Verification Of Web Service Composition And Compensation By Using FSP

By

Farhana Sultana

&

Fahmida Rahman

Computer Science and Engineering

East West University

Fall 2015

Verification Of Web Service Composition And Compensation By Using FSP

Submitted By

Farhana Sultana

ID: 2011-3-60-004

&

Fahmida Rahman

ID: 2012-1-60-025

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

Fall 2015

Declaration

We, hereby certify that our thesis work solely to be our own scholarly work. To the best of our knowledge, it has not been collected from any source without the due acknowledgement and permission. It is being submitted in fulfilling the requirements for the degree of Bachelor of Science in Computer Science and Engineering. It has not been submitted, either in whole or in part for a degree or examination at this or any other university.

Farhana Sultana

(2011-3-60-004)

Fahmida Rahman

(2012-1-60-025)

Letter of Acceptance

The Project entitled **"Verification of Web Service Composition And Compensation By Using FSP"** submitted by Farhana Sultana [2011-3-60-004] and Fahmida Rahman [2012-1-60-025] to the Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh in the semester of Fall 2015 is approved satisfactory in partial fulfillment of requirements for award of the degree of Bachelor of Science in Computer Science and Engineering.

_____

Dr. Shamim Hasnat Ripon

SUPERVISOR and CHAIRPERSON

Department of Computer Science and Engineering

East West University

Dhaka-1212, Bangladesh

Abstract

A platform-independent software component available in the distributed environment of the Internet is titled as Web service. Many Business organizations are publishing their applications functionalities on the web. A web service has a limited functionality alone. So to support business to business interactions it is a crying need to aggregate web services and assembled them in a goal oriented interface. To provide atomicity to a transaction where multiple partners are involved handling faults are both difficult and critical. A possible solution of the problem would be that the system designer can provide a mechanism to compensate the actions that cannot be undone automatically. In this project we have composed a car broker system and implemented a compensation mechanism that will compensate all services from their point of cancelation. We have modeled the service choreography in FSP and used LTSA tool to animate the transitions. Every model should be verified before implementation, we tried to verify the system composition using property processes available in FSP.

Acknowledgement

It is with enormous appreciation that we acknowledge the contribution of our supervisor, **Dr. Shamim Hasnat Ripon**, Chairperson and Associate Professor of Department of Computer Science and Engineering, East West University. Without his proper guidance, encouragement and support this thesis would have remained a dream. We consider it as an honor to work with him. We are also indebted to our parents, other professors of the department and friends for their support and encouragements. Finally, thanks to the Almighty, who gave us the patience to complete the task successfully.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLE

CHAPTER 1

Introduction

1.1 Introduction and Motivation

Business transactions need multiple partner involvement, coordination and interaction with each other. Many business companies or enterprises publish their applications functionalities on the web using a web service format. Web services are defined as self-contained, modular units of application logic, which provide business functionality to other applications through an Internet connection. Each service provider is a self-contained software system having its own threads of control.

In this technological era business applications like web services allows greater efficiency and availability for business. A web service alone has a limited functionality which may not be sufficient to respond to the user's request. Whereas a composition of several web services can achieve a specific goal. From a user perspective, the composition might be considered as a simple web service, even though it is composed of several web services. In an essence, the aggregation is a collaboration of many Web service providers.

Models are simplified representations of real-world entities. We model something to better understand it. We can use models to focus on interesting aspects, visualize potential outcomes and create mechanisms to test and verify an approach. We need model checking to verify correctness properties such as the absence of deadlocks and similar critical states that can cause the system to crash. Every model should be verified before implementation. There are various languages to model a system and verify it properly. BPEL, cCSP and FSP are most handful language to model a system with their notations. Among them FSP has the most expressive and powerful approach to visualize the system. To provide atomicity to a transaction handling faults where multiple partners are involved are both difficult and critical. A possible solution of the problem would be that the system designer can provide a mechanism to compensate the actions that cannot be undone automatically. In BPEL compensation is expressed in a XML notation, in cCSP it is expressed in a compensation pair but it can be expressed in FSP as a separate process and represented elaborately.

1.2 Objectives

The objectives of our project are as follows:

- Analyzing the web service composition in respect to the composition mechanism orchestration and choreography.
- Modeling a composition using Finite State Process (FSP) notations and Labeled Transition System Analyzer (LTSA) tool.
- Introducing a compensation mechanism as a fault handler that could handle all the failed transactions and could manage all the compensation processes of every component processes.
- Verifying the designed model as it is specified in the model specification. Ensuring that in a concurrent execution all synchronizing points executes properly and no deadlock and such critical states occur that violate the correctness properties.

1.3 Contribution

Our contributions in the project are as follows:

We have used Finite State Process (FSP) notations to describe the model and LTSA tool to generate the corresponding Labeled transition Diagrams. We select Car Broker Web Service as our model. We analyzed the model and identified various components of the web service as well as the composition among the services. Then implement the system according to their interactions.

We have implemented a compensation mechanism which will describe the compensation actions from the point of cancelation. Each process has its own compensation process and a main compensation process to control the whole mechanism.

We added some safety properties to verify synchronizations among processes in a concurrent execution and checked the correctness properties such as the absence of deadlocks and similar critical states that can cause the system to crash.

1.4 Outline

Chapter 1: Firstly we represent about our motivation to work, Specify our objectives and then the contribution that we have made.

Chapter 2: Web service composition and two ways to compose the web services (Choreography and orchestration). Then a brief description is given about Finite State Process (FSP) which is used to specify our model and about LTSA tool to compile FSP notations. After that we discussed about Compensating CSP (cCSP).

Chapter 3: This chapter describes about car broker service composition including the contribution of each web services in the system and how the compensation process works.

Chapter 4: The coding representation of our service in FSP.

Chapter 5: Define some Safety properties in order to verify our web service composition and compensation by composing them with required safety properties.

Chapter 6: Differences of using notations in FSP and cCSP are discussed in brief.

Chapter 7: At last, in this chapter we summarized our work and give a definition about our future plan.

CHAPTER 2

Background

2.1 Web Service and Composition

Web services are distributed, independent processes which communicate with each other through the exchange of messages. The coordination between business processes is particularly crucial as it includes the logic that makes a set of different software components become a whole system. Web services provided by various organizations can be interconnected to implement business collaborations, leading to composite web services. Business collaborations require interactions driven by explicit process models. Web services are driven by the paradigm of the so called service oriented architecture (SOA), which describes the relationships, that exists among service providers, service consumers, and service brokers and there by provides an abstract execution environment for web services. We refer to a service implemented by combining the functionality provided by other web services as a *composite service*, and the process of developing a composite web service as *service composition.* [1, 2]

There are two key aspects in web service composition those are choriography and orchestration.

2.1.1 Orchestration

"A single director in control"

In Orchestration several web services are involved in an operation. In the operation one central process, can be a web service, leads the other web services and coordinates the execution of different parts of the operation on different web services. As all the data are exchanged via the central coordinator of the orchestration so it needs to understand the specific composition logic and other web services need not to know that they are being incorporated in a composition process and taking part in a larger business process. Every

component service considers the central coordinator just as one consumer of its service. Orchestration describes how web services interact with each other through messages, including the business logic and execution order. [3, 4]

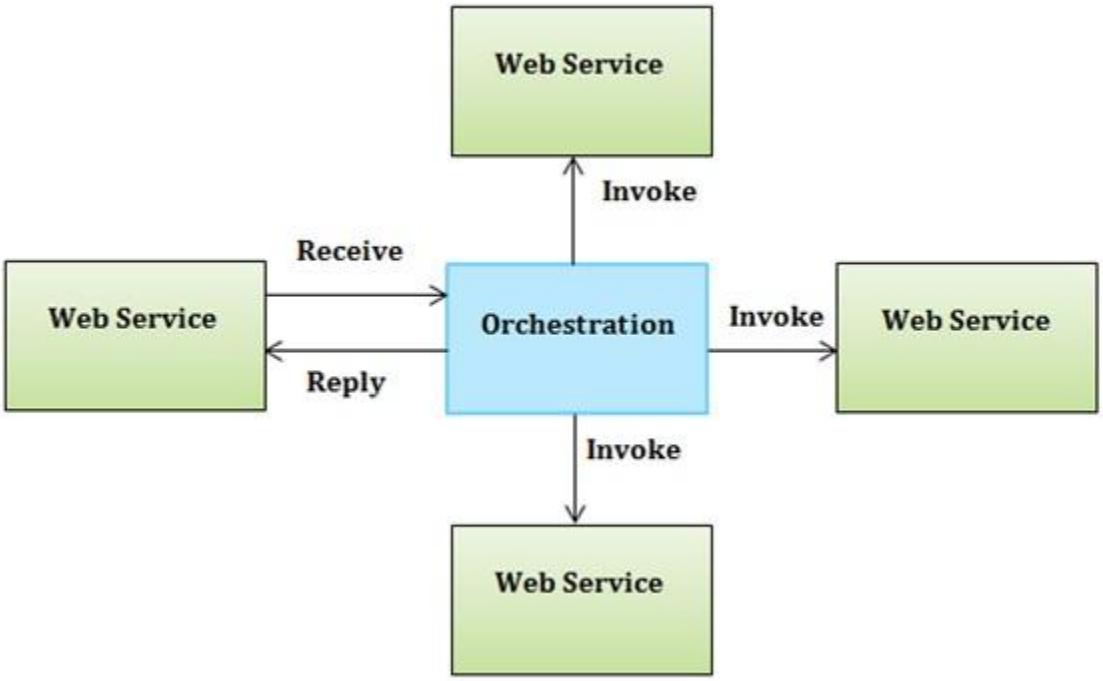

Figure 2.1: Composition of Web Services with Orchestration [3]

2.1.2 Choreography

"Distributed control"

Choreography is based on collaboration; it does not rely on a central coordinator. In choreography each web service needs to be aware of the business process. All participants need to know when to execute its operations, what messages to exchange, when to exchange the messages and with whom it to interact. [3, 4]

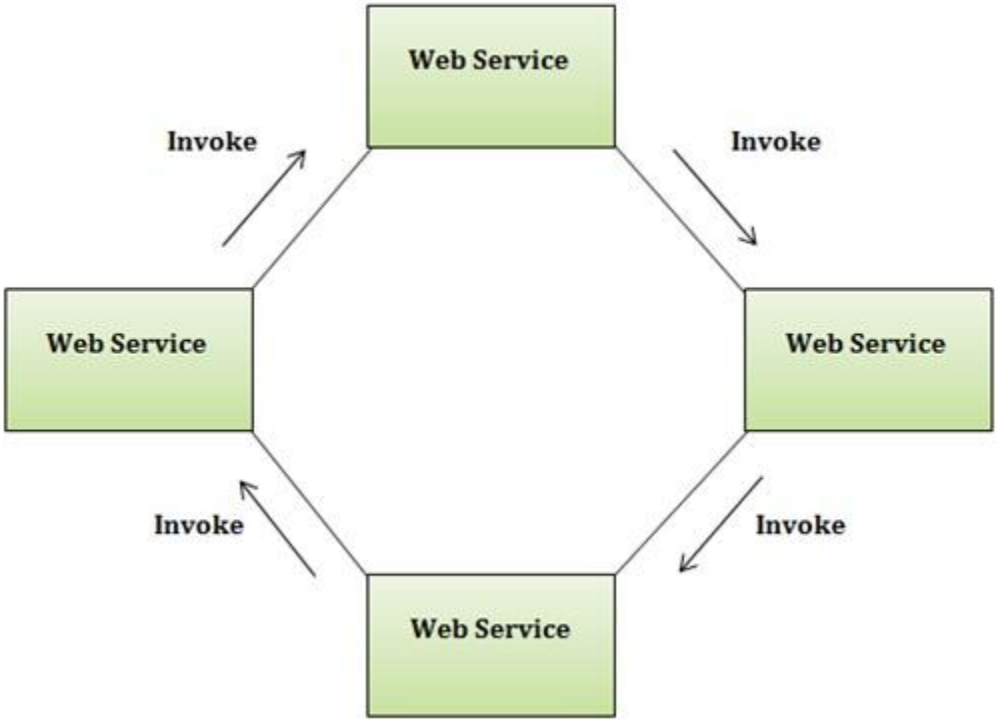

Figure 2.2: Composition of Web Services with Choreography [3]

2.2 FSP

FSP stands for Finite State Processes. Finite State Processes is an algebraic notation to describe process models. The constructed FSP can be used to model the exact transition of workflow processes through a modeling tool such as the Labeled Transition System Analyzer (LTSA), which provides compilation of an FSP into a Labeled Transition System. Models are described using state machines, known as Labeled Transition Systems LTS. These are described textually as finite state processes (FSP) and displayed and analyzed by the LTSA analysis tool. This tool gives an opportunity to test the model workflows before implementing the model. LTS is the graphical form and FSP is the algebraic form. [5]

FSP consists of Action Prefix, Process Definition, Choice, Indexed Processes and Actions, Guarded Actions, properties, Constant and Range Declarations, Variable Declaration, Process Alphabets and so on.

2.2.1 Modeling Processes in FSP

A service can be a process or a composition of several processes. A process is the execution of a sequential program. It is modeled as a finite state machine which transits from state to state by executing a sequence of atomic actions. In practical terms, an action might be a communication, a signals, or perhaps, traditional execution of a task. [6]

In FSP processes are two types such as Primitive Processes and Composite Processes.

Primitive Processes

Primitive processes are defined using action prefix, choice and recursion. Both action labels and local process names can be indexed or non-indexed.

Action Prefix "->"

Action prefix defines a transition between states. If x is an action and P a process then the action prefix (x->P) describes a process that initially engages in the action x and then behaves exactly as described by P. The action prefix operator "->" always has an action on its left and a process on its right. In FSP, identifiers beginning with a lowercase letter denote actions and identifiers beginning with an uppercase letter denote processes. A primitive process definition is terminated by a full stop. [6]

The following definition describes the process `CLOCK` which repeatedly engages in the action `tick`.

```
CLOCK = (tick -> CLOCK).
```

The LTS corresponding to the definition above is:

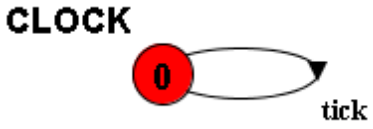

Figure 2.3: LTSA representation of `CLOCK` process.

Choice "|"

Choice is represented as a state with more than one outgoing transition. Choice operator "|" can express a choice of more than two actions. Choices are of two types, Deterministic and

Non-Deterministic. The FSP language provides mechanisms for deterministic and non-deterministic choice. Their definitions are as follows:

Deterministic Choice: If x and y are actions then (x->P | y->Q) describes a process which initially engages in either of the actions x or y. The execution of action x will have subsequent behavior described by P. Similarly, the execution of y will have subsequent behavior described by Q.

The example describes the behavior of a dispensing machine which dispenses coffee if the red button is pressed and tea if the blue button is pressed.

```
DRINKS = (red->coffee->DRINKS | blue->tea->DRINKS).
```

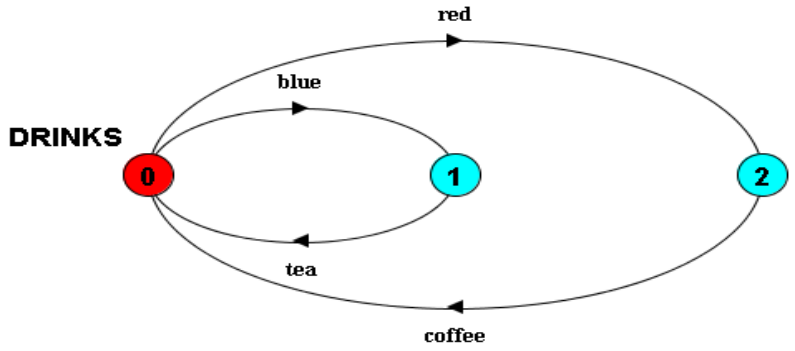

Figure 2.4: LTSA representation of Deterministic process `DRINKS`.

Non-deterministic Choice: The process (x->P | x->Q) is said to be *non-deterministic* since after the action x, it may behave as either P or Q. The COIN process defined below and drowns as a state machine in Figure is an example of a non-deterministic process. [6, 7]

```
COIN = (toss->heads->COIN | toss->tails->COIN).
```

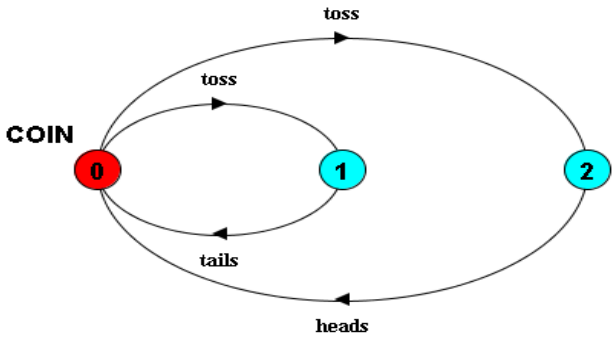

Figure 2.5: LTSA representation of Non-deterministic process `COIN`.

Conditional

A conditional takes the form: if *expr* then *local_process* else *local_process*. FSP supports only integer expressions. A non-zero expression value causes the conditional to behave as the local process of the then part; a zero value causes it to behave as the local process of the else part. The else part is optional, if omitted and expr evaluates to zero the conditional becomes the STOP process.

Example:

```
LEVEL = (read[x:0..2] -> if x>=1 then (high -> LEVEL)   else
(low -> LEVEL)).
```

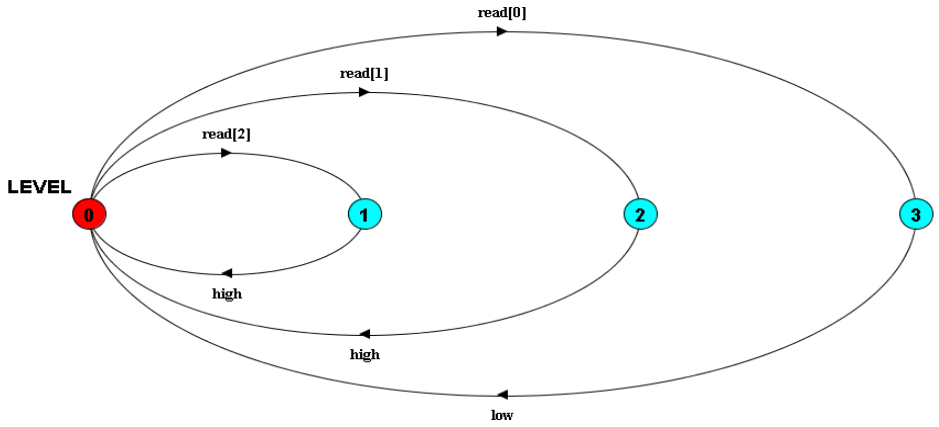

Figure 2.6: LTSA representation of `LEVEL` process.

Guarded Actions

It is often useful to define particular actions as conditional, depending on the current state of the machine. We use Boolean guards to indicate that a particular action can only be selected if its guard is satisfied. The choice (When B x->P | y->Q) means that when the guard B is true then the actions x and y are both eligible to be chosen, otherwise if B is false then the action x cannot be chosen. The example below is a process that encapsulates a count variable. The count can be increased by `inc` operations and decreased by `dec` operations. The count is not allowed to exceed N or be less than zero. [6]

```
COUNT (N=3)    = COUNT[0],

COUNT[i:0..N] = (when(i<N) inc->COUNT[i+1]

                |when(i>0) dec->COUNT[i-1]).
```

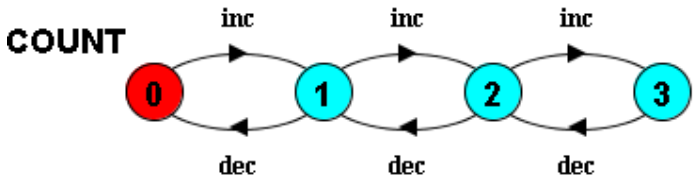

Figure 2.7: LTSA representation of `COUNT` process.

FSP supports only integer expressions; consequently, the value zero is used to represent false and any non-zero value represents true.

Sequential Composition in FSP

If P is a sequential process and Q is a local process, then P;Q represents the sequential composition such that when P terminates, P;Q becomes the process Q.

Composite Processes

Composite processes are defined using parallel composition, relabeling and hiding.

Parallel Composition in FSP

If P and Q are two processes then (P || Q) represents the concurrent execution of P and Q. The operator || is the parallel composition operator. Parallel composition yields a process, which is represented as a state machine in the same way as any other process. The state machine representing the composition generates all possible interleavings of the traces of its component processes. Composite process definitions are always preceded by "||" to distinguish them from primitive process definitions. For example, the process:

```
ITCH = (scratch->STOP).
```

has a single trace consisting of the action `scratch`. The process :

```
CONVERSE = (think->talk->STOP).
```

has the single trace `think->talk`. The composite process:

```
||CONVERSE_ITCH = (ITCH || CONVERSE).
```

has the following traces

```
think->talk->scratch
```

```
think->scratch->talk

scratch->think->talk
```

The state machine representing the composition is formed by the Cartesian product of its constituents. [6]

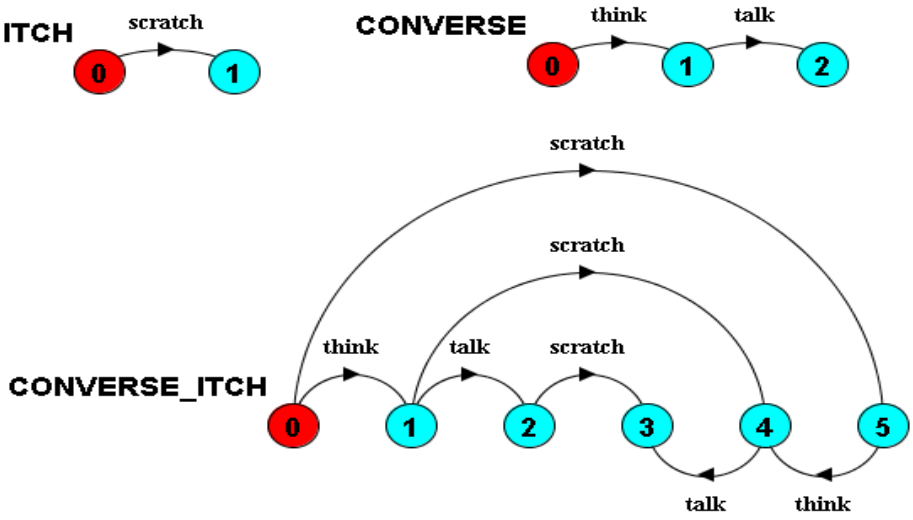

Figure 2.8: LTSA representation of Composition `CONVERSE_ITCH`.

Modeling interaction - Shared Actions

If processes in a composition have actions in common, these actions are said to be shared. Concurrent processes that share actions interact with each other for synchronization. A shared action must be executed at the same time by all the processes that participate in that shared action while unshared actions may be arbitrarily interleaved. For an example, a process that manufactures an item and then signals that the item is ready for use by a shared `ready` action. A user can only use the item after `ready` action occurs. Two items can be made before the first is used which is an undesirable behavior and we do not wish the `MAKER` process to get ahead in this way. The solution is to ensure that the user indicates that the item is used. The `used` action is shared with the `MAKER` who now cannot proceed to manufacture another item until the first is used. The interaction between `MAKER` and `USER` in such a way is an example of a handshake. A handshake is an action acknowledged by another action. Handshake protocols are widely used to structure interactions between processes. [6]

```
MAKER = (make->ready->used->MAKER).
```

```
USER = (ready->use->used->USER).

||MAKER_USER = (MAKER || USER).
```

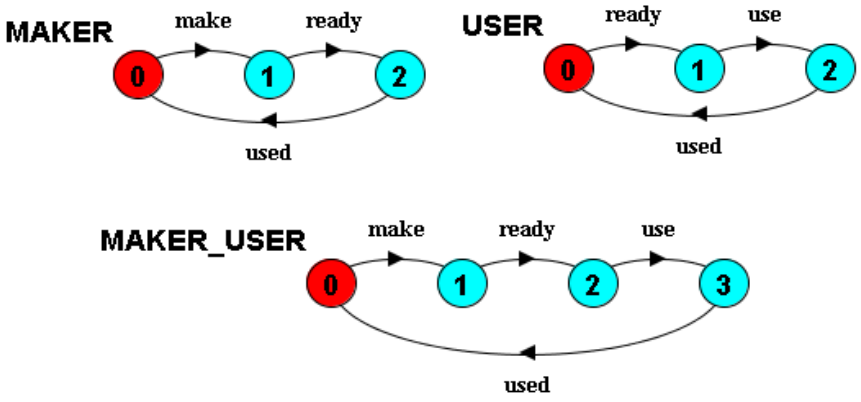

Figure 2.9: LTSA representation of Composition `MAKER_USER`.

Relabeling Actions in FSP

Relabeling functions are applied to processes and change the names of action labels. This is usually done to ensure that composite processes synchronize on the desired actions. /{newlabel_1/oldlabel_1,...newlabel_n/oldlabel_n} is the general form of the relabeling function. For an example, a server process that provides some service and a client process that invokes the service are described below.

```
CLIENT = (call->wait->continue->CLIENT).

SERVER = (request->service->reply->SERVER).
```

Using relabeling we can associate `call` action of the `CLIENT` with the `request` action of the `SERVER` and similarly the `reply` and the `wait` actions.

```
||CLIENT_SERVER = (CLIENT || SERVER)
                  / {call/request reply/wait}.
```

The effect of applying the relabeling function can be seen in the state machine as the label `call` replaces `request` in `SERVER` and `reply` replaces `wait` in `CLIENT`. [6]

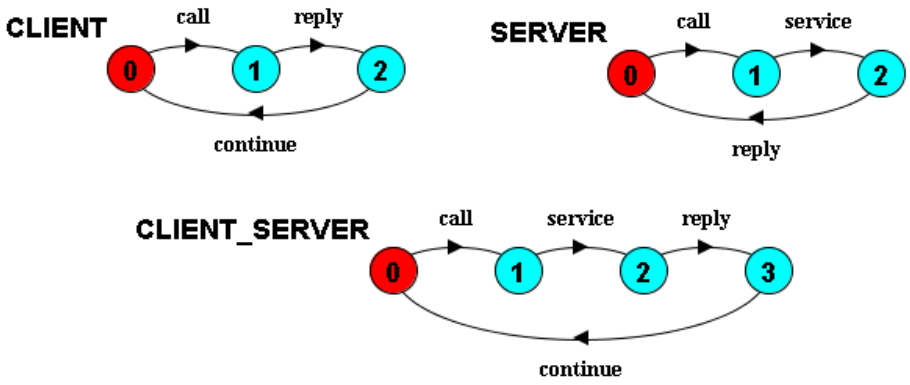

Figure 2.10: LTSA representation of Relabeling in `CLIENT_SERVER`.

Hiding "\" and "@"

Hiding removes action names from the alphabet of a process and thus makes these concealed actions "silent". By convention, these silent actions are labeled "tau". The general form of a hiding expression is \ {set of labels to be hidden}. Sometimes it is more convenient to state the set of action labels, which are visible and hide all other labels. This is expressed by @ {set of visible labels}. [6]

2.2.2 Property Processes to verify the System

Safety Properties

Safety properties are specified to LTSA as deterministic primitive processes which contain no silent (tau) transitions (no hiding). Safety property processes are denoted by the keyword property. They are composed with a target system to ensure that the specified property holds for that system. Composing a property process with a set of processes does not affect their normal operation. If behavior can occur that violates the safety property, then a transition to the ERROR state results. For example, the following property specifies that only behavior in which knock occurs before enter is acceptable. [6]

```
property POLITE = (knock->enter->POLITE).
```

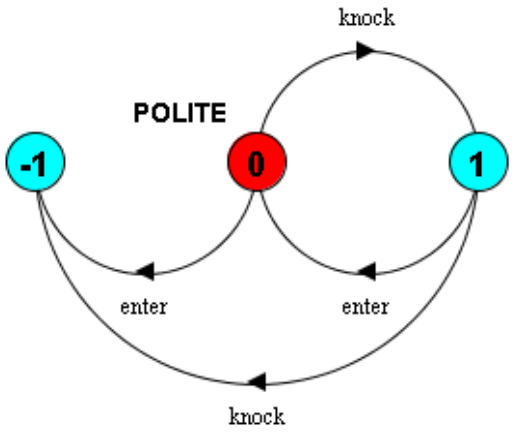

Figure 2.11: LTSA representation of Property `POLITE`.

2.3 Compensating CSP (cCSP)

Compensating CSP (cCSP) is a language defined to model long running business transactions within the framework of standard CSP process algebra. Processes in cCSP are modeled in terms of the atomic events they can engage in. The processes are categorized into standard and compensable processes. [8]

A standard process does not have any compensation. The basic unit of the standard processes is an atomic event (A). The other operators are the sequential (P;Q), and the parallel composition (P || Q), the choice operator (P□Q), the interrupt handler (P ▷ Q), the empty process SKIP, raising an interrupt THROW, and yielding to an interrupt YIELD. A process that is ready to terminate is also willing to yield to an interrupt. In a parallel composition, throwing an interrupt by one process synchronizes with yielding in another process. [8]

Compensation is part of a compensable process that is used to compensate a failed transaction. The sequential composition of compensable processes is defined in such a way that the compensations of the completed tasks will be accumulated in reverse to the order of their original composition, whereas compensations from the compensable parallel processes will be placed in parallel. In order to support failed transaction, compensation operators are introduced. We use notations, such as, P,Q,.. to identify standard processes, and PP,QQ,.. to identify compensable processes. The basic way of constructing a compensable process is through a compensation pair (P ÷ Q), which is constructed from two standard processes, where P is called the forward behavior that executes during normal execution, and Q is the associated compensation that is designed to compensate the effect of P when needed. [8]

The language provides operators that support sequencing, choice and parallel composition of processes. SKIPP, THROWW, and YIELDD are the compensable counterpart of the corresponding standard processes and they are defined by pairing an empty compensation with them, e.g., SKIPP = SKIP ÷ SKIP. [8]

Table 2.1: cCSP syntax

Standard Processes		**Compensable Processes**	
P,Q ::= A	(atomic event)	PP,QQ ::= P ÷ Q	(compensation pair)
P ;Q	(sequential composition)	PP ;QQ	
P □ Q	(choice)	PP □ QQ	
P || Q	(parallel composition)	PP || QQ	
SKIP	(normal termination)	SKIPP	
THROW	(throw an interrupt)	THROWW	
YIELD	(yield to an interrupt)	YIELDD	
P ⊳Q	(interrupt handler)		
[PP]	(transaction block)		

CHAPTER 3

Car Broker Service Composition

3.1 Car Broker Web Service

Car Broker Web service is an online service designed for supporting customers to negotiate car purchases and arranges loans for these. The main web service Broker uses two separate partner web services: A Supplier to find quote then deliver car to Broker according to the order; A Lender, named LoanStar to arrange loans for these. Each web service can operate separately and can be used in other web services. In this model a Buyer provides a need for a car model to the Broker, Broker receives order from Buyer (the order contains the details about the car); according to Buyer's order Broker first uses its business partner Supplier to find the available quotes for car models. Broker receives the quotes from Supplier and chooses a suitable quote for Buyer. After that, Broker simultaneously send Quote to Buyer, Order to Supplier and request Loan to LoanStar to arrange a loan for the Buyer for the selected quote. The order will be completed without any error if and only if the Buyer confirms the order, Supplier confirms the delivery and LoanStar approves the loan. A loan can be refused due to a failure in the loan assessment, a Buyer can reject the quoted offer and a Supplier may fail to provide proper quote or model. In either case an interrupt can be raised. It needs to invoke a compensation for the action so that the whole works never fail for all these negative possibilities. For an example, a negative acknowledgement is thrown by LoanStar. This negative message will be received by a compensation process of LoanStar which will compensate all actions performed by LoanStar and will throw an interrupt to the Main Compensation process. After stage Supplier, Buyer and Broker will be terminated by yielding an interrupt message combination. These messages will be received by the compensation processes of Supplier, Buyer and Broker respectively and will compensate all actions those are already performed by these processes. The behavior of the Car Broker Web Service is defined by combining the behavior of Broker, Buyer, Supplier and LoanStar, where the processes synchronize over their set of actions within a composition and between compositions. It also defined by how compensations are handled in each case. Before we can start writing the FSP process definition we have to become familiar with the web services involved in our business process.

3.1.1 BUYER

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 the quote satisfied the Buyer, Buyer sends a confirmation message to Broker or else reject the quote by throwing a message.

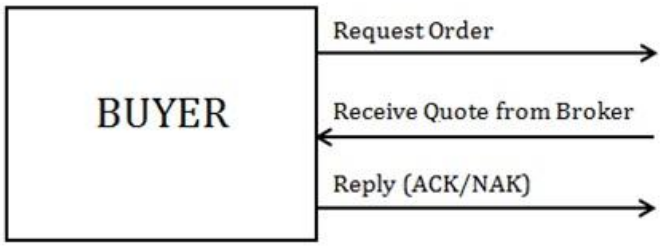

Figure 3.1: Architectural view of Buyer

3.1.2 BROKER

After receiving the order from the Buyer the Broker requests the Supplier for available quotes. Then select a quote from the received quotes. Broker then simultaneously send quote to Buyer, give an order to the Supplier and requests for loan to the LoanStar. Broker accepts all the positive acknowledgements from Buyer, Supplier and LoanStar if Buyer accepts the quote, Supplier is able to provide the requested model and LoanStar approved the loan then we can say that the order is complete. If any compensation is required to run for other processes, Main Compensation process will also run the Brokers compensation.

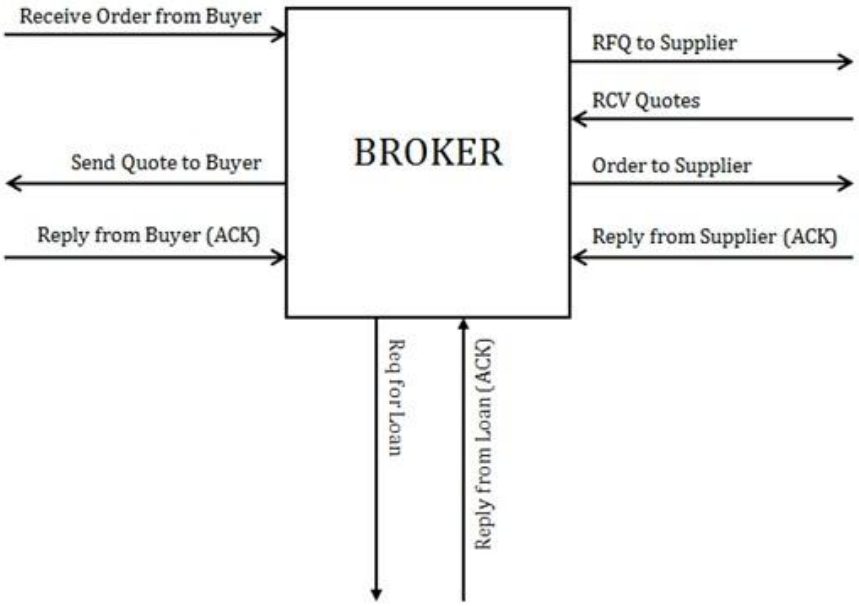

Figure 3.2: Architectural view of Broker Web Service

3.1.3 SUPPLIER

Supplier is a service that receives a quotation request from Broker in accordance to the order of a particular car model by Buyer. Getting the request for quotation, Supplier collects Quotes from all of its associated partners and passes the accumulated quotes to the Broker. Supplier receives a final order from Broker while Broker selects a suitable quote for Buyer. If the Supplier is able to manage the desired car model ready to supply, it acknowledges Broker by a positive reply. Otherwise it throws a failure message.

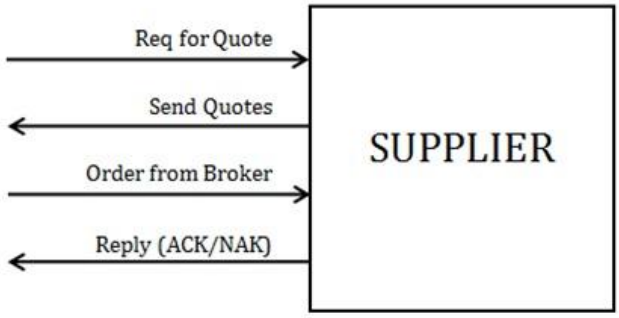

Figure 3.3: Architectural view of Supplier Web Service

3.1.4 LOANSTAR

LoanStar assumed as a lender web service that offers loans to online Buyers. After a detailed assessment of the loan LoanStar can either approve the loan or reject the loan. If the assessment outcome is positive loan request is granted and LoanStar sends a positive acknowledgement to Broker. Otherwise loan request is canceled.

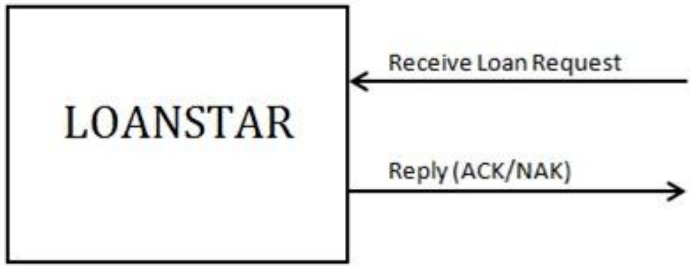

Figure 3.4: Architectural view of LoanStar Web Service

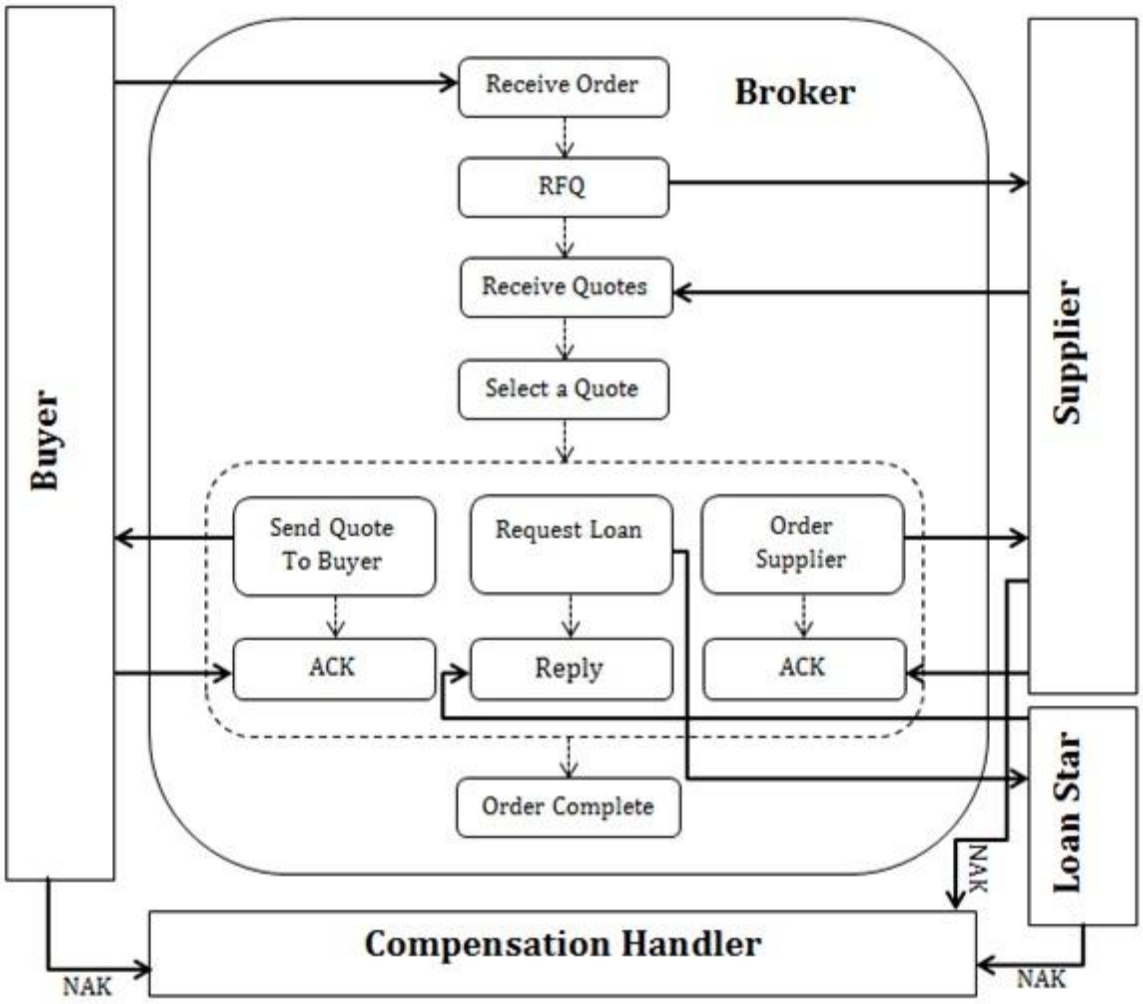

Figure 3.5: Architectural view of Car Broker Web Service

Overall Transaction in Message sequence Chart

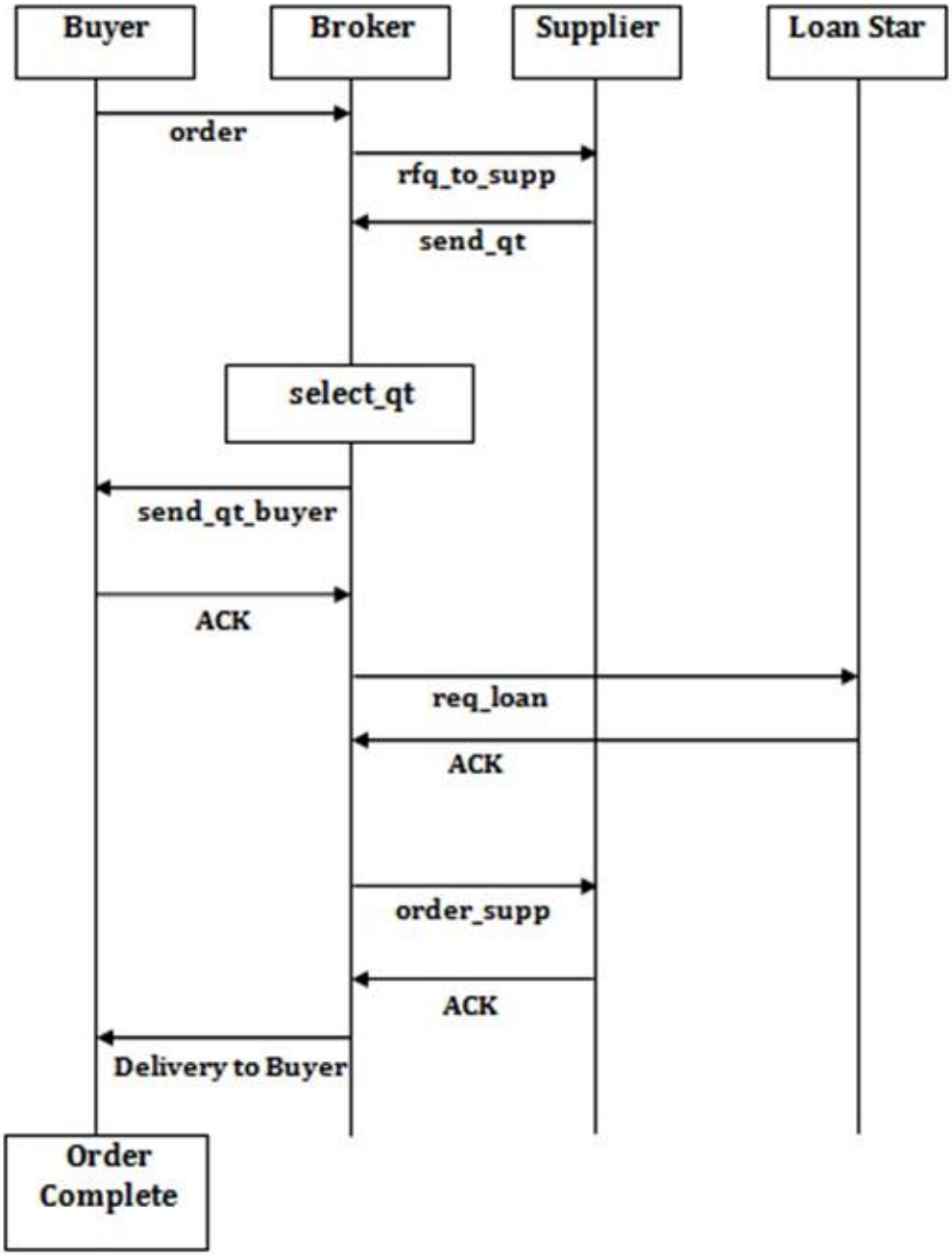

Figure 3.6: Overall Transaction in Message sequence Chart

3.2 Compensation in Car Broker Web Service

3.2.1 Compensation

To give a proper service it needs interaction between services. One service can call another service and need to deals with error occurs during interaction. If any negative acknowledgement is thrown by any service it is considered as a fault or error of the system for which service cannot be continued anymore. That's why we have to handle errors by compensating the services. A mechanism is used to handle the errors that can arise in any stage of communication between services is called compensation. Using the compensation mechanism all services can reach in their initial state from where they have been interrupted.

3.2.2 Compensation Mechanism of Car Broker Web Service

In our model while a negative acknowledgement is thrown by a service this message is received by the compensation process of this service. The reverse actions are performed to compensate the service from where the interruption occurs. Simultaneously the compensation process throws an interrupt to the Main Compensation process CMAIN. CMAIN throws a combination of messages that will be received by the compensation processes of the respective services attached to the system and runs the compensating actions in parallel.

In the case of rejection, Buyer will throw a message that will be received by a compensation process which will compensate Buyer's activities and will throw an interrupt to the Main Compensation process to compensate others.

Broker's compensation process runs in two phases. Compensation process will cancel all the requests placed by the Broker to other processes in phase two and then reverse all sequential actions of Broker declared in phase one.

The failure message of the Supplier will be received by a compensation process which will compensate Supplier's activities and will throw an interrupt to the Main Compensation process to compensate others.

Performed activities of LoanStar are compensated by the compensation process of LoanStar and simultaneously other services will be compensated through Main Compensation process.

A message sequence chart has been included to how compensation mechanism works in our system.

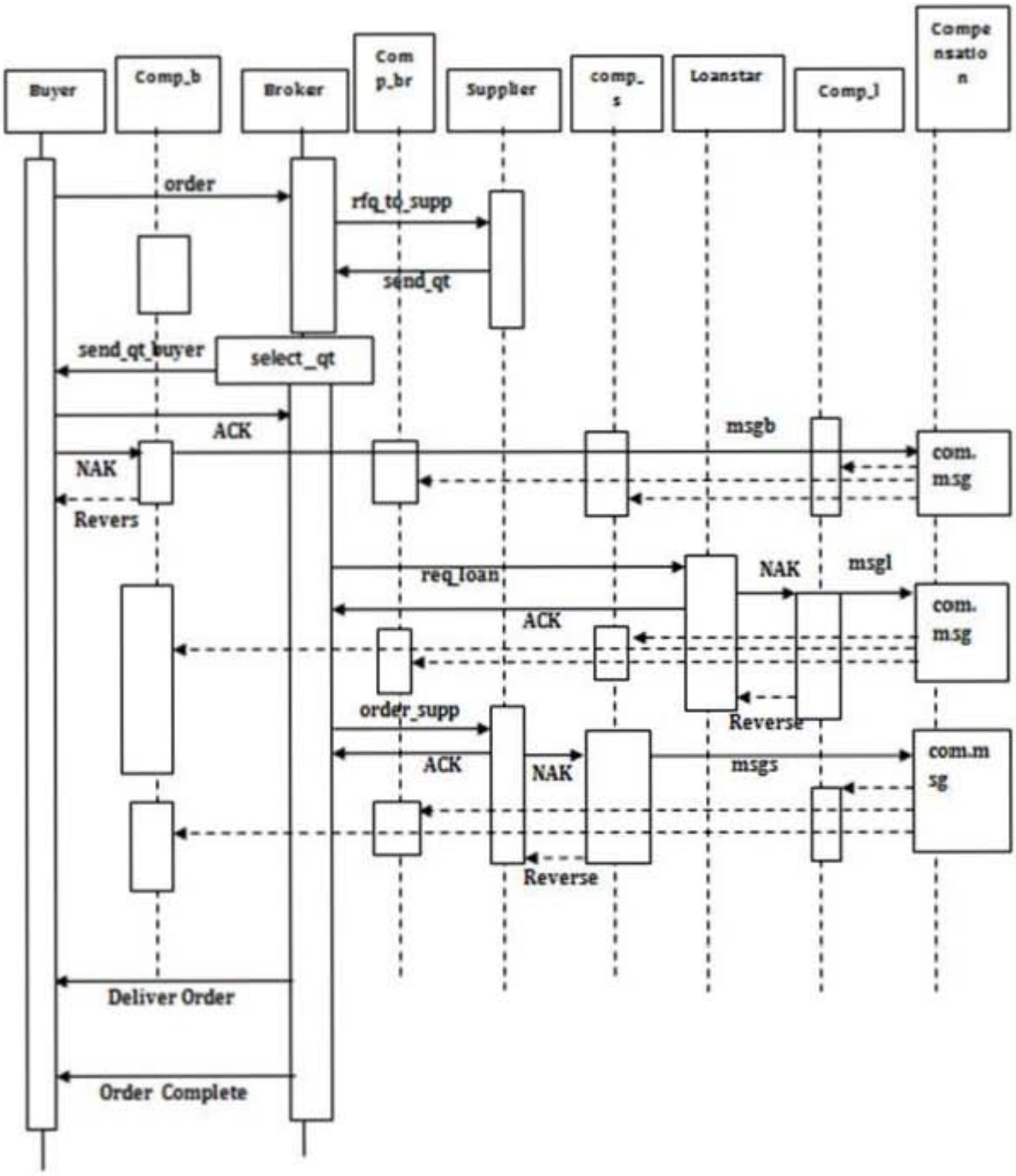

Figure 3.7: Overall Transaction in Message sequence Chart with Compensation

CHAPTER 4

Service Composition in FSP

4.1 Coding Representation

In our system we have four major processes which have their own compensation process and safety property to ensure a good composition. In FSP we modeled the system like that, Buyer will send an order to Broker in order to purchase a car, Broker will collect quotes for the requested car model from Supplier and select a quote then send it to Buyer and request a loan on behalf of Buyer to LoanStar and submit an order for car to Supplier at a time. The order is completed after getting all positive acknowledgements. When a negative acknowledgement is thrown by Buyer, LoanStar or Supplier will be received by their respective compensation process and a message will be sent by the process to main compensation process to compensate other processes. Here main compensation process use messaging system to communicate with other compensation processes.

4.2 Modeling the Car Broker Service in FSP

Car Broker Service is divided into four major services. Each service contains its own general process and its compensation process. Main Compensation Process handles any kind of anomaly that occurs in the System.

4.2.1 Declaring Original Processes

BUYER

Buyer process consists of a sequence of actions. The process starts the service by giving an `order` for a car to Broker. When Buyer receives a quote from Broker named as `rcv_qt` action in the process it checks whether the quote is suitable or not. If the quote is suitable then it gives a positive acknowledgement `send_b_ack` to Broker or else denies the quote by throwing a negative acknowledgement `send_b_nak`.

```
BUYER = (order->rcv_qt->reply->(send_b_ack->BUYER|send_b_nak-
>thrwb->END)).
```

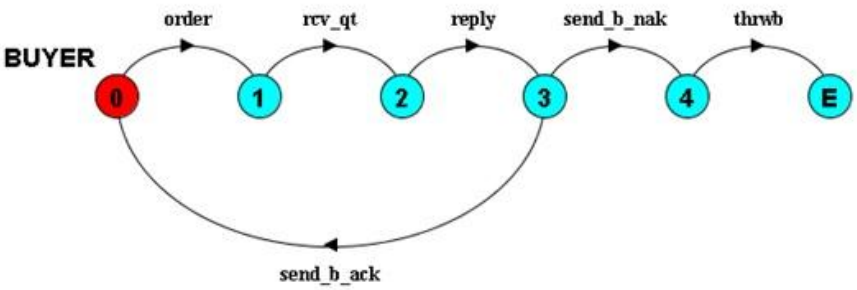

Figure 4.1: LTSA representation of Buyer Process

BROKER

Broker service is a process interacts with other three partner service processes. Broker process works in two phases. Phase one consists of several sequential actions and Phase two is the concurrent execution of three parallel processes.

BROKER Phase One

Phase one starts by receiving an order from Buyer labeled as `rcv_order`. According to Buyer's order Broker request for the quotes to the Supplier by `rfq_to_supp`. `rcv_qt_supp` action represents that Broker receives the quotes from Supplier. Broker finds the best possible quote for the requested car model by Buyer represented by the action `select_qt`.

```
BRK_PHASE1 = (rcv_order->rfq_to_supp->rcv_qt_supp->select_qt-
>END).
```

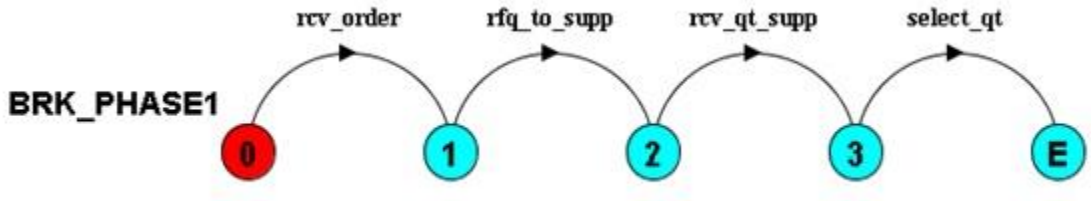

Figure 4.2: LTSA representation of `BRK_PHASE1` Process

BROKER Phase Two

After selecting the quote in Phase two, Broker simultaneously send quote to Buyer, order to Supplier and request loan to LoanStar by using the process `REQ`. The order will be completed without any error if and only if Buyer, Supplier and LoanStar all send positive acknowledgements to Broker. These acknowledgements received in the process `RCV` and the respective receiving messages are `rcv_buyerack`, `rcv_suppack`, `rcv_loanack`. Broker Phase Two is the composition of these two processes `REQ and RCV` and the composition is titled as `BRK_PHASE2`.

```
REQ1 = (select_qt->send_qt_buyer->reply->END).

REQ2 = (select_qt->order_supp->reply->END).

REQ3 = (select_qt->req_loan->reply->END).


RCV1 = (reply->rcv_buyerack->END).

RCV2 = (reply->rcv_suppack->END).

RCV3 = (reply->rcv_loanack->END).


||REQ = (REQ1||REQ2||REQ3).

||RCV = (RCV1||RCV2||RCV3).

||BRK_PHASE2 = (REQ||RCV).
```

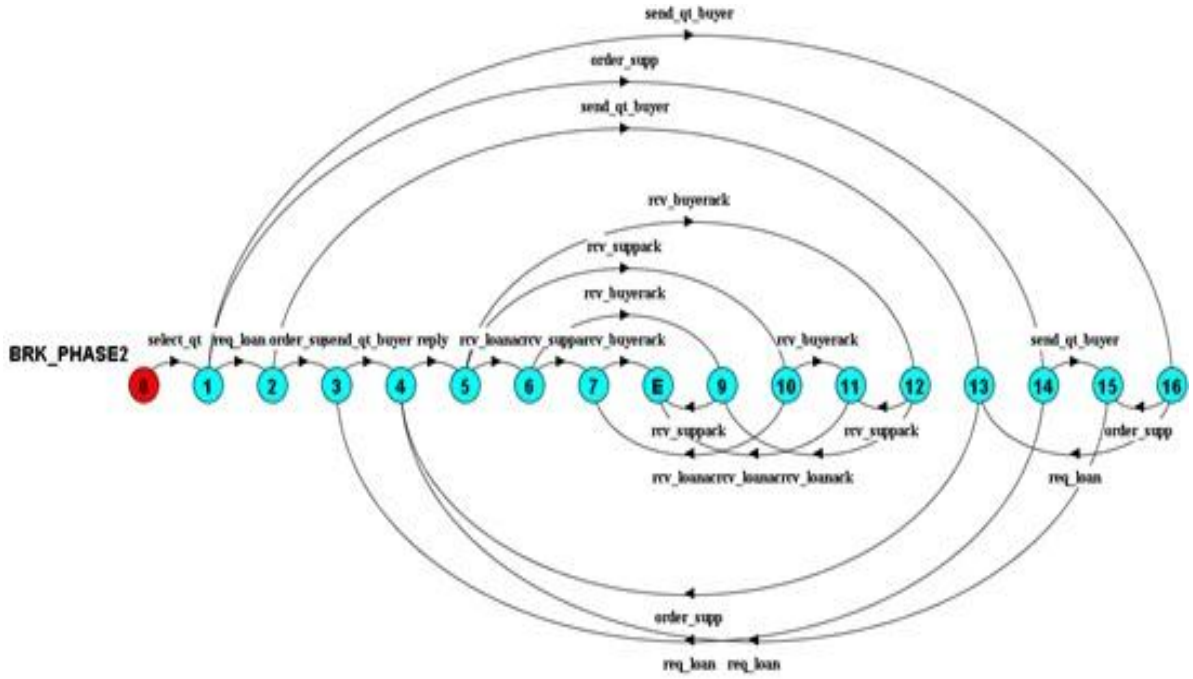

Figure 4.3: LTSA representation of `BRK_PHASE2` Process

Finally the Broker process is the composition of two phases; `BRK_PHASE1`, the sequential part of Broker and `BRK_PHASE2`, the parallel part of Broker.

```
||BROKER = (BRK_PHASE1||BRK_PHASE2).
```

SUPPLIER

Supplier receives a request for quotes from the Broker by `rcv_rfq`. According to the request, Supplier sends accumulated quotes to the Broker by `send_qt`. After selecting the appropriate quote Broker sends an order for car and that is received by the action labeled `rcv_brk_order` at the Suppliers end. If the Supplier able to deliver the order it confirms Broker by sending an acknowledgement `send_s_ack` otherwise it rejects the order and sends a negative acknowledgement `send_s_nak`.

```
SUPPLIER = (rcv_rfq->send_qt->rcv_brk_order->reply->(send_s_ack-
>SUPPLIER|send_s_nak->thrws->END)).
```

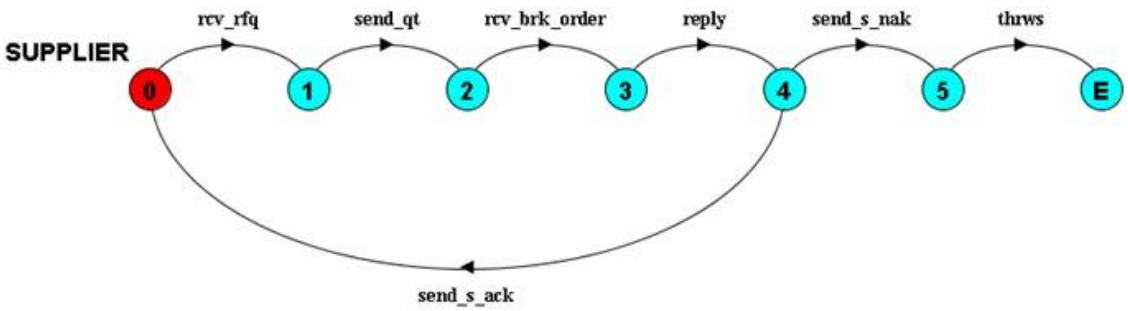

Figure 4.4: LTSA representation of Supplier Process

LOANSTAR

After selecting the quote Broker sends a request to its business partner LoanStar to arrange a loan for Buyer. This request is received in LoanStar by the action `rcv_req`. Loanstar confirms the approval of the loan by sending an acknowledgement `send_l_ack`. Loanstar rejects the loan request using `send_l_nak` if it is not able to arrange the loan for the Buyer.

```
LOANSTAR = (rcv_req->reply->(send_l_ack->LOANSTAR|send_l_nak-
>thrwl->END)).
```

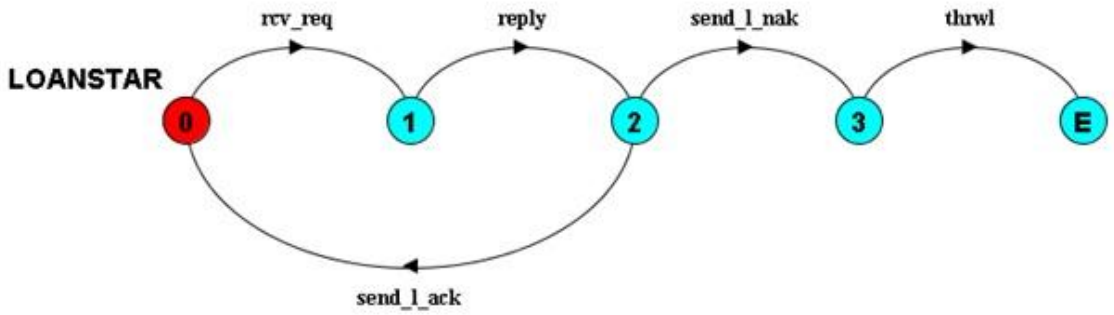

Figure 4.5: LTSA representation of LoanStar Process

4.2.2 Declaring Compensation Processes

Compensation Process for BUYER

Buyer's compensation process is completed with two processes. One process is consists of compensating actions to compensate Buyer and another contains a message which alerts Main Compensation Process that a negative acknowledgement is thrown from Buyer.

```
COMP_B = (thrwb->cancel_rcv_qt->cancel_order->END).
```

```
MSGB = (thrwb->msgb->END).
```

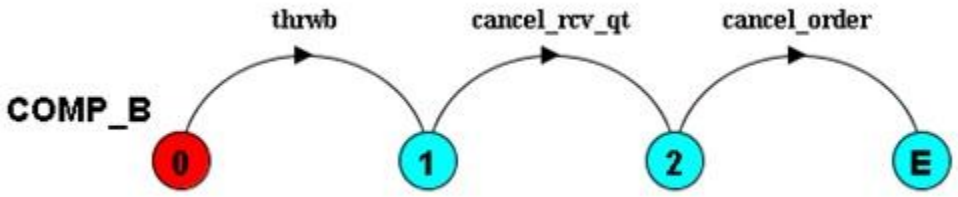

Figure 4.6: LTSA representation of `COMP_B` Process

The compensation process `COMP_B` consists of reverse actions those were performed by Buyer before sending a negative acknowledgement. The action `thrwb` is the synchronizing shared action for those processes who tries to execute it.

When `COMP_B` runs, at the same time another process `MSGB` also run in parallel. `MSGB` throws a message `msgb` which is received by the Main Compensation Process. `msgb` indicates that Buyer process is not running anymore; so getting this message Main Compensation Process will take necessary steps to compensate others.

Compensation Process for BROKER

Broker's Compensation Process `COMP_BRK` completed in two phases; one is the compensation of Phase two of Broker Process then the compensation of Phase one of the Broker Process. `COMP_BRK` is composed of two separate processes, `BRK_PHASE2_COMP` and `BRK_PHASE1_COMP`.

```
||COMP_BRK = (BRK_PHASE2_COMP||BRK_PHASE1_COMP).
```

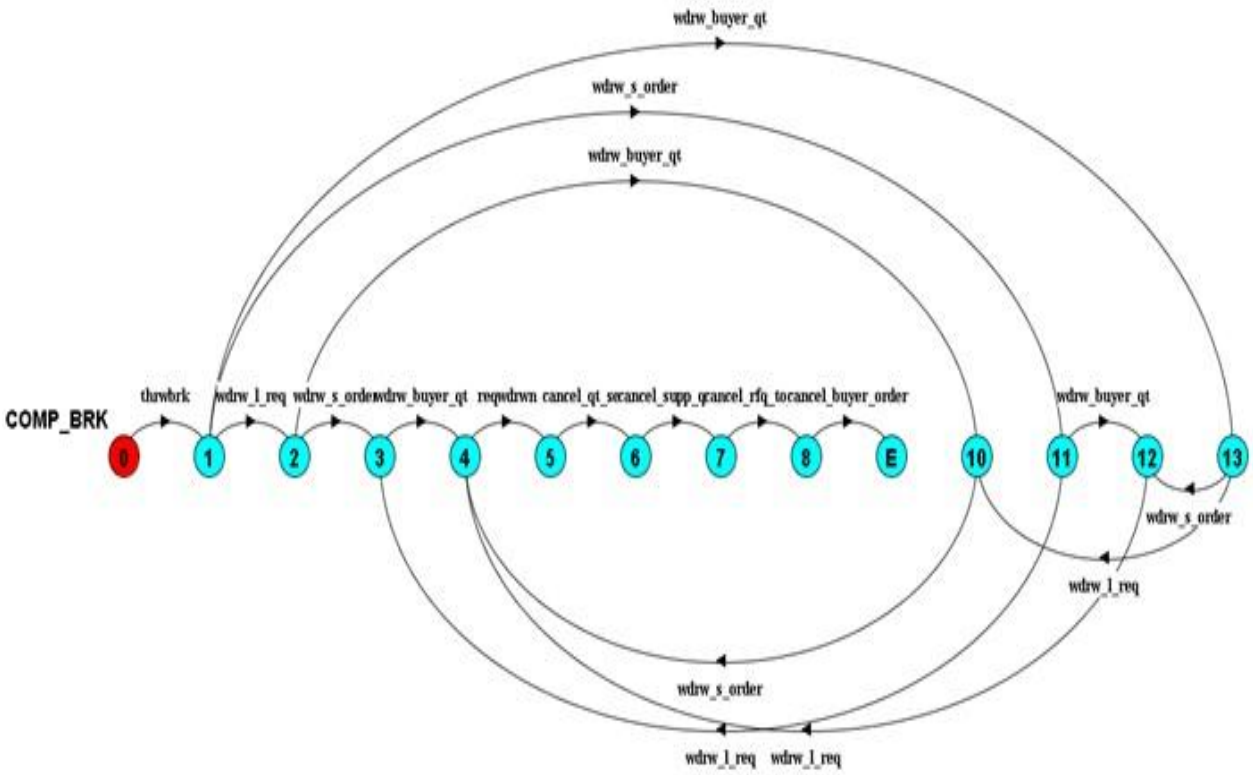

Figure 4.7: LTSA representation of `COMP_BRK` Process

Compensation Process of Broker's Phase Two

```
CMP_REQ1 = (thrwbrk->wdrw_buyer_qt->reqwdrwn->END).
CMP_REQ2 = (thrwbrk->wdrw_s_order->reqwdrwn->END).
CMP_REQ3 = (thrwbrk->wdrw_l_req->reqwdrwn->END).

||BRK_PHASE2_COMP = (CMP_REQ1||CMP_REQ2||CMP_REQ3).
```

`BRK_PHASE2_COMP` is the compensation process of Broker's parallel part. After getting an interrupt from Main Compensation Process Broker's phase two compensation process withdraws all placed request to its partner processes with the actions `wdrw_buyer_qt`, `wdrw_s_order` and `wdrw_l_req`. These actions are composed in parallel with the

separate respective processes CMP_REQ1, CMP_REQ2 and CMP_REQ3 in BRK_PHASE2_COMP process. reqwdrwn indicates that all requests are successfully withdrawn.

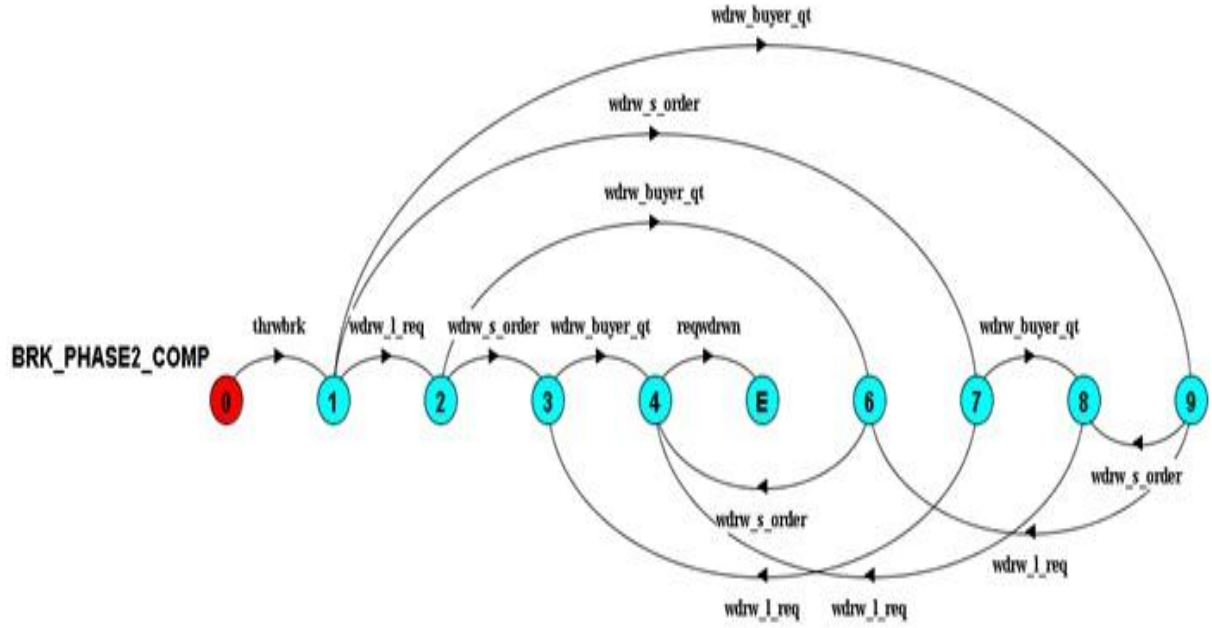

Figure 4.8: LTSA representation of BRK_PHASE2_COMP Process

Compensation Process of Broker's Phase One

```
BRK_PHASE1_COMP = (reqwdrwn->cancel_qt_select-
>cancel_supp_qt_rcv->cancel_rfq_to_supp->cancel_buyer_order-
>END).
```

Broker's Phase One Compensation Process BRK_PHASE1_COMP starts after successfully withdrawing all requests placed by Broker to its partner processes. Process BRK_PHASE1_COMP consists a sequence of actions that cancels every actions performed by Broker after receiving an order from Buyer till selecting a quote among various quotes sent by supplier.

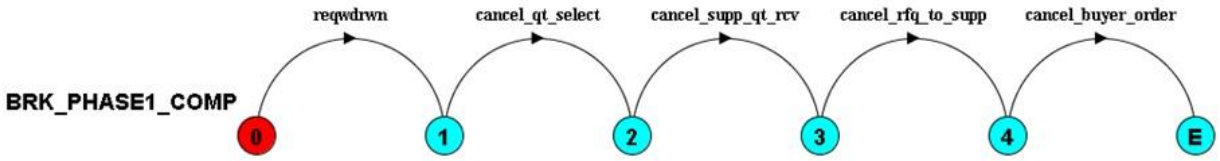

Figure 4.9: LTSA representation of `BRK_PHASE1_COMP` Process

Compensation Process for Supplier

Supplier's Original Compensation process is composed of two processes. One is Supplier's compensation process and another is a messaging system alerts the Main Compensation Process that a negative acknowledgement is thrown from Supplier.

```
COMP_S = (thrws->cancel_brk_order->cancel_qt->cancel_rfq->END).
```

```
MSGS = (thrws->msgs->END).
```

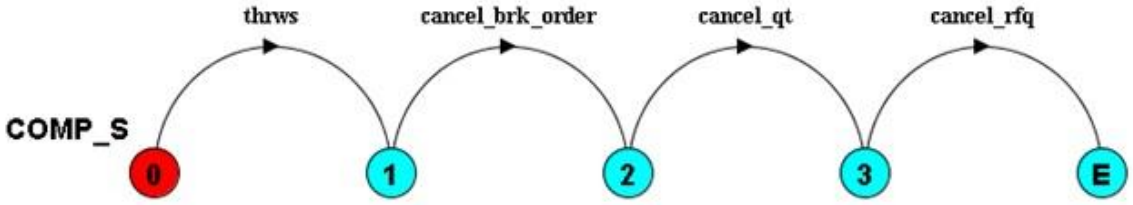

Figure 4.10: LTSA representation of `COMP_S` Process

`COMP_S`, the compensation process of Supplier reverse the actions which are already done by Supplier before sending a negative acknowledgement. This process is synchronized through a shared action `thrws` while Supplier needs to be compensated.

When `COMP_S` runs, at the same time another process `MSGS` also run in parallel. `MSGS` throws a message `msgs` which is received by the Main Compensation Process. This message indicates that Supplier process is not able to run anymore and the compensation process of supplier is running; after getting this message Main Compensation Process will take necessary steps to compensate others.

Compensation Process for LoanStar

Process `COMP_L` and `MSGL` are used to compensate LoanStar's activities. While `COMP_L` runs, simultaneously a process `MSGL` runs.

```
COMP_L = (thrwl->cancel_loan_req->END).

MSGL = (thrwl->msgl->END).
```

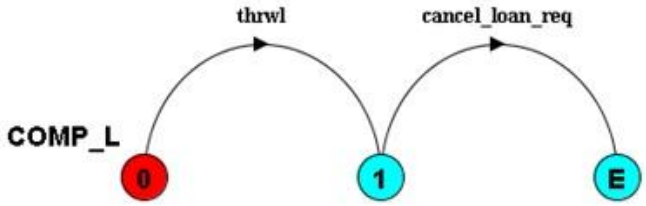

Figure 4.11: LTSA representation of `COMP_L` Process

`COMP_L` compensates LoanStar's actions those took place before sending a negative acknowledgement. A shared action `thrwl` is used to synchronize with those processes that are willing to execute the compensation process of LoanStar.

`MSGL` throws a message `msgl` which is received by the Main Compensation Process. `msgl` indicates that LoanStar rejects the loan request. So Main Compensation Process has to take necessary steps to compensate others.

4.2.3 Main Compensation Process

Main Compensation process is a messaging system. When a negative acknowledgement is given by any partner processes of Broker, a message is received by the Main Compensation Process. From the message, Main Compensation Process identifies that from which process the interrupt is thrown. On the basis of the sent message a combination of messages is generated by the Main Compensation Process to compensate other processes except the process from where the message was received.

```
CMAIN = (msgb->COMP_EXCPT_BUYER|msgs->COMP_EXCPT_SUPP|msgl-
>COMP_EXCPT_LOAN),

COMP_EXCPT_BUYER = FROM_BUYER;END,

COMP_EXCPT_SUPP = FROM_SUPP;END,

COMP_EXCPT_LOAN = FROM_LOAN;END.
```

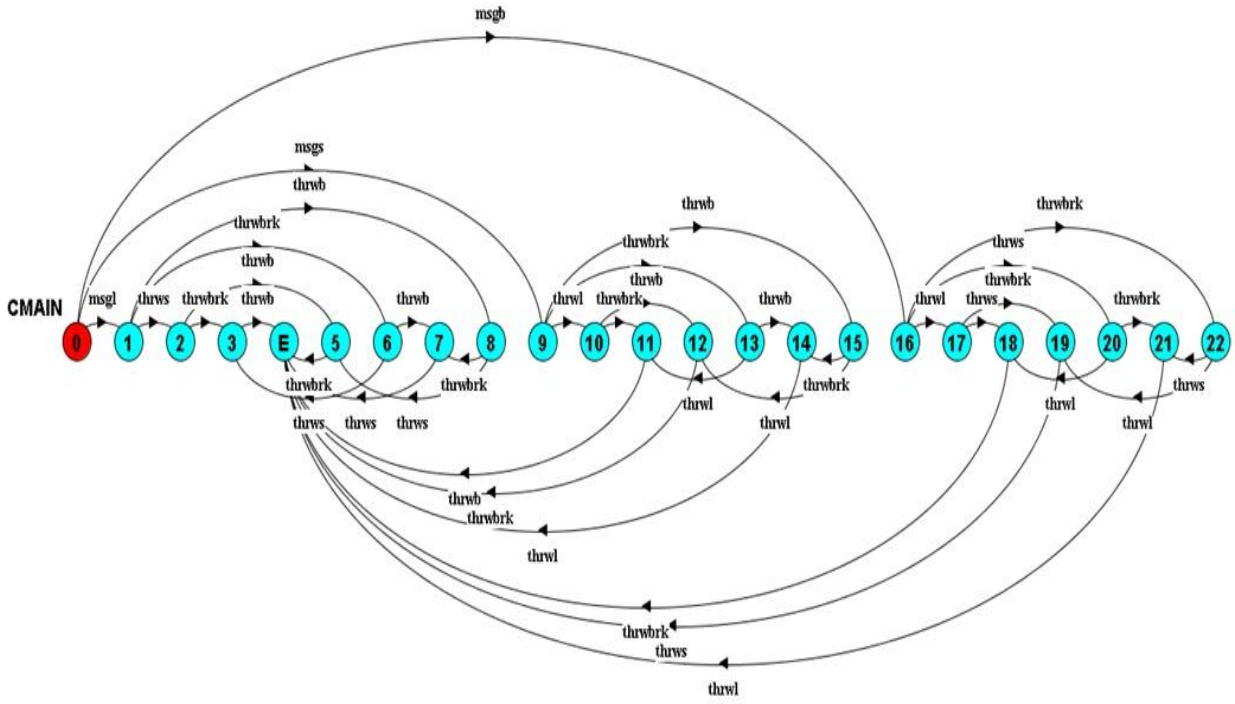

Figure 4.12: LTSA representation of `CMAIN` Process

Message Composition to Compensate Processes

When a message `msgb` is thrown from the compensation process of Buyer it is synchronized with a process of Main Compensation process `CMAIN`, named as `FROM_BUYER`. `FROM_BUYER` consisting `thrwbrk`, `thrws` and `thrwl` actions that synchronizes with the compensation process of Broker, Supplier and LoanStar respectively.

```
BUYERMSG_TO_COMP_BRK = (thrwbrk->END).

BUYERMSG_TO_COMP_S = (thrws->END).

BUYERMSG_TO_COMP_L = (thrwl->END).

||FROM_BUYER =
(BUYERMSG_TO_COMP_BRK||BUYERMSG_TO_COMP_S||BUYERMSG_TO_COMP_L).
```

`msgs` is thrown from the compensation process of Supplier. This synchronized with the process `FROM_SUPP` which is a part of the Main Compensation Process. `FROM_SUPP` is a parallel composition of thrwb, `thrwbrk` and `thrwl` actions that synchronizes with the compensation process of Buyer, Broker and LoanStar respectively.

```
SUPPMSG_TO_COMP_B = (thrwb->END).

SUPPMSG_TO_COMP_BRK = (thrwbrk->END).

SUPPMSG_TO_COMP_L = (thrwl->END).

||FROM_SUPP =
(SUPPMSG_TO_COMP_B||SUPPMSG_TO_COMP_BRK||SUPPMSG_TO_COMP_L).
```

After receiving `msgl` from the compensation process of LoanStar, Main Compensation Process throws a combination of messages `thrwb`, `thrwbrk` and `thrws` using the process `FROM_LOAN` to the compensation process of Buyer, Broker and Supplier respectively in order to compensate them.

```
LOANMSG_TO_COMP_B = (thrwb->END).

LOANMSG_TO_COMP_BRK = (thrwbrk->END).

LOANMSG_TO_COMP_S = (thrws->END).

||FROM_LOAN =
(LOANMSG_TO_COMP_B||LOANMSG_TO_COMP_BRK||LOANMSG_TO_COMP_S).
```

4.2.4 Final Compositions

```
||CARBROKERSERVICE = (

     BUYER||BROKER||SUPPLIER||LOANSTAR||
     MSGB||MSGS||MSGL||CMAIN||
     COMP_B||COMP_BRK||COMP_S||COMP_L||
     SAFE_COMP_B||SAFE_COMP_S||SAFE_COMP_L||
     SAFE_MSG_BRK||SAFE_MSG_B||SAFE_MSG_S||SAFE_MSG_L|
     SAFE_SYSTEM||SAFE_REQ1||SAFE_REQ2||SAFE_REQ3

                    )

                         /{

                         rcv_order/order,

                         rcv_rfq/rfq_to_supp,

                         rcv_qt_supp/send_qt,


                         rcv_qt/send_qt_buyer,

                         rcv_req/req_loan,

                         rcv_brk_order/order_supp,


                         rcv_buyerack/send_b_ack,

                         rcv_loanack/send_l_ack,

                         rcv_suppack/send_s_ack

                         }.
```

CARBROKERSERVICE Process is the parallel composition of all engaged processes to the system with all safety properties (that will be discussed in Chap. 5). All major services, their compensation processes, all message throwing processes, CMAIN the main compensation process all together composed in CARBROKERSERVICE. All the services have been synchronized with each other through the relabeling functions.

CHAPTER 5

Composition Verification

5.1 Property Processes for Verification

If a safety property is composed in parallel with a process and no trace violation is generated after the composition we can ensure that the safety property verifies that process. If any trace violation occurs we can say that the safety property could not verify the process.

5.2 Property Processes to Verify Compensation

5.2.1 Verifying Buyer Compensation

```
property SAFE_COMP_B = (send_b_nak->cancel_rcv_qt->SAFE_COMP_B).
```

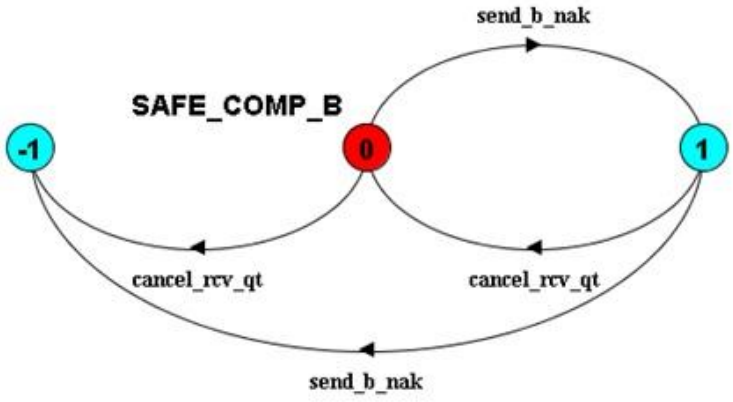

Figure 5.1: LTSA representation of safety property `SAFE_COMP_B`

The property process `SAFE_COMP_B` ensures that when a negative acknowledgement is thrown from Buyer it actually synchronizes with the compensation process of Buyer. The property is described with two actions `send_b_nak` and `cancel_rcv_qt`. `send_b_nak` is an action of Buyer process, when a negative acknowledgement is thrown and `cancel_rcv_qt` is the first action of Buyer's compensation process after the synchronizing shared action `thrwb`.

When the property process is composed with Buyer and its compensation process `COMP_B` in `BSAFE`, if the two actions of our property process found in a sequential manner and

does not show any trace violations in the resulting LTS, we can say that they have been synchronized with each other successfully and satisfied the condition of our property process, otherwise not.

```
||BSAFE = (BUYER||COMP_B||SAFE_COMP_B).
```

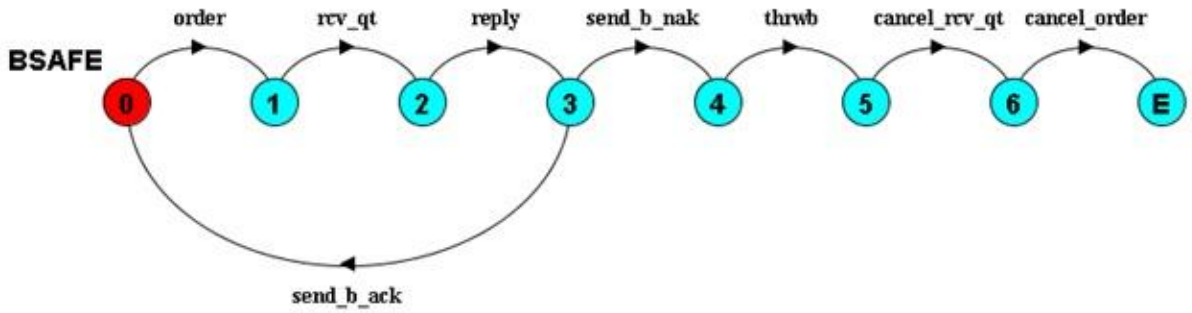

Figure 5.2: LTSA representation of `BSAFE` process

5.2.2 Verifying Supplier Compensation

```
property SAFE_COMP_S = (send_s_nak->cancel_brk_order-
>SAFE_COMP_S).
```

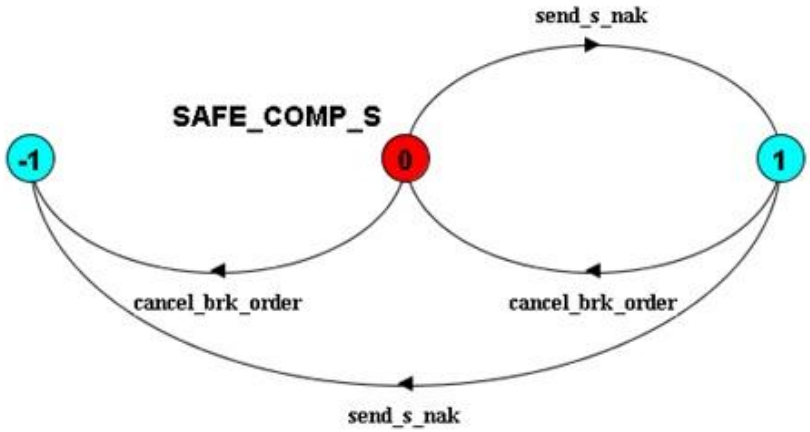

Figure 5.3: LTSA representation of safety property `SAFE_COMP_S`

The property process SAFE_COMP_S is described with two actions send_s_nak and cancel_brk_order. send_s_nak is an action of Supplier process, when a negative acknowledgement is thrown and cancel_brk_order is the first action of Supplier's compensation process after the synchronizing shared action thrws.

When the property process is composed with Supplier and its compensation process COMP_S in SSAFE, the above mentioned two actions of our property process remain in sequential manner in the resulting LTS and does not show any trace violations we can say that they have been synchronized with each other successfully and satisfied the condition of our property process, otherwise not.

```
||SSAFE=(SUPPLIER||COMP_S||SAFE_COMP_S).
```

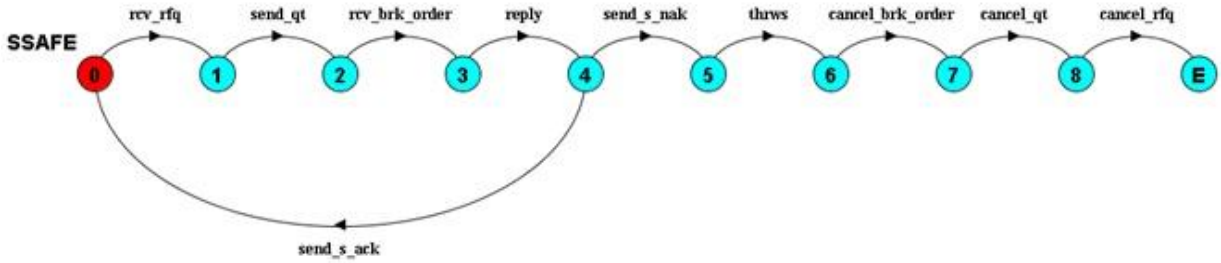

Figure 5.4: LTSA representation of SSAFE process

5.2.3 Verifying LoanStar Compensation

```
property SAFE_COMP_L = (send_l_nak->cancel_loan_req-
>SAFE_COMP_L).
```

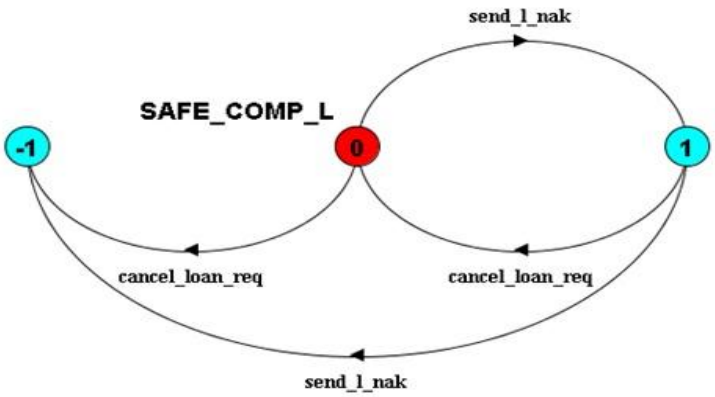

Figure 5.5: LTSA representation of safety property SAFE_COMP_L

Using the property process SAFE_COMP_L we tried to ensure that when a negative acknowledgement is thrown from LoanStar using send_l_nak it actually received by the compensation process of LoanStar and then start to execute the compensation action cancel_loan_req to compensate LoanStar.

After composing the property process with LoanStar and its compensation process COMP_L in LSAFE we can observe the resulting LTS. If there is no trace violations is observed we can say that they have been synchronized with each other successfully and satisfied the condition of our property process, otherwise not.

```
||LSAFE = (LOANSTAR||COMP_L||SAFE_COMP_L).
```

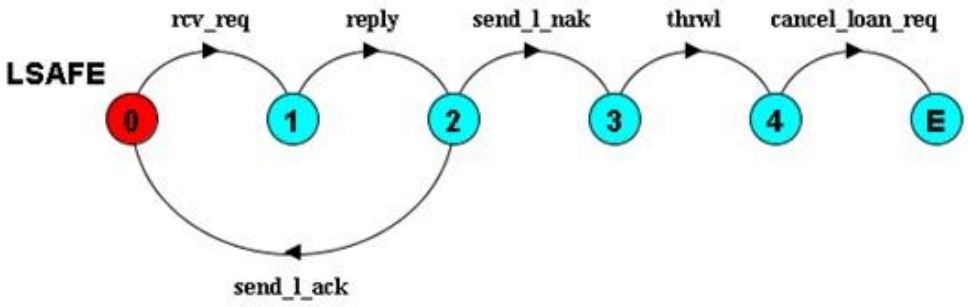

Figure 5.6: LTSA representation of LSAFE process

5.3 Verifying Main Compensation Mechanism

```
property SAFE_MSG_BRK = (msgb->thrwbrk->SAFE_MSG_BRK
                         |msgl->thrwbrk->SAFE_MSG_BRK|msgs-
>thrwbrk->SAFE_MSG_BRK).

property SAFE_MSG_B = (msgs->thrwb->SAFE_MSG_B|msgl->thrwb-
>SAFE_MSG_B).

property SAFE_MSG_S = (msgb->thrws->SAFE_MSG_S|msgl->thrws-
>SAFE_MSG_S).

property SAFE_MSG_L = (msgb->thrwl->SAFE_MSG_L|msgs->thrwl-
>SAFE_MSG_L).
```

To verify the main compensation mechanism we defined four different property processes where SAFE_MSG_BRK is defined to confirm that when a negative acknowledgement is received from Buyer, Supplier or LoanStar, CMAIN will surely run the compensation process of Broker by throwing the message thrwbrk. Here thrwbrk is the synchronizing shared action of the compensation process of Broker.

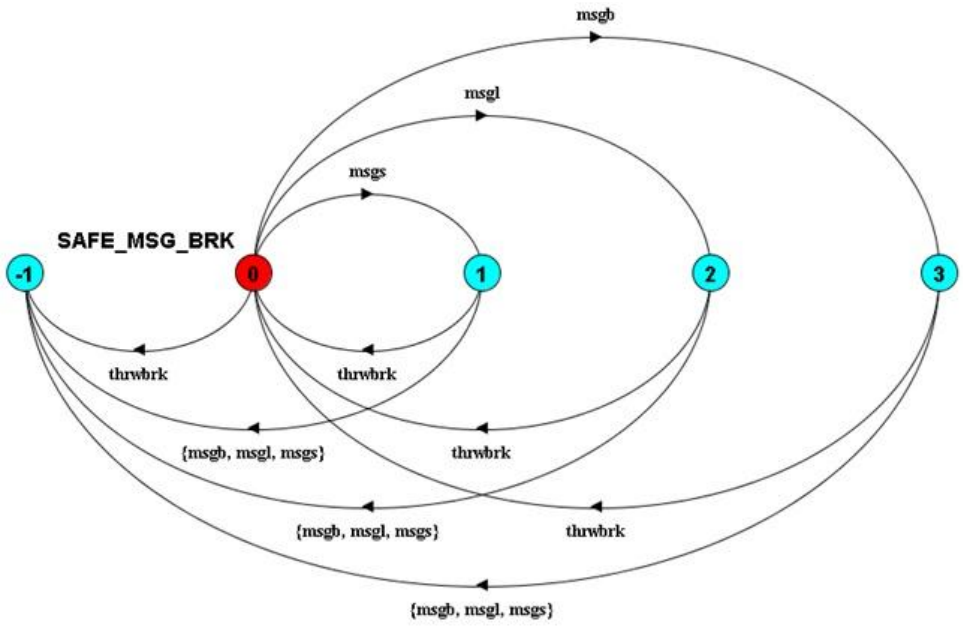

Figure 5.7: LTSA representation of safety property SAFE_MSG_BRK

SAFE_MSG_B is defined to confirm that CMAIN will surely run the compensation process of Buyer by throwing the message thrwb when a negative acknowledgement is received from Supplier or LoanStar. Here thrwb is the synchronizing shared action of the compensation process of Buyer.

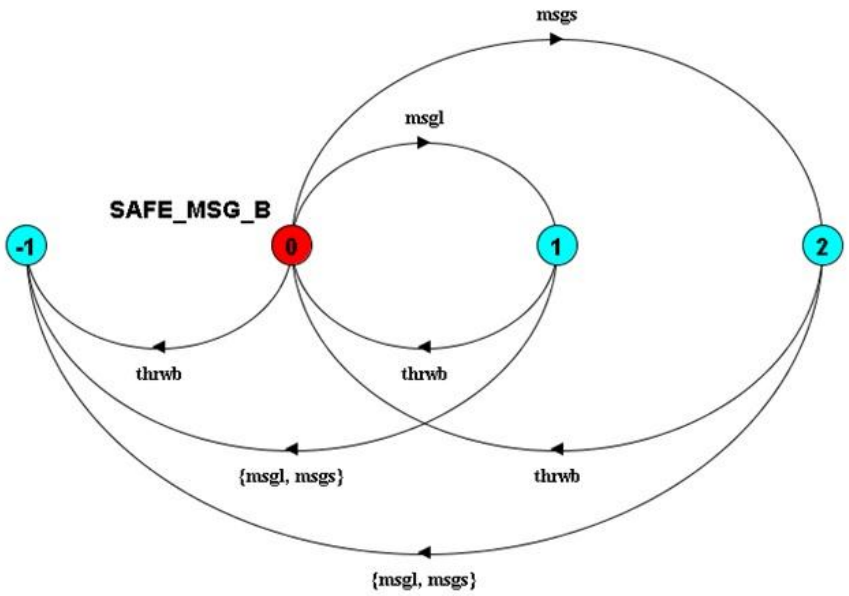

Figure 5.8: LTSA representation of safety property `SAFE_MSG_B`

`SAFE_MSG_S` ensures that when a negative acknowledgement is received from Buyer or LoanStar, `CMAIN` will surely run the compensation process of Supplier by throwing the message `thrws`.

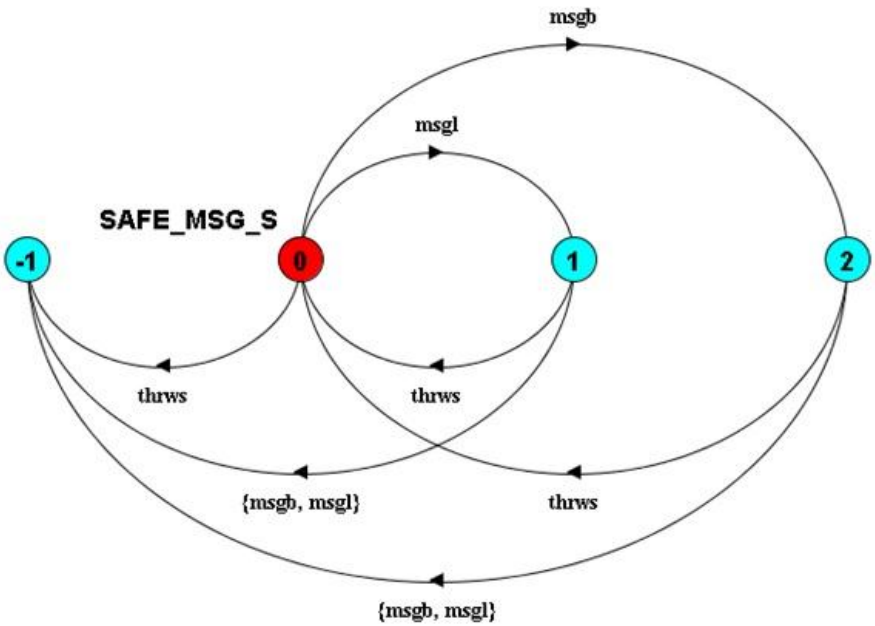

Figure 5.9: LTSA representation of safety property `SAFE_MSG_S`

SAFE_MSG_L is defined to ensure that when a negative acknowledgement is received from Buyer or Supplier, CMAIN will synchronize with the compensation process of LoanStar throwing the message thrwl.

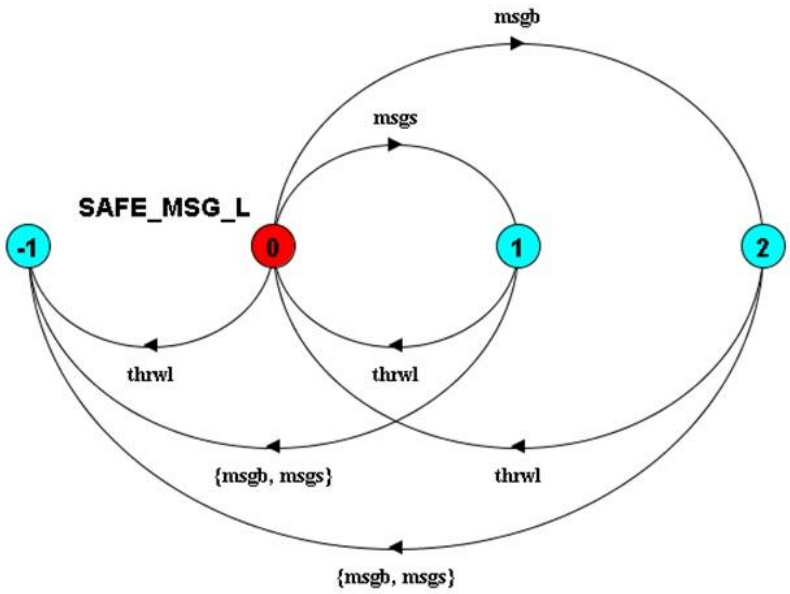

Figure 5.10: LTSA representation of safety property SAFE_MSG_L

msgb, msgs and msgl are used by CMAIN to receive a negative acknowledgement from Buyer, Supplier and LoanStar respectively.

```
||CMAIN_CHECK=(CMAIN||COMP_B||COMP_BRK||COMP_S||COMP_L||SAFE_MSG
_BRK||SAFE_MSG_B||SAFE_MSG_S||SAFE_MSG_L).
```

We can be sure that CMAIN must execute all the compensation processes engaged in the system if and only if there is no trace violation when CMAIN is composed in parallel with the property processes named SAFE_MSG_BRK, SAFE_MSG_B, SAFE_MSG_S, SAFE_MSG_L and all the compensation processes in CMAIN_CHECK, otherwise it will violates the main compensation mechanism properties.

5.4 Verifying System Composition

```
property SAFE_SYSTEM = (rcv_order->rcv_rfq->rcv_qt_supp-
>select_qt->SAFE_SYSTEM).
```

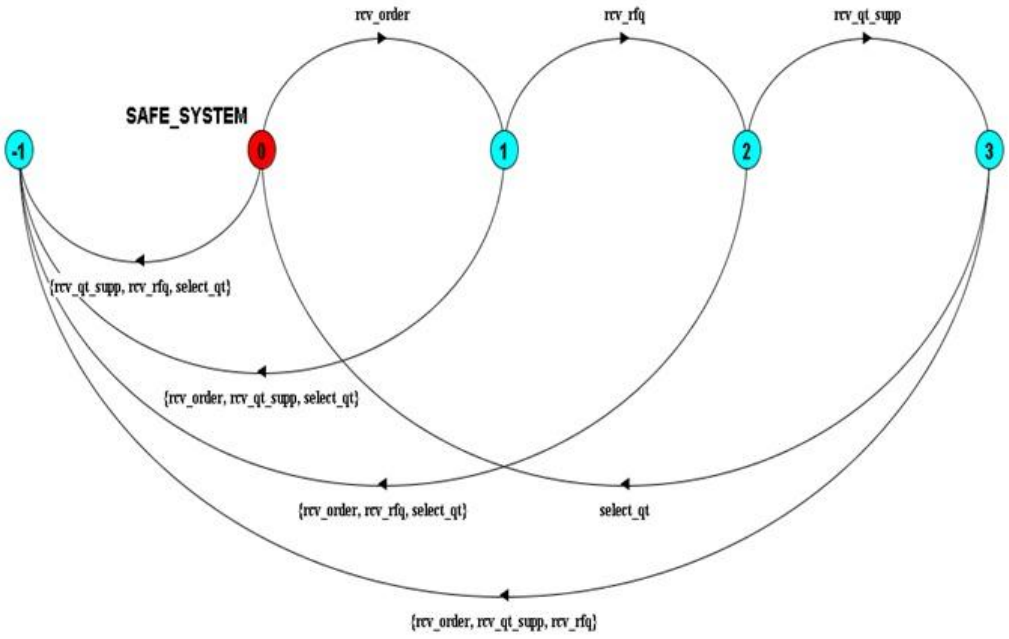

Figure 5.11: LTSA representation of safety property `SAFE_SYSTEM`

The property process `SAFE_SYSTEM` is defined to ensure that Buyer, Broker and Supplier processes synchronized correctly in their desired synchronizing points in the system up to quote selection.

```
property SAFE_REQ1 = (select_qt->rcv_qt->SAFE_REQ1).

property SAFE_REQ2 = (select_qt->rcv_brk_order->SAFE_REQ2).

property SAFE_REQ3 = (select_qt->rcv_req->SAFE_REQ3).
```

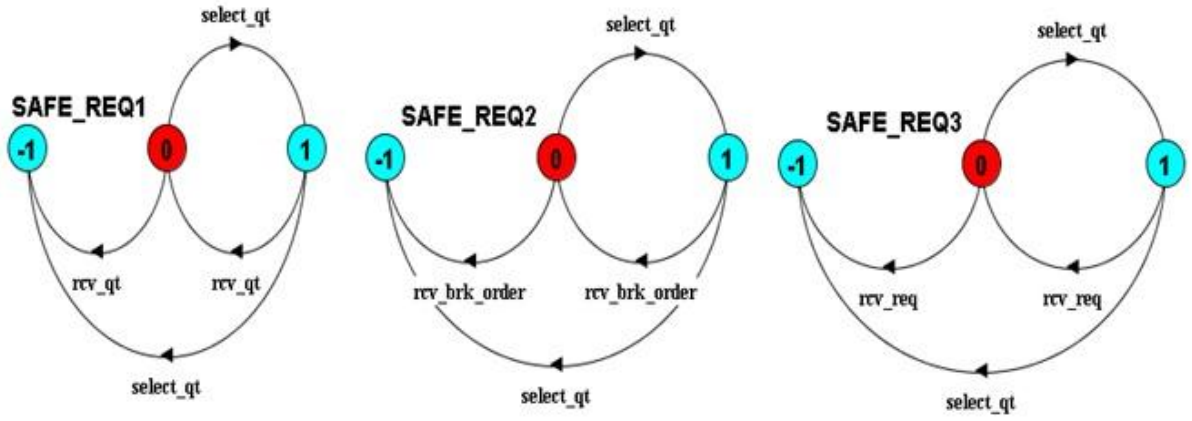

Figure 5.12: LTSA representation of safety property SAFE_REQ1, SAFE_REQ2, SAFE_REQ3

After selecting quote, a quote is sent to Buyer, a loan request is placed to LoanStar and an order is placed to Supplier simultaneously. These requests are received by those three service processes using rcv_qt, rcv_brk_order, rcv_req actions. SAFE_REQ1, SAFE_REQ2 and SAFE_REQ3, these three safety properties ensures that the requests placed in parallel to Buyer, Supplier and LoanStar by Broker is successfully received.

If the property SAFE_SYSTEM and SAFE_REQ1, SAFE_REQ2, SAFE_REQ3 are composed in parallel with the processes BUYER, BROKER, SUPPLIER and LOANSTAR in MAINSYSTEM_CHECK and the traces of MAINSYSTEM_CHECK does not show any violation in LTS diagram we can say that our system is verified with the written SAFE_SYSTEM property process which ensures that the system has been synchronized successfully. On the other hand we can also say that the requests are made by the Broker are successfully received by its partner services. If there any violation occurs in the traces we can say that system synchronization might not ok or else there is a fault in the message passing system. As Including LOANSTAR in the composition of MAINSYSTEM_CHECK, it generates too many states. So, by omitting LOANSTAR from the composition we can generate the LTSA representation of the process MAINSYSTEM_CHECK.

```
||MAINSYSTEM_CHECK =
(BUYER||BROKER||SUPPLIER||LOANSTAR||SAFE_SYSTEM||SAFE_REQ1||SAFE
_REQ2||SAFE_REQ3)

                        /{

                              rcv_order/order,
```

```
        rcv_rfq/rfq_to_supp,
        rcv_qt_supp/send_qt,
        rcv_qt/send_qt_buyer,
        rcv_req/req_loan,
        rcv_brk_order/order_supp,
        rcv_buyerack/send_b_ack,
        rcv_loanack/send_l_ack,
        rcv_suppack/send_s_ack
    }.
```

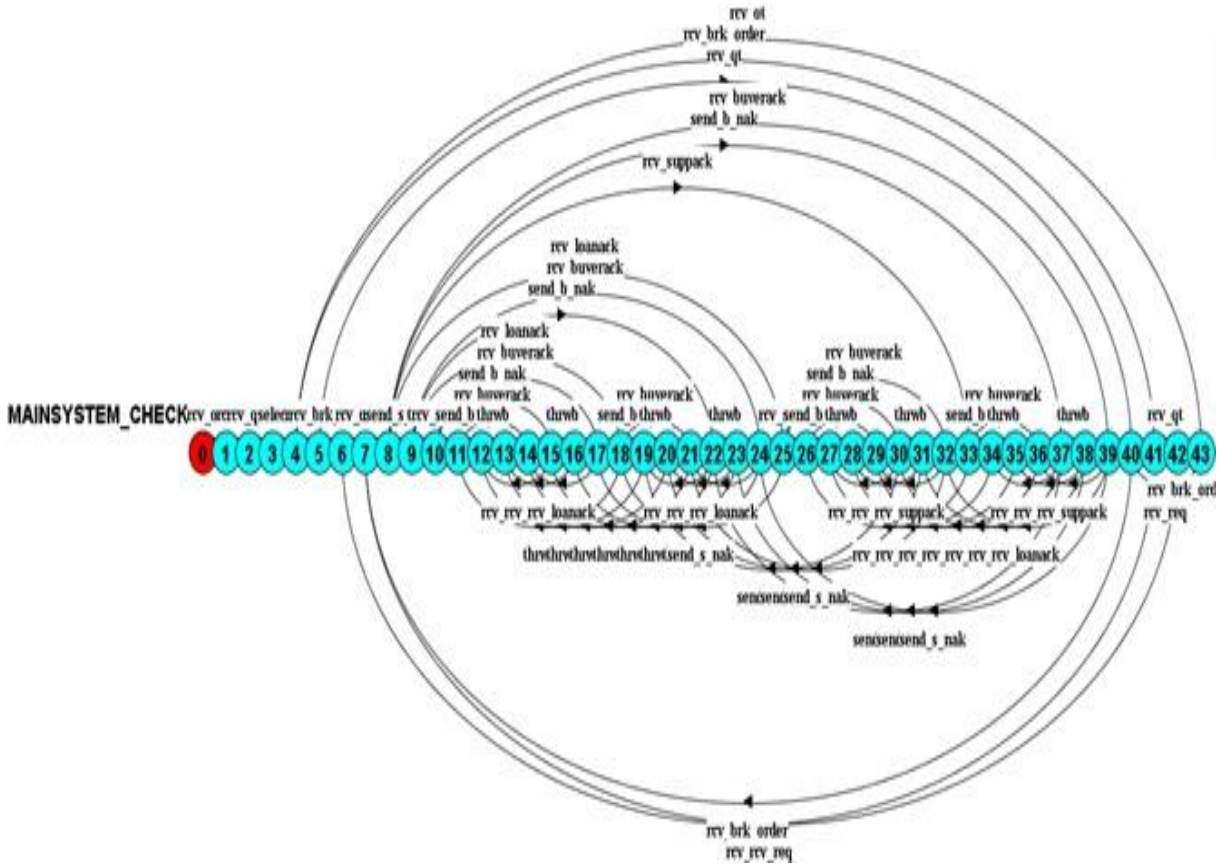

Figure 5.13: LTSA representation of `MAINSYSTEM_CHECK`

CHAPTER 6

Comparison With cCSP

6.1 Introduction

The languages cCSP and FSP provides operators that support sequencing, choice, parallel composition of processes, compensation operators in order to support failed transaction and so on. In some cases the way notations are used in cCSP are similar they are in FSP but most of the cases they are used in a different way. In this chapter comparison of using notations in cCSP and FSP are discussed in brief.

6.2 Sequential Composition

Sequential composition is done in the same way as cCSP and FSP. If two processes P and Q are composed sequentially then process Q starts just after the execution of the actions of the process P. Sequential composition in both FSP and cCSP are denoted by P;Q.

A sequential composition in FSP takes the form:

```
P = (act1->act2->END).
Q = (act3->END).
SEQUENCE=P;Q;END.
```

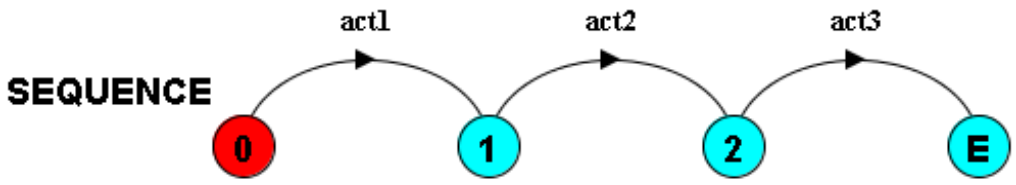

Figure 6.1: LTSA representation of `SEQUENCE`

Where P and Q are sequential processes and END is a local process.

A sequential composition in cCSP takes the form:

$Supplier \triangleq GetOrder?\ a : Qty\ ;\ ProcessOrder(a)$

Supplier gets an order of an item of quantity a then process the order.

6.3 Choice

In a process choice operator is involved where a process is initially engages in more than one action but executes either of the actions. In cCSP "□", in FSP "|" operator represented as a choice operator.

In FSP choices are of two types: Deterministic and Non-deterministic. In deterministic choice the process initially might engages with two or more different actions leading to different succeeding behaviors. On the other hand in non-deterministic choice the process initially might engages with the same action leading to different succeeding behaviors.

Deterministic Choice in FSP:

The example describes the behavior of a dispensing machine which dispenses coffee if the red button is pressed and tea if the blue button is pressed.

```
DRINKS = (red->coffee->DRINKS | blue->tea->DRINKS).
```

Non-Deterministic Choice in FSP:

```
COIN = (toss->heads->COIN | toss->tails->COIN).
```

Choice in cCSP:

$SendQuote(c) \triangleq Quote.c\ ;\ (Ack?\ Accept\ ; SKIPP \ \square\ Ack?\ Reject\ ; THROWW\ )$

Here, after receiving the quote reply might be *Accept* or *Reject*. [9]

6.4 Parallel Composition

A parallel operator is used to compose two or more processes which will execute at the same time in parallel coordinating with each other.

In cCSP (P || Q) describes the parallel composition of the processes P and Q. It allows all the possible interleaving of actions of two processes. In a parallel composition, throwing an interrupt by one process synchronizes with yielding in another process. Parallel operator synchronizes and interleave over observable events. The process $(P \parallel_x Q)$ represents the parallel composition of processes P and Q, synchronizing over the set of events *X*, events not in *X* can occur independently.

In cCSP representation:

$ProcessOrder(m)\ \triangleq RFQ.m\ ; Quote\ ?\ q\ :\ FQ;$

$c \in q \cdot ((Sendorder(c) \| Loan(a)) \| SendQuote(c))$

After selecting a quote Broker send order to supplier, place a request to LoanStar and Send Quote to Buyer simultaneously. That is represented here as the parallel composition of Sendorder(c), Loan(a) and SendQuote(c) [2, 9]

Synchronizing over the events from the SET B:

$System = [Broker \parallel_B Supplier]$

$B = \{RFQ, Quote, Order, Cancel\}$

In FSP (P || Q) describes the concurrent execution of P and Q. In concurrent processes same actions are considered as shared. Shared actions interact with each other for synchronizations. In a parallel composition if there is no shared action between processes the composition will generates all possible sequence of actions. To ensure that the composite processes are synchronized on their desired actions in FSP, relabeling functions are used in parallel compositions.

In FSP representation:

Without relabeling

```
ITCH = (scratch->STOP).

CONVERSE = (think->talk->STOP).

||CONVERSE_ITCH = (ITCH || CONVERSE).
```

With relabeling

```
CLIENT = (call->wait->continue->CLIENT).

SERVER = (request->service->reply->SERVER).

||CLIENT_SERVER = (CLIENT || SERVER)
                  / {call/request reply/wait}.
```

6.5 Compensation Pair

Compensation is part of a compensable process that is used to compensate a failed transaction. The compensations are accumulated during the execution of the processes. The compensations are defined in such a way that when an interrupt occurs at any stage of the transaction, the appropriate compensations are executed for the actions that already did take place.

In cCSP the basic way of constructing a compensable process is through a compensation pair (P÷Q), which is constructed from two standard processes, where P is called the *forward* behavior that executes during normal execution, and Q is called the associated compensation that is designed to compensate the effect of P when the forward behavior of P throws an interrupt. [2, 9]

In FSP compensation process and the main process are two separate processes. The main process and its compensation process are composed in parallel. If any interrupt occurs in the main process the process throw a message, that message is received by the compensation process with the collaboration of a shared action. Thus both processes synchronize and the compensation process runs the compensating actions of those actions that have already took place by the main process. Here P is the main process also called *forward* behavior and Q is its corresponding compensation process, contains the reverse actions done by P. P_Q is the parallel composition of the main process P with its compensation process Q that resembles to the compensation pair (P÷Q) as referred in cCSP.

```
P = (act1->(ack->P|nak->throw->END)).

Q = (throw->cancel_act1->END).

||P_Q = (P||Q).
```

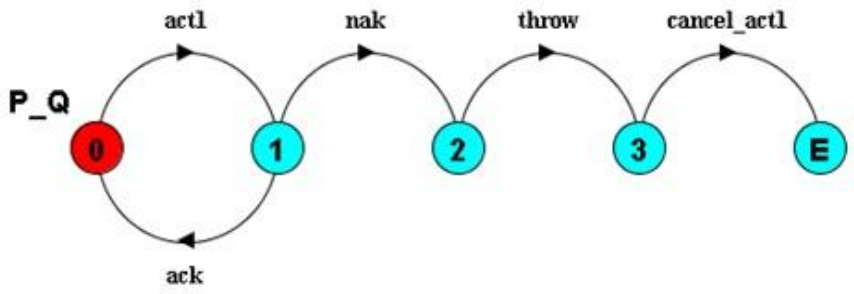

Figure 6.2: LTSA representation of the compensation process `P_Q`.

Table 6.1: Comparison between cCSP and FSP

Syntax	**cCSP**	**FSP**
Sequential Composition of two process(P and Q)	P ; Q	P= (action1->END). Q= (action2->END). SEQUENCE=P;Q ;END.
Choice	P □ Q	Deterministic Choice: (x->P| y->Q) Non-Deterministic Choice: (x->P| x->Q)
Parallel Composition (without relabeling "/")	P || Q	P= (a ->b -> END). Q = (c-> END). ||SYS = (P || Q).
Parallel Composition (with relabeling "/")	$P \|\|_x Q$	P= (a ->b ->e->P). Q = (c->d->Q). ||SYS = (P || Q)/ {c/b, e/d}.
Compensation Pair	P ÷ Q	P=(act1->(ack->P|nak->throw->END)). Q = (throw->cancel_act1->END). ||P_Q = (P||Q).

CHAPTER 7

Conclusion

7.1 Summary

We have analyzed about web services and its composition. We have modeled the Car Broker Web Service by composing several web services to create a composite web service in a choreographic manner. In order to deal with transaction errors we included compensation mechanism in our system. We have used FSP notations to model and verify our desired system. We have created a comparison table of cCSP and FSP to visualize the similarities and variations of notations to describe a system.

7.2 Future Work

Our future plan is to add some other property processes in the system and observing the impacts on our verification mechanism. We also want to model and verify service orchestration with another system including compensation in a similar manner that we have followed in this project.

Appendix A

A.1 Buyer Web Service

```
BUYER = (order->rcv_qt->reply->(send_b_ack->BUYER|send_b_nak-
>thrwb->END)).

/**Buyer Compensation Process**/

COMP_B = (thrwb->cancel_rcv_qt->cancel_order->END).

MSGB = (thrwb->msgb->END).
```

A.2 Broker Web Service

```
/**Start of BROKER Section**/

BRK_PHASE1 = (rcv_order->rfq_to_supp->rcv_qt_supp->select_qt-
>END).


REQ1 = (select_qt->send_qt_buyer->reply->END).

REQ2 = (select_qt->order_supp->reply->END).

REQ3 = (select_qt->req_loan->reply->END).


RCV1 = (reply->rcv_buyerack->END).

RCV2 = (reply->rcv_suppack->END).

RCV3 = (reply->rcv_loanack->END).


||REQ=(REQ1||REQ2||REQ3).
```

```
||RCV=(RCV1||RCV2||RCV3).

||BRK_PHASE2=(REQ||RCV).

||BROKER =(BRK_PHASE1||BRK_PHASE2).

/**END of BROKER Section**/


/**Broker Compensation Process**/

CMP_REQ1 = (thrwbrk->wdrw_buyer_qt->reqwdrwn->END).

CMP_REQ2 = (thrwbrk->wdrw_s_order->reqwdrwn->END).

CMP_REQ3 = (thrwbrk->wdrw_l_req->reqwdrwn->END).


||BRK_PHASE2_COMP = (CMP_REQ1||CMP_REQ2||CMP_REQ3).


BRK_PHASE1_COMP = (reqwdrwn->cancel_qt_select-
>cancel_supp_qt_rcv->cancel_rfq_to_supp->cancel_buyer_order-
>END).


||COMP_BRK = (BRK_PHASE2_COMP||BRK_PHASE1_COMP).

/**end of Broker Compensation Process**/
```

A.3 Supplier Web Service

```
SUPPLIER = (rcv_rfq->send_qt->rcv_brk_order->reply->(send_s_ack-
>SUPPLIER|send_s_nak->thrws->END)).

/**Supplier Compensation Process**/

COMP_S = (thrws->cancel_brk_order->cancel_qt->cancel_rfq->END).

MSGS = (thrws->msgs->END).
```

A.4 LoanStar Web Service

```
LOANSTAR = (rcv_req->reply->(send_l_ack->LOANSTAR|send_l_nak-
>thrwl->END)).

/**LoanStar Compensation Process**/

COMP_L = (thrwl->cancel_loan_req->END).

MSGL = (thrwl->msgl->END).
```

A.5 Main Compensation Process

```
/**Throwing messages to compensate other processes except
Buyer**/

BUYERMSG_TO_COMP_BRK = (thrwbrk->END).

BUYERMSG_TO_COMP_S = (thrws->END).

BUYERMSG_TO_COMP_L = (thrwl->END).


||FROM_BUYER =
(BUYERMSG_TO_COMP_BRK||BUYERMSG_TO_COMP_S||BUYERMSG_TO_COMP_L).


/**Throwing messages to compensate other processes except
Supplier**/

SUPPMSG_TO_COMP_B = (thrwb->END).

SUPPMSG_TO_COMP_BRK = (thrwbrk->END).

SUPPMSG_TO_COMP_L = (thrwl->END).


||FROM_SUPP =
(SUPPMSG_TO_COMP_B||SUPPMSG_TO_COMP_BRK||SUPPMSG_TO_COMP_L).
```

```
/**Throwing messages to compensate other processes except
LoanStar**/

LOANMSG_TO_COMP_B = (thrwb->END).

LOANMSG_TO_COMP_BRK = (thrwbrk->END).

LOANMSG_TO_COMP_S = (thrws->END).


||FROM_LOAN =
(LOANMSG_TO_COMP_B||LOANMSG_TO_COMP_BRK||LOANMSG_TO_COMP_S).


/**Main Compensation Process**/

CMAIN = (msgb->COMP_EXCPT_BUYER|msgs->COMP_EXCPT_SUPP|msgl-
>COMP_EXCPT_LOAN),

            COMP_EXCPT_BUYER = FROM_BUYER;END,

            COMP_EXCPT_SUPP = FROM_SUPP;END,

            COMP_EXCPT_LOAN = FROM_LOAN;END.


/**End of Main Compensation Process**/
```

A.6 Property Processes

```
property SAFE_COMP_B = (send_b_nak->cancel_rcv_qt->SAFE_COMP_B).

||BSAFE=(BUYER||COMP_B||SAFE_COMP_B).


property SAFE_COMP_S = (send_s_nak->cancel_brk_order-
>SAFE_COMP_S).

||SSAFE=(SUPPLIER||COMP_S||SAFE_COMP_S).
```

```
property SAFE_COMP_L = (send_l_nak->cancel_loan_req-
>SAFE_COMP_L).

||LSAFE=(LOANSTAR||COMP_L||SAFE_COMP_L).


property SAFE_MSG_BRK = (msgb->thrwbrk->reqwdrwn-
>cancel_buyer_order->SAFE_MSG_BRK

                          |msgl->thrwbrk->reqwdrwn-
>cancel_buyer_order->SAFE_MSG_BRK

                                |msgs->thrwbrk->reqwdrwn-
>cancel_buyer_order->SAFE_MSG_BRK).


property SAFE_MSG_B = (msgs->thrwb->cancel_order-
>SAFE_MSG_B|msgl->thrwb->cancel_order->SAFE_MSG_B).


property SAFE_MSG_S = (msgb->thrws->cancel_rfq->SAFE_MSG_S|msgl-
>thrws->cancel_rfq->SAFE_MSG_S).


property SAFE_MSG_L = (msgb->thrwl->cancel_loan_req-
>SAFE_MSG_L|msgs->thrwl->cancel_loan_req->SAFE_MSG_L).


||CMAIN_CHECK=(CMAIN||COMP_B||COMP_BRK||COMP_S||COMP_L||SAFE_MSG
_BRK||SAFE_MSG_B||SAFE_MSG_S||SAFE_MSG_L).


property SAFE_SYSTEM = (rcv_order->rcv_rfq->rcv_qt_supp-
>select_qt->SAFE_SYSTEM).


property SAFE_REQ1 = (select_qt->rcv_qt->SAFE_REQ1).

property SAFE_REQ2 = (select_qt->rcv_brk_order->SAFE_REQ2).

property SAFE_REQ3 = (select_qt->rcv_req->SAFE_REQ3).
```

```
||MAINSYSTEM_CHECK =
(BUYER||BROKER||SUPPLIER||LOANSTAR||SAFE_SYSTEM||SAFE_REQ1||SAFE
_REQ2||SAFE_REQ3)
                    /{
                         rcv_order/order,
                         rcv_rfq/rfq_to_supp,
                         rcv_qt_supp/send_qt,

                         rcv_qt/send_qt_buyer,
                         rcv_req/req_loan,
                         rcv_brk_order/order_supp,

                         rcv_buyerack/send_b_ack,
                         rcv_loanack/send_l_ack,
                         rcv_suppack/send_s_ack
                    }.
```

A.7 Car Broker Web Service

```
||CARBROKERSERVICE = (

     BUYER||BROKER||SUPPLIER||LOANSTAR||
     MSGB||MSGS||MSGL||CMAIN||
     COMP_B||COMP_BRK||COMP_S||COMP_L||
     SAFE_COMP_B||SAFE_COMP_S||SAFE_COMP_L||
     SAFE_MSG_BRK||SAFE_MSG_B||SAFE_MSG_S||SAFE_MSG_L||
     SAFE_SYSTEM||SAFE_REQ1||SAFE_REQ2||SAFE_REQ3

                          )

                         /{

                              rcv_order/order,

                              rcv_rfq/rfq_to_supp,

                              rcv_qt_supp/send_qt,


                              rcv_qt/send_qt_buyer,

                              rcv_req/req_loan,

                              rcv_brk_order/order_supp,


                              rcv_buyerack/send_b_ack,

                              rcv_loanack/send_l_ack,

                              rcv_suppack/send_s_ack

                         }.
```

References

[1] Florian Daniel, Barbara Pernici, *Politecnico deo Milano, Italy,* "Web Service Orchestration and Choreography: Enabling Business Processes on the Web", Chapter XII, January 2006.

[2] Shamim Ripon, Mohammad Salah Uddin and Aoyan Barua, "Web Service composition – BPEL vs cCSP Process Algebra", *Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh.*

[3] Abdaladhem Albreshne, Patrik Fuhrer, Jacques Pasquier, "Web Services Orchestration and Composition Case Study of Web services Composition", September 2009.

[4] B. Margolis with J. Sharpe, "Based on SOA for Business Developer, Concepts, BPEL, and SCA", McPress, Lewisville, TX, 2007.

[5] Howard Foster, Sebastian Uchitel, Jeff Magee, Jeff Kramer, "Model-based Verification of Web Service Compositions", *Department of Computing, Imperial College London*.

[6] Jeff Magee, Jeff Kramer, "Concurrency: State Models and Java Programs", Text Book, 2nd Edition, John Wiley & Sons, Ltd, 2006.

[7] Mark Austin, John Johnson, "Compositional Behavior Modeling and Formal Validation of Canal System Operations with Finite State Automata", ISR Technical report 2011-04, The Institute for Systems Research.

[8] Shamim H. Ripon, *Department of Computing Science, University of Glasgow, UK*, Michael Butler, *School of Electronics and Computer Science, University of Southampton, UK*, "Formalizing cCSP Synchronous Semantics in PVS".

[9] Shamim H. Ripon, *Department of Computing Science, University of Glasgow, UK*, "Process Algebraic Support for Web Service Composition".