Verification of Web Service Composition Using "SPIN"

Submitted by

Mohammad Mohibul Hasan
Abdur Razzaq Akhunji

Supervised by

Dr. Shamim Hasnat Ripon
Chairperson and Associate Professor
Department of Computer Science and Engineering

A project submitted in partial fulfillment for the degree of
Bachelor of Science in Computer Science and Engineering

In the

Faculty of Science and Engineering
Department of Computer Science and Engineering

September 2015

Verification of Web Service Composition Using "SPIN"

Supervised by

Dr. Shamim Hasnat Ripon
Chairperson and Associate Professor
Department of Computer Science and Engineering

Submitted by

Mohammad Mohibul Hasan
2011-1-60-007
Abdur Razzaq Akhunji
2011-1-60-013

A project submitted in partial fulfillment for the degree of
Bachelor of Science in Computer Science and Engineering

In the

Faculty of Science and Engineering
Department of Computer Science and Engineering

East West University

September 2015

Declaration by Candidates

We hereby declare that, the work presented in this project is to the best of our knowledge and belief, original, except as acknowledged in the text and that the material has not been submitted, either in whole or in part, for a degree at this or any other university.

_____

Mohammad Mohibul Hasan
(2011-1-60-007)

_____

Abdur Razzaq Akhunji
(2011-1-60-013)

Letter of Acceptance

The project entitled "**Web Service Composition Verification Using SPIN**" submitted by Mohammad Mohibul Hasan (ID: 2011-1-60-007) and Abdur Razzaq Akhunji (2011-1-60-013), to the department of Computer Science and Engineering, East West University, Dhaka, Bangladesh is accepted by the department in partial fulfillment of requirements for the Award of the Degree of Bachelor of Science in Computer Science and Engineering on September 2015

_____

Dr. Shamim Hasnat Ripon
SUPERVISOR and CHAIRPERSON
Department of Computer Science and Engineering
East West University,
Dhaka-1212, Bangladesh

Abstract

Web service composition and its verification is an important part to get an error free web service. In a web service there are huge business transaction occurs. Sometimes in business transaction, it needs to deal with faults that can arise in any stage of transaction. So always roll back mechanism is not possible to solve this problem. In order to achieve a common business goal, the protocol of interaction must be correct. In this project we have established a Broker system and encode this overall model into PROMELA (Model checking Language). For verify this model we also have used linear temporal logic and convert it to never claim property to verify this model accuracy. If any oversight happened then a compensation mechanism has been used to handle the situation.

Acknowledgement

Firstly, we would like to thank Almighty Allah for give us mental and psychical strength to complete this project. We are cordially grateful to our honorable supervisor **Dr. Shamim H. Ripon,** Chairperson and Associate Professor of Department of Computer Science and Engineering, East West University, Dhaka Bangladesh for providing us this opportunity to test our skills in the best possible manner. He enlightened, encouraged and provided us with ingenuity to transform our vision into reality. Without his help, it will not be possible to complete our project work in due time.

TABLE OF CONTENTS

CHAPTER III

CHAPTER IV

LIST OF FIGURES

Table

Appendix

Chapter I

Introduction

1.1 Introduction

Web service plays an important role in business management in these days. Many business companies run their company by using web service. It makes the life easier for the business people also the others. It is important to check the performance of a web service system when it starts. Performance means, whether every process inside the service is working properly or there are any transactions in that web services has fault, all these issues become important to maintain a good web service. When multiple partners involve in a web service then it became composite web service or we can say web service composition. Web service composition's verification is an important part to get a good web service. In a web service there are business transaction occurs. Sometimes in business transaction, it needs to deal with faults that can arise in any stage of transaction. In usual database transaction, a rollback mechanism is used to handle faults in order to provide atomicity to a transaction. It takes lot of times to fulfill the transaction if this kind of problem happens; it calls Long Run Transaction (LRT). Sometimes it is not possible to roll back the total transaction. It is both difficult and critical while multiple partners are involved in their interactive nature. LRT also not able to handle in which point the fault has occurred. I.e. sent message cannot be unsent in this case separate mechanism can be used to handle this. Possible solution of this problem may be that system designer can provide a mechanism to compensate the action that cannot be undone automatically [1]. SPIN is a powerful model checker that verifies the correctness of distributed communication models. This project report represents a SPIN based formal verification approach of a web service composition.

1.2 Motivation

In web service composition existing techniques for creating business procedure are not good enough to handle errors while each service transfer data in between cross organizational opponents. Since all this methods are not flexible enough to handle the technical transections' errors that web services introduce. For solving this problem compensation mechanism has a good impact on web service composition. In our project we have used this mechanism on our web service composition and then we have tried to verify the overall model.

1.3 Objective

Web service composition verification we have some basic objectives have to show which are

- Verification of web service choreography.
- Compensation mechanism of our service.
- Verification of the model by checking Linear Temporal Logic (LTL) property

1.4 Contribution

Our contribution of this project are given below:

- First we set a model. We have choose a Buyer-Broker system as our model (web service) and try to represent it in PROMELA language by using SPIN tool.
- After that, we have tried to check whether this model perform correctly or not.
- Further added, we have used Linear Temporal logic (LTL) so that we can verify our implementation.
- What will happen if the model works perfectly or there will be any error? As this situation we have tried to implement compensation mechanism for solving this problematic situation.

1.5 Outline

- Basically chapter I is an introductory part.
- In chapter II there is a brief explanation of "SPIN" tool which specify how to work with SPIN. There are also have some description about PROMELA. After that, we discussed a general description of linear temporal logic property and web service composition.
- In chapter II there is an overview of Broker System in service composition model that how it works with message sequence chart.
- In chapter IV we have encoded our web service (Broker System) into PROMELA.
- In chapter V simulation and verification of the broker system shows its correctness using LTL property.
- In chapter VI conclude it with sort summary and future work.

Chapter II

Background

2.1 SPIN

SPIN is a temporal logic checker works on automata based property. SPIN stands for Simple PROMELA Interpreter. It is a tool for analyzing the model system specified in PROMELA. Therefore, it uses PROMELA as a modeling language. SPIN can use the entire check of high-level models of concurrent system. It also executes parallel processes. It is designed, to describe systems of asynchronous communication. SPIN is design for various distributed system like data communication protocol, multithread code, client server protocol, network application etc. We also can check some different properties of state by the SPIN tool. It can use in two basic modes: as a simulator and as a verifier. In simulation mode, SPIN uses to get a quick impression behavior type, which is captured in the system model. Verifier means correctness of detail model of the system under validation. Spin can be used as a LTL model checking system **[2].** It supports correctness requirements expressible in linear time temporal logic. It can also be used to check safety and liveness properties. If there any violation, it produces an error trace. Using this error trace, a user can run a simulation of the execution that leads to the violation.

2.2 PROMELA

PROMELA is a verification modeling language. If we elaborate it, then it becomes Process Meta Language. It uses to verify the logic and parallel system. Given a program in PROMELA, Spin can verify the model for correctness by performing random or iterative simulations of the modeled system's execution, or it can generate a C program that performs a fast exhaustive verification of the system state space. During simulations and verifications, SPIN checks for the absence of deadlocks, unspecified receptions, and not executable code **[3]**. There has several characteristics in PROMELA, those are modeling language for channel systems with a finite number of processes, synchronous and asynchronous channel-based communication and shared variables. The verifier can also be used to prove the correctness of system invariants and it can find non-progress execution cycles.

PROMELA programs consist of processes, message channels, and variables. Processes are global objects that represent the concurrent entities of the distributed system **[3]**. Message channels and variables can be declared either globally or locally within a process. Processes specify behavior, channels and global variables define the environment in which the processes run.

2.2.1 *mtype* Declaration

An ***mtype*** declaration represents some variable, which has some constant value. In C language we can use #define keyword to define some constant value here ***mtype*** also dose the same thing as well as C but in different representation.

```
mtype = {ack, buyer, order, confirm}

#define ack true
#define buyer 3
#define  order 2
```

Range of ***mtype*** is 0….255, as it is a keyword and in spin tool it use as a data type. It can obtain the values from the range of symbolic names that is declared. This data type can also be used inside ***chan*** declarations, for specifying the type of message fields.

2.2.2 Channel Declaration

Channels are used for transferring messages on those processes, which are activated. It is declared by the keyword **chan**. Channels by default store messages in first-in first-out order. Channel statement can be declared such as below:

```
chan   chan_name   = [  buffer size, N  ] of {  type   }
```

After *chan*, we write the name of the channel name, after that inside third brace we declare the size of channel and then what type of data we use that declares in second brace. The buffer size indicates how many types we can use in type.

2.2.3 Message Passing

To send data from one process to another it needs message passing. Message passing makes through the channel. Declaration of message passing is shown in below:

```
chan Byr_Brkr  = [2] of {mtype, bool}
chan Byr_supplr[3] = [0] of {byte};
chan Channel;
```

In Byr_Brkr channel, it can hold two messages and the message type is shown inside the brace. Here, message consists of two part those are mtype and bool. However, in Byr_supplr there are three channels and here it can hold only one type of message that is byte.
The statement

```
qname! expr
```

It sends the value of the expression expr to the channel name qname. If there are several expression in that case by default the send statement is only executable if the target channel is not full otherwise it blocks.
The statement:

```
qname ? msg;
```

It receives the message and retrieves it from the head of the channel, then stores it in the variable msg. The channels pass messages by first-in-first-out order. The receive statement is only executes while the sender channel is not empty. It will show error if there are more or fewer message fields then declaration for the message channel that is addressed.

2.2.4 Control Flow Constructs

There are three control flow constructs in PROMELA. They are the selection, the repetition and the unconditional jump.

2.2.4.1 Selection

In selection part, it can choose any of statement depending on the condition. Here we consider two values a and b give a simple introduction of selection part.

```
if
:: (a > b) -> option1
:: (a < b) -> option2
:: else break;
fi
```

The selection structure contains two execution sequences, each preceded by a double colon. Here only one sequence will go to be executed based on the condition. A sequence can be selected only if its first statement is executable. The first statement after double colon is called guard statement. This guard statement determines whether the execution sequence that follows is selectable for execution or not. . If not all guards are executable, then the process will block until one of them is selected. In the example above, the guards are mutually exclusive, but they need not be. If more than one guard is executable, one of the corresponding sequences is selected non-deterministically.

2.2.4.2 Repetition

Repetition (loop) a logical extension of the selection structure is the repetition structure. For example:

```
do
:: count = count + 1
:: count = count-1
:: (count == 0) -> break
od
```

Only one option can be selected at a time in the structure of repetition in PROMELA. After the option completes, the execution of the structure is repeated. Break statement is used

here to terminate the structure normally. It transfers the control to the instruction that immediately follows the repetition structure.

2.2.4.3 Unconditional Jump

Unconditional jumps is the another way to break a loop and that is the goto statement. For example, one can modify the example above as follows:

```
do
:: count = count + 1
:: a = b + 2
:: (count == 0) -> goto done
od
done:
skip;
```

The goto in this example jumps to a label named done. A label can only appear before a statement. To jump at the end of the program, for example, a dummy statement skip is useful: it is a place-holder that is always executable and has no effect.

2.2.5 Active proctype

The active keyword, when added before any proctype definition, it means an instance of that proctype will be active in the initial system state. If, multiple proctype needs to run then it run as follows

```
active proctype A() { ... }
active [4] proctype B() { ... }
```

2.2.6 init

init is another way to active any process. It works as main process we can call function inside it as follows:

```
proctype prc_x(byte x)
{
     printf("x: %d, pid: %d\n",x,_pid)
}
init
{
     run prc_x(0);
     run prc_x(1)
}
```

We can run several processes by using init. It's working action is similar as main function of other languages.

2.2.7 Atomic

Atomic sequence makes a way to avoid the test and set problem. It indicates that the sequence is to be executed as one indivisible unit, non-interleaved with other processes. In the interleaving of process executions no process can execute statements from the moment that the first statement of an atomic sequence is executed until the last one has completed. example:

```
atomic {
       if
       :: a = 1
       :: a = 2
       fi;
       if
       :: b = 1
       :: b = 2
```

```
    fi
}
```

In this example, the variables a and b are assigned a single value, with no possible interleaving Statements from any other process. There are four possible ways to execute this atomic sequence.

2.2.8 Assertion

An important language construct in PROMELA that needs a little explanation is the *assert* statement. Statements of the form:

```
assert(any_boolean_condition)
```

The assert statement takes any valid PROMELA expression as its argument. The expression works each time the statement is executed. If the expression evaluates to false then an assertion violation is reported.

2.3 LTL (Linear Temporal Logic)

It is important to find out desired property of a model by using model checking method. LTL operators actually check if the model satisfies this property. Therefore, SPIN accepts correctness properties expressed in linear temporal logic (LTL). SPIN performs the conversion to Büchi automata mechanically based on a simple on-the-fly construction [3]. The automata that are generated formally accept only those system executions that satisfy the corresponding LTL formula.

By using LTL we can find out different types of properties such as safety property, liveness property etc. which helps to find the desired checking of a model. A formula of LTL is built from atomic propositions and from operators that include the operators of the propositional

calculus as well as temporal operators. Some basic operators, which use in LTL is shown in below:

Operator	SPIN	General Symbol
Always	[ ]	□
Eventually	<>	◇
Until	U	U

Table 2.1: Temporal Operator

General representation of a LTL property is given below:

```
ltl [ name ] '{'  formula '}'
```

Inside the name brace, we include here the process name, and inside the formula brace we insert formula to see which property we want to represent like liveness property or safety property. Second operators can either be abbreviated with the symbols shown above, or spelled out in full as (always, eventually, until, implies, and equivalent). Below the example will show that the two properties are equivalent.

```
ltl p1 { []<> p }
ltl p2 { always eventually p }
```

The model checker will perform an automatic negation of the formula to find counter-examples from these properties.

SPIN version 6 and later an improvement is introduced that is LTL formula can contain any propositional formula so there is no restriction to use the lower-case propositional symbols as before. Now it can write as follows

```
ltl p3 { eventually (a > b) implies len(q) == 0 }
```

The operands of an LTL formula are often one-character symbols, such as p , q , r , but they can also be symbolic names, provided that they start with a lowercase character, to avoid confusion with some of the temporal operators which are in uppercase. The names or symbols must be defined to represent Boolean expressions on global variables from the model. The names or symbols are normally defined with macro definitions.

2.4 Web Service Composition

Web service is an emerging technology of building complex in distributed system. It is focusing on interoperability, support of efficient integration of distributed processes. Different services provided by different organization, perform basic activities that combined in suitable ways, allow for the definition of different complex business process [4]. Web service supports the interaction among different partners by providing a model of synchronous or asynchronous exchange of message. There are two key aspects in web service composition those are Choreography, Orchestration.

2.4.1 Choreography

Choreography is that where different party involves and interacts with each other for a particular work [Figure 2.1]. Choreography manages the sequence of messages of each party where they involved in web services for the purpose of business process.

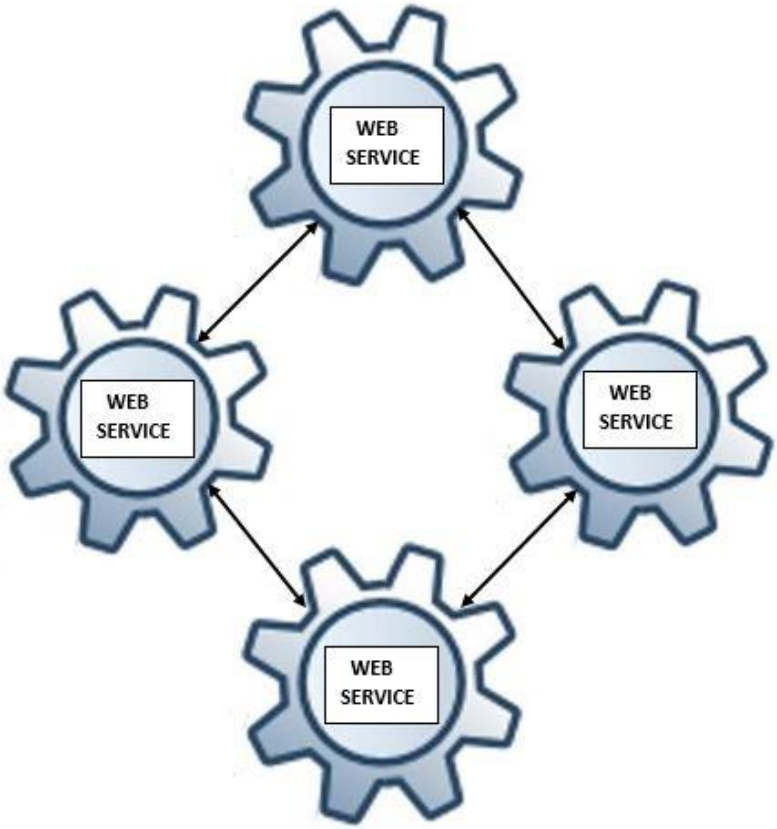


Figure 2.1: Web Services Choreography

2.4.2 Orchestration

Orchestration works inside one party mechanism [Figure2.2]. In a single party, whatever work is done inside it that is managed by orchestration. It can interact with both internal and external web service. The interactions occur at the message level in orchestration.

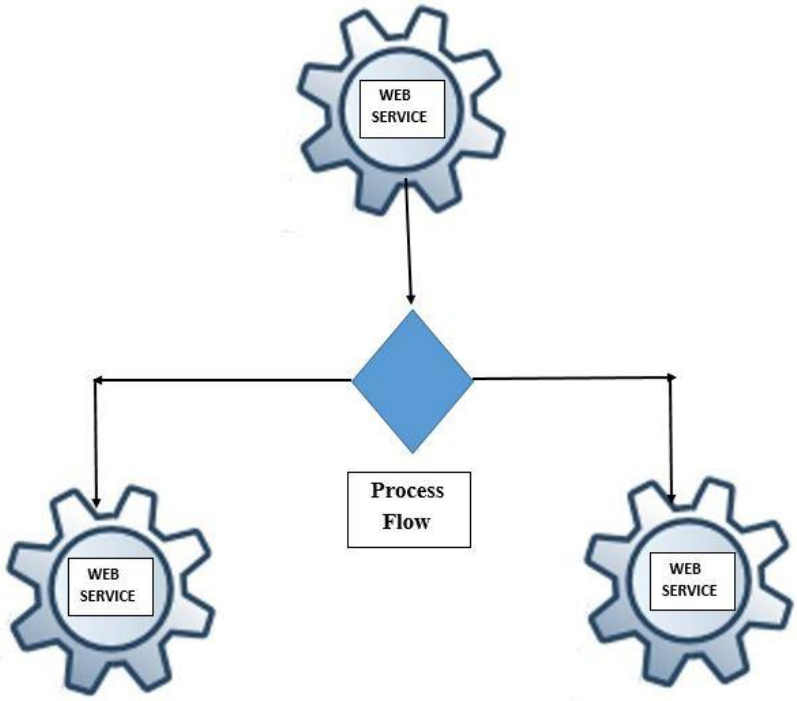


Figure 2.2: Web Services Orchestration

CHAPTER III

Web Service Composition and Compensation

3.1 Web Service Composition

Web Service technology provides a platform so that we can develop distributed service. Composition of web services has received much interest to support business-to-business or enterprise application integration. Web service is define as self-contained, modular units of application logic, which provide business functionality to other applications via an Internet connection. Web services support the interaction of business partners and their processes by providing a stateless model of “atomic” synchronous or asynchronous message exchanges. These “atomic” message exchanges can be compose into longer business interactions by providing message exchange protocols that show the mutually visible message exchange behavior of each of the partners involved. The issue of how web services are to be described can be resolved in various ways. [5]

We choose Broker system web service as an example to our verification. In broker system, web service negotiates purchase for its buyers and arranges loans for these. In this model, a buyer send a model request to broker for his/her desire products. Broker then uses its business partner Supplier to find the possible quote for the given model. Then another business model Loan Center arranges loan to buyer for the selected quote. Some notifications go to buyer about the quotes and the necessary loan arrangements. Both buyer and Loan

Center can cause interrupt to be invoke. Several negative possibilities might occur while we are running this entire process. For examples: loan can be refused due to failure of loan assessment, again customer can reject loan offer or quotes offer, Supplier may not provide proper quote to the broker etc. It needs to take some necessary steps so that the whole works never fail for all of these negative possibilities. Therefore, compensation needs to provide not to fall of all of these problems.

3.2 Buyer

In our system buyer can buy products from online. A buyer send request to broker for the products. When one gets his/her desire products from the broker then he/she can choose and order the product. If the overall process completed then buyer receives an acknowledgement, which will indicate the entire work is done.

Figure 3.1: Buyer Process

3.3 Broker

Broker manages the major part of communication while a product is buying. At first buyer sends product details to the broker. After that dealing with supplier, broker provides good product details to the buyer. If buyer satisfied with the product then broker deals with loan dealer to give loan for buyer. In addition, it sends order request to the supplier for the selected product. If the Loan Center rejects loan then a loan refusal fault will occur. Since the product has already ordered, so compensation requires the order to be cancel. The refusal is then return to the customer.

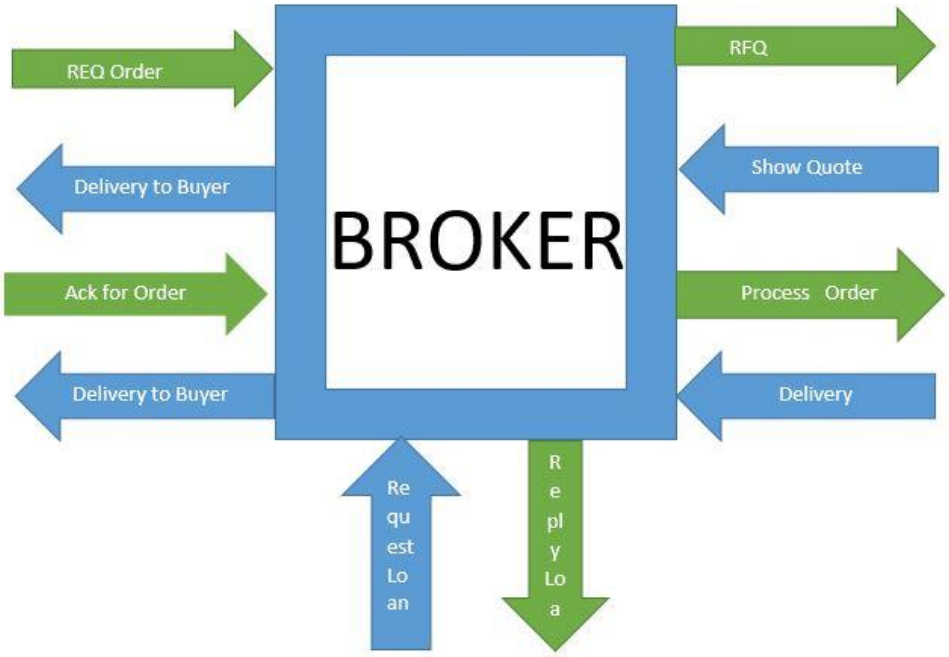


Figure 3.2: Broker Process

3.4 Supplier

First, customer sends product details to broker. The request for a quotation passes from Broker to Supplier. Supplier and Buyer doses not interact with each other. When Buyer orders for the product to Broker, Broker contacts with Supplier and makes all arrangement to give buyer the appropriate product.

Figure 3.3: Supplier Process

3.5 Loan Center:

A loan service is a frequent example for business processes. Loan Center offers a loan to customer, who submits a proposal contains on loan amount. Lender asks for a certain amount who takes loan also gives some condition. If Buyer agrees with condition, then lender gives loan to the Broker. After that, complete buying process is resume.

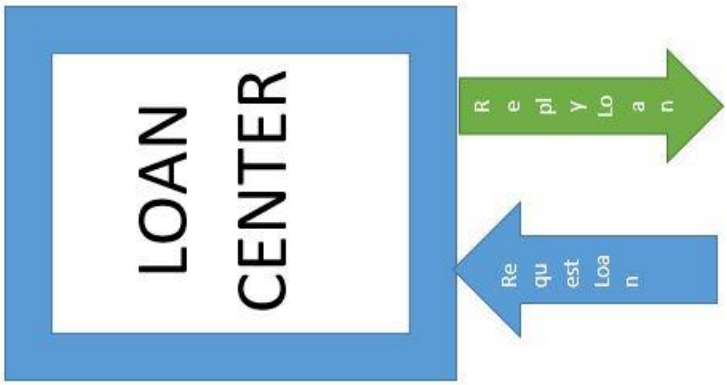


Figure 3.4: Loan Center

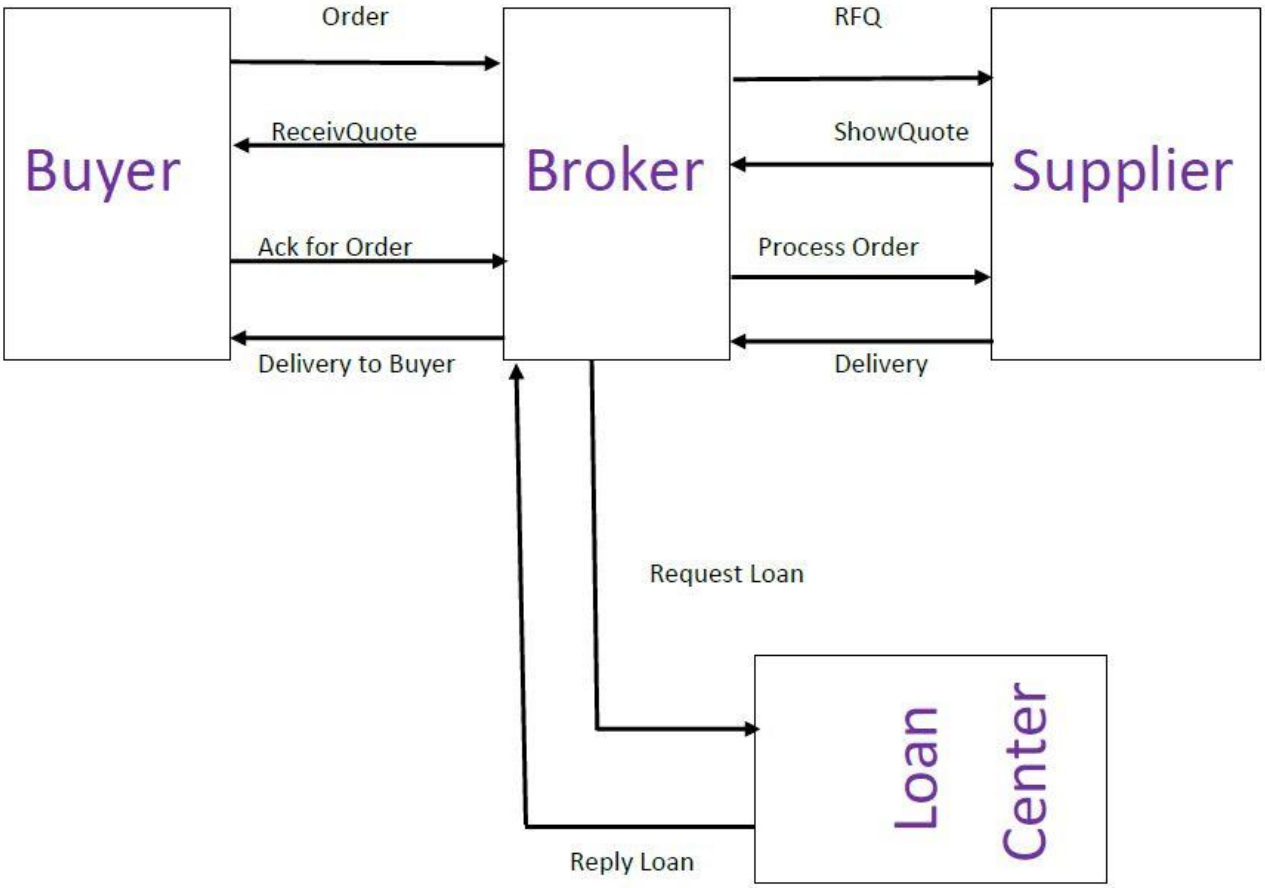


Figure 3.5: Overall diagram of BUYER BROKER system

Overall Transection in Message Sequence Chart:

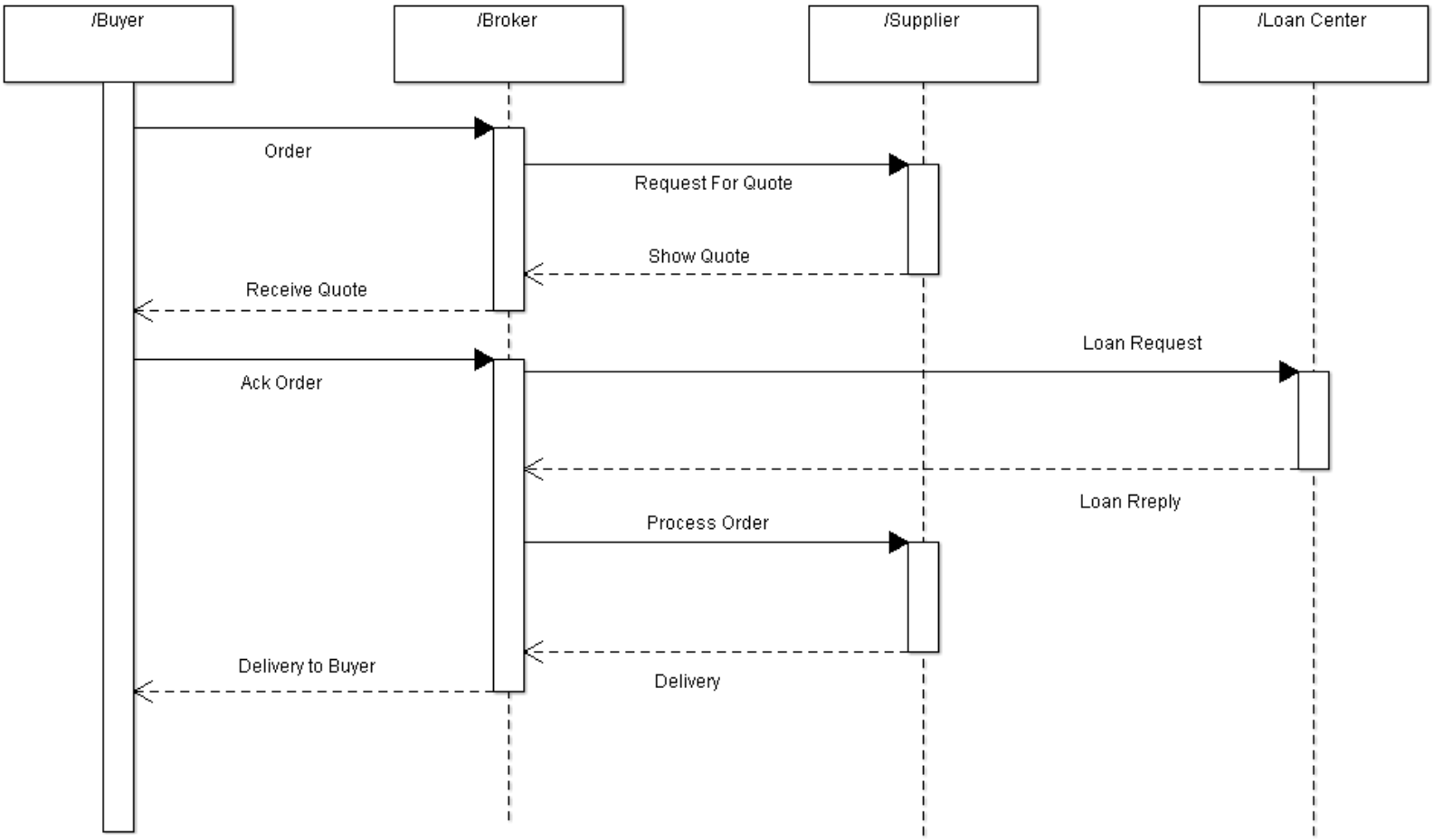


Figure 3.6: Buyer-Broker Message Sequence Chart

3.6 Compensation

Business transection typically involve coordination and interaction between multiple partners. This transection involve hierarchies of active and these activities need to orchestrate. Business transection need to deal with faults that can arise in any stage of the transecting in usual database transection. A rollback mechanism is used to handle faults in order to provide automaticity to a transection. However, for transection that that required long period to complete. Also called LONG RUNING TRANSECTION (LRT), rollback is not always possible .LRs are usually interactive that communicate with several activities .handling faults where multiple partner are involved are both difficult and critical. Due to their interactive nature, LRTs are not able to be check pointed, e.g. a sent message cannot be unsent. In such case, a separate mechanism i.e. required to handle faults. A possible solution of the problem would be that the system designer could provide a mechanism to compensate the action that cannot be undone automatically. [6]

3.7 Compensation Mechanism of Web Service

In our model, we have used a compensation mechanism to handle errors. Errors are mainly occur during message passing. Since every transection reply to other services only when previous transection sent a confirmation message, so absent of confirmation message fails the overall transection process. For solving this problem, we have used a guard bite to identify where the transection process failed. If guard bit replies with false value then it call a process named “Abort” , which stars from that point where guard bit found a false value and it pass a message to those whose are already transect message for completing the overall services. In chapter 4 section 4.6.5, we have implemented this mechanism by using PROMELA. A message sequence chart has been included to show how compensation mechanism sent abort message to other process, when any process sent any negative

confirmation or something like that. A message sequence chart with cancelation process has been shown in figure 3.6.It represents how cancelation message will be passed in overall model.

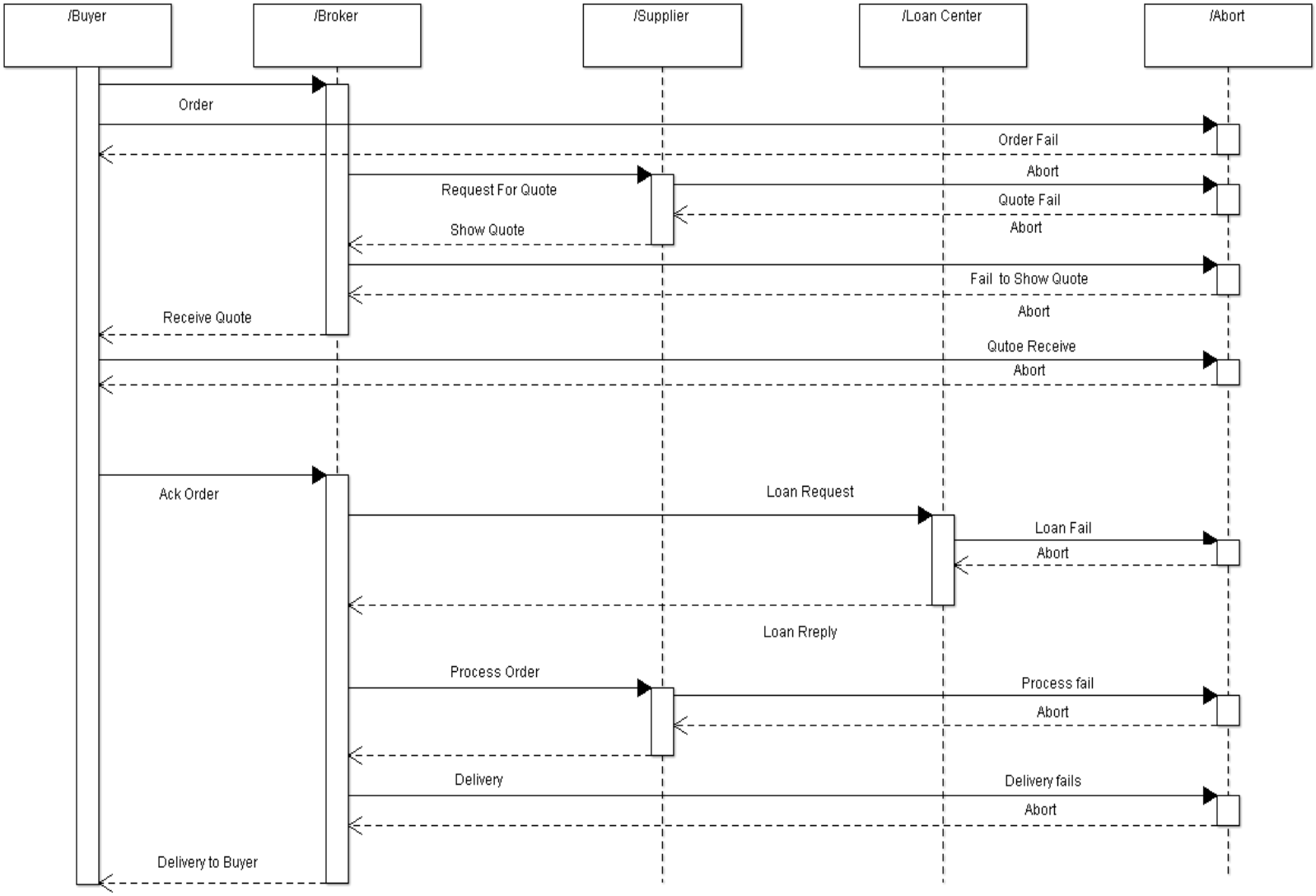


Figure 3.7: Message Sequence chart for Abort process

CHAPTER IV

Service Choreography in PROMELA

4.1 Encoding Process

In this section we designed a model in PROMELA which is a logical representation of message based interaction among the process, so that we can explain web service choreography. Each section of the model contain one process that runs parallel. This model consist with a BUYER, BROKER, SUPPLYER and LOAN center section. Once BUYER need to order something then it contact with BROKER system. Basically all other communication system is occurred by BROKER system. After receiving order request, BROKER collect information from SUPPLYER and LOAN center through message passing communication and relevant order response send to the BUYER section. In the meantime BROKER system communicate with other system via message or relevant task with respective response of other system. Since each section are related with others, so any negative response can be occurred any time. For solving this situation we used compensation.

4.2 Declaration of *mtype*

To build this model at first we declare some message type variable that called *"mtype*" in PROMELA. *"mtype"* refers model variable lists in PROMELA . We have used this data type for specifying message passing along with channel declaration. This"*mtype*" variable is not

initialized whether by default it will be initialized to zero. Corresponding individual name for each message are represented by this data type. The subsequent set of data are defined as

```
mtype = {
Order,QuoteS,QuoteB,Ack,RFQ,recived,Reqln,Reply,Abort_request,order,
splrordr,feadback,Cancel_Order } ;
```

BUYER system firstly sent an order message and BROKER receive it. After that, BROKER ask quotes information to SUPPLYER and relevant response are represented by SUPPLYER through QuoteS. BROKER receive it and another message named QuoteB pass to the BUYER system. So when BUYER take it decision for transecting message then it sent an ACK message to BORKER and then this system sent a message for processing order to SUPPLYER by splrorder. Finally last system sent delivery to BROKER and final BUYER received message from BROKER. In the meantime BROKER system contact LOAN center by Reqln for lending and relevant response is passing by Reply message. Cancel_Order and Abort_request message transection occurred when any negative response transfer from one system to others.

4.3 Channel Declaration

Channels are the track or the transection rout of each message passing. In this section we highlighted the channels construction that has been used to communicate services with each other. Channels are defined for each service that contains message type data. Channel declaration have to be global, otherwise message's data cannot be received. We have used two types channel for message passing. One type for regular message transection and other used when compensation occurred. Each channel contains data length and type of data which can be Boolean type or byte etc. In our model we have used message type and Boolean type.

```
//Channels for transecting message
3    chan buyer_broker1 = [2] of {mtype,bool};
4    chan buyer_broker2 = [2] of {mtype,bool};
5    chan broker_buyer1 = [2] of {mtype,bool};
6    chan broker_buyer2 = [2] of {mtype,bool};
7    chan broker_buyer3 = [2] of {mtype,bool};
8    chan broker_supplyer1 =[2] of {mtype,bool};
9    chan broker_supplyer2 =[2] of {mtype,bool};
10   chan Brokerto_ohers1 =[2 ] of {mtype,bool};
11   chan Brokerto_ohers2 =[2 ] of {mtype,bool};

12   //Channels for transecting abort message

13   chan abortbuyer_broker1=[1] of {mtype};
14   chan abortbuyer_broker2=[1] of {mtype};
15   chan abortbroker_supplier1=[1] of {mtype};
16   chan abortbroker_supplier2=[1] of {mtype};
17   chan abortbroker_buyer1=[1] of {mtype};
18    chan abortsupplyer_broker2=[1] of {mtype};
19   chan abortbroker_buyer2=[1] of {mtype};
20    chan abortsupplier_brokrt2=[1]of{mtype}
```

Message type data named *"mtype"* contains information between two services and Boolean type contain confirmation data that has been confirmed by most recent previous transection among two services.

4.4 Boolean types Representation

Since Boolean type represent only true and false value, that's why we use this property for checking the data type whether the most recent message transection is occurred or not. Each service transection message contains one confirmation towards next service.

```
23   bool buyerorder_confirmation=true, rfq_confirmation=true;
24   bool quots_confirmationS=true, quots_confirmationByr=true;
25   bool ack_confirmation=true, loanreq_confirmation=true;
26   bool  brokerorder_confirmation=true, Quote_confirmationBr,
     loanereply_confirmation=true;
27   bool splrordr_confirmation=true, feadback_confirmation=true;
```

All Boolean value initialized by true, because we consider that no negative confirmation come from any transection. If any situation occur when any service send any negative confirmation then service process made Boolean value negative for completing the abort process.

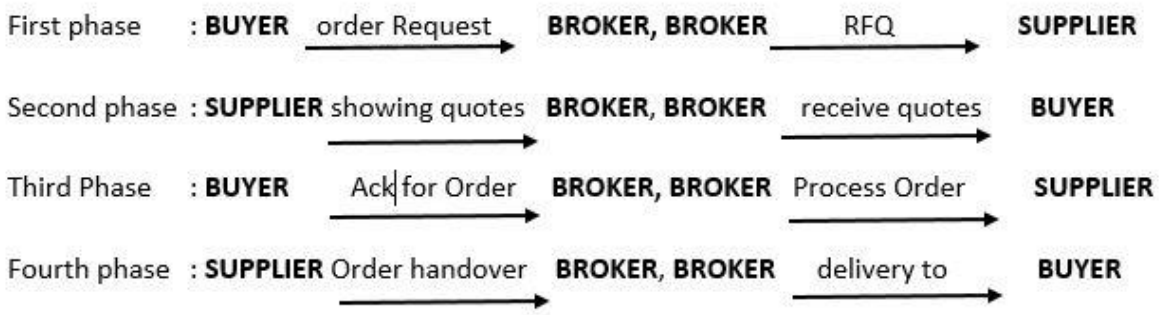
4.5 Process Creation

Each service has its own process where formal methods and statement are written and message sending and receiving to its destination also include in this portion. In PROMELA each process is represented by *"proctype"*, where it refers process type. Process type is used for declaring new process behavior. A process type consist with a name and body whether it has any parameters or not. Process can be created at any portion in the execution and it starts executing after run statement. Also it execute by active process where process need not any run statement. In our model we have used five process for representing the overall service architecture and these are *proctype* BUYER, BROKER, SUPPLIER, LOAN Center and Abort. Every process has its own methods and statement for checking the overall service, whether it work or not. A sample process that has been used in our model is given below:

```
active proctype LoanStar()
123  {
124  Brokerto_ohers1?Reqln(loanreq_confirmation);
125  if
126  ::loanreq_confirmation==true-
>Brokerto_ohers1!Reply(loanereply_confirmation); flag=1;
127  ::else->run Abort();
128  fi;
129
130  }
```

4.6 General Model Representation in PROMELA

To encoding our model in PROMELA we have divided our model into few fragments. Each fragments has a logical representation of PROMELA code. In first section a transaction will occur from BUYER to SUPPLIER and relevant response will be passed by second section. Third section will describe about Oder confirmation and Loan services and relative confirmation will be shown in last section. Overall section will be follow like below:



Following phases are works as below:

4.6.1 First Phase:

First BUYER sent an ORDER request to BROKER service and BROKER receive it. Since BROKER always sent request to BUYER in first starting phase, so we used atomic statement. After receiving order request BROKER sent RFQ (request for quotes) to SUPPLIER. Following codes are given below

```
34   atomic
35   {
36   buyer_broker1 ! Order(buyerorder_confirmation);
37    }
64   atomic
65   {
66   buyer_broker1 ? Order(buyerorder_confirmation);
69   }
70   if
71   ::buyerorder_confirmation==true>broker_supplyer1!
     RFQ(rfq_confirmation);flag= flag+1;
72   ::else->run Abort();
73   fi
102 broker_supplyer1?RFQ(rfq_confirmation);
```

4.6.2 Second Phase:

SUPPLIER will inform about quotes information to BROKER whether there are any quotes or not and relevant information will provides to BUYER for its next transection.

```
103  if
104  ::rfq_confirmation==true-
>broker_supplyer2!QuoteS(quots_confirmationS);flag=flag+5;
105  ::else->run Abort();
106  fi;
     broker_supplyer2?QuoteS(quots_confirmationS);
75   if
76   ::quots_confirmationS==true-
     >broker_buyer2!QuoteB(quots_confirmationByr); flag=
     flag+1;
77   ::else -> run Abort();
78   fi;
```

4.6.3 Third Phase:

In this section BUYER send it's acknowledge for order to BROKER. BROKER service then ask loan for LOAN Center if negative response occurred then an abort process sent cancel order message to all other services as like as every section whether any negative response occurred. If negative response sent absent message then BROKER send order process message to SUPPLIER.

```
42   if
43   ::quots_confirmationByr== true->buyer_broker1
     !Ack(ack_confirmation);
44   ::quots_confirmationByr== false->run Abort();
45   fi;
79   buyer_broker1 ?Ack(ack_confirmation);
80   if
81   ::ack_confirmation==true-
     >Brokerto_ohers1!Reqln(loanreq_confirmation);
82   ::else->run Abort();
83   fi;
     Brokerto_ohers1?Reqln(loanreq_confirmation);
125  if
126  ::loanreq_confirmation==true-
     >Brokerto_ohers1!Reply(loanereply_confirmation); flag=1;
127  ::else->run Abort();
128  fi;
84   Brokerto_ohers1?Reply(loanereply_confirmation);
85   if
86   ::loanereply_confirmation==true-
     >broker_supplyer1!order(brokerorder_confirmation);
87   ::else->run Abort();
88   fi;
     broker_supplyer1?order(broker order_confirmation );
```

4.6.4 Fourth Phase:

SUPPLIER completed its task and relevant result sent to the BROKER. Finally BROKER received SYPPLIER delivery and send this message through channel to broker for final delivery. By this process overall transection come to their finishing point.

```
108  if
109  ::brokerorder_confirmation==true-
     >broker_supplyer1!splrordr(splrordr_confirmation);
110  ::else->run Abort();
111  fi;
     broker_supplyer1?splrordr(splrordr_confirmation);
90   if
91   ::splrordr_confirmation==true-> broker_buyer3!feadback(
     feadback_confirmation);
92   ::else->run Abort();
93   fi;
94   broker_buyer3 ? feedback ( feedback-confirmation )
```

4.6.5 Abort Process

This process will not be active until any negative confirmation passes any message to the others process. For checking this condition we have always checked a Boolean value which comes from previous transection. If they find absence of this value then it calls the Abort process which send a cancellation message to other process which already transferred message among the process. Basically this part describe how we implemented compensation mechanism in our web service. This process are given as below

```
proctype Abort()
104       {
105       if
106       ::ack_confirmation==false->Abrt=Abrt+1-
     >abortsupplier_brokrt2!Cancel_Order(Abrt);
107       ::brokerorder_confirmation==false->Abrt=Abrt+1-
     >abortsupplier_brokrt2!Cancel_Order(Abrt);
108       fi;
109       }
```

CHAPTER V

Web Service Composition Simulation

In this section we have simulated our model that has been represented by a simulation tool named “SPIN”. Its basic is made with model checking. Simulation modeling is used to understand whether, under what conditions, and in which ways a part could fail and what loads it can tolerate. Basically, “SPIN” is a professional software tool for specifying and verifying concurrent and distributed systems. Models, written in a simple language called PROMELA, can be simulated randomly or interactively. After representing our BUYER – BROKER web service model in PROMELA we have run this by a “SPIN” event named “Simulate” to check whether our model is working properly or not. If our Simulation comes with an absence of error then a simple message based figure comes to a window to represent a sequence of the overall model. Then verification and other related works are done by different process and steps. Sequential simulation process are described in the below section. [7]

5.1 Web Service Composition Simulation Result

Basically “SPIN” is a model checking simulator and it doesn’t represent a formal analysis, it actually provides a limited form support for verification like if statement, assertion checking, atomic condition and much more. While we have simulated our model it represented an impression that how overall model working in a certain condition .If we consider that each services are received positive response from other and no one sent Cancel request to other

service then a sequential message represent the overall model to understand that all condition have worked properly. Related figure are given below

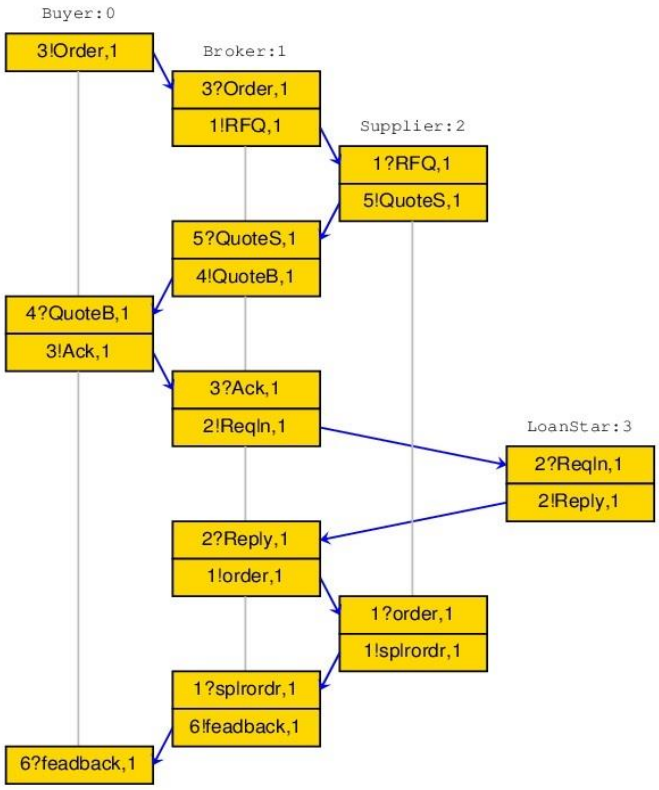


Figure 5.1: Simulation symbol of BUYER-BROKER Web Services

5.2 Automata View

Automata view refers, the study of the mathematical properties and its visualization that recognizes strings containing an even or odd number to symbolize the transection of one state/event to another. Our model consist of five independent process one BUYER, one BROKER, one SUPPLIER, one LOAN Center (as Loan Star) and one Abort which we have said before. Each corresponding process communicate with others by respective involvement. In

this section we describe about automata view of all this processes and then verifying this model using the language of PROMELA by the validation tool “SPIN”.

5.2.1 BUYER Process

The BUYER process automata figure is shown in the figure number 5.1. This process is consist with some message passing services. BUYER send an order request message to BROKER through their own channel and wait until BROKER response. After receiving BROKER response BUYER send ACK message to Broker and wait for final delivery. In PROMELA code “Feedback” signify the final delivery confirmation message. Since PROMELA made an automata view by the sequential line of code, therefor BUYER received lots of cancel order message which is not used in overall model. Because of while we designed our model negative confirmation can be arisen at any point, so we had to keep a special process for this situation. And that’s why few extra state has been shown in our automata view.

5.2.2 BROKER Process

BROKER process contains the maximum transection and provides maximum services among the process. BROKER communicates the BUYER, SUPPLIER and Loan Center to executing its task. BROKER received BUYER Order request and ACK confirmation and replies with quotes confirmation and final delivery .Also BROKER communicate with SUPPLIER to receive quotes information and final order delivery message against quotes information and BUYER final order confirmation.in addition BROKER process also communicate with Loan Center for handling the loan request and receive. All individual sequential transection of BROKER are shown as below figure no 5.1

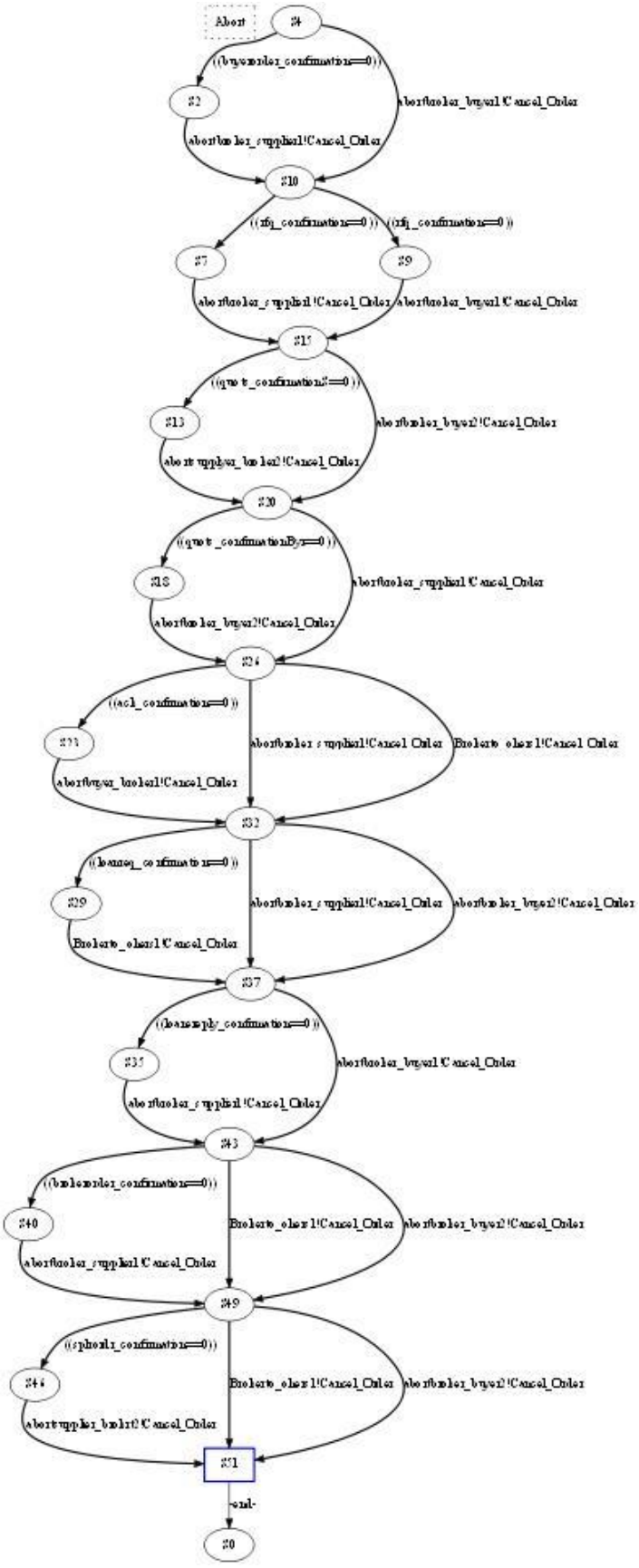


Figure 5.2: Automata View of Abort process

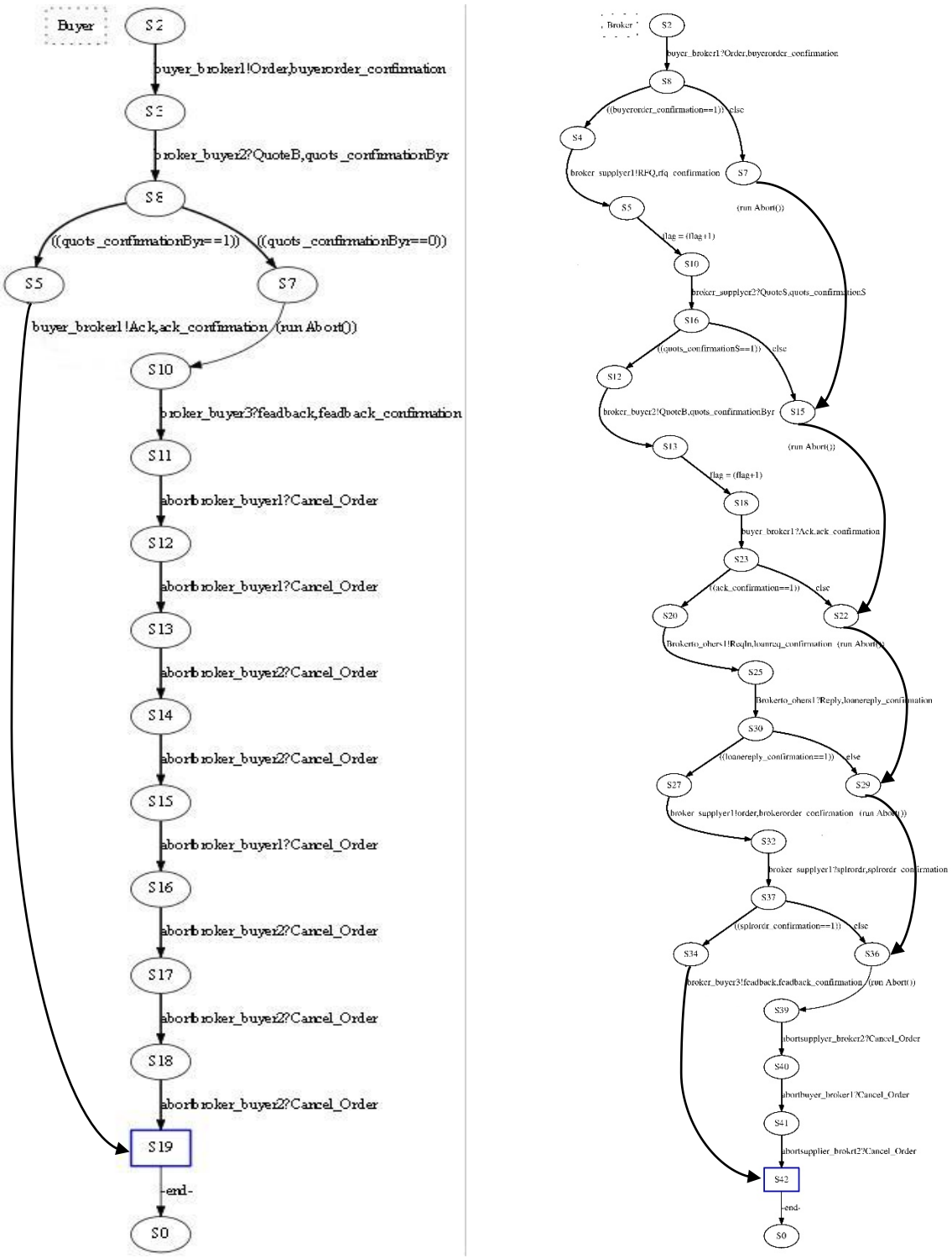


Figure 5.3: Automata view of BUYER-BROKER web services

5.2.3 SUPPLIER Process

SUPPLIER received BROKER quotes request and send quotes information to the source. After that BROKER send its final order confirmation to SUPPLIER and wait until its response .Then SUPPLIER starts it processing order and make a reply to BROKER via feedback message. All these steps are shown in SUPPLIRE automata view figure 5.2.

5.2.4 LOAN Center Process

In section 5 we have describe about message sequence chart of the general model and we have seen that LOAN center process become active only while BROKER system request a loan. So this process make it transection by “Replyln” or Cancel Order message. So its automata view is very simple .Relevant figure are shown as below in figure 5.2

5.2.5 ABORT Process

Our model is consist with all positive response which means there are no negative response among the process or the services. So what will happen if any service sent any negative response? For solving this situation an abort process has been established in this model. In chapter three already we have discussed about compensation mechanism. In this part we have handle our model’s error by using compensation mechanism. Each transection check previous Boolean type value for next transection. So if it acquire any negative or false value then we call abort process. So this process sent cancel order message to the services that has been already used for transection. Actually in Abort automata view, process represent the compensation mechanism that has been used. All these steps are shown in Abort automata view figure 5.2.

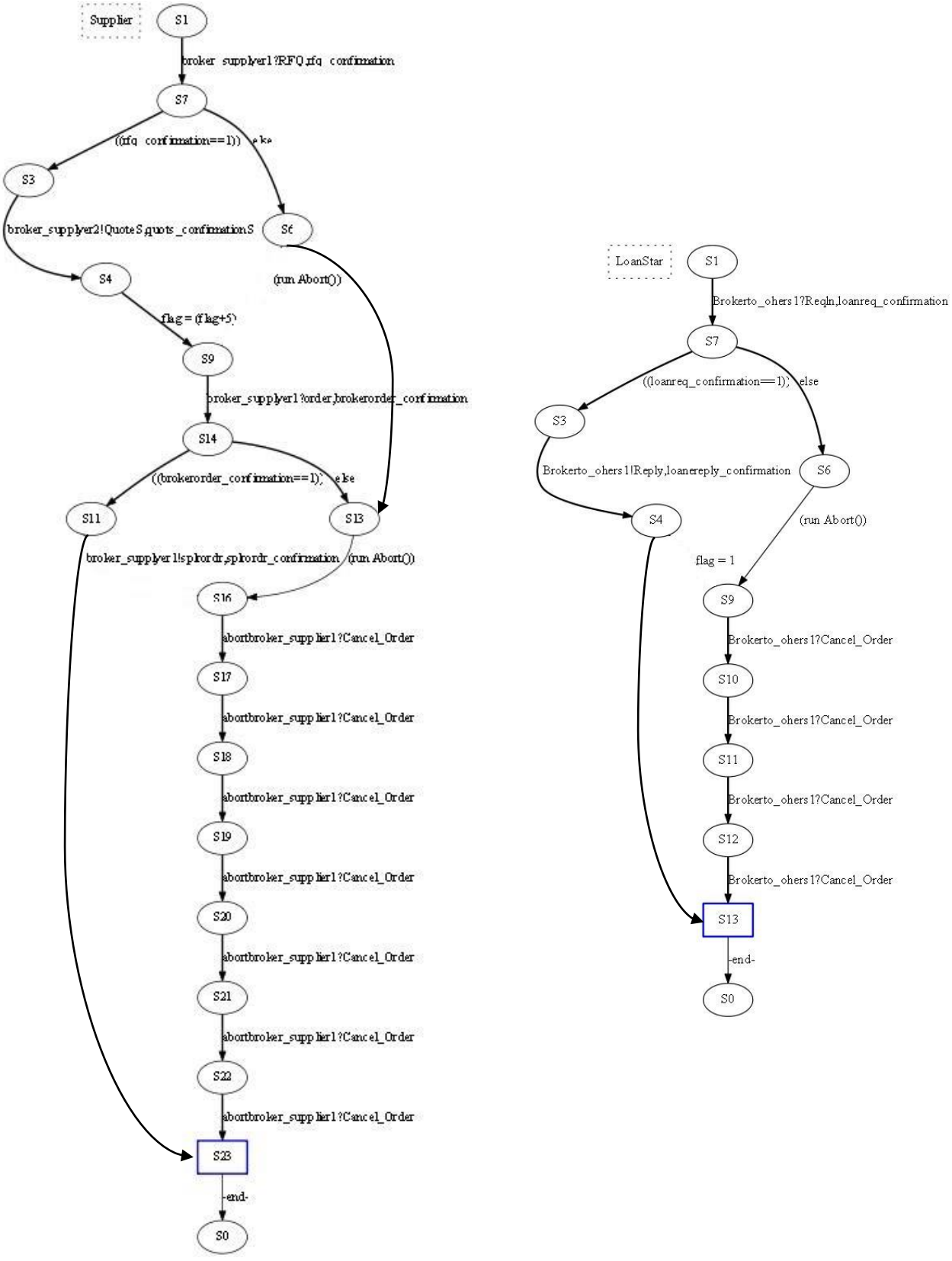


Figure 5.4: Automata view of SUPPLIER, LOAN Center web service

5.3 Verification

In verification mode, SPIN analyses the model against the properties considering all possible executions performing an exhaustive search on the state space. We use PROMELA as the formal specification language. We define our requirements as LTL (Linear Temporal Logic) formulae and convert these to never claims in PROMELA. In the following sections, all the formulae and related definitions are coded in PROMELA. [8]

5.3.1 LTL Verification

LTL refers liner temporal logic. Which is an infinite sequence of states where each point in time has a unique successor, based on a linear-time perspective. In our model we have used this logics and special syntax for justify weather our model has a fault point or it just worked properly. LTL has a lots of criteria or property for checking the model. So we have choose shifty and liveness property for Verify our model. [9]

5.3.1.1 Safety

For measuring the correctness of this composition we can perform a comprehensive verification with spin to prove some safety properties, such as absence of deadlock, unreachable code, and unspecified receptions. In this properties check the loan request confirmation and loan reply confirmation message send and received supply order confirmation. This property definite below

It is always true that if loan request confirmation and loan reply confirmation become true then it implies that supply order confirmation will be true.

The LTL representation are given below

```
ltl   p{[]((loanreq_confirmation==true)&&(loanereply_confirmation==true))
-> ((splrordr_confirmation==true))}
```

First we have written some LTL code and then convert this logic into never claim property of PROMELA to check weather this logic is working on overall code or not.

5.3.1.2 Liveness

LTL liveness property refers that in the future in any time something good will happen. So for checking this we have used a simple LTL. We have written some LTL code and then convert this logic into never claim property to check weather this logic is working or not.

. The formal representation of testing this property are described by below

"If Buyer order confirmation becomes true it implies that eventually state value will be equal to 5"

Here the LTL representation of checking the liveness property

```
ltl p { (buyerorder_confirmation==true)->(<>(state_Value==5))}
```

Chapter VI

Conclusion

6.1 Summary

Web service composition is an important issue in this modern age. Many services are now running as a web service to feel comfort for the consumer. Therefore, it is also important to have a verified web service. While a web service is running may issues came out from that, many changes can happen during a running services. In that case, it is not possible to run that service repeatedly from the beginning. Therefore, its verification is important to figure out the proper problem. Also if there is possibility to use some compensation mechanism for the failure part then web service will be more efficient and strong for working.

6.2 Future Work

Our future plan is to make the compensation part much stronger. We are also interested to make a comparison table between our project and compensating CSP to improve this project.

Appendix

A.1 PROMELA ENCODING

```
1    mtype={Order,QuoteS,QuoteB,Ack,RFQ,
     recived,Reqln,Reply,Abort_request,order,
2    splrordr,feadback,Cancel_Order};
3    chan buyer_broker1 = [2] of {mtype,bool};
4    chan buyer_broker2 = [2] of {mtype,bool};
5    chan broker_buyer1 = [2] of {mtype,bool};
6    chan broker_buyer2 = [2] of {mtype,bool};
7    chan broker_buyer3 = [2] of {mtype,bool};
8    chan broker_supplyer1 =[2] of {mtype,bool};
9    chan broker_supplyer2 =[2] of {mtype,bool};
10   chan Brokerto_ohers1 =[2 ] of {mtype,bool};
11   chan Brokerto_ohers2 =[2 ] of {mtype,bool};
13   chan abortbuyer_broker1=[1] of {mtype};
14   chan abortbuyer_broker2=[1] of {mtype};
15   chan abortbroker_supplier1=[1] of {mtype};
16   chan abortbroker_supplier2=[1] of {mtype};
17   chan abortbroker_buyer1=[1] of {mtype};
18   chan abortsupplyer_broker2=[1] of {mtype};
19   chan abortbroker_buyer2=[1] of {mtype};
20   chan abortsupplier_brokrt2=[1]of{mtype}
23   bool buyerorder_confirmation=true, rfq_confirmation=true;
24   bool quots_confirmationS=true, quots_confirmationByr=true;
25   bool ack_confirmation=true, loanreq_confirmation=true;
```

```
26  boolbrokerorder_confirmation=true,Quote_confirmationBr,
    loanereply_confirmation=true;
27  bool splrordr_confirmation=true, feadback_confirmation=true;
29  byte flag=0;
31  active proctype Buyer()
32  {
34  atomic
35  {
36  buyer_broker1 ! Order(buyerorder_confirmation);
37   }
39  broker_buyer2?QuoteB(quots_confirmationByr);
42  if
43  ::quots_confirmationByr==true->buyer_broker1
    !Ack(ack_confirmation);
44  ::quots_confirmationByr== false->run Abort();
45  fi;
47  broker_buyer3?feadback( feadback_confirmation);
48  abortbroker_buyer1?Cancel_Order;
52  abortbroker_buyer1?Cancel_Order;
53  abortbroker_buyer2?Cancel_Order;
54  abortbroker_buyer2?Cancel_Order;
55  abortbroker_buyer1?Cancel_Order;
56  abortbroker_buyer2?Cancel_Order;
57  abortbroker_buyer2?Cancel_Order;
58  abortbroker_buyer2?Cancel_Order;
60  }
62  active proctype Broker()
63  {
64  atomic
65  {
66  buyer_broker1 ? Order(buyerorder_confirmation);
67  //flag=1;
```

```
69  }
70  if
71  ::buyerorder_confirmation==true-
    >broker_supplyer1!RFQ(rfq_confirmation);flag= flag+1;
72  ::else->run Abort();
73  fi;
74  broker_supplyer2?QuoteS(quots_confirmationS);
75  if
76  ::quots_confirmationS==true-
    >broker_buyer2!QuoteB(quots_confirmationByr); flag= flag+1;
77  ::else -> run Abort();
78  fi;
79  buyer_broker1 ?Ack(ack_confirmation);
80  if
81  ::ack_confirmation==true-
    >Brokerto_ohers1!Reqln(loanreq_confirmation);
82  ::else->run Abort();
83  fi;
84  Brokerto_ohers1?Reply(loanereply_confirmation);
85  if
86  ::loanereply_confirmation==true-
    >broker_supplyer1!order(brokerorder_confirmation);
87  ::else->run Abort();
88  fi;
89  broker_supplyer1?splrordr(splrordr_confirmation);
90  if
91  ::splrordr_confirmation==true->broker_buyer3!feadback(
    feadback_confirmation);
92  ::else->run Abort();
93  fi;
95  abortsupplyer_broker2?Cancel_Order;
96  abortbuyer_broker1?Cancel_Order;
```

```
97   abortsupplier_brokrt2?Cancel_Order;
98   }
100  active proctype Supplier()
101  {
102  broker_supplyer1?RFQ(rfq_confirmation);
103  if
104  ::rfq_confirmation==true-
     >broker_supplyer2!QuoteS(quots_confirmationS);
          flag=flag+5;
105  ::else->run Abort();
106  fi;
107  broker_supplyer1?order(brokerorder_confirmation);
108  if
109  ::brokerorder_confirmation==true-
     >broker_supplyer1!splrordr(splrordr_confirmation);
110  ::else->run Abort();
111  fi;
113  abortbroker_supplier1?Cancel_Order;
114  abortbroker_supplier1?Cancel_Order;
115  abortbroker_supplier1?Cancel_Order;
116  abortbroker_supplier1?Cancel_Order;
117  abortbroker_supplier1?Cancel_Order;
118  abortbroker_supplier1?Cancel_Order;
119  abortbroker_supplier1?Cancel_Order;
120  }
122  active proctype LoanStar()
123  {
124  Brokerto_ohers1?Reqln(loanreq_confirmation);
125  if
126  ::loanreq_confirmation==true-
     >Brokerto_ohers1!Reply(loanereply_confirmation); flag=1;
127  ::else->run Abort();
```

```
128  fi;
130  Brokerto_ohers1?Cancel_Order;
131  Brokerto_ohers1?Cancel_Order;
132  Brokerto_ohers1?Cancel_Order;
133  Brokerto_ohers1?Cancel_Order;
137  }
139   proctype Abort()
140  {
142  if
143  ::(buyerorder_confirmation==false)-
     >abortbroker_supplier1!Cancel_Order;
144  ::abortbroker_buyer1!Cancel_Order;
145  fi;
147  if
148  ::rfq_confirmation==false-
     >abortbroker_supplier1!Cancel_Order;
149  ::rfq_confirmation==false->abortbroker_buyer1!Cancel_Order;
150  fi;
152  if
153  ::quots_confirmationS==false-
     >abortsupplyer_broker2!Cancel_Order;
154  ::abortbroker_buyer2!Cancel_Order;
155  fi;
157  if
158  ::quots_confirmationByr==                                false-
     >abortbroker_buyer2!Cancel_Order;
159  ::abortbroker_supplier1!Cancel_Order;
160  fi;
162  if
163  ::ack_confirmation==false->
164  atomic
165  {
```

```
166  abortbuyer_broker1!Cancel_Order;
167  abortbroker_supplier1!Cancel_Order;
168  Brokerto_ohers1!Cancel_Order;
169  }
170  fi;
172  if
173  ::loanreq_confirmation==false-
     >Brokerto_ohers1!Cancel_Order;
174  ::abortbroker_supplier1!Cancel_Order;
175  ::abortbroker_buyer2!Cancel_Order;
176  fi;
178  if
179  ::loanereply_confirmation==false-
     >abortbroker_supplier1!Cancel_Order;
180  ::abortbroker_buyer1!Cancel_Order;
181  fi;
183  if
184  ::brokerorder_confirmation==false-
     >abortbroker_supplier1!Cancel_Order;
185  ::Brokerto_ohers1!Cancel_Order;
186  ::abortbroker_buyer2!Cancel_Order;
187  fi;
189  if
190  ::splrordr_confirmation==false-
     >abortsupplier_brokrt2!Cancel_Order;
191  ::Brokerto_ohers1!Cancel_Order;
192  ::abortbroker_buyer2!Cancel_Order;
193  fi;
194  }
205  ltl
     p{[]((loanreq_confirmation==true)&&(buyerorder_confirmation
     ==false))->(splrordr_confirmation==false)}
```

```
207  ltl
     p{[]((loanreq_confirmation==true)&&(loanereply_confirmation
     ==true))-> (!(splrordr_confirmation==true))}
215  ltl
p { (rfq_confirmation==true)-> (<>(quots_confirmationByr==true))}
```

A.2 LTL Verification

A.2.1 LTL Codes

```
ltl
p{[]((quots_confirmationS==true)&&(ack_confirmation==true))>(fea
dback_confirmation==true)}

ltl
p   {[]((quots_confirmationS==false)&&(ack_confirmation==false))-
>(feadback_confirmation==true)}

 ltl
p{[]((buyerorder_confirmation==true)&&(flag==1))-
>((quots_confirmationS==true)&&(flag==5))}
```

```
ltl
 p{(buyerorder_confirmation==true)-
>(<>(splrordr_confirmation==true))}

ltl
p {    (loanreq_confirmation==true)->(splrordr_confirmation==true)
}
```

A.2.1.1 Liveness

```
ltl
p { (rfq_confirmation==true)-> (<>(quots_confirmationByr==true))}
```

A.2.1.2 Safety

```
ltl
p{[]((loanereply_confirmation==false)&&(splrordr_confirmation==
false))->(feadback_confirmation==true) }
```

Reference:

[1] Shamim H Ripon .Process Algebraic Support for Web Service Composition

[2] Shamim Ripon, Sumaya Mahbub and K. M. Imtiaz-ud-Din. Verification of a Security Adaptive Protocol Suite Using SPIN. Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh

[3] Concise PROMELA Reference [http://spinroot.com]

[4] Raman Kazhamiakin, Marco Pistore. Formal Verification of Requirement using SPIN: A Case Study on Web Service

[5] Biplav Srivastava, Jana Koehler. Web Service Composition - Current Solutions and Open Problems http://user.enterpriselab.ch/~takoehle/publications/bpm/icaps03-ws.pdf

[6] Process algebraic support for web service composition. SH Ripon - ACM SIGSOFT Software Engineering Notes, 2010.Cited by 3

[7] Ben-Ari Mordechai. "Principles of the Spin Model Checker"[http://www.springer.com/]

[8] SH Ripon, SF Ahmed, YR AfrozaYasmin, KM Imtiaz-Ud-Din. Formal Analysis of a Ranked Neighbour MANET Protocol Suite. International Journal of Future Computer and Communication (IJFCC) 3 (5)

[9] Linear Temporal Logic.www.cs.colostate.edu/~france/CS614/Slides/Ch5-Summary.pdf