED_1 = new JButton("S_ED_1" );
JButton ButtonR_Elected_2 = new
JButton("R_Elected_2" );
JButton ButtonS_ED_2 = new JButton("S_ED_2" );
JButton ButtonR_Elected_3 = new
JButton("R_Elected_3" );
JButton ButtonS_ED_3 = new JButton("S_ED_3" );

        /*the variables for Process*/
        static myGUI traceTable;
public String state=" ";
private int noOfevents=0;
//private Message message;


public participant0(String name){
         this.name =name;
         traceTable=new
myGUI(name,"participant0");
state="initial";
}

static class myGUI extends JFrame
{
String[] headerStr = {"No.","Event", "Type", "Before
state", "After state", "Channel", "Value"};
DefaultTableModel dm = new
DefaultTableModel(headerStr, 60);
JTable table = new JTable(dm);
JPanel center=new JPanel();

JLabel instanceLabel;
JTextField instanceField;

JLabel eventsLabel;
JLabel processLabel;
JTextField processField;

myGUI(String a, String b){
setTitle("cycle");
setLocation(200,200);
setSize(30,30);

JPanel top =new JPanel();
top.setBackground(Color.gray);
instanceLabel= new JLabel("Instance");
top.add(instanceLabel);

instanceField=new JTextField(a,15);
Font g =new Font("Roman",Font.PLAIN,12);
top.setFont(g);
top.add(instanceField);

processLabel= new JLabel("Process");
top.add(processLabel);

processField=new JTextField(b,15);
Font h =new Font("Roman",Font.ITALIC,12);
top.setFont(h);
top.add(processField);

getContentPane().add(top, BorderLayout.NORTH);

JPanel middle =new JPanel();
middle.setBackground(Color.green);
eventsLabel= new JLabel("Possible event(s):");
middle.add(eventsLabel);
getContentPane().add(middle,
BorderLayout.WEST);

center.setBackground(Color.gray);
getContentPane().add(center,
BorderLayout.CENTER);

JPanel record =new JPanel();
table.setPreferredScrollableViewportSize(new
Dimension(200, 150));
getContentPane().add(new JScrollPane(table),
BorderLayout.SOUTH);

pack();
setVisible(true);

}

}


private void send(String messageName,String
currentName,String participantName,String state)
{
try {
        currentName = name;
participant1_p1_factory =
ServiceFactory.newInstance();
participant1_p1_service =
(Service)participant1_p1_factory.createService(ne
w QName(participant_p1_qnameService));
participant1_p1_port = new
QName(participant1_p1_qnamePort);

participant1_p1_call =
participant1_p1_service.createCall(participant1_p1
_port);
participant1_p1_call.setTargetEndpointAddress("htt
p://localhost:8080/cycle1-jaxrpc/participant1?wsdl");
participant1_p1_call.setProperty(Call.SOAPACTIO
N_USE_PROPERTY, new Boolean(true));
participant1_p1_call.setProperty(Call.SOAPACTIO
N_URI_PROPERTY, "");
participant1_p1_call.setProperty(ENCODING_STY
LE_PROPERTY, URI_ENCODING);

participant1_p1_call.addParameter("String_1", new
QName(NS_XSD, "string"), ParameterMode.IN);
participant1_p1_call.addParameter("String_2", new
QName(NS_XSD, "string"), ParameterMode.IN);
participant1_p1_call.addParameter("String_3", new
QName(NS_XSD, "string"), ParameterMode.IN);
participant1_p1_call.addParameter("String_4", new
QName(NS_XSD, "string"), ParameterMode.IN);

participant1_p1_call.setReturnType(null);
participant1_p1_call.setOperationName(new
QName(BODY_NAMESPACE_VALUE,
"createMessage"));


Object [] params =
{ messageName,currentName,participantName,stat
e };
```

```
participant1_p1_call.invokeOneWay(params);



} catch (Exception ex) {
ex.printStackTrace();
}
}

public void createMessage(String type, String writer,
String reader, String channel) {

if(type.equals(" XXXX_XXXX ") && state==null){ }

else if(type.equals("V0") && state.equals("initial"))
R_Election_0(writer,type,"R_Election_0");
else if(type.equals("V1") && state.equals("initial"))
R_Election_1(writer,type,"R_Election_1");
else if(type.equals("V2") && state.equals("initial"))
R_Election_2(writer,type,"R_Election_2");
else if(type.equals("V3") && state.equals("initial"))
R_Election_3(writer,type,"R_Election_3");
else if(type.equals("V0") && state.equals("initial"))
R_Elected_0(writer,type,"R_Elected_0");
else if(type.equals("V1") && state.equals("initial"))
R_Elected_1(writer,type,"R_Elected_1");
else if(type.equals("V2") && state.equals("initial"))
R_Elected_2(writer,type,"R_Elected_2");
else if(type.equals("V3") && state.equals("initial"))
R_Elected_3(writer,type,"R_Elected_3");
else
/*it will be a warnning window exposed*/
        System.out.println("sdsdfsadfas");
}

public void REN0(){
if(state=="REN0")
S_Elected_0();
}

public void REN1(){
if(state=="REN1")
S_Election_1();
}

public void REN2(){
if(state=="REN2")
S_Election_2();
}

public void REN3(){
if(state=="REN3")
S_Election_3();
}

public void RED1(){
if(state=="RED1")
S_ED_1();
}

public void RED2(){
if(state=="RED2")
S_ED_2();
}

public void RED3(){
if(state=="RED3")
S_ED_3();
}

public void R_Election_0(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_0);
ButtonR_Election_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){

transformState("REN0");
displayTrace( "R_Election_0", "Read", "initial",
"REN0", "election", "V0");
REN0();
ButtonR_Election_0.setVisible(false);
}
});
}

public void R_Election_1(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_1);
ButtonR_Election_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){

transformState("REN1");
displayTrace( "R_Election_1", "Read", "initial",
"REN1", "election", "V1");
REN1();
ButtonR_Election_1.setVisible(false);
}
});
}

public void R_Election_2(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_2);
ButtonR_Election_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){

transformState("REN2");
displayTrace( "R_Election_2", "Read", "initial",
"REN2", "election", "V2");
REN2();
ButtonR_Election_2.setVisible(false);
}
});
}

public void R_Election_3(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_3);
ButtonR_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){

transformState("REN3");
displayTrace( "R_Election_3", "Read", "initial",
"REN3", "election", "V3");
REN3();
ButtonR_Election_3.setVisible(false);
}
});
}

public void R_Elected_0(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_0);
ButtonR_Elected_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){

transformState("IMBoss");
displayTrace( "R_Elected_0", "Read", "initial",
"IMBoss", "elected", "V0");
ButtonR_Elected_0.setVisible(false);
}
```

```
});
}

public void R_Elected_1(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_1);
ButtonR_Elected_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){

transformState("RED1");
displayTrace( "R_Elected_1", "Read", "initial",
"RED1", "elected", "V1");
RED1();
ButtonR_Elected_1.setVisible(false);
}
});
}

public void R_Elected_2(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_2);
ButtonR_Elected_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){

transformState("RED2");
displayTrace( "R_Elected_2", "Read", "initial",
"RED2", "elected", "V2");
RED2();
ButtonR_Elected_2.setVisible(false);
}
});
}

public void R_Elected_3(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_3);
ButtonR_Elected_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){

transformState("RED3");
displayTrace( "R_Elected_3", "Read", "initial",
"RED3", "elected", "V3");
RED3();
ButtonR_Elected_3.setVisible(false);
}
});
}

public void S_Election_3(){
traceTable.center.add(ButtonS_Election_3);
ButtonS_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{

send("V3",name,"p1","PassOn_election");
transformState("initial");
}
catch(Exception f){System.out.println(name + " :
S_Election_3- send error");}

displayTrace( "S_Election_3", "Create", "REN3",
"initial=", "PassOn_election", "V3");
ButtonS_Election_3.setVisible(false);
}

});

}

public void S_ED_3(){
traceTable.center.add(ButtonS_ED_3);
ButtonS_ED_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
send("V3",name,"p1","PassOn_elected");
transformState("3_Boss");
}
catch(Exception f){System.out.println(name + " :
S_ED_3- send error");}

displayTrace( "S_ED_3", "Create", "RED3",
"3_Boss", "PassOn_elected", "V3");
ButtonS_ED_3.setVisible(false);
}

});

}

public void S_Elected_0(){
System.out.println(name+" : S_Elected_0");

traceTable.center.add(ButtonS_Elected_0);
ButtonS_Elected_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
        send("V0",name,"p1","elected");
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p0-
send error");}
displayTrace( "S_Elected_0", "Write", "REN0",
"initial=", "PassOn_elected", "V0");
ButtonS_Elected_0.setVisible(false);
}

});


}

public void S_Election_1(){
System.out.println(name+" : S_Election_1");

traceTable.center.add(ButtonS_Election_1);
ButtonS_Election_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
        send("V1",name,"p1","election");
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p0-
send error");}
displayTrace( "S_Election_1", "Write", "REN1",
"initial=", "PassOn_election", "V1");
ButtonS_Election_1.setVisible(false);
}

});


}

public void S_Election_2(){
System.out.println(name+" : S_Election_2");

traceTable.center.add(ButtonS_Election_2);
ButtonS_Election_2.addActionListener( new
ActionListener(){
```

```
public void actionPerformed(ActionEvent e){
try{
        send("V2",name,"p1","election");
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p0-
send error");}
displayTrace( "S_Election_2", "Write", "REN2",
"initial=", "PassOn_election", "V2");
ButtonS_Election_2.setVisible(false);
}

});


}

public void S_ED_1(){
System.out.println(name+" : S_ED_1");

traceTable.center.add(ButtonS_ED_1);
ButtonS_ED_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
        send("V1",name,"p1","elected");
transformState("1_Boss");
}
catch(Exception f){System.out.println(name + " : p0-
send error");}
displayTrace( "S_ED_1", "Write", "RED1", "1_Boss",
"PassOn_elected", "V1");
ButtonS_ED_1.setVisible(false);
}

});


}

public void S_ED_2(){
System.out.println(name+" : S_ED_2");

traceTable.center.add(ButtonS_ED_2);
ButtonS_ED_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
        send("V1",name,"p1","elected");
transformState("2_Boss");
}
catch(Exception f){System.out.println(name + " : p0-
send error");}
displayTrace( "S_ED_2", "Write", "RED2", "2_Boss",
"PassOn_elected", "V2");
ButtonS_ED_2.setVisible(false);
}

});


}


public void displayTrace(String ev, String ty, String
be, String af, String ch, String va){
traceTable.table.setValueAt((new
Integer(noOfevents)).toString(), noOfevents,0);
traceTable.table.setValueAt(ev, noOfevents,1);
traceTable.table.setValueAt(ty, noOfevents,2);
traceTable.table.setValueAt(be, noOfevents,3);
traceTable.table.setValueAt(af, noOfevents,4);
traceTable.table.setValueAt(ch, noOfevents,5);
traceTable.table.setValueAt(va, noOfevents,6);
noOfevents++;
}

public void transformState(String s){
         state=s;
         System.out.println(name +" : " + state);
       }


public static void main(String[] args) throws
Exception
{
participant0 p0= new participant0 ("p0");
}
}
```

E.3 Participant1 Application

E.3.1 Interface

```
package participant1_p1;

import java.rmi.Remote;
import java.rmi.RemoteException;


public interface participant1_p1_IF extends Remote
{
public void createMessage(String type, String writer,
String reader, String channel) throws
RemoteException;
}
```

E.3.2 Implementation

```
package participant1;

import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.JAXRPCException;
import javax.xml.namespace.QName;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.ParameterMode;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.Font;
import javax.swing.text.*;
import javax.swing.table.*;


public class participant1 extends JFrame
implements participant1_IF
{
public String name = "p1";
public static String BODY_NAMESPACE_VALUE =
"urn:Foo";
public static String
ENCODING_STYLE_PROPERTY =
"Javax.xml.rpc.encodingstyle.namespace.uri";
public static String NS_XSD =
"http://www.w3.org/2001/XMLSchema";
public static String URI_ENCODING =
"http://schemas.xmlsoap.org/soap/encoding/";

/*this is the first participant*/
public static String participant2_p2_qnameService =
"participant2_p2_Service";
public static String participant2_p2_qnamePort =
"participant2_IF";

public Call participant2_p2_call;
public ServiceFactory participant2_p2_factory;
public Service participant2_p2_service;
public QName participant2_p2_port;

JButton ButtonR_Election_0 = new
JButton("R_Election_0" );
JButton ButtonS_Election1_0 = new
JButton("S_Election1_0" );
JButton ButtonR_Election_1 = new
JButton("R_Election_1" );
JButton ButtonS_Elected_1 = new
JButton("S_Elected_1" );
JButton ButtonR_Election_2 = new
JButton("R_Election_2" );
JButton ButtonS_Election_2 = new
JButton("S_Election_2" );
JButton ButtonR_Election_3 = new
JButton("R_Election_3" );
JButton ButtonS_Election_3 = new
JButton("S_Election_3" );
JButton ButtonR_Elected_1 = new
JButton("R_Elected_1" );
JButton ButtonR_Elected_2 = new
JButton("R_Elected_2" );
JButton ButtonS_ED_2 = new JButton("S_ED_2" );
JButton ButtonR_Elected_3 = new
JButton("R_Elected_3" );
JButton ButtonS_ED_3 = new JButton("S_ED_3" );


        /*the variables for Process*/
        static myGUI traceTable;
public String state=" ";
private int noOfevents=0;
public String name;
//private Message message;

public participant1 (String name){
this.name= name;
traceTable=new myGUI(name,"participant1");
state="initial";
}

static class myGUI extends JFrame{
String[] headerStr = {"No.","Event", "Type", "Before
state", "After state", "Channel", "Value"};
DefaultTableModel dm = new
DefaultTableModel(headerStr, 60);
JTable table = new JTable(dm);
JPanel center=new JPanel();

JLabel instanceLabel;
JTextField instanceField;

JLabel eventsLabel;
JLabel processLabel;
JTextField processField;

myGUI(String a, String b){
setTitle("cycle");
setLocation(200,200);
setSize(30,30);

JPanel top =new JPanel();
top.setBackground(Color.gray);
instanceLabel= new JLabel("Instance");
top.add(instanceLabel);

instanceField=new JTextField(a,15);
Font g =new Font("Roman",Font.PLAIN,12);
top.setFont(g);
top.add(instanceField);

processLabel= new JLabel("String ");
top.add(processLabel);

processField=new JTextField(b,15);
Font h =new Font("Roman",Font.ITALIC,12);
top.setFont(h);
top.add(processField);

getContentPane().add(top, BorderLayout.NORTH);

JPanel middle =new JPanel();
middle.setBackground(Color.green);
eventsLabel= new JLabel("Possible event(s):");
middle.add(eventsLabel);
getContentPane().add(middle,
BorderLayout.WEST);

center.setBackground(Color.gray);
getContentPane().add(center,
BorderLayout.CENTER);

JPanel record =new JPanel();
table.setPreferredScrollableViewportSize(new
Dimension(200, 150));
getContentPane().add(new JScrollPane(table),
BorderLayout.SOUTH);

pack();
setVisible(true);

}

}
```

```
private void send(String messageName,String
currentName,String participantName,String state)
{
try {

participant2_p2_factory =
ServiceFactory.newInstance();
participant2_p2_service =
(Service)participant2_p2_factory.createService(ne
w QName(participant2_p2_qnameService));
participant2_p2_port = new
QName(participant2_p2_qnamePort);

participant2_p2_call =
participant2_p2_service.createCall(participant2_p2
_port);
participant2_p2_call.setTargetEndpointAddress("htt
p://localhost:8080/cycle2-jaxrpc/participant2?wsdl");
participant2_p2_call.setProperty(Call.SOAPACTIO
N_USE_PROPERTY, new Boolean(true));
participant2_p2_call.setProperty(Call.SOAPACTIO
N_URI_PROPERTY, "");
participant2_p2_call.setProperty(ENCODING_STY
LE_PROPERTY, URI_ENCODING);

participant2_p2_call.addParameter("String_1", new
QName(NS_XSD, "string"), ParameterMode.IN);
participant2_p2_call.addParameter("String_2", new
QName(NS_XSD, "string"), ParameterMode.IN);
participant2_p2_call.addParameter("String_3", new
QName(NS_XSD, "string"), ParameterMode.IN);
participant2_p2_call.addParameter("String_4", new
QName(NS_XSD, "string"), ParameterMode.IN);

participant2_p2_call.setReturnType(null);
participant2_p2_call.setOperationName(new
QName(BODY_NAMESPACE_VALUE,
"createMessage"));


Object [] params =
{ messageName,currentName,participantName,stat
e };
participant2_p2_call.invokeOneWay(params);



} catch (Exception ex) {
ex.printStackTrace();
}
}

public void createMessage(String type, String writer,
String reader, String channel)
{


if(type.equals(" XXXX_XXXX ") && state==null){ }
else if(type.equals("V0") && state.equals("initial"))
R_Election_0(writer,type,"R_Election_0");
else if(type.equals("V1") && state.equals("initial"))
R_Election_1(writer,type,"R_Election_1");
else if(type.equals("V2") && state.equals("initial"))
R_Election_2(writer,type,"R_Election_2");
else if(type.equals("V3") && state.equals("initial"))
R_Election_3(writer,type,"R_Election_3");

else if(type.equals("V1") && state.equals("initial"))
R_Elected_1(writer,type,"R_Elected_1");
else if(type.equals("V2") && state.equals("initial"))
R_Elected_2(writer,type,"R_Elected_2");
else if(type.equals("V3") && state.equals("initial"))
R_Elected_3(writer,type,"R_Elected_3");
else
/*it will be a warnning window exposed*/
        System.out.println("sdsdfsadfas");
}

// public void setMessage(Message m) {
//      message= m;
//}



public void REN0(){
if(state=="REN0")
S_Election1_0();
}

public void REN1(){
if(state=="REN1")
S_Elected_1();
}

public void REN2(){
if(state=="REN2")
S_Election_2();
}

public void REN3(){
if(state=="REN3")
S_Election_3();
}

public void RED2(){
if(state=="RED2")
S_ED_2();
}

public void RED3(){
if(state=="RED3")
S_ED_3();
}

public void R_Election_0(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_0);
ButtonR_Election_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN0");
displayTrace( "R_Election_0", "Read", "initial",
"REN0", "election", "V0");
REN0();
ButtonR_Election_0.setVisible(false);
}
});
}

public void R_Election_1(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_1);
ButtonR_Election_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN1");
displayTrace( "R_Election_1", "Read", "initial",
"REN1", "election", "V1");
REN1();
ButtonR_Election_1.setVisible(false);
}
});
}

public void R_Election_2(String from, String
message, String current_state){
```

```
traceTable.center.add(ButtonR_Election_2);
ButtonR_Election_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN2");
displayTrace( "R_Election_2", "Read", "initial",
"REN2", "election", "V2");
REN2();
ButtonR_Election_2.setVisible(false);
}
});
}

public void R_Election_3(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_3);
ButtonR_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN3");
displayTrace( "R_Election_3", "Read", "initial",
"REN3", "election", "V3");
REN3();
ButtonR_Election_3.setVisible(false);
}
});
}

public void R_Elected_1(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_1);
ButtonR_Elected_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("IMBoss");
displayTrace( "R_Elected_1", "Read", "initial",
"IMBoss", "elected", "V1");
ButtonR_Elected_1.setVisible(false);
}
});
}

public void R_Elected_2(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_2);
ButtonR_Elected_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("RED2");
displayTrace( "R_Elected_2", "Read", "initial",
"RED2", "elected", "V2");
RED2();
ButtonR_Elected_2.setVisible(false);
}
});
}

public void R_Elected_3(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_3);
ButtonR_Elected_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("RED3");
displayTrace( "R_Elected_3", "Read", "initial",
"RED3", "elected", "V3");
RED3();
ButtonR_Elected_3.setVisible(false);
}
});
}

public void S_Election1_0(){
traceTable.center.add(ButtonS_Election1_0);
ButtonS_Election1_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
send("V1",name,"p2","PassOn_election");
transformState("initial");
}
catch(Exception f){System.out.println(name + " :
S_Election1_0- send error");}

displayTrace( "S_Election1_0", "Create", "REN0",
"initial=", "PassOn_election", "V1");
ButtonS_Election1_0.setVisible(false);
}

});

}

public void S_ED_3(){
traceTable.center.add(ButtonS_ED_3);
ButtonS_ED_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
        send("V3",name,"p2","PassOn_elected");
transformState("3_Boss");
}
catch(Exception f){System.out.println(name + " :
S_ED_3- send error");}

displayTrace( "S_ED_3", "Create", "RED3",
"3_Boss", "PassOn_elected", "V3");
ButtonS_ED_3.setVisible(false);
}

});

}

public void S_Elected_1(){
System.out.println(name+" : S_Elected_1");

traceTable.center.add(ButtonS_Elected_1);
ButtonS_Elected_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
        send("V1",name,"p2","elected");
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p1-
send error");}
displayTrace( "S_Elected_1", "Write", "REN1",
"initial=", "PassOn_elected", "V1");
ButtonS_Elected_1.setVisible(false);
}

});


}

public void S_Election_2(){
System.out.println(name+" : S_Election_2");

traceTable.center.add(ButtonS_Election_2);
ButtonS_Election_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
        send("V2",name,"p2","election");
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p1-
send error");}
displayTrace( "S_Election_2", "Write", "REN2",
"initial=", "PassOn_election", "V2");
ButtonS_Election_2.setVisible(false);
}

});


}

public void S_Election_3(){
System.out.println(name+" : S_Election_3");

traceTable.center.add(ButtonS_Election_3);
ButtonS_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
send("V3",name,"p2","election");
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p1-
send error");}
displayTrace( "S_Election_3", "Write", "REN3",
"initial=", "PassOn_election", "V3");
ButtonS_Election_3.setVisible(false);
}

});



}

public void S_ED_2(){
System.out.println(name+" : S_ED_2");

traceTable.center.add(ButtonS_ED_2);
ButtonS_ED_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
send("V2",name,"p2","elected");
transformState("2_Boss");
}
catch(Exception f){System.out.println(name + " : p1-
send error");}
displayTrace( "S_ED_2", "Write", "RED2", "2_Boss",
"PassOn_elected", "V2");
ButtonS_ED_2.setVisible(false);
}

});


}


public String getname(){
return name;
}

public void displayTrace(String ev, String ty, String
be, String af, String ch, String va){
traceTable.table.setValueAt((new
Integer(noOfevents)).toString(), noOfevents,0);
traceTable.table.setValueAt(ev, noOfevents,1);
traceTable.table.setValueAt(ty, noOfevents,2);
traceTable.table.setValueAt(be, noOfevents,3);
traceTable.table.setValueAt(af, noOfevents,4);
traceTable.table.setValueAt(ch, noOfevents,5);
traceTable.table.setValueAt(va, noOfevents,6);
noOfevents++;
}

public void transformState(String s){
        state=s;
        System.out.println(name +" : " + state);
      }


public static void main(String[] args) throws
Exception
{
participant1 p1= new participant1 ("p1");
}
}
```

E.4 Participant2 Application

E.4.1 Interface

```
package participant2;

import java.rmi.Remote;
import java.rmi.RemoteException;


public interface participant2_IF extends Remote
{
public void createMessage(String type, String writer,
String reader, String channel) throws
RemoteException;
}
```

E.4.2 Implementation

```
package participant2;

import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.JAXRPCException;
import javax.xml.namespace.QName;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.ParameterMode;

import java.awt.*;
import java.awt.event.*;
import java.awt.FontMetrics;
import javax.swing.*;
import javax.swing.text.*;
import javax.swing.table.*;

public class participant2 extends JFrame
implements participant2_IF
{
public String name = "p2";
public static String BODY_NAMESPACE_VALUE =
"urn:Foo";
public static String
ENCODING_STYLE_PROPERTY =
"Javax.xml.rpc.encodingstyle.namespace.uri";
public static String NS_XSD =
"http://www.w3.org/2001/XMLSchema";
public static String URI_ENCODING =
"http://schemas.xmlsoap.org/soap/encoding/";

/*this is the first participant*/
public static String participant3_p3_qnameService =
"participant3_p3_Service";
public static String participant3_p3_qnamePort =
"participant3_IF";

public Call participant3_p3_call;
public ServiceFactory participant3_p3_factory;
public Service participant3_p3_service;
public QName participant3_p3_port;

        JButton ButtonR_Election_0 = new
JButton("R_Election_0" );
JButton ButtonS_Election2_0 = new
JButton("S_Election2_0" );
JButton ButtonR_Election_1 = new
JButton("R_Election_1" );
JButton ButtonS_Election2_1 = new
JButton("S_Election2_1" );
JButton ButtonR_Election_2 = new
JButton("R_Election_2" );
JButton ButtonS_Elected_2 = new
JButton("S_Elected_2" );
JButton ButtonR_Election_3 = new
JButton("R_Election_3" );
JButton ButtonS_Election_3 = new
JButton("S_Election_3" );
JButton ButtonR_Elected_2 = new
JButton("R_Elected_2" );
JButton ButtonR_Elected_3 = new
JButton("R_Elected_3" );
JButton ButtonS_ED_3 = new JButton("S_ED_3" );


        /*the variables for Process*/
        static myGUI traceTable;
public String state=" ";
private int noOfevents=0;

//private Message message;

public participant2(String name) {
this.name =name;
          traceTable=new
myGUI(name,"participant2");
          state = "initial";
}


static class myGUI extends JFrame{
String[] headerStr = {"No.","Event", "Type", "Before
state", "After state", "Channel", "Value"};
DefaultTableModel dm = new
DefaultTableModel(headerStr, 60);
JTable table = new JTable(dm);
JPanel center=new JPanel();

JLabel instanceLabel;
JTextField instanceField;

JLabel eventsLabel;
JLabel processLabel;
JTextField processField;

myGUI(String a, String b){
setTitle("cycle");
setLocation(200,200);
setSize(30,30);

JPanel top =new JPanel();
top.setBackground(Color.gray);
instanceLabel= new JLabel("Instance");
top.add(instanceLabel);

instanceField=new JTextField(a,15);
Font g =new Font("Roman",Font.PLAIN,12);
top.setFont(g);
top.add(instanceField);

processLabel= new JLabel("Process");
top.add(processLabel);

processField=new JTextField(b,15);
Font h =new Font("Roman",Font.ITALIC,12);
top.setFont(h);
top.add(processField);

getContentPane().add(top, BorderLayout.NORTH);

JPanel middle =new JPanel();
middle.setBackground(Color.green);
eventsLabel= new JLabel("Possible event(s):");
middle.add(eventsLabel);
getContentPane().add(middle,
BorderLayout.WEST);

center.setBackground(Color.gray);
getContentPane().add(center,
BorderLayout.CENTER);

JPanel record =new JPanel();
table.setPreferredScrollableViewportSize(new
Dimension(200, 150));
getContentPane().add(new JScrollPane(table),
BorderLayout.SOUTH);

pack();
```

```
setVisible(true);

}

}


private void send(String messageName,String
currentName,String participantName,String state)
{
try {
        currentName = name;
participant3_p3_factory =
ServiceFactory.newInstance();
participant3_p3_service =
(Service)participant3_p3_factory.createService(ne
w QName(participant3_p3_qnameService));
participant3_p3_port = new
QName(participant3_p3_qnamePort);

participant3_p3_call =
participant3_p3_service.createCall(participant3_p3
_port);
participant3_p3_call.setTargetEndpointAddress("htt
p://localhost:8080/cycle3-jaxrpc/participant3?wsdl");
participant3_p3_call.setProperty(Call.SOAPACTIO
N_USE_PROPERTY, new Boolean(true));
participant3_p3_call.setProperty(Call.SOAPACTIO
N_URI_PROPERTY, "");
participant3_p3_call.setProperty(ENCODING_STY
LE_PROPERTY, URI_ENCODING);

participant3_p3_call.addParameter("String_1", new
QName(NS_XSD, "string"), ParameterMode.IN);
participant3_p3_call.addParameter("String_2", new
QName(NS_XSD, "string"), ParameterMode.IN);
participant3_p3_call.addParameter("String_3", new
QName(NS_XSD, "string"), ParameterMode.IN);
participant3_p3_call.addParameter("String_4", new
QName(NS_XSD, "string"), ParameterMode.IN);

participant3_p3_call.setReturnType(null);
participant3_p3_call.setOperationName(new
QName(BODY_NAMESPACE_VALUE,
"createMessage"));


Object [] params =
{ messageName,currentName,participantName,stat
e };
participant3_p3_call.invokeOneWay(params);



} catch (Exception ex) {
ex.printStackTrace();
}
}
public void createMessage(String type, String writer,
String reader, String channel)
{
if(type==" XXXX_XXXX " && state==null){ }
else if(type.equals("V0") && state.equals("initial"))
R_Election_0(writer,type,"R_Election_0");
else if(type.equals("V1") && state.equals("initial"))
R_Election_1(writer,type,"R_Election_1");
else if(type.equals("V2") && state.equals("initial"))
R_Election_2(writer,type,"R_Election_2");
else if(type.equals("V3") && state.equals("initial"))
R_Election_3(writer,type,"R_Election_3");

else if(type.equals("V2") && state.equals("initial"))
R_Elected_2(writer,type,"R_Elected_2");
else if(type.equals("V3") && state.equals("initial"))
R_Elected_3(writer,type,"R_Elected_3");
else
/*it will be a warnning window exposed*/
        System.out.println("sdsdfsadfas");
}

public void REN0(){
if(state=="REN0")
S_Election2_0();
}

public void REN1(){
if(state=="REN1")
S_Election2_1();
}

public void REN2(){
if(state=="REN2")
S_Elected_2();
}

public void REN3(){
if(state=="REN3")
S_Election_3();
}

public void RED3(){
if(state=="RED3")
S_ED_3();
}

public void R_Election_0(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_0);
ButtonR_Election_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN0");
displayTrace( "R_Election_0", "Read", "initial",
"REN0", "election", "V0");
REN0();
ButtonR_Election_0.setVisible(false);
}
});
}

public void R_Election_1(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_1);
ButtonR_Election_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN1");
displayTrace( "R_Election_1", "Read", "initial",
"REN1", "election", "V1");
REN1();
ButtonR_Election_1.setVisible(false);
}
});
}

public void R_Election_2(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_2);
ButtonR_Election_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN2");
displayTrace( "R_Election_2", "Read", "initial",
"REN2", "election", "V2");
REN2();
ButtonR_Election_2.setVisible(false);
```

```
}
});
}

public void R_Election_3(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_3);
ButtonR_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN3");
displayTrace( "R_Election_3", "Read", "initial",
"REN3", "election", "V3");
REN3();
ButtonR_Election_3.setVisible(false);
}
});
}

public void R_Elected_2(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_2);
ButtonR_Elected_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("IMBoss");
displayTrace( "R_Elected_2", "Read", "initial",
"IMBoss", "elected", "V2");
ButtonR_Elected_2.setVisible(false);
}
});
}

public void R_Elected_3(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_3);
ButtonR_Elected_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("RED3");
displayTrace( "R_Elected_3", "Read", "initial",
"RED3", "elected", "V3");
RED3();
ButtonR_Elected_3.setVisible(false);
}
});
}

public void S_Election2_1(){
traceTable.center.add(ButtonS_Election2_1);
ButtonS_Election2_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
send("V2",name,"p3","PassOn_election");
transformState("initial");
}
catch(Exception f){System.out.println(name + " :
S_Election2_1- send error");}

displayTrace( "S_Election2_1", "Create", "REN1",
"initial=", "PassOn_election", "V2");
ButtonS_Election2_1.setVisible(false);
}

});

}

public void S_ED_3(){
traceTable.center.add(ButtonS_ED_3);
ButtonS_ED_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
        send("V3",name,"p3","PassOn_elected");
transformState("3_Boss");
}
catch(Exception f){System.out.println(name + " :
S_ED_3- send error");}

displayTrace( "S_ED_3", "Create", "RED3",
"3_Boss", "PassOn_elected", "V3");
ButtonS_ED_3.setVisible(false);
}

});

}

public void S_Election2_0(){
System.out.println(name+" : S_Election2_0");

traceTable.center.add(ButtonS_Election2_0);
ButtonS_Election2_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
        send("V2",name,"p3","election");
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p2-
send error");}
displayTrace( "S_Election2_0", "Write", "REN0",
"initial=", "PassOn_election", "V2");
ButtonS_Election2_0.setVisible(false);
}

});


}

public void S_Elected_2(){
System.out.println(name+" : S_Elected_2");

traceTable.center.add(ButtonS_Elected_2);
ButtonS_Elected_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
        send("V2",name,"p3","elected");
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p2-
send error");}
displayTrace( "S_Elected_2", "Write", "REN2",
"initial=", "PassOn_elected", "V2");
ButtonS_Elected_2.setVisible(false);
}

});


}

public void S_Election_3(){
System.out.println(name+" : S_Election_3");

traceTable.center.add(ButtonS_Election_3);
ButtonS_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
send("V3",name,"p3","election");
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p2-
send error");}
displayTrace( "S_Election_3", "Write", "REN3",
"initial=", "PassOn_election", "V3");
ButtonS_Election_3.setVisible(false);
}

});


}



public String getname(){
return name;
}

public void displayTrace(String ev, String ty, String
be, String af, String ch, String va){
traceTable.table.setValueAt((new
Integer(noOfevents)).toString(), noOfevents,0);
traceTable.table.setValueAt(ev, noOfevents,1);
traceTable.table.setValueAt(ty, noOfevents,2);
traceTable.table.setValueAt(be, noOfevents,3);
traceTable.table.setValueAt(af, noOfevents,4);
traceTable.table.setValueAt(ch, noOfevents,5);
traceTable.table.setValueAt(va, noOfevents,6);
noOfevents++;
}

public void transformState(String s){
        state=s;
        System.out.println(name +" : " + state);
      }



public static void main(String[] args) throws
Exception
{
participant2 p2= new participant2 ("p2");
}
}
```

E.5 Participant3 Application

E.5.1 Interface

```
package participant3;

import java.rmi.Remote;
import java.rmi.RemoteException;


public interface participant3_IF extends Remote
{
public void createMessage(String type, String writer,
String reader, String channel) throws
RemoteException;
}
```

E.5.2 Implementation

```
package participant3;

import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.JAXRPCException;
import javax.xml.namespace.QName;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.ParameterMode;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.text.*;
import javax.swing.table.*;

public class participant3 extends JFrame
implements participant3_IF
{
public String name ="p3";
public static String BODY_NAMESPACE_VALUE =
"urn:Foo";
public static String
ENCODING_STYLE_PROPERTY =
"Javax.xml.rpc.encodingstyle.namespace.uri";
public static String NS_XSD =
"http://www.w3.org/2001/XMLSchema";
public static String URI_ENCODING =
"http://schemas.xmlsoap.org/soap/encoding/";

/*this is the first participant*/
public static String participant0_p0_qnameService =
"participant0_p0_Service";
public static String participant0_p0_qnamePort =
"participant0_IF";

public Call participant0_p0_call;
public ServiceFactory participant0_p0_factory;
public Service participant0_p0_service;
public QName participant0_p0_port;


        /*the variables for Process*/
        static myGUI traceTable;
```

```
public String state=" ";
private int noOfevents=0;
//private Message message;

JButton ButtonR_Election_0 = new
JButton("R_Election_0" );
JButton ButtonS_Election3_0 = new
JButton("S_Election3_0" );
JButton ButtonR_Election_1 = new
JButton("R_Election_1" );
JButton ButtonS_Election3_1 = new
JButton("S_Election3_1" );
JButton ButtonR_Election_2 = new
JButton("R_Election_2" );
JButton ButtonS_Election3_2 = new
JButton("S_Election3_2" );
JButton ButtonR_Election_3 = new
JButton("R_Election_3" );
JButton ButtonS_Elected_3 = new
JButton("S_Elected_3" );
JButton ButtonR_Elected_3 = new
JButton("R_Elected_3" );


public participant3(String name) {
this.name =name;
        traceTable=new
myGUI(name,"participant3");
        state="initial";
}


static class myGUI extends JFrame{
String[] headerStr = {"No.","Event", "Type", "Before
state", "After state", "Channel", "Value"};
DefaultTableModel dm = new
DefaultTableModel(headerStr, 60);
JTable table = new JTable(dm);
JPanel center=new JPanel();

JLabel instanceLabel;
JTextField instanceField;

JLabel eventsLabel;
JLabel processLabel;
JTextField processField;

myGUI(String a, String b){
setTitle("cycle");
setLocation(200,200);
setSize(30,30);

JPanel top =new JPanel();
top.setBackground(Color.gray);
instanceLabel= new JLabel("Instance");
top.add(instanceLabel);

instanceField=new JTextField(a,15);
Font g =new Font("Roman",Font.PLAIN,12);
top.setFont(g);
top.add(instanceField);

processLabel= new JLabel("Process");
top.add(processLabel);

processField=new JTextField(b,15);
Font h =new Font("Roman",Font.ITALIC,12);
top.setFont(h);
top.add(processField);

getContentPane().add(top, BorderLayout.NORTH);

JPanel middle =new JPanel();
middle.setBackground(Color.green);
eventsLabel= new JLabel("Possible event(s):");
middle.add(eventsLabel);
getContentPane().add(middle,
BorderLayout.WEST);

center.setBackground(Color.gray);
getContentPane().add(center,
BorderLayout.CENTER);

JPanel record =new JPanel();
table.setPreferredScrollableViewportSize(new
Dimension(200, 150));
getContentPane().add(new JScrollPane(table),
BorderLayout.SOUTH);

pack();
setVisible(true);

}

}


private void send(String messageName,String
currentName,String participantName,String state)
{
try {
        currentName = name;
participant0_p0_factory =
ServiceFactory.newInstance();
participant0_p0_service =
(Service)participant0_p0_factory.createService(ne
w QName(participant0_p0_qnameService));
participant0_p0_port = new
QName(participant0_p0_qnamePort);

participant0_p0_call =
participant0_p0_service.createCall(participant0_p0
_port);
participant0_p0_call.setTargetEndpointAddress("htt
p://localhost:8080/cycle0-jaxrpc/participant0?wsdl");
participant0_p0_call.setProperty(Call.SOAPACTIO
N_USE_PROPERTY, new Boolean(true));
participant0_p0_call.setProperty(Call.SOAPACTIO
N_URI_PROPERTY, "");
participant0_p0_call.setProperty(ENCODING_STY
LE_PROPERTY, URI_ENCODING);

participant0_p0_call.addParameter("String_1", new
QName(NS_XSD, "string"), ParameterMode.IN);
participant0_p0_call.addParameter("String_2", new
QName(NS_XSD, "string"), ParameterMode.IN);
participant0_p0_call.addParameter("String_3", new
QName(NS_XSD, "string"), ParameterMode.IN);
participant0_p0_call.addParameter("String_4", new
QName(NS_XSD, "string"), ParameterMode.IN);

participant0_p0_call.setReturnType(null);
participant0_p0_call.setOperationName(new
QName(BODY_NAMESPACE_VALUE,
"createMessage"));


Object [] params =
{ messageName,currentName,participantName,stat
e };
participant0_p0_call.invokeOneWay(params);



} catch (Exception ex) {
ex.printStackTrace();
```

```
}
}

public void createMessage(String type, String writer,
String reader, String channel) {
if(type==" XXXX_XXXX " && state==null){ }
else if(type.equals("V0") && state.equals("initial"))
R_Election_0(writer,type,"R_Election_0");
else if(type.equals("V1") && state.equals("initial"))
R_Election_1(writer,type,"R_Election_1");
else if(type.equals("V2") && state.equals("initial"))
R_Election_2(writer,type,"R_Election_2");
else if(type.equals("V3") && state.equals("initial"))
R_Election_3(writer,type,"R_Election_3");

else if(type.equals("V3") && state.equals("initial"))
R_Elected_3(writer,type,"R_Elected_3");
else
/*it will be a warnning window exposed*/
        System.out.println("sdsdfsadfas");
}


public void REN0(){
if(state=="REN0")
S_Election3_0();
}

public void REN1(){
if(state=="REN1")
S_Election3_1();
}

public void REN2(){
if(state=="REN2")
S_Election3_2();
}

public void REN3(){
if(state=="REN3")
S_Elected_3();
}

public void R_Election_0(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_0);
ButtonR_Election_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN0");
displayTrace( "R_Election_0", "Read", "initial",
"REN0", "election", "V0");
REN0();
ButtonR_Election_0.setVisible(false);
}
});
}

public void R_Election_1(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_1);
ButtonR_Election_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN1");
displayTrace( "R_Election_1", "Read", "initial",
"REN1", "election", "V1");
REN1();
ButtonR_Election_1.setVisible(false);
}
});
}

public void R_Election_2(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_2);
ButtonR_Election_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN2");
displayTrace( "R_Election_2", "Read", "initial",
"REN2", "election", "V2");
REN2();
ButtonR_Election_2.setVisible(false);
}
});
}

public void R_Election_3(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Election_3);
ButtonR_Election_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("REN3");
displayTrace( "R_Election_3", "Read", "initial",
"REN3", "election", "V3");
REN3();
ButtonR_Election_3.setVisible(false);
}
});
}

public void R_Elected_3(String from, String
message, String current_state){
traceTable.center.add(ButtonR_Elected_3);
ButtonR_Elected_3.addActionListener(new
ActionListener(){
public void actionPerformed(ActionEvent e){
transformState("IMBoss");
displayTrace( "R_Elected_3", "Read", "initial",
"IMBoss", "elected", "V3");
ButtonR_Elected_3.setVisible(false);
}
});
}

public void S_Election3_2(){
traceTable.center.add(ButtonS_Election3_2);
ButtonS_Election3_2.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
send("V3", name ,"p0","PassOn_election");
transformState("initial");
}
catch(Exception f){System.out.println(name + " :
S_Election3_2- send error");}

displayTrace( "S_Election3_2", "Create", "REN2",
"initial=", "PassOn_election", "V3");
ButtonS_Election3_2.setVisible(false);
}

});

}

public void S_Elected_3(){
traceTable.center.add(ButtonS_Elected_3);
ButtonS_Elected_3.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
        send("V3",name,"p0","PassOn_elected");
transformState("initial");
}
catch(Exception f){System.out.println(name + " :
S_Elected_3- send error");}

displayTrace( "S_Elected_3", "Create", "REN3",
"initial=", "PassOn_elected", "V3");
ButtonS_Elected_3.setVisible(false);
}

});

}

public void S_Election3_0(){
System.out.println(name+" : S_Election3_0");

traceTable.center.add(ButtonS_Election3_0);
ButtonS_Election3_0.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
send("V3",name,"p0","election");
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p3-
send error");}
displayTrace( "S_Election3_0", "Write", "REN0",
"initial=", "PassOn_election", "V3");
ButtonS_Election3_0.setVisible(false);
}

});


}

public void S_Election3_1(){
System.out.println(name+" : S_Election3_1");

traceTable.center.add(ButtonS_Election3_1);
ButtonS_Election3_1.addActionListener( new
ActionListener(){
public void actionPerformed(ActionEvent e){
try{
        send("V3",name,"p0","election");
transformState("initial");
}
catch(Exception f){System.out.println(name + " : p3-
send error");}
displayTrace( "S_Election3_1", "Write", "REN1",
"initial=", "PassOn_election", "V3");
ButtonS_Election3_1.setVisible(false);
}

});


}


public String getname(){
return name;
}

public void displayTrace(String ev, String ty, String
be, String af, String ch, String va){
traceTable.table.setValueAt((new
Integer(noOfevents)).toString(), noOfevents,0);
traceTable.table.setValueAt(ev, noOfevents,1);
traceTable.table.setValueAt(ty, noOfevents,2);
traceTable.table.setValueAt(be, noOfevents,3);
traceTable.table.setValueAt(af, noOfevents,4);
traceTable.table.setValueAt(ch, noOfevents,5);
traceTable.table.setValueAt(va, noOfevents,6);
noOfevents++;
}

public void transformState(String s){
          state=s;
          System.out.println(name +" : " + state);
        }



public static void main(String[] args) throws
Exception
{
participant3 p3= new participant3 ();
}
}
```

References

Agarwal, R., G. Bruno and M. Torchiano (2001). Model based Web Applications. In Proceeding of 4th International Conference on Information Technology (CIT), Gopalpur-on-Sea, India.

Ahmed, K., S. Ancha and A. Cioroianu (2001). "Professional Java XML". Birmingham, Wrox Press Ltd. **186100401X**:(1st edition).

Alitalia (2008). "Alitalia.com". from http://www.alitalia.com/.

Andrews, G. (1991). "Paradigms for Process Interaction in Distributed Programs". ACM Computing Surveys **23**(1): 49-90.

Apache (2002). "Apache Tomcat". from http://tomcat.apache.org/.

Apache (2007). "WSIF: Web Service Invocation Framework". from http://ws.apache.org/wsif/.

Apfelbaum, L. and J. Doyle (1997). Model Based Testing. In Proceeding of 10th International Software Quality Week Conference, San Francisco, California, USA.

Artho, C. and A. Biere (2001). Applying Static Analysis to Large-scale, Multi-threaded Java Programs. In Proceeding of 13th Australian Software

Engineering Conference (ASWEC'01), Canberra, Australia. IEEE Computer Society.

Banks, A., J. Challenger, P. Clarke, D. Davis, R. P. King, K. Witting, A. Donoho, T. Holloway, J. Ibbotson and S. Todd (2002). Specification: HTTPR Specification. from ftp://www6.software.ibm.com/software/developer/library/ws-httprspec.pdf.

Basin, D., S. Friedrich, J. Posegga and H. Vogt (1999). Java ByteCode Verification by Model Checking System Abstract. "Computer Aided Verification". Heidelberg, Springer Berlin. **978-3-540-66202-0: 1633/1999:** 681.

BEA (2003). "BEA WebLogic Server". from http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/weblogic/server.

BEA, IBM, Microsoft, SAP-AG and Siebel-Systems (2003). "Business Process Execution Language for Web Services Specification". from http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf.

Beizer, B. (1995). "Black-Box Testing: Techniques for Functional Testing of Software and Systems", John Wiley & Sons. **0471120944**.

Beydeba, S. and V. Gruhn (2002). Class Specification Implementation Graphs and their Application in Regression Testing. In Proceeding of 26th Annual International Computer Software and Applications Conference (COMPSAC 2002), Oxford, England, UK.

Boyapati, C., S. Khurshid and D. Marinov (2002). Korat: Automated Testing Based on Java Predicates. In Proceeding of 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), Rome, Italy. ACM.

Callahan, J., F. Schneider and S. Easterbrook (1996). Automated Software Testing Using Model-Checking. In Proceeding of 1996 SPIN Workshop, Rutgers, USA.

Chan, F., T. Y. Chen and T. H. Tse (1997). "On the Effectiveness of Test Case Allocation Schemes in Partition Testing". Information and Software Technology **39**(10): 719-726.

Chang, E. and R. Roberts (1979). "An Important Algorithm for Decentralized Extrema-finding in Circular Configurations of Processors". IEEE Transactions on Computers **22**(5): 281-283.

Chen, H. Y., T. Y. Tse and Y. T. Deng (2000). "ROCS: An Object-oriented Class-level testing System based on the Relevant Observable ContextS technique". Information and Software Technology **42**(10): 677-686.

Chen, W. K., T. Y. Chen and T. H. Tse (2002). An Overview of Integration testing techniques for Object-Oriented Programs. In Proceeding of 2nd ACIS Annual International Conference on Computer and Information Science (ICIS 2002), Mt. Pleasure, Michigan, USA. International Association for Computer and Information Science.

Cheon, Y. and G. Leavens (2002). A Runtime Assertion Checker for the Java Modelling Language (JML). In Proceeding of International Conference on Software Engineering Research and Practice (SERP'02), Las Vegas, Nevada, USA. CSREA Press.

Clarke, E., O. Grumberg and D. Peled (2000). "Model Checking", The MIT Press. **0-262-03270-8**.

Colouris, G., J. Dollimore and T. Kindberg (2001). "Distributed Systems Concepts and Design", Addison-Wesley. **0201619180**:(3rd edition).

Corbett, J., M. Dwyer, J. Hatcliff, S. Laubach, C. S. Pasareanu and R. H. Zheng (2000). Bandera: Extracting Finite-state Models from Java Source Code. In Proceeding of 2000 International Conference on Software Engineering, Limerick, Ireland.

Damelio, R. (1996). "Basics of Process Mapping", Productivity Press. **0527763160**:(1st edition).

Davenport, T. (1992). "Process Innovation: Reengineering Work through Information Technology", Boston: Harvard Business School Press. **0875843662**.

Demartini, C., R. Iosif and R. Sisto (1999). "A Deadlock Detection Tool for Concurrent Java Programs". Software: Practice and Experience **29**(7): 577-603.

Dijkstra, E. W. (1965). "Co-Operating Sequential Processes". **Technical Report EWD-123**: 43-112. Academic Press, New York, USA.

Eclipse (2007). "Ecipse IDE for Java Developers". from www.eclipse.org.

Esser, M. and P. Struss (2006). Fault-model-based Test Generation for Embedded Software. In Proceeding of 20th International Joint conference on Artificial Intelligence IJCAI-07, Hyderabad, India.

Expedia (2008). "Expedia.co.uk". from http://www.expedia.co.uk/Default.aspx.

Frankel, D. (2003). "Model Driven Architecture: Applying MDA to Enterprise Computing", Wiley Publishing, Inc. **0471319201**.

Garcia-Molina, H. (1982). "Elections in Distributed Computer Systems". IEEE Transactions on Computer **31**(1): 48-59.

Gargantini, A. and C. Heitmeyer (1999). Using Model Checking to Generate Tests from Requirements Specifications. In Proceeding of Joint 7th European Software Engineering Conference and 7th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Toulouse, France. Springer-Verlag.

Godefroid, P. (1997). "VeriSoft: A tool for the automatic analysis of concurrent reactive software ". Computer Aided Verification: 476-479.

Goschl, S. and H. Sneed, Eds. (2002). A Case Study of Testing a Distributed Internet-System. "Software Testing, Verification and Reliability".77-92.

Gottschalk, K. (2000). Web Services Architecture Overview: The next stage of evolution for e-business. from http://www.ibm.com/developerworks/web/library/w-ovr/.

Grønmo, R., D. Skogan, I. Solheim and J. Oldevik (2004). Model-Driven Web Services Development. In Proceeding of 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'04), Taipei, China. IEEE Computer Society.

Halvorson, M. (1998). "Microsoft Visual Basic Professional 6.0 Step by Step", Microsoft Press. **1-57231-809-0**.

Hammer, M. and J. Champy (2001). "Reengineering the Corporation", Nicholas Prealey Publishing Ltd. **1857880978**:(3rd Revised edition).

Hansen, B. (1975). "The Programming Language Concurrent Pascal". IEEE Transactions on Software Engineering **1**(2): 199-207.

Harold, E. R. (1997). "Java Network Programming", O'Reilly & Associates, Inc. **0596007213**:(3rd edition).

Hausmann, J. H., R. Heckel and M. Lohmann (2005). "Model-Based Development of Web Services Descriptions Enabling a Precise Matching Concept". Web Services Research **2**(2): 67-84.

Havelund, K. (1999). Java PathFinder A Translator from Java to PROMELA. "Theoretical and Practical Aspects of SPIN Model Checking". Berlin, Springer **LNCS 1680:** 152.

Havelund, K. (2000). Using Runtime Analysis to Guide Model Checking of Java Programs. In Proceeding of 7th SPIN Workshop, California, USA.

Havelund, K. and T. Pressburger (2000). "Model Checking Java Programs Using Java PathFinder". International Journal on Software Tools for Technology Transfer **2**(4).

HCI (2008). "Flow Charting". from http://www.hci.com.au/hcisite2/toolkit/flowchar.htm.

Henderson, P., Y. Howard and R. Walters (2001). "A Tool for Evaluation of the Software Development Process". Journal of Systems and Software **59**(3): 355-362.

Henderson, P. and R. Walters (1999). Component Based systems as an aid to Design Validation. In Proceeding of 14th IEEE Conference on Automated Software Engineering, ASE'99. IEEE Computer Society Press.

Hoare, C. (1985). "Communication Sequential Processes", Prentice Hall. **0-13-153271-5**.

Holzmann, G. J. (1990). "Design and Validation of Computer Protocols". Englewood Cliffs, NJ, Prentice Hall. **0135399254**.

Holzmann, G. J. (1997). "The Model Checker SPIN". IEEE Transactions on Software Engineering **23**(5).

Holzmann, G. J. (2003). "The Spin Model Checker: Primer and Reference Manual", Addison-Wesley. **0321228626**.

Holzmann, G. J. and R. Joshi (2004). Model-Driven Software Verification. "Model Checking Software". Berlin, Springer **LNCS 2989:** 76-91.

IEEE (1987). "IEEE Standard 610-1987 IEEE Standard for Computer Applications Terminology", IEEE Computer Society. **9789998267978**.

IEEE (1998). "IEEE 829-1998 Standard for Software Test Documentation". **9780738157477**.

JBoss (2003). "JBoss Enterprise Application Platform". from http://www.jboss.com/products/platforms/application.

Kalian, A. and A. Watson (2003). "Modelling the building cladding attainment process". Business Process Managent Journal **10**.

Kalian, A., A. Watson, E. Agbasi, C. Anumba and A. Gibb (2004). "Modelling the Building Cladding Attainment Process". Business Process Managent Journal **10**(6): 712-723.

Kansomkeat, S. and W. Rivepiboon (2003). Automated-generating test case using UML statechart diagrams. In Proceeding of 2003 annual research conference of the South African Institute of Computer Scientists and Information Technologists on Enablement through technology. SAICSIT.

Kaveh, N. and W. Emmerich (2001). Deadlock Detection in Distributed Object Systems. In Proceeding of Joint 8th European Software Engineering Conference (ESEC) and the 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9), Vienna, Austria. ACM Press.

Kleppe, A., J. Warmer and W. Bast (2003). "MDA Explained: The Model Driven Architecture - Practice and Promise", Addison-Wesley. **032119442X**.

Krichen, M. and S. Tripakis (2004). Black-box Conformance Testing for Real-time Systems. In Proceeding of 11th International SPIN Workshop on Model Checking of Software (SPIN'04), Barcelona, Spain. Springer Verlag.

Lerda, F., N. Sinha and M. Theobald (2003). "Symbolic Model Checking of Software". Electronic Notes in Theoretical Computer Science **89**(3).

Leuschel, M. and M. J. Butler (2003). ProB: A Model Checker for B. "FME 2003: Formal Methods". Berlin, Springer. **978-3-540-40828-4: LNCS 2805:** 855-874.

Long, B. and P. Strooper (2001). A Case Study in Testing Distributed Systems. In Proceeding of Third International Symposium on Distributed Objects and Applications, Rome, Italy. IEEE Computer Society.

Lynch, N. A. (1996). "Distributed Algorithms", Morgan Kaufmann. **1558603484**:(1st edition).

Magee, J. and J. Kramer (1999). "Concurrency: State Models and Java Programs", John Wiley & Sons. **0471987107**:(1st edition).

McMillan, K. (2000). "The SMV System". from http://www.cs.cmu.edu/~modelcheck/smv/smvmanual.ps.

Microsoft (1996). "DCOM". from http://msdn.microsoft.com/en-us/library/ms809340.aspx.

Microsoft (2005). "Visual Studio 2005". from http://msdn.microsoft.com/en-us/library/ms950416.aspx.

Microsoft (2008). ".NET Framework". from http://msdn.microsoft.com/en-us/netframework/default.aspx.

Milner, R. (1989). "Communication and Concurrency", Prentice Hall. **0131149849**.

Milner, R. (1993). The Polyadic Pi-calculus: a tutorial. "Logic and Algebra of Specification", Springer-Verlag. **0387558136:** 203-246.

Monson-Haefel, R. and D. Chappell (2001). "Java Message Service", O'Reilly & Associates, Inc. **284177208X**.

Mukhar, K., J. Weaver, R. Phillips and J. Crume (2003). "Beginning J2EE 1.4: From Novice to Professional", Wrox Press Ltd. **1-86100-833-3**.

Nagappan, R., R. Skoczylas and R. Sriganesh (2003). "Developing Java Web Service: Architecting and Developing Secure Web service Using Java", Wiley Publishing, Inc. **0-471-23640-3**.

Omega (2008). "Omegatravel.net". from http://www.omegatravel.net/.

OMG (1993). "The Common Object Request Broker: Architecture and Specification", QED Publish Co. **0471587923**.

OMG (2007). "Model Driven Architecture". from http://www.omg.org/mda.

Ozu, N., R. Anderson and W. A. Team (2001). "Professional XML (Programmer to Programmer)", Wrox Press Inc. **1861005059**:(2nd edition).

Parasoft (2003). "Parasoft JTest". from http://www.parasoft.com/jsp/products/home.jsp?product=Jtest.

Patton, R. (2000). "Software Testing". Indianapolis, USA, SAMS. **0672319837**.

Phalp, K., P. Henderson, G. Abeysinghe and R. Walters (1998). "RolEnact - Role Based Enactable Models of Business Processes". Information and Software Technology **40**(3): 123-133.

Ruggiero, R. (2003). JMS/Web Services/WS-Reliability. from http://www.creativematch.co.uk/viewnews/?88460.

Savage, S., G. Nelson, P. Sobalvarro and T. Anderson (1997). Eraser: A Dynamic Data Race Detector for Multi-Threaded Programs. In Proceeding of 16th ACM Symposium on Operating System Principles, St. Malo, France.

Schmit, B. A. and S. Dustdar (2005). Model-driven Development of Web Service Transactions. In Proceeding of Second GI-Workshop XML for Business Process Management, Karlsruhe, Germany.

Sneed, H. (1998). Automated Test Case Specification for Integration Testing of Distributed Objects. In Proceeding of EuroStar98, München, Germany.

Sommerville, I. (2001). "Software Engineering", Addison-Wesley. **0201398151**:(6th edition).

Stevens, P. and R. Pooley (2000). "Using UML Software Engineering with Objects and Components", Addison-Wesley. **0201648601**:(Revised edition).

Stoller, S. D. and Y. A. Liu (2001). Transformations for Model Checking Distributed Java Programs. In Proceeding of 8th International SPIN Workshop on Model Checking of Software, Toronto, Ontario. Springer-Verlag.

SUN (2003a). "Java Remote Invocation- Distributed Computing for Java (White paper)". from http://java.sun.com/marketing/collateral/javarmi.html.

SUN (2003b). "RMI: Remote Method Invocation". from http://java.sun.com/products/jdk/rmi/index.html.

SUN (2003c). "Sun's Java Tutorials". from http://java.sun.com/docs/books/tutorial/.

SUN (2007). "Java API for XML Messaging (JAXM)". from http://java.sun.com/webservices/jaxm/index.jsp.

Thompson, S. (2000). A Survey on Model Checking Java Programs. from www.cs.toronto.edu/~chechik/courses99/csc2108/projects/5.ps. Technical Report CSRG-407.

Tsai, W. T., L. Yu and A. Saimi (2003). "Scenario-based Object-Oriented Test Frameworks for Testing Distributed Systems". Distributed Computing Systems: 288-294.

Tufarolo, J., J. Ives and T. Hyon (1999). Automated Distributed System Testing: Application of an RTI Verification System. In Proceeding of 1999 Winter Simulation Conference.

Tufarolo, J., J. Nielsen, S. Symington, R. Weatherly, A. Wilson and T. Hyon (1998). Automated Distributed System Testing: designing an RTI Verification

System. In Proceeding of 31st conference on Winter simulation: Simulation - a bridge to the future, Phoenix, Arizona, USA.

Visser, W., K. Havelund, G. Brat and S. Park (2000). Model Checking Programs. In Proceeding of 15th IEEE International Conference on Automated Software Engineering (ASE'00), Grenoble, France.

W3C (2001). "WSDL: Web Service Definition Language". from http://www.w3.org/TR/wsdl.

Walters, R. (2002a). A Graphically Based Language for Communicating, Executing and Analysing Models of Software Systems. In Proceeding of 26th Annual Internation Computer Software and Applications Conference (COPSA 2002), Oxford, England.

Walters, R. (2002b). "A Graphically based Language for Constructing, Executing and Analysing Models of Software Systems", PhD thesis.

Walters, R. (2005). "Automating Checking of Models built using a Graphically Based Formal Modelling Language". Journal of Systems and Software **76**: 55-64.

Xing, G., M. R. Lyu and N. T. Shatin (2000). Testing, Reliable, and Interoperability issues in the CORBA Programming Paradigm. In Proceeding of 1999 Asia-Pacific Software Engineering Conference (APSEC'99), Takamatsu, Kagawa, Japan.

Yamaura, T. and A. Onoma (2002). Hypothesis Testing for Module Test in Software Development. In Proceeding of 26th Annual International Computer Software and Applications Conference (COMPSAC 2002), Oxford, England, UK.