SPIN Based Verification Of Service Composition And Compensation

Nusrat Jahan Eka

2013-1-60-013

Tasneem Jannat Sultan

2013-1-60-027

Nuzat Nawal

2013-1-60-046

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

Department of Computer Science and Engineering
East West University
Dhaka-1212, Bangladesh

April, 2017

Declaration

We, hereby, declare that the work presented in this thesis is the outcome of the investigation performed by us under the supervision of Dr. Shamim H. Ripon , Associate Professor , Department of Computer Science and engineering, East West University. We also declare that no part of this thesis has been or is being submitted elsewhere for the award of any degree or diploma.

Signature

.....

(Nusrat Jahan Eka)

(ID: 2013-1-60-013)

Signature

.....

(Tasneem Jannat Sultan)

(ID: 2013-1-60-027)

Signature

.....

(Nuzat Nawal)

(ID: 2013-1-60-046)

Letter of Acceptance

This thesis entitled "SPIN Based Verification Of Service Composition And Compensation" submitted by Nusrat Jahan Eka (ID:2013-1-60-013), Tasneem Jannat Sultan (ID:2013-1-60-027) And Nuzat Nawal (ID:2013-1-60-046), to the Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh is accepted by the department in partial fulfillment of requirements for Award of the degree of Bachelor of Science in Computer Science and Engineering on Aril, 2017.

Supervisor

.....

(Dr. Shamim H. Ripon)

Associate Professor, Department of Computer Science and Engineering

East West University, Dhaka, Bangladesh

Chairperson

.....

(Dr. Ahmed Wasif Reza)

Chairperson (Acting) and Associate Professor

Department of Computer Science and Engineering

East West University, Dhaka, Bangladesh

Abstract

In a web service, there occur business transactions. Sometimes there may occur some sorts of difficulties in the terms of transactions. To provide a good web service, it's composition needs to get verified and those difficulties need to get handled. And so our motivation is to verify composition and compensation of the web service. We have used car broker web service as our composition to verify and compensate the web service to provide a good web service. And so we have composed and compensated this service and verified this through the **LTL** verifications using **PROMELA** language in **SPIN** tool.

Acknowledgments

First of all thanks to ALMIGHTY ALLAH for the uncountable blessings on us. We would like to express our deepest gratitude to our supervisor, Dr. Shamim H. Ripon, for his excellent guidance, caring, patience. He was consistently there to help and to guide us, whenever we ran into a trouble spot or had a question about our work. It would have been tough to complete the work without his guidance.

We would like to express our gratitude to the faculty members of the Department of Computer Science and Engineering (CSE), East West University for giving us valuable feedbacks and supporting us throughout the process.

Finally, we must express our very profound gratitude to our families and friends who provided us continuous encouragement and supported us through thick and thin.

Nusrat Jahan Eka
April, 2017

Tasneem Jannat Sultan
April, 2017

Nuzat Nawal
April, 2017

Table of Contents

List of Figures

Chapter 1

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 is important to verify a web service in order to provide a good web service. A good web service means, every process inside the service is working properly or if there are any faulty in the transactions, then those should be handled. When multiple partners are involved in a web service then it becomes composite web service or we can say web service composition. In a web service, there business transaction occurs. Sometimes in business transaction, there can arise any faulty in any stage of transactions. In usual database transaction, a rollback mechanism is used to handle faulty 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 represents a SPIN based formal verification approach of a web service composition and compensation.

1.2 Motivation

In web service composition existing techniques for creating business procedure are not good enough to handle errors while each service transfers data in between cross organizational opponents. Since all this methods are not flexible enough to handle the errors of the technical transactions of that web services. For solving this problem, compensation mechanism has a good impact on web service composition. We choose car broker web service as an example to our verification. We have simulated our model that has been represented by a simulation tool named "SPIN". Its basic is model checking and our purpose is to check whether our model is working properly or not. We have used compensation mechanism on our web service composition and then we have also tried to verify the overall model.

- Analyze and verify
  - The composition of web service.
  - Compensation mechanism.
- Verify correctness of the model.

1.3 Contribution

We have made the following contributions in this project:

1. We have analyzed the feature of Car Broker web service by using SPIN tool in PROMELA language. Here, we identified various components of car broker web service.
2. Several negative possibilities can occur in the transaction of web service. So in this case, we have used compensation mechanism to solve this problem.

3. Finally we have used Linear Temporal Logic (LTL) property *where safety and liveness property is used* to verify our implementation.

1.4 Outline

- Chapter 1: Basically chapter I is an introductory part.
- Chapter 2: In chapter II named Background and there is an explanation of service composition and its types in which our service composition works. There is also an brief explanation of 'SPIN' tool which specifies how to work with SPIN. There are also some descriptions about 'PROMELA'. After that, we have discussed a general description of LTL (linear temporal logic) property and web service composition.
- Chapter 3: In chapter III there is an overview of our Car Broker System in service composition model that how it works with message sequence chart.
- Chapter 4: In chapter IV there is a compensation overview of our Car Broker System model and how the compensation mechanism works, shown in the sequence chart.
- Chapter 5: In chapter V we have simulated and modeled our web service in SPIN tool and have encoded our web service (Car Broker Web Service) into PROMELA language.
- Chapter 6: In chapter VI verification of the Car Broker Web Service and shows its correctness using LTL (linear temporal logic) with safety and liveliness property. property.
- Chapter 7: In chapter VII we have concluded our whole work with a short summary.

Chapter 2

Background

2.1 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[2]. 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.

2.1.1 Orchestration

Orchestration means executable Process. Orchestration describes how web services interact with each other, including the business logic and execution order of the interactions. WS BPEL is one of the languages which executes on an orchestration engine.

2.1.2 Choreography

Choreography describes externally observable interactions between web services. It describes multi party contacts. Our Car broker web service follows the Choreography mechanism. It tracks the sequence of messages that may involve multiple parties and multiple sources, including customers, suppliers, and partners. It typically associated

with the public message exchanges that occur between multiple web services, rather than a specific business process that is executed by a single party.

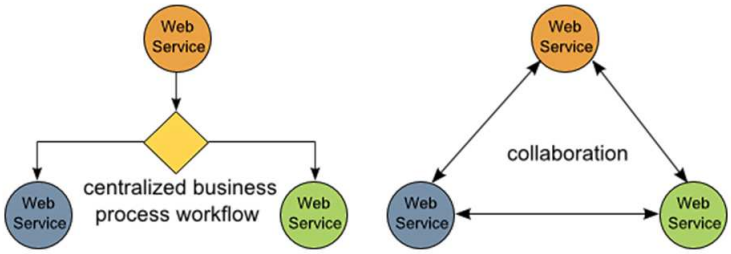

Figure 2.1: Web Service Composition(Orchestration and Choreography).

2.2 Compensation

In Car broker web service several errors can occur during message passing. For example, supplier can send negative quotation to broker if there is not any suitable quotation or supplier may send ordernack message to broker if there is any difficulty to deliver the car.

To handle these kinds of errors, we have used compensation mechanism, where if one process sends any negative response to another then other processes will be informed about that response.

2.3 SPIN

SPIN is a simple PROMELA interpreter where the tool is used to verify the correctness of software model. PROMELA as a modeling language and SPIN is a tool for analyzing the model system specified in PROMELA. The tool can be used for the formal verification of multi-threaded software applications. SPIN is design for various distributed system

like data communication protocol, multi-thread code, client server protocol, network application etc. SPIN can use the entire check of high-level models of concurrent system. It also executes parallel processes. The state of different properties can be checked by this tool. It can use in two basic modes: as a simulator and as a verifier[3]. 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 full LTL model checking system, supporting all 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.

2.3.1 Promela

Promela means Process Meta Language which provides a way for making abstraction of distributed system. During simulations and verifications SPIN checks for the absence of deadlocks, unspecified receptions, and not executable code[4]. It is a verification modeling language where each model of the environment can be verified with spin . PROMELA program consists of *processes*,*message*,*channels*,and *variables*. PROMELA processes behave like real processes where each process is a global object. Message channels and variables can be declared either globally or locally within a process.

2.3.2 mtype Declaration

The introduction of symbolic names can be specified by mtype declaration. In PROMELA language, there can be multiple mtype declaration. If multiple declarations are given, they are equivalent to a single mtype declaration that contains the concatenation of all separate lists of symbolic names.

```
mtype = { ack, nak, order }

is functionally equivalent to the sequence of macro definitions:
```

```
#define ack  5
#define nak  4
#define order   2
```

This data type can be used inside channel declarations, for specifying the type of message fields.

2.3.3 Channel Declaration

Channels are used to transfer messages between active processes. Channels are declared using the keyword *'chan'*, either locally or globally, much like integer variables. The keyword *'chan'* can be followed by one or more names, the syntax is-

```
chan pname=[2] of {mtype,bool}%
```

In this case the channel is named pname which is an array of two elements.

2.3.4 Process

In PROMELA, each process is represented by *'proctype'*, where it refers process type. The state of a variable or of a message channel can only be changed or inspected by processes.

```
proctype sender()
{
   byte state;
    state = 3;
}
```

In this example, the process type consists of a name which is 'sender'. New processes can be spawned using the *'run'* statement, which takes an argument consisting of the name of a *'proctype'*, from which a process is then instantiated. The *'run'* operator can be used

in the body of the *'proctype'* definitions, not only in the initial process. This allows for dynamic creation of processes in PROMELA.

2.3.5 Message passing

Message channels are used to model the transfer of data from one process to another. Message channels are declared by,

```
chan sender  = [2] of {mtype, bit, short, chan};
chan receiver = [0] of {byte};
chan Channel;
```

In this example, the sender channel can store up to two messages in the channel; the message type is indicated between the braces, in this case each message consists of 4 parts.

```
chan producer = [5] of {short}
```

This declares a buffered channel that can store up to five messages of type short (it's capacity is 5). The statement:

```
sender ! ack;
```

sends the value of the expression *ack* to the channel named *'sender'*, that is, it appends the value to the tail of the channel. The statement:

```
receiver ? msg;
```

Receives the message, retrieves it from the head of the channel, and stores it in the variable msg. The channels pass messages in first-in-first-out order. A rendezvous port can be declared as a message channel with the store length of zero. For example, the following:

```
chan port = [0] of {byte}
```

It defines a rendezvous port that can pass messages of type byte. Message interactions via such rendezvous ports are by definition synchronous, i.e. sender or receiver (the one that arrives first at the channel) will block for the contender that arrives second (receiver or sender). When a buffered channel has been filled to its capacity (sending is "capacity" number of outputs ahead of receiving inputs), the default behavior of the channel is to become synchronous, and the sender will block on the next sending. Observe that there is no common message buffer shared between channels. Increasing complexity, as compared to using a channel as unidirectional and point to point, it is possible to share channels between multiple receivers or multiple senders, and to merge independent data-streams into a single shared channel. From this follows that a single channel may also be used for bidirectional communication.

2.3.6 Control flow constructs

There are three control flows constructed in PROMELA. They are the case *selection*, the *repetition* and the *unconditional jump*.

2.3.7 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.

2.3.8 Repetition

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

```
do
:: count = count + 1
:: count = count-1
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 instruction that immediately follows the repetition structure.

2.3.9 Unconditional Jump

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

```
L1: do
:: c = a+b
:: c = a*b
:: (count == 0) -> goto L1
od
```

```
skip;
```

The *'goto'* statement 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. In case selection the simplest construct is the selection structure. Using the relative values of two variables 'p' and 'q', for example, one can write:

```
if
:: (p == q) -> option1
:: (p != q) -> option2
fi
```

The selection structure contains two execution sequences, each preceded by a double colon. One sequence from the list will be executed. A sequence can be selected only if its first statement is executable. The first statement of a control sequence is called a guard. 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. If all guards are not executable, the process will block until one of them can be selected.

2.3.10 Assertions

An important statement is constructed in PROMELA that needs a little explanation is the 'assert' statement. Statements of the form:

```
assert(any_boolean_condition)
```

that's always executable. If a Boolean condition specified holds the statement, has no effect. If, however, the condition does not necessarily hold, the statement will produce an error during verifications with Spin.

2.3.11 Active Proctype

The active keyword can be prefixed to any *'proctype'* definition. If the keyword is present, an instance of that *'proctype'* will be active in the initial system state. Multiple instantiations of that *'proctype'* can be specified with an optional array suffix of the keyword. Example:

```
active proctype sender() { ... }
active [2] proctype receiver() { ... }
```

2.4 Linear Temporal Logic (LTL)

SPIN is a tool which verifies the correctness of the system expressed in LTL (linear temporal logic). SPIN performs the conversion to Bchi automata mechanically based on a simple on-the-fly construction [5]. The automata that are generated formally accept only those (infinite) system executions that satisfy the corresponding LTL formula. In LTL we can find out different types of properties such as safety property, liveliness property etc, which helps to check the model properly. As noted briefly above, we use correctness requirements to formalize system behaviors that are claimed to be impossible, i.e., to formalize the potential violation of correct system behavior. Each positive LTL formula can, of course, be turned into a negative one, and vice versa, by prefixing it with a logical negation operator. 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. The temporal operators are as shown in Table 2.1.

The easiest way to specify an LTL property is to specify it inline. The formula is specified globally with the following syntax:

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

The name is optional, but can be useful when specifying multiple formulae. (Each such formula follows the same basic format.) The formula has the grammar outlined

Operator	Math	SPIN
always	□	[]
eventually	◊	<>
until	ʊ	U

Figure 2.2: LTL Formula.

above, with some extensions. First, white space (newlines, spaces, tabs) can be used anywhere to separate operands and operators. Second, the names of operators can either be abbreviated with the symbols shown above, or spelled out in full as always, eventually, until, implies, and equivalent. [6]. The following two are equivalent:

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

The properties stated in this way are taken as positive properties that must be satisfied by the model. The model checker will perform an automatic negation of the formula to find **counter-examples**. Another improvement in the specification of LTL formula in Spin version 6 and later is that the inline LTL formula can contain any propositional state formula, i.e., it is not restricted to the lower-case propositional symbols from before, where each propositional symbol has to be defined in macro definitions. This means that we can write:

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

Spin can translate LTL formulae into PROMELA, 'never' claims with command line option -f. The *'never'* claims it is generated to encode the Buchi acceptance conditions from the LTL formula. Formally, any omega-run that satisfies the LTL formula is guaranteed to correspond to an accepting run of the never claim. 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.

Chapter 3

Web Service Composition

3.1 Introduction

Web services are self-describing and platform-independent applications that can be invoked over the Web. 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 defined as self-contained, modular units of application logic, which provide business functionality to other applications via an Internet connection. In many cases, a single service is not sufficient to respond to the user's request and often services should be combined through services composition to achieve a specific goal[7]. 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 composed into longer business interactions by providing message exchange protocols that shows 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.

3.2 Car Broker Web Service

We have chosen Car Broker Web Service as an example to our verification. In the car broker web service, the broker negotiates with supplier for car purchase for customer or buyer and arranges loan. The main web service, broker uses two separate partners. A

supplier to find suitable quotation for buyer then delivers car to broker according to the order and a **loan-center** to arrange loan. In this model the **buyer** starts the process by sending an order message to **broker** for a car. Broker then sends a request message for a suitable quotation of that car model to supplier. Broker receives the quotes from supplier and after that, Broker simultaneously sends quotes to buyer, buyer acknowledges about that quote and then broker sends request for loan message to loan-center. Loan-center receives that and sends loan reply to broker. Then broker sends this loan reply to buyer. Buyer receives and confirm for the car order to broker. Broker then sends an order message to supplier. Supplier sends an order acknowledgement message to broker. Broker sends a deliver message to buyer and buyer sends received car message to broker. Then finally broker will send this received message to supplier and to loan-center. So, we can say that all transactions have been successfully done.

3.3 Buyer

Buyer is a person who intends to buy a product from online. In this system at first buyer gives an order to broker to purchase a car. In according to the order, buyer receives a suitable quote from broker. The buyer can either accept or reject the quote. If buyer sends an acknowledgement then broker will send loan reply message to buyer. Then buyer finally confirms for the car and broker sends a deliver message and buyer will send a received message to broker which means buyer got that ordered car successfully. Fig 3.1 shows the architectural view of buyer.

3.4 Broker

Broker leads an important part, because broker communicates with all other processes. After receiving the order from the buyer, broker requests supplier for available quotes. Then broker sends this quote message from supplier to buyer. Then broker receives an

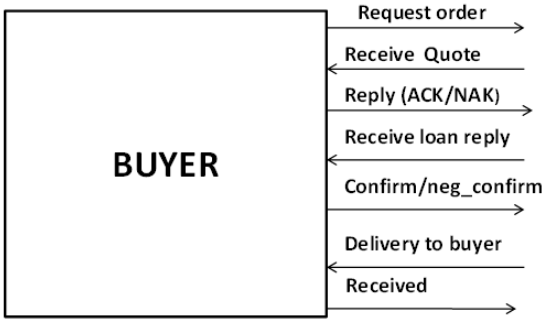

Figure 3.1: Architectural view of Buyer.

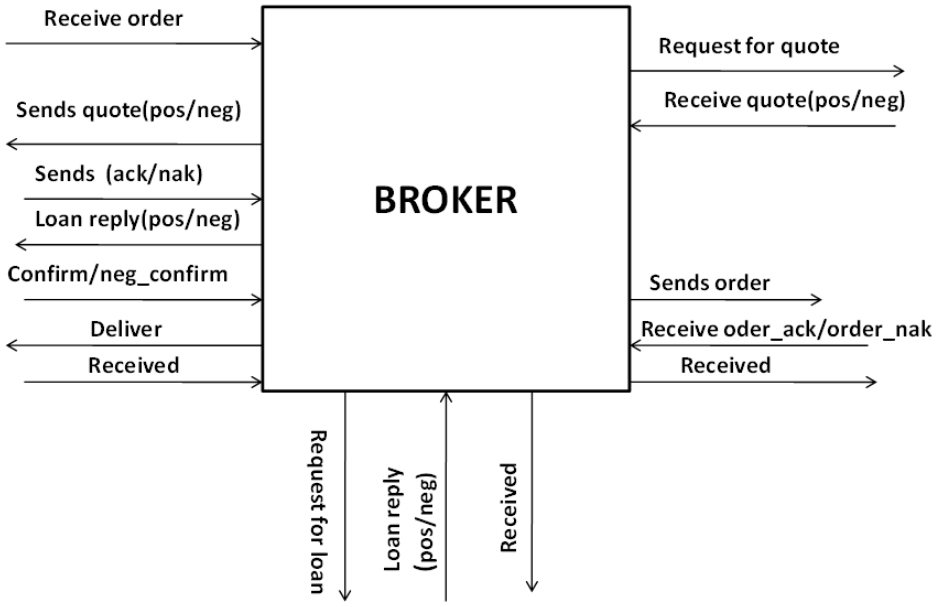

Figure 3.2: Architectural view of Broker.

acknowledgement message from buyer and sends a request for loan message to loan-center. Broker receives loan reply from loan-center and sends this loan reply message to buyer. Then broker gets a final confirmation message from buyer. Then broker sends an order message to supplier for that car. Then broker receives an order acknowledgement message from supplier for the ordered car. Then broker sends a deliver message to buyer. Then broker receives a received message from buyer. Then broker sends this received message to supplier and to loan-center, which means buyer got that ordered car successfully. Fig 3.2 shows the architectural view of broker.

3.5 Supplier

Supplier receives a request for quote message from broker and sends quote reply message to broker. Then supplier receives an order message for that car from broker. Then supplier sends an order acknowledgement message to broker. Then finally supplier receives a received message from broker, which means buyer got that ordered car successfully. Fig 3.3 shows the architectural view of Supplier and Fig 3.4 shows the architectural view of loan center.

3.6 Loan Center

Loan Center is assumed as a lender web service that offers loans to online buyers via broker. Loan-center receives a request for loan message from broker. Then loan-center sends loan reply to broker. After that, loan-center receives a received message from broker, which means buyer got that ordered car successfully. Fig 3.5 shows the architectural view of Car broker and Fig 3.6 shows the Sequential chart without compensation.

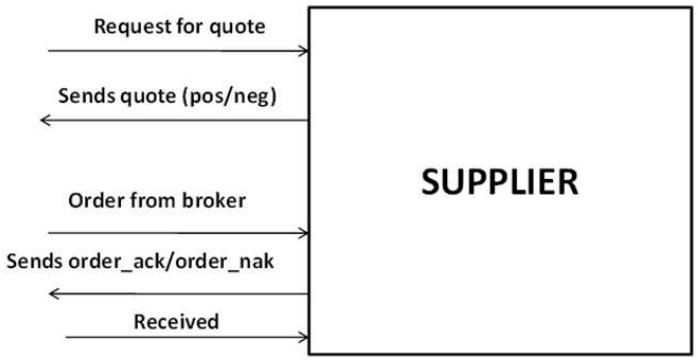

Figure 3.3: Architectural view of sup.

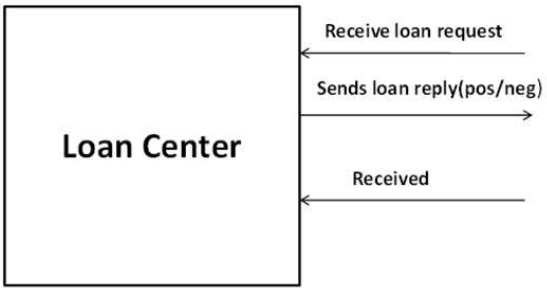

Figure 3.4: Architectural view of loan center.

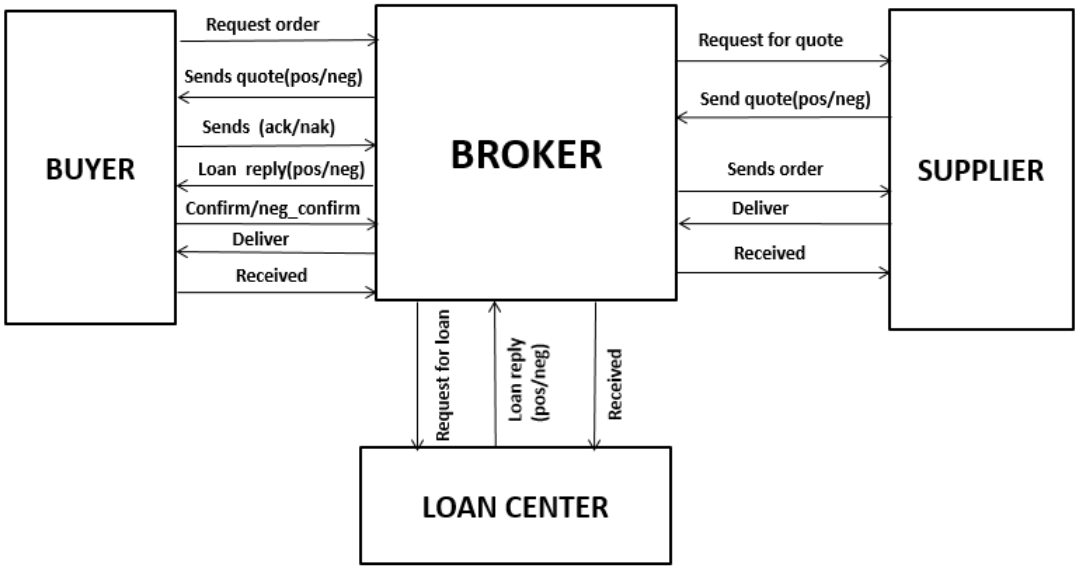

Figure 3.5: Architectural view of Car broker Web Service.

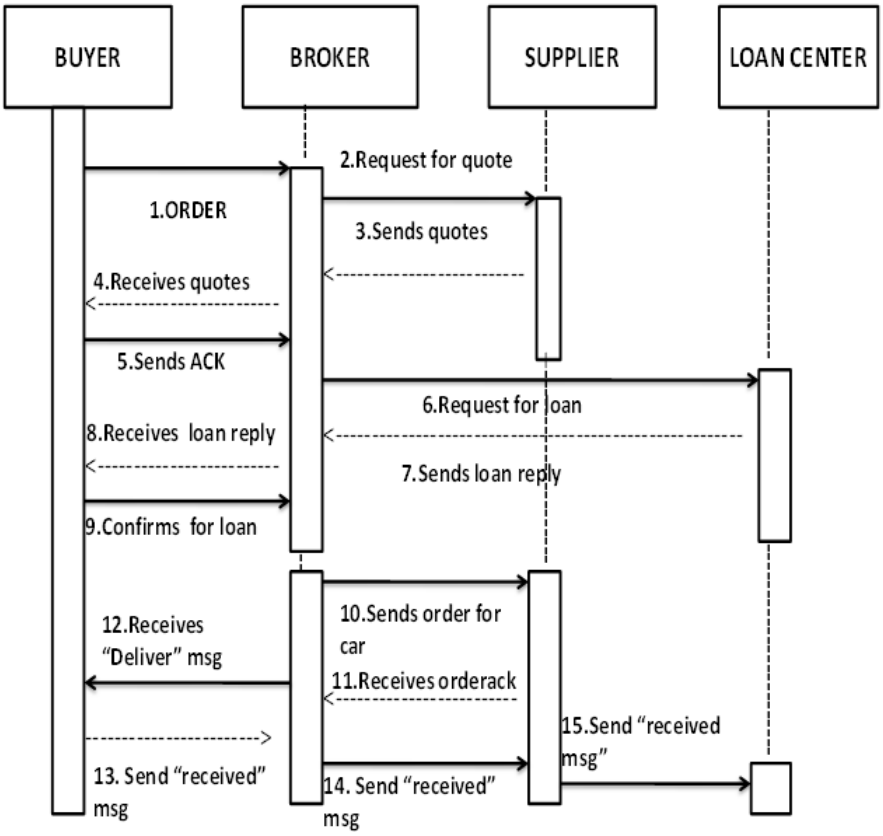

Figure 3.6: Sequential chart without compensation

Chapter 4

Compensation Composition

4.1 Compensation

Business transaction typically means coordination and interaction between multiple partners. This transaction involves hierarchies of active. Business transaction needs to deal with faulty or complications or difficulties that can arise in any stage of the transaction. A rollback mechanism is used to handle those problems in order to provide automaticity to a transaction. However, some transaction requires long period to complete. It's called LONG RUNNING TRANSACTION (LRT), sometimes rollback is not always possible .LRs are usually interactive that communicates with several activities. Handling faulty where multiple partners are involved is 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. is required to handle faulty. 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[8]

4.2 Web Service Compensation

In any stage of transactions, any type of difficulties can occur. For these circumstances, we used compensation mechanism to handle these kinds of situations. To ensure that all positive or negative messages which are sent, are have transmitted accurately to those particular channels in processes through Boolean variables. If there any type of difficulties

occurs then, those assigned true Boolean variables will be false. Then by checking these Boolean variables at each transaction, particular message will be sent to complete the service.

4.3 Compensations of Processes

In the part of **buyer's** compensation, two cases will occur. **First**, when buyer gets suitable or positive quote message which was sent by broker from supplier after requesting, then buyer can cancel the order, if buyer wants to. If buyer wants to cancel the order, buyer will send cancel order message to broker. After receiving this, broker will send this cancel order message to supplier. **Second**, when buyer gets positive loan message which was sent by broker from loan-center after requesting, then buyer can cancel the order, if buyer wants to. If buyer wants to cancel the order, buyer will send cancel order message to broker. After receiving this, broker will send this cancel order message to supplier and to loan-center. In the part of **supplier's** compensation, two cases will occur. **First**, when supplier gets request for quote message from broker, supplier can give negative quote message to broker, if supplier fails to give suitable quotation. If supplier fails to give suitable quotation, broker will send this message to buyer. After receiving this, buyer will send the cancel order message to broker. Then broker will send this cancel order message to supplier. **Second**, after supplier gets the order message from broker, supplier may fail to deliver the car, if there is any difficulty. If supplier fails to deliver the ordered car, supplier will send message, after receiving message broker will send negative deliver message to buyer. After receiving this message, buyer will send cancel order message to broker. Then broker will send this cancel order message to supplier and to loan-center. In the part of **loan-center's** compensation, one case will occur. When loan-center gets the request message from broker after a confirmation from buyer, loan-center may not give loan or may give negative loan message to broker. If loan-center sends no loan message

to broker, broker will send this message to buyer. After receiving this message, buyer will send cancel order message to broker. Then broker will send this message to supplier and to loan-center.

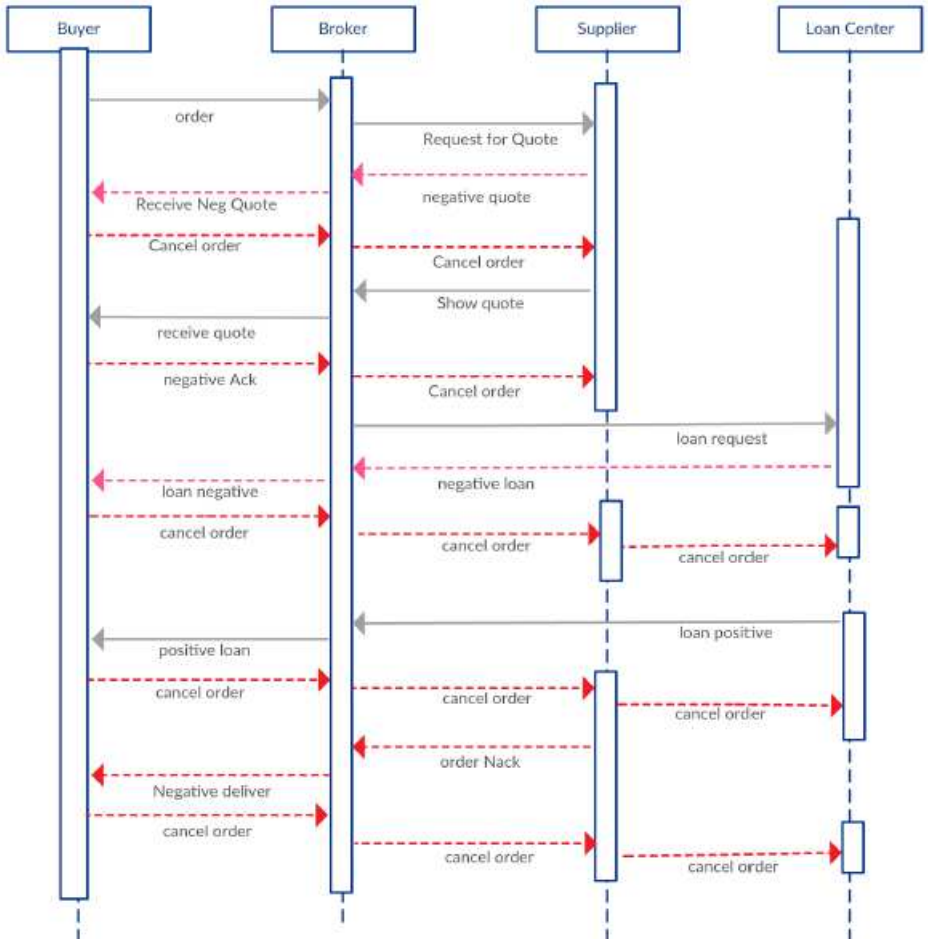

Figure 4.1: Sequential chart of handling compensation

Chapter 5

Modeling in SPIN

In this section we designed a model in **SPIN** tool using **PROMELA** language which logically represents the message based interactions among the processes to explain web service choreography. Each section of the model contains one process that runs parallel. This model consists of a **buyer**, **broker**, **supplier** and **loan-center** processes. When **buyer** orders, it contacts with **broker**. The **broker** system communicates with other systems; **suppler** and **loan-center** and collects information through message passing communication and sends due responses to the **buyer**. Any negative response can occur at any time, because there can occur any sort of difficulties in any system. And so, we have used **compensation** to solve these kinds of scenarios.

5.1 Declaration of mtype

In **PROMELA "mtype"** declaration allows for the introduction of symbolic names for constant values. In this model we use this mtype variable to specify message passing through channel. Corresponding individual name for each message are represented by this data type.

```
mtype={order, ack, quote, rfq, rloan, suporder, orderack, deliver,
received, confirm, neg_quote, loan_pos, loan_neg, cancelorder,
ordernack, deliverneg, cancelorder1, cancelorder2 }
```

Firstly, **buyer** will send an order message to broker. **broker** receives order message

and sends **rfq** message to **supplier** for quotation.**Supplier** receives rfq message and sends quote_confirmation message to broker which can be positive or negative (quote or neg_quote). broker receives quote_confirmation message and sends this to buyer. buyer sends an ack message to broker to confirm. broker receives this message and sends rloan message to loan-center. loan_center receives this message and sends loanrply_confirmation message to broker which can be positive or negative (loan_pos or loan_neg). broker sends this message to buyer. buyer receives this and will send confirmation message to broker which can be positive or negative (confirm or cancelorder1). broker receives this message and sends suporder message to supplier. supplier will send a suplyrorder_confirmation message to broker which can be positive or negative (orderack or ordernack). broker receives this message and sends deliver_confirmation message to buyer which can be positive or negative (deliver or deliverneg).

We have used compensation mechanism to handle difficulties in transactions, when there happens. There can be happen some sort of difficulties in transactions. Some of them are written below: After receiving rfq message from broker, if supplier sends quote_confirmation message to broker which is negative (neg_quote). broker receives this neg_quote message and sends this to buyer.buyer sends a cancelorder message to broker to cancel the order. broker receives this message and sends this cancel message to supplier. Even after, broker sends positive quotation (quote) message from supplier to buyer, buyer can cancel the order, and then broker will send cancel message (cancelorder) to supplier. Even after sending positive loanrply_confirmation message (loan_pos) to buyer, buyer may send a cancelorder message to broker. Then, in terms of canceling the order progress, broker will send this cancelorder message to supplier and loan-center. Even, loan_center might be unable to give loan, so loan-center will send negative loanrply_confirmation message (loan_neg) to broker. Then broker will send this message to buyer. And so, buyer will cancel the order by sending cancelorder message to broker. After receiving this, broker will send this cancelorder message to supplier and to loan-center. Even, there

might occur any faulty about supplying the car. Before that, all other transactions were successfully done, but supplier may not be able to supply the ordered car. So, supplier sends a negative suplyrorder_confirmation (ordernack) to broker. Then broker sends a negative deliver_confirmation message (deliverneg) to buyer. After receiving this, buyer will send a cancelorder message to broker. Then broker will send this cancelorder message to supplier and to loan-center. Even, loan-center might be unable to give loan, so loan-center will send negative loanrply_confirmation message (loan_neg) to broker. Then broker will send this message to buyer. And so, buyer will cancel the order by sending cancelorder message to broker. After receiving this, broker will send this cancelorder message to supplier and to loan_center. Even, there might occur any faulty about supplying the car. Before that, all other transactions were successfully done, but supplier may not be able to supply the ordered car. So, supplier sends a negative suplyrorder_confirmation (ordernack) to broker. Then broker sends a negative deliver_confirmation message (deliverneg) to buyer. After receiving this, buyer will send a cancelorder message to broker. Then broker will send this cancelorder message to supplier and to loancenter.

5.2 Channel Declaration

Message channels are used to model the exchange of data between processes. We have used one type of channel for regular message transaction including messages for compensations. Channel declaration have to be global, otherwise message's data cannot be received. Each channel contains data length and in this model we have used message type and Boolean type. Message type data named **"mtype"** contains information between two services and **Boolean** type contains confirmation data that has been confirmed by most recent previous transaction among two services.

```
chan buyer_broker1 =[2] of {mtype,bool};
chan buyer_broker3=[2] of {mtype,bool};
```

```
chan buyer_broker4=[2] of {mtype,bool};
chan buyer_broker5=[2] of {mtype,bool};
chan buyer_broker6=[2] of {mtype,bool};
chan buyer_broker7=[2] of {mtype,bool};
chan buyer_broker8=[2] of {mtype,bool};
chan buyer_broker9=[2] of {mtype,bool};
chan buyer_broker10=[2] of {mtype,bool};
chan broker_buyer2=[2] of {mtype,bool};
chan broker_buyer1=[2] of {mtype,bool};
chan broker_buyer3=[2] of {mtype,bool};
chan broker_buyer4=[2] of {mtype,bool};
chan broker_buyer5=[2] of {mtype,bool};
chan broker_buyer6=[2] of {mtype,bool};
chan broker_supplier1=[2] of {mtype,bool};
chan broker_supplier2=[2] of {mtype,bool};
chan broker_supplier3=[2] of {mtype,bool};
chan broker_supplier4=[2] of {mtype,bool};
chan broker_supplier5=[2] of {mtype,bool};
chan broker_supplier6=[2] of {mtype,bool};
chan broker_supplier7=[2] of {mtype,bool};
chan broker_supplier8=[2] of {mtype,bool};
chan broker_supplier9=[2] of {mtype,bool};
chan broker_supplier10=[2] of {mtype,bool};
chan broker_supplier11=[2] of {mtype,bool};
chan supplier_broker1=[2] of {mtype,bool};
chan broker_loanstar1=[2] of {mtype,bool};
chan broker_loanstar2=[2] of {mtype,bool};
```

```
chan broker_loanstar3=[2] of {mtype,bool};
chan broker_loanstar4=[2] of {mtype,bool};
chan broker_loanstar5=[2] of {mtype,bool};
chan broker_loanstar6=[2] of {mtype,bool};
chan loanstar_broker1=[2] of {mtype,bool};
```

5.3 Boolean types Representation

Since **Boolean** type represents only true or false value, that's why we use this property for checking the data type whether the most recent message transaction is occurred or not. Each service transaction message contains one confirmation towards next service. We have initialized some **Boolean** values by true and some are by false. We considered the fact that, negative occurrences will not be initialized by true, but only when it will occur, then it will be true.

```
bool buyerorder_confirmation=true,confirmation=true;
bool quote_confirmation=true,quote_confirmationbyr=true;
bool ack_confirmation=true,loanreq_confirmation=true;
bool brokerorder_confirmation=true;
bool suplyrorder_confirmation=true,deliver_confirmation=true;
bool loanrply_confirmation=true;
bool loanrply_confirmationbyr=true;
bool cancelorder_confirmationbyr=true;
bool cancelorder_confirmationbrk=true;
bool deliverneg_confirmation=true;
bool cancelorder1_confirmation=true;
bool ok_confirmation=true;
```

5.4 Process Creation

In **PROMELA** each process is represented by **"proctype"**. It refers to the process type and it is used for declaring process behavior. A process type consists of a name and a 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 executes by active processes where processes need not any run statement. In our model we have used four processes and these are buyer, broker, supplier, loan-center. Each process has its own methods for checking the overall service. A sample process that has been used in our model is given below:

```
active proctype loanstar()
{
L5: do::
broker_loanstar1?rloan(loanreq_confirmation);
/*reqest for loan after quote pos & buyer ok_cnfrm*/
if
::loanrply_confirmation==true->
broker_loanstar1!loan_pos(loanrply_confirmation); /*loan pos to broker*/
::loanrply_confirmation==false->
loanstar_broker1!loan_neg(loanrply_confirmation);->
broker_loanstar2?cancelorder(loanrply_confirmationbyr);
->goto L5/*loan neg to broker so buyer canceled*/
fi;
/*when loan positive*/
if
::loanrply_confirmation==true->
broker_loanstar5?received(confirmation);
```

```
->goto L5  /*car received*/
::ok_confirmation==false->
broker_loanstar6?cancelorder1(cancelorder1_confirmation);
->goto L5 /*loan pos but buyer canceled*/
::suplyrorder_confirmation==false->
broker_loanstar4?cancelorder1(cancelorder1_confirmation);
->goto L5 /*supplier ordernack (no car) buyer canceled*/
fi;
od;
}
```

5.5 Model Representation

To encoding our model in SPIN, we can represent the whole transaction method in one section including compensation mechanism. In buyer process, buyer only communicates with broker. buyer gives order to broker

```
buyer_broker1! order(buyerorder_confirmation);
```

and receives quote reply from broker.

```
::quote_confirmationbyr==true->broker_buyer2?quote(quote_confirmationbyr);
/* receive quote from broker */
::quote_confirmationbyr==true-> buyer_broker1!ack(ack_confirmation);
 /* buyer send ack to broker */
```

After sending acknowledgement message to broker, buyer receives loan reply from broker.

```
::loanrply_confirmation==true->
```

```
broker_buyer6?loan_pos(loanrply_confirmation);->goto L8
```

After receiving positive quotation and loan, buyer finally confirms for the car and sends a confirm message to broker.

```
::loanrply_confirmation==true-> buyer_broker9!confirm(ok_confirmation)
;->goto L11
```

At last, buyer receives a deliver message from broker and sends a received message to broker.

```
::suplyrorder_confirmation==true->broker_buyer1?deliver(deliver_confirmation)
;-> buyer_broker6!received(confirmation);->goto L1;
```

buyer will cancel the order for three reasons, when quote is negative or loan is negative or supply order confirmation is false

```
::quote_confirmationbyr==false->broker_buyer3?
neg_quote(quote_confirmationbyr)
;->buyer_broker5!cancelorder1(cancelorder1_confirmation)
;->goto L1

::loanrply_confirmation==false->broker_buyer4?loan_neg
(loanrply_confirmationbyr)
;->buyer_broker3!cancelorder(cancelorder_confirmationbyr)
;->goto L1
::suplyrorder_confirmation==false->broker_buyer5?deliverneg
(deliverneg_confirmation)
;->buyer_broker4!cancelorder1(cancelorder1_confirmation)
;->goto L1
```

Even after receiving positive quotation message or positive loan message buyer can cancel the order, if wants to. buyer receives positive quote but can cancel the order -

```
::quote_confirmationbyr==true->broker_buyer2?quote(quote_confirmationbyr);
::ack_confirmation==false->buyer_broker8!cancelorder1
(cancelorder1_confirmation)
;->goto L1 /*quote pos but buyer canceled*/
```

buyer receives positive loan reply but can cancel the order-

```
::loanrply_confirmation==true->broker_buyer6?loan_pos(loanrply_confirmation)
;->goto L8
::ok_confirmation==false->buyer_broker10!cancelorder1
(cancelorder1_confirmation);->goto L1
```

In broker process, it communicates with all processes and passes messages of these processes among them. broker receives order message from buyer.

```
buyer_broker1?order(buyerorder_confirmation);
::buyerorder_confirmation==true->broker_supplier1!rfq(confirmation);
/* broker send rfq to supplier*/
```

broker gets quotation reply from supplier that can be positive or negative.

```
::quote_confirmation==true->broker_supplier2?quote(quote_confirmation)
;->broker_buyer2!quote(quote_confirmationbyr);
::quote_confirmation==false->broker_supplier3?neg_quote(quote_confirmation);
```

If broker gets acknowledgement message then requests for loan to loan-center

```
buyer_broker1?ack(ack_confirmation); /*ack sent after quote sent*/
::ack_confirmation==true->broker_loanstar1!rloan(loanreq_confirmation);
```

broker will receive loan reply from loan-center which can be positive or negative and will send loan reply to buyer.

```
::loanrply_confirmation==true->broker_loanstar1?loan_pos
(loanrply_confirmation);->broker_buyer6!loan_pos
(loanrply_confirmation);->goto L2
```

```
::loanrply_confirmation==false->loanstar_broker1?loan_neg
(loanrply_confirmation)
;->broker_buyer4!loan_neg(loanrply_confirmationbyr);
```

If broker gets confirmation from buyer after sending positive loan, then orders car to supplier.

```
::ok_confirmation==true->buyer_broker9?confirm(ok_confirmation)
;->broker_supplier1!suporder(brokerorder_confirmation)
;->goto L12
```

broker gets an order confirmation from supplier which can be positive or negative If broker gets positive order confirmation, then sends deliver message to buyer.

```
::suplyrorder_confirmation==true->broker_supplier2?orderack
(suplyrorder_confirmation);->broker_buyer1!
deliver(deliver_confirmation);->buyer_broker6?received(confirmation)
;->broker_supplier8!received(confirmation)
;->broker_loanstar5!received(confirmation);->goto L4
```

Then broker receives a received confirmation message from buyer and sends it to supplier and to loan-center. All these messages were sent from broker to others, when

broker received positive messages from others. We have used compensation mechanism and how broker handles compensation, is given below: If supplier sends negative quotation, broker sends this to buyer and buyer will cancel the order and broker will send this cancel message to supplier.

```
::quote_confirmation==false-> broker_supplier3?neg_quote(quote_confirmation)
;->broker_buyer3!neg_quote(quote_confirmationbyr)
;->buyer_broker5?cancelorder1(cancelorder1_confirmation)
;->broker_supplier6!cancelorder1(cancelorder1_confirmation)
;->goto L4
```

Even after sending positive quotation buyer can cancel order, then broker will send cancel message to supplier.

```
::ack_confirmation==false->buyer_broker8?cancelorder1
(cancelorder1_confirmation);->broker_supplier9!cancelorder
(cancelorder_confirmationbrk);->goto L4
```

If loan-center cannot give loan, after sending this to buyer, broker will get cancel order from buyer. Then broker will send this cancel message to supplier and to loan-center.

```
::loanrply_confirmation==false->buyer_broker3?
cancelorder(cancelorder_confirmationbyr);->broker_supplier4!cancelorder
(cancelorder_confirmationbrk)
;-> broker_loanstar2!cancelorder(cancelorder_confirmationbyr)
;->goto L4
```

Even after sending positive loan to buyer, buyer can cancel order. Then after receiving cancel order message, broker will send this cancel order message to supplier and to loan-center.

```
::ok_confirmation==false->buyer_broker10?cancelorder1
(cancelorder1_confirmation)
;->broker_supplier11!cancelorder1(cancelorder1_confirmation)
;-> broker_loanstar6!cancelorder1(cancelorder1_confirmation)
;->goto L4 /*loan pos but buyer canceled order*/
```

If for some reason supplier fails to send car, broker gets an ordernack confirmation from supplier. broker informs buyer and buyer will send cancel order message. Then broker will send this cancel message to supplier and to loan-center.

```
::suplyrorder_confirmation==false->
supplier_broker1?ordernack(suplyrorder_confirmation)
;->broker_buyer5!deliverneg(deliverneg_confirmation)
;->buyer_broker4?cancelorder1(cancelorder1_confirmation)
;->broker_supplier7!cancelorder1(cancelorder1_confirmation)
;->broker_loanstar4!cancelorder1(cancelorder1_confirmation);->goto L4
```

In supplier process, it only communicates with broker. supplier gets request message from broker, then sends quotation reply to broker.

```
broker_supplier1?rfq(confirmation); /*request for quotation*/
::quote_confirmation==true->broker_supplier2!quote(quote_confirmation);
::quote_confirmation==false->broker_supplier3!neg_quote(quote_confirmation)
;->broker_supplier6?cancelorder1(cancelorder1_confirmation);->goto L6
```

After sending positive quotation, supplier gets an order message for car from broker. Then supplier sends supply order confirmation message to broker. If supplier can supply the car then receives a received message from broker.

```
::ok_confirmation==true->broker_supplier1?suporder(brokerorder_confirmation)
;->goto L7
```

```
::brokerorder_confirmation==true->
broker_supplier2!orderack(suplyrorder_confirmation)
;->broker_supplier8?received(confirmation);->goto L6
```

supplier may receive cancel order message, even after giving the positive quotation to broker. If buyer wants to cancel the order even after getting the positive quotation, so supplier will get a cancel order message from broker.

```
::ack_confirmation==false->broker_supplier9?cancelorder
(cancelorder_confirmationbrk);->goto L6
```

Even after, other transactions are positive,before the order of car from broker, supply may get a cancel order message from broker.

```
::ok_confirmation==false->broker_supplier11?cancelorder1
(cancelorder1_confirmation);->goto L6
```

After sending positive quotation, for other difficulty, supplier may get a cancel order message from broker.

```
::loanrply_confirmation==false->broker_supplier4?
cancelorder(cancelorder_confirmationbrk);->goto L6
```

If there are some difficulties in transactions of supplier and so we have used compensation mechanism in supplier process. After getting the request message for quotation, supplier may give negative quotation to broker and so will receive a cancel order message from broker.

```
::quote_confirmation==false->broker_supplier3!neg_quote(quote_confirmation)
;->broker_supplier6?cancelorder1(cancelorder1_confirmation);->goto L6
```

After getting order message from broker, supplier may send ordernack message to broker and so will get a cancel order message from broker.

```
::suplyrorder_confirmation==false->supplier_broker1!ordernack
(suplyrorder_confirmation)
;->broker_supplier7?cancelorder1(cancelorder1_confirmation);-> goto L6
```

In the loan-center process, loan-center only communicates with broker. loan-center gets request for loan from broker and sends loan reply to broker.

```
broker_loanstar1?rloan(loanreq_confirmation);
::loanrply_confirmation==true->broker_loanstar1!loan_pos
(loanrply_confirmation);
::loanrply_confirmation==false->loanstar_broker1!loan_neg
(loanrply_confirmation)
;->broker_loanstar2?cancelorder(loanrply_confirmationbyr);->goto L5
```

If loan-center has given loan then it will receive a received message from broker.

```
::loanrply_confirmation==true->broker_loanstar5?received(confirmation);
Even after giving loan, loan may receive a cancel order message from broker.
::ok_confirmation==false-> broker_loanstar6?cancelorder1
(cancelorder1_confirmation);->goto L5
```

There might happen some difficulties in the transactions of loan-center and broker, and so we have used compensation mechanism. If loan-center cannot give loan then it will send negative loan reply and broker will send a cancel order message.

```
::loanreq_confirmation==false->loanstar_broker1!loan_neg
(loanrply_confirmation)
;->broker_loanstar2?cancelorder(loanrply_confirmationbyr);
```

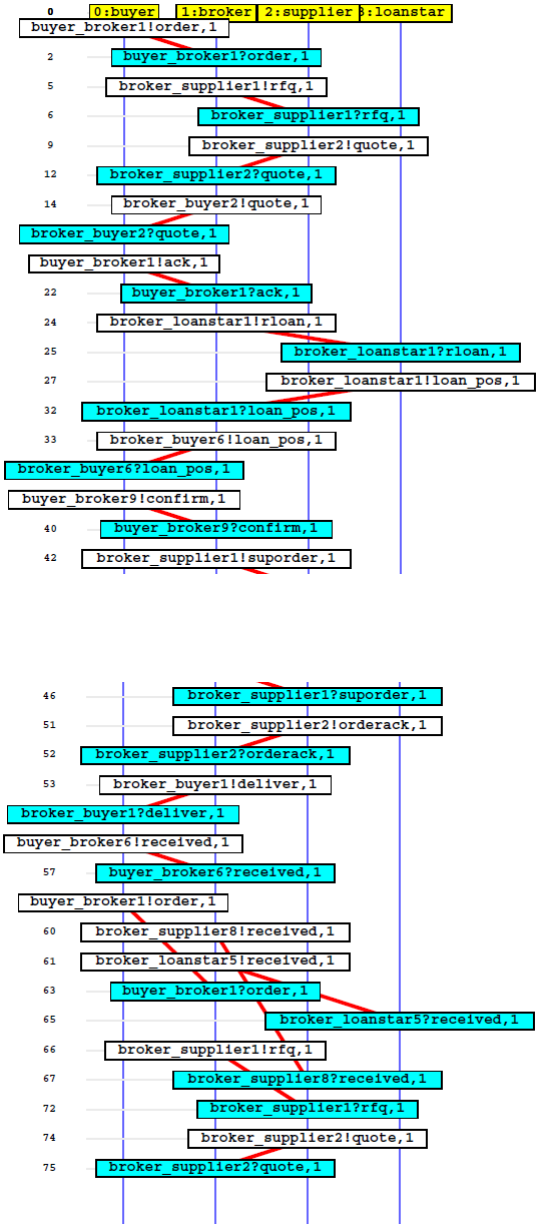

Figure 5.1: Simulation result of car broker web service

5.6 Auotmata 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 transaction of one state or event to another. It is a representation of state diagram which indicates all the transactions in a process. Our model consists of four independent processes buyer, broker, supplier, loan-center named as loanstar. Each automata view includes the compensation mechanism, because we have used compensation mechanism in each process of its.

5.6.1 BUYER

Buyer is a person who intends to buy a product from online. In this system at first buyer gives an order to broker to purchase a car. In according to the order, buyer receives a suitable quote from broker. The buyer can either accept or reject the quote. If buyer sends an acknowledgment then broker will send loan reply message to buyer. Then buyer finally confirms for the car and broker sends a deliver message and buyer will send a received message to broker which means buyer got that ordered car successfully. When buyer gets suitable or positive quote message which was sent by broker from supplier, then buyer can cancel the order, if buyer wants to. If buyer wants to cancel the order, buyer will send cancel order message to broker. After receiving this, broker will send this cancel order message to supplier. When buyer gets positive loan message which was sent by broker from loan-center after requesting, then buyer can cancel the order, if buyer wants to. If buyer wants to cancel the order, buyer will send cancel order message to broker. After receiving this, broker will send this cancel order message to supplier and to loan-center. Buyer will cancel the order, if gets negative quote or negative loan or negative deliver message.

5.6.1.1 Automata View of BUYER

Figure 5.2 shows the Automata View of Buyer.

5.6.2 BROKER

Broker leads an important part, because broker communicates with all other processes. After receiving the order from the buyer, broker requests supplier for available quotes. Then broker sends this quote message from supplier to buyer. Then broker receives an acknowledgment message from buyer and sends a request for loan message to loan-center. Broker receives loan reply from loan-center and sends this loan reply message to buyer. Then broker gets a final confirmation message from buyer. Then broker sends an order message to supplier for that car. Then broker receives an order acknowledgment message from supplier for the ordered car. Then broker sends a deliver message to buyer. Then broker receives a received message from buyer. Then broker sends this received message to supplier and to loan-center, which means buyer got that ordered car successfully. How broker handles compensation is written below: If supplier sends negative quotation, broker sends this to buyer and buyer will cancel the order and broker will send this cancel message to supplier. Even after sending positive quotation buyer can cancel order, then broker will send cancel message to supplier. If loan-center cannot give loan, after sending this to buyer, broker will get cancel order from buyer. Then broker will send this cancel message to supplier and to loan-center. Even after sending positive loan to buyer, buyer can cancel order. Then after receiving cancel order message, broker will send this cancel order message to supplier and to loan-center. If for some reason supplier fails to send car, broker gets an ordernack confirmation from supplier and broker informs buyer and buyer will send cancel order message. Then broker will send this cancel message to supplier and to loan-center.

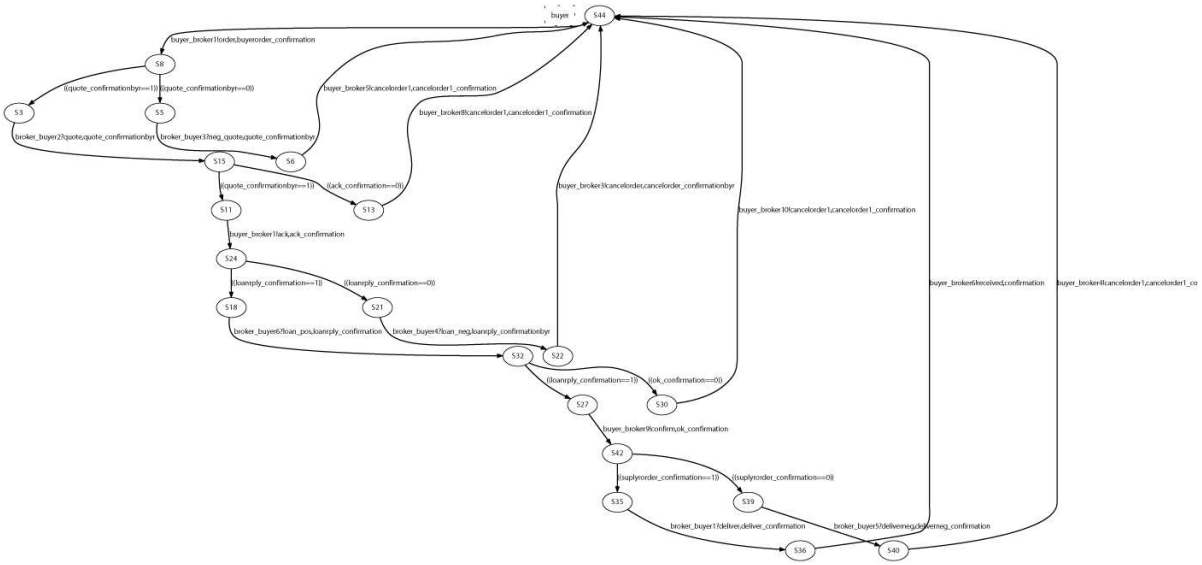

Figure 5.2: Automata view of Buyer

5.6.2.1 Automata View of broker

Figure 5.3 shows the Automata View of Broker.

5.6.3 SUPPLIER

Supplier receives a request for quote message from broker and sends quote reply message to broker. Then supplier receives an order message for that car from broker. Then supplier sends an order acknowledgement message to broker. Then finally supplier receives a received message from broker, which means buyer got that ordered car successfully. When supplier gets request for quote message from broker, supplier can give negative quote message to broker, if supplier fails to give suitable quotation. If supplier fails to give suitable quotation, broker will send this message to buyer. After receiving this, buyer will send the cancel order message to broker. Then broker will send this cancel order message to supplier. After supplier gets the order message from broker, supplier may fail to deliver the car, if there is any difficulty. If supplier fails to deliver the ordered car, supplier will send message, after receiving message broker will send negative deliver message to buyer. After receiving this message, buyer will send cancel order message to broker. Then broker will send this cancel order message to supplier and to loan-center.

5.6.3.1 Automata View of Supplier

Figure 5.4 shows the Automata View of Supplier.

5.6.4 LOAN CENTER

Loan Center is assumed as a lender web service that offers loans to online buyers via broker. Loan-center receives a request for loan message from broker. Then loan-center sends loan reply to broker. After that, loan-center receives a received message from broker, which means buyer got that ordered car successfully. When loan-center gets the

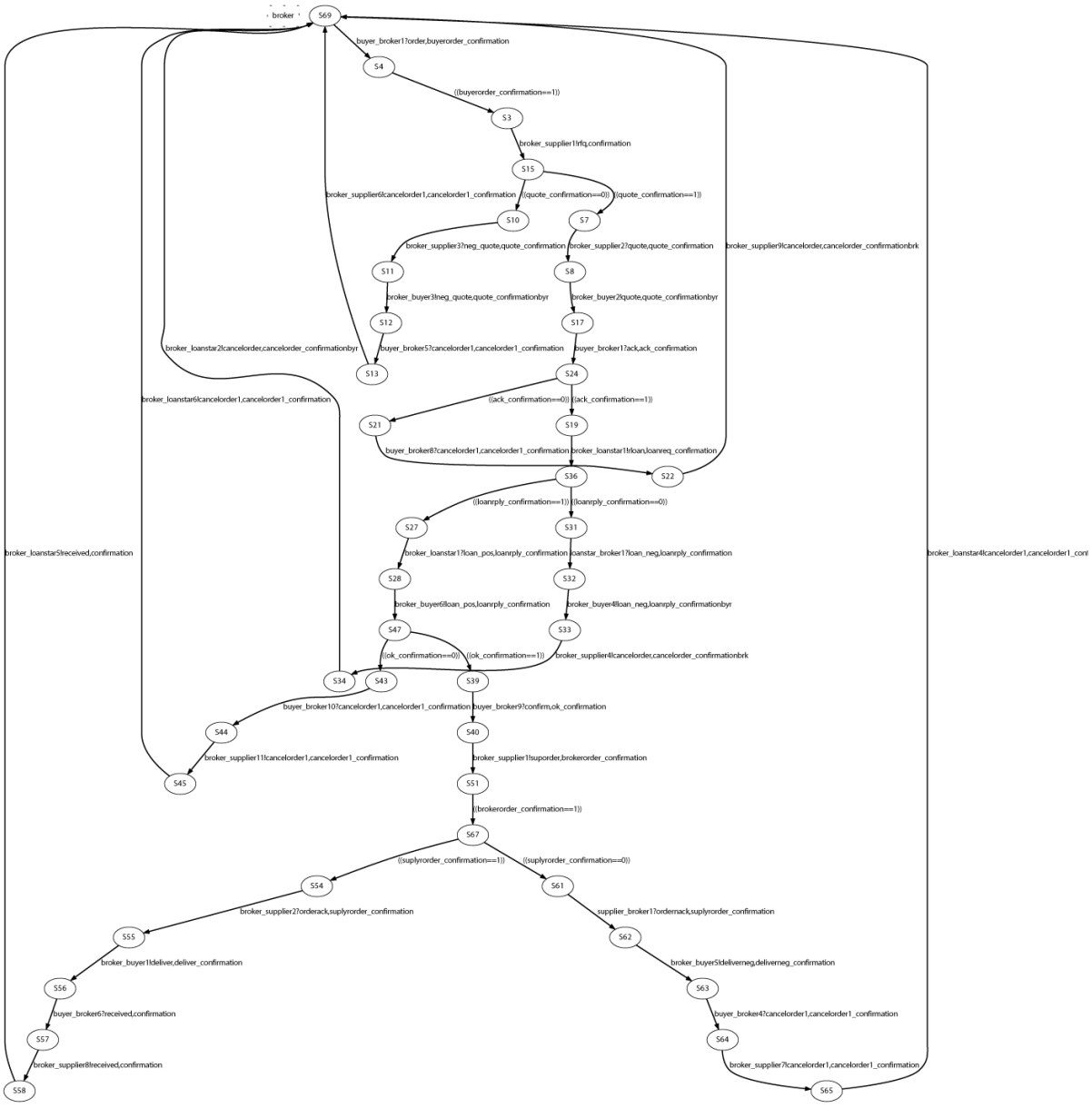

Figure 5.3: Automata view of Broker

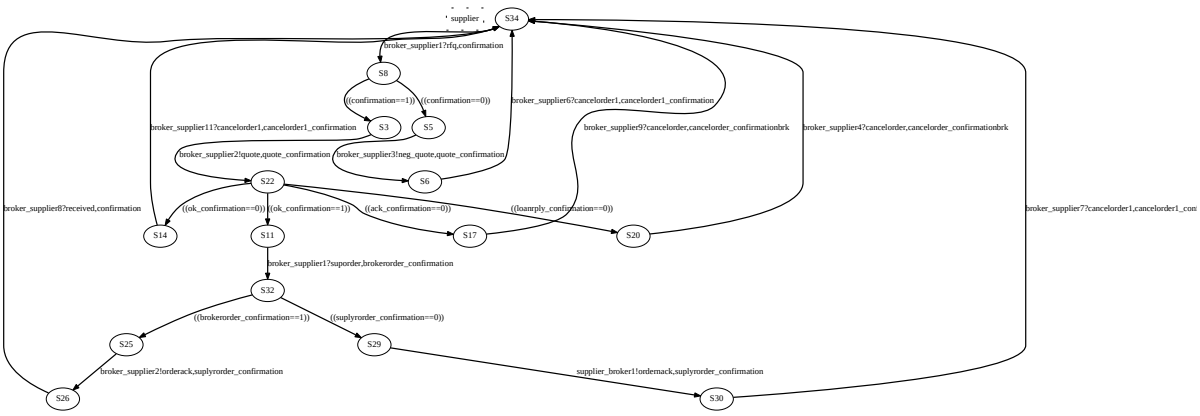

Figure 5.4: Automata view of Supplier

request message from broker after a confirmation from buyer, loan-center may not give loan or may give negative loan message to broker. If loan-center sends no loan message to broker, broker will send this message to buyer. After receiving this message, buyer will send cancel order message to broker. Then broker will send this message to supplier and to loan-center.

5.6.4.1 Automata View of loan Center

Figure 5.5 shows the Automata View of Loan Center.

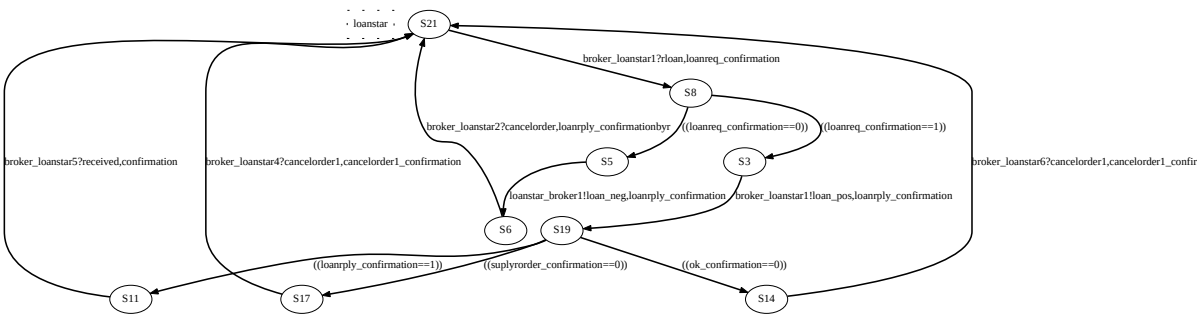

Figure 5.5: Automata view of Loan Center

Chapter 6

Verification

6.1 Verification

In verification model, SPIN allows checking whether an execution of a system satisfies or violates the correctness of the system. We use PROMELA as the formal specification language. We define our requirement as LTL (Linear Temporal Logic) formula. In the following sections, all the formula and related definitions are coded in PROMELA.

6.1.1 LTL Verification

LTL refers linear temporal logic. LTL formulas are evaluated on sequences of states (linear). Several model-checking tools use LTL as a property specification language. The model checker SPIN is a prominent example of such an automated verification tool. LTL has a lot of property for checking the model. so we have used choose safety and liveness property for verify our model.

6.1.1.1 Safety

For measuring the correctness of this composition we can perform a comprehensive verification with spin to prove some safety properties. In this properties check the loan reply confirmation and buyer confirmation and receive supply order confirmation. This property definite below- It is always true that if the loan reply confirmation and buyer confirmation become true then it implies that supply order confirmation will be true.

the LTL representation are given below-

```
Ltl p{[]((loanrply_confirmation==true) &&(ok_confirmation==true))->
((suplyrorder_confirmation==true))}
```

we have written some ltl code to check whether this logic is logic is working on overall code or not.

6.1.1.2 Liveness:

LTL liveness property refers that in the future in any time comething good will happen. so for checking this we have used a simple LTL . The formal representation of testing this property are described below- if the ok confirmation and loan reply confirmation become true then eventually the receive confirmation from buyer. here the LTL representation of checking the liveness property -

```
ltl p{(ok_confirmation==true) && (suplyrorder_confirmation==true)
->(<>(confirmation==true))}
```

Chapter 7

Conclusion

7.1 Summary

In a web service, there occur business transactions. Sometimes there may occur any sorts of difficulties or complications in the terms of transactions. To provide a good web service to the customers, its composition needs to get verified and those complications are needed to get handled. And so our motivation was to verify composition and compensation of the web service. We have used car broker web service as our example in SPIN to verify this model's composition and to use compensation mechanism to provide a good web service model.

Bibliography

[1] S. H. Ripon, *Process Algebraic Support for Web Service Composition.*

[2] M. P. Raman Kazhamiakin, "A case study on web service."

[3] M. Ben-Ari, *Principles of the Spin Model Checker*, ISBN: 1846287693, 9781846287695.

[4] S. M. Shamim Ripon and K. M. I. ud Din, "Verification of a security adaptive protocol suite using spin."

[5] A. Y. Y. R. Shamim H Ripon, Syed Fahin Ahmed and K. M. Imtiaz-UD-Din., "Formal analysis of a ranked neighbour manet protocol suite," 2014.

[6] "Linear temporal logic," www.cs.colostate.edu/ france/CS614/Slides/Ch5-Summary.pdf.

[7] P. Maggi and R. Sisto, "Using spin to verify security properties of cryptographic protocols."

[8] P. S. Ryan, "Modelling and analysis of security protocols, addison-wesley, harlow, england," 2001.

Appendix A

PROMELA ENCODING

```
mtype={order,ack,quote,rfq,rloan,suporder,orderack,deliver,received,confirm,
neg_quote,loan_pos,loan_neg,cancelorder,ordernack,deliverneg,
cancelorder1,cancelorder2}

chan buyer_broker1 =[2] of {mtype,bool};
chan buyer_broker3=[2] of {mtype,bool};
chan buyer_broker4=[2] of {mtype,bool};
chan buyer_broker5=[2] of {mtype,bool};
chan buyer_broker6=[2] of {mtype,bool};
chan buyer_broker7=[2] of {mtype,bool};
chan buyer_broker8=[2] of {mtype,bool};
chan buyer_broker9=[2] of {mtype,bool};
chan buyer_broker10=[2] of {mtype,bool};
chan broker_buyer2=[2] of {mtype,bool};
chan broker_buyer1=[2] of {mtype,bool};
chan broker_buyer3=[2] of {mtype,bool};
chan broker_buyer4=[2] of {mtype,bool};
chan broker_buyer5=[2] of {mtype,bool};
chan broker_buyer6=[2] of {mtype,bool};
chan broker_supplier1=[2] of {mtype,bool};
```

```
chan broker_supplier2=[2] of {mtype,bool};
chan broker_supplier3=[2] of {mtype,bool};
chan broker_supplier4=[2] of {mtype,bool};
chan broker_supplier5=[2] of {mtype,bool};
chan broker_supplier6=[2] of {mtype,bool};
chan broker_supplier7=[2] of {mtype,bool};
chan broker_supplier8=[2] of {mtype,bool};
chan broker_supplier9=[2] of {mtype,bool};
chan broker_supplier10=[2] of {mtype,bool};
chan broker_supplier11=[2] of {mtype,bool};
chan supplier_broker1=[2] of {mtype,bool};
chan broker_loanstar1=[2] of {mtype,bool};
chan broker_loanstar2=[2] of {mtype,bool};
chan broker_loanstar3=[2] of {mtype,bool};
chan broker_loanstar4=[2] of {mtype,bool};
chan broker_loanstar5=[2] of {mtype,bool};
chan broker_loanstar6=[2] of {mtype,bool};
chan loanstar_broker1=[2] of {mtype,bool};
bool buyerorder_confirmation=true,confirmation=true;
bool quote_confirmation=true,quote_confirmationbyr=true;
bool ack_confirmation=true,loanreq_confirmation=true;
bool brokerorder_confirmation=true;
bool suplyrorder_confirmation=true,deliver_confirmation=true;
bool loanrply_confirmation=true;
bool loanrply_confirmationbyr=true;
bool cancelorder_confirmationbyr=true;
bool cancelorder_confirmationbrk=true;
```

```
bool deliverneg_confirmation=true;
bool cancelorder1_confirmation=true;
bool ok_confirmation=true;
active proctype buyer()
{
L1: do::
buyer_broker1! order(buyerorder_confirmation);
if
::quote_confirmationbyr==true->broker_buyer2?quote(quote_confirmationbyr);
::quote_confirmationbyr==false->broker_buyer3?neg_quote(quote_confirmationbyr)
;->buyer_broker5!cancelorder1(cancelorder1_confirmation);->goto L1
fi;

if
::quote_confirmationbyr==true->buyer_broker1!ack(ack_confirmation);
::ack_confirmation==false->buyer_broker8!cancelorder1
(cancelorder1_confirmation);->goto L1
fi;

if
::loanrply_confirmation==true->broker_buyer6?loan_pos(loanrply_confirmation)
;->goto L8
::loanrply_confirmation==false->broker_buyer4?loan_neg
(loanrply_confirmationbyr)
;->buyer_broker3!cancelorder(cancelorder_confirmationbyr)
;->goto L1
fi;
```

```
L8 : if
::loanrply_confirmation==true->buyer_broker9!confirm(ok_confirmation)
;->goto L11
::ok_confirmation==false-> buyer_broker10!cancelorder1
(cancelorder1_confirmation);->goto L1

fi;

L11 : if
::suplyrorder_confirmation==true->broker_buyer1?deliver(deliver_confirmation)
;->buyer_broker6!received(confirmation);->goto L1;
::suplyrorder_confirmation==false-> broker_buyer5?deliverneg
(deliverneg_confirmation)
;->buyer_broker4!cancelorder1(cancelorder1_confirmation);->goto L1
fi;

od;
}

active proctype broker()
{
L4:do::
buyer_broker1?order(buyerorder_confirmation);
if
::buyerorder_confirmation==true-> broker_supplier1!rfq(confirmation);
fi;
```

```
if
::quote_confirmation==true->broker_supplier2?quote(quote_confirmation)
;->broker_buyer2!quote(quote_confirmationbyr);
::quote_confirmation==false->broker_supplier3?neg_quote(quote_confirmation)
;->broker_buyer3!neg_quote(quote_confirmationbyr)
;->buyer_broker5?cancelorder1(cancelorder1_confirmation)
;->broker_supplier6!cancelorder1(cancelorder1_confirmation);->goto L4
fi;
buyer_broker1?ack(ack_confirmation);
::ack_confirmation==true-> broker_loanstar1!rloan(loanreq_confirmation);
::ack_confirmation==false-> buyer_broker8?cancelorder1
(cancelorder1_confirmation)
;->broker_supplier9!cancelorder(cancelorder_confirmationbrk);->goto L4
fi;

if
::loanrply_confirmation==true->broker_loanstar1?loan_pos
(loanrply_confirmation)
;-> broker_buyer6!loan_pos(loanrply_confirmation);->goto L2
::loanrply_confirmation==false-> loanstar_broker1?loan_neg
(loanrply_confirmation)
;->broker_buyer4!loan_neg(loanrply_confirmationbyr);
fi;

if
::loanrply_confirmation==false-> buyer_broker3?cancelorder
(cancelorder_confirmationbyr)
```

```
;->broker_supplier4!cancelorder(cancelorder_confirmationbrk)
; ->broker_loanstar2!cancelorder(cancelorder_confirmationbyr);->goto L4
fi;
L2 : if
::ok_confirmation==true->buyer_broker9?confirm(ok_confirmation)
;->broker_supplier1!suporder(brokerorder_confirmation);->goto L12
fi;
L12 : if
::brokerorder_confirmation==true->goto L9
fi;

/*when car is ordered and reply from supplier*/
L9 : if
::suplyrorder_confirmation== true->broker_supplier2?orderack
(suplyrorder_confirmation)
;->broker_buyer1!deliver(deliver_confirmation)
;->buyer_broker6?received(confirmation)
;->broker_supplier8!received(confirmation)
;->broker_loanstar5!received(confirmation)
;->goto L4
::suplyrorder_confirmation==false-> supplier_broker1?ordernack
(suplyrorder_confirmation)
;-> broker_buyer5!deliverneg(deliverneg_confirmation)
;-> buyer_broker4?cancelorder1(cancelorder1_confirmation)
;->broker_supplier7!cancelorder1(cancelorder1_confirmation)
;->broker_loanstar4!cancelorder1(cancelorder1_confirmation);->goto L4
fi;
```

```
od;
}

active proctype supplier()
{
L6: do::

broker_supplier1?rfq(confirmation); /*request for quotation*/
if
::quote_confirmation==true-> broker_supplier2!quote(quote_confirmation);
::quote_confirmation==false-> broker_supplier3!neg_quote(quote_confirmation)
;->broker_supplier6?cancelorder1(cancelorder1_confirmation); ->goto L6
fi;
/*when quote pos*/
if
::ok_confirmation==true->broker_supplier1?suporder(brokerorder_confirmation)
;->goto L7
::ok_confirmation==false->broker_supplier11?
cancelorder1(cancelorder1_confirmation);->goto L6
::ack_confirmation==false->broker_supplier9?
cancelorder(cancelorder_confirmationbrk);->goto L6
::loanrply_confirmation==false->broker_supplier4?
cancelorder(cancelorder_confirmationbrk);->goto L6

fi;
/*when supplier got order for car*/
```

```
L7 : if
::brokerorder_confirmation==true-> broker_supplier2!orderack
(suplyrorder_confirmation)
;->broker_supplier8?received(confirmation);->goto L6
::suplyrorder_confirmation==false-> supplier_broker1!ordernack
(suplyrorder_confirmation)
;->broker_supplier7?cancelorder1(cancelorder1_confirmation);->goto L6
fi;
od;
}
active proctype loanstar()
{
L5:do::
broker_loanstar1?rloan(loanreq_confirmation);
if
::loanrply_confirmation==true->broker_loanstar1!loan_pos
(loanrply_confirmation);
::loanrply_confirmation==false-> loanstar_broker1!loan_neg
(loanrply_confirmation)
;->broker_loanstar2?cancelorder(loanrply_confirmationbyr);->goto L5
fi;
/*when loan positive*/
if
::loanrply_confirmation==true->broker_loanstar5?received(confirmation)
;->goto L5
::ok_confirmation==false-> broker_loanstar6?cancelorder1
(cancelorder1_confirmation);->goto L5
```

```
::suplyrorder_confirmation==false-> broker_loanstar4?
cancelorder1(cancelorder1_confirmation);->goto L5
fi;
od;
}
ltl p{((loanrply_confirmation==true)&&(ok_confirmation==true))
-> (<>(suplyrorder_confirmation==true))}
ltl p{[](ok_confirmation==true) && (suplyrorder_confirmation==true)
->((confirmation==true))}
ltl p{[](suplyrorder_confirmation==false)->((cancelorder1_confirmation==true))}
```