# Analyzing and Verification of Web Service Composition

By

**Maliha Tunzira**

&

**Khadeja Akter**


Computer Science and Engineering

East West University

Summer 2019

# Analyzing and Verification of Web Service Composition

Submitted By

**Maliha Tunzira**

ID : 2013-3-60-010

&

**Khadeja Akter**

ID : 2014-3-60-073

A project submitted in partial fulfillment 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

Summer 2019

# Declaration

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

_____

Maliha Tunzira

(2013-3-60-010)

_____

Khadeja Akter

(2014-3-60-073)

# Letter of Acceptance

The Project entitled " Analyzing and Verification of Web Service Composition" submitted by Maliha Tunzira [2013-3-60-010] and Khadeja Akter [2014-3-60-073] to the Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh in the semester of Summer 2019 is approved satisfactory in partial fulfillments for the award of the degree of Bachelor of Science in Computer Science and Engineering.

_____

Dr. Shamim Hasnat Ripon

Professor,

Department of Computer Science and Engineering

East West University

Dhaka, Bangladesh.

_____

Dr. Taskeed Jabid

Associate Professor & Chairperson,

Department of Computer Science and Engineering

East West University

Dhaka, Bangladesh.

# Abstract

In the internet's distributed setting, a platform-independent software component is called Web Service. Many business institutions publish advanced features of their apps on the internet. A Web Service has only a confined feature. It is therefore a crying need to integrate Web Services and organize them into an objective-oriented framework to promote company connections. This is both hard and sensitive to provide error handling for a transaction various participants in managing defects. We have created a Real Estate Web Service system in this venture. We modeled the FSP choreography of service and used the LTSA tool to visualize the transformations. We attempted to check the structure of the scheme using ownership procedures applicable in FSP before implementing for every model.

# Acknowledgement

It is with enormous appreciation that we acknowledge the contribution of our supervisor, **Dr. Shamim Hasnat Ripon**, Professor of Department of Computer Science and Engineering, East West University. This thesis would have stayed a dream without his adequate guidance, motivation and help. We find working with him to be an accomplishment. We are also indebted for their assistance and encouragement to our parents, other faculty members of the department and friends. Finally thanks to the Almighty, who gave us the willpower to effectively finish the thesis.

# Table of Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Introduction and Motivation

We live in such era where consumers are online looking for information or purchase something they want. This buying behavior trend increased the importance of web services in businesses. Many business companies or enterprise publish their applications functionalities on the web using a web service format. A web service is a technology independent service that helps different applications to communication with each other. Web services are known as self-contained, modular units of application logic, which provide business functionality to other applications through an Internet connection.

In this mechanical period, business applications like web service permit more prominent effectiveness and accessibility for business. A web service alone has a constrained usefulness which may not be adequate to react the client's solicitation. While a composition of a few web service can accomplish a particular objective. From a client's point of view, the composition may be considered as a straightforward web service, despite the fact that its made out of a few web service. In a quintessence, the total is a coordinated effort of many web service providers.

As models are simplified representations of real-world entities, we made model to better understand it. To focus on particular interesting aspects, visualize potential outcomes and create mechanisms to test and verify an approach. We need model checking to confirm accuracy properties, for example, the absence of deadlocks and similar critical states that can make the framework or the system crash. Each model ought to be confirmed before implementation. There are different languages to model a system and check it properly. BPEL, FSP, cCSP are the most usable language to construct a model a system with their notations. Among them FSP has the most expressive and ground-breaking way to deal with envision the system. To give atomicity to an exchange taking care of multiple partners where different accomplices are included are both troublesome and basic. A solution to the problem is to provide such mechanism to control the actions that cannot be undone automatically.

## 1.2 Objectives

The objectives of our project are as follows :

- Analyzing the web service composition in respect to the composition mechanism choreography.
- Modeling a composition using Finite State Process (FSP) notations and Labeled Transition System Analyzer (LTSA) tool.
- 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 contribution in the project as follows :

We have utilized Finite State Process (FSP) documentations to depict the model and LTSA tool to create the relating Labeled change outlines. We dissected the model and distinguished different segments of the web administration just as the piece among the administrations. Then implement the system according to their interactions.

We added some safety properties to confirm synchronizations among procedures in a simultaneous execution and checked the accuracy properties, for example, the nonappearance of deadlocks and comparable basic expresses that can make framework pound.

## 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 a way to compose the web service (Choreography). 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.

**Chapter 3 :** This chapter describes about Real Estate service composition including the contribution of each web services in the system.

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

**Chapter 5 :** Define some safety properties in order to verify our web service.

**Chapter 6 :** 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 computing components that are autonomous and interacting. That solves particular duties ranging from easy demands to complicated business processes and communicating using SOAP standard XML messages. They are autonomous platform self-describing, self-contained, and can be released, situated, and accessed through conventional Internet protocols.

Composition of Web service is a mixture of several current services to produce a composite value-added service. It aims to combine and coordinate a set of services to achieve functionality that can't be achieved through current services. Service community attempts suppose that all services are based on an interface model.

There are two methods for describing the sequence of stuff that make up the composite workflows: orchestration and choreography[1][2][3].

## 2.2 Choreography

Service choreography is collaborative and enables every side engaged in the communication to define their role. In choreography, a worldwide view specifies the logic of message-based relationships between the members.

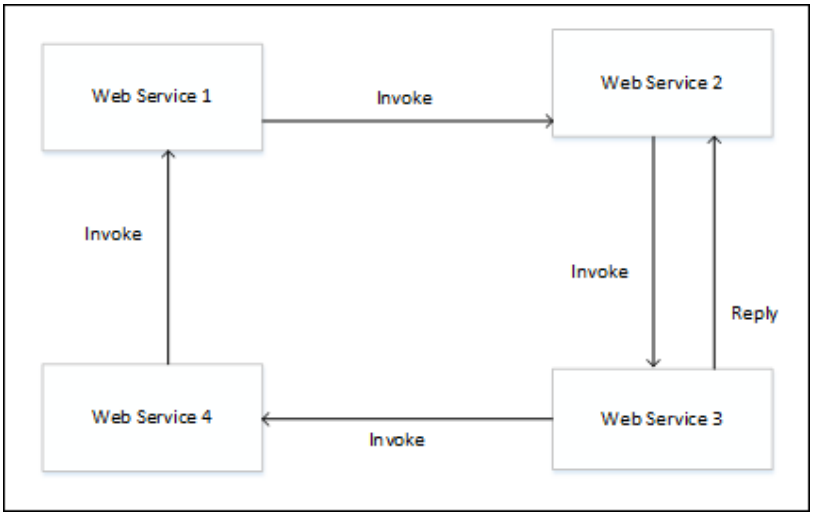


Figure 2.1: Web service choreography

Choreography of Web services relates to a Web service's public protocol, defining the form and order of messages passed between such a Web service and its customers or associates. It reflects a worldwide overview of the measurable conduct of each service involved in communication, characterized by the public exchange of messages, communication rules, and arrangements between two or more nodes of business operations. Normally, it is connected with connections occurring between various Web services instead of a particular business mechanism executed by a single party.

The normal Web Services Choreography Translation Language supports the choreography mechanism. As a coating to bridge the gap between current orchestration techniques, choreography is been suggested[1][4].

## 2.3 Modelling processes in FSP

FSP stands for Finite State Processes. Finite State Processes is a logarithmic documentation to portray procedure models. The constructed FSP can be utilized to demonstrate the careful progress of work process forms through a modeling tool such as Labeled Transition System Analyzer (LTSA), which gives gathering of FSP into a Labeled Transition System (LTS). These are depicted literally as Finite State Processes and showed and broke down by the Labeled Transition System Analyzer LTSA analysis Tool. This tool offers chance to test the model work processes before actualizing the model. LTS is the graphical structure and FSP is the algebraic structure[5][6].

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. [verification compensation, EWU]

A service may be a multi-process process or structure. A process is a continuous program implementation. It is designed as a finite state machine by implementing a series of atomic activities that traverses from state to state. Practically speaking, an action could be a message, a signal, or maybe a typical job implementation[5].

There are two types of Finite State 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 may be indexed which greatly increases the descriptive power of FSP.

**Action Prefix "->"**

Action prefix describes a process involving action a and then acting as outlined by P. More operationally, a transition between states is defined by the action prefix. The following recursive definition describes the process CLOCK which repeatedly engages in the action tick.

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

The LTS corresponding to the definition above is:

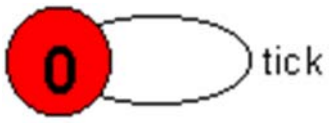



Figure 2.2 : LTSA Representation of Clock 1

**Choice "|"**

**Determine**

Choice defines a process that originally involves either a or b actions. Subsequent behavior is defined by P after the first action has been conducted if the first event was a, or by Q if the first event was b. The LTS corresponding to this phase has out of the original state two feasible movements a and b. The example describes the behavior of a dispensing machine which dispenses coffee if the black button is pressed and tea if the white button is pressed.

```
DRINKS = (black -> coffee -> DRINKS | white -> tea ->
DRINKS).
```

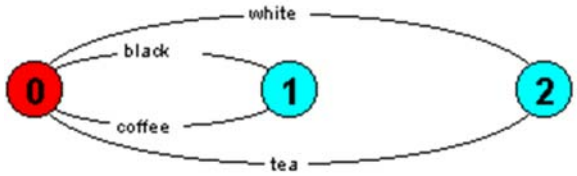



Figure 2.3 : LTSA Representation of Choice effect

**Non-determinism**

In not distinguishing between inner and external choices, FSP follows CCS rather than CSP. Consequently, non-deterministic choice is demonstrated merely by having the same action leading to two distinct successor behaviors as shown in the instance of throwing a coin.

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

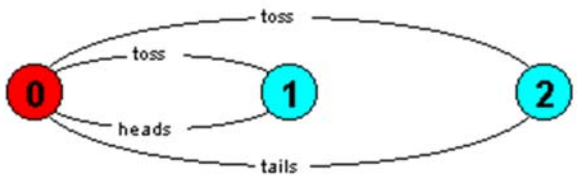



Figure 2.4 : LTSA Representation of Choice effect

**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..3] ->
      if x>=2 then
        (high -> LEVEL)
      else
        (low -> LEVEL)).
```

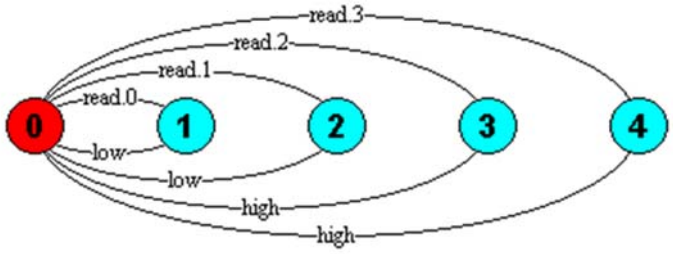



Figure 2.5 : LTSA Representation of Condition

**Guards**

A guarded transition takes the form `(when B a -> P)` which means that the action a is eligible when the guard B is true, otherwise a cannot be chosen for execution. The following example uses guards to define a bounded semaphore:

```
const Max = 4
range Int = 0..Max

BSEMA(Init=0) = BSEMA[Init],
BSEMA[v:Int]  = (when (v<Max) up   -> BSEMA[v+1]
                |when (v>0)   down -> BSEMA[v-1]
                   ).
```

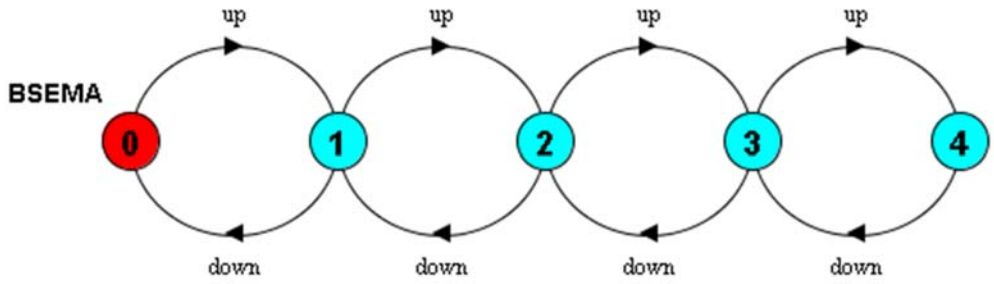



Figure 2.6 : LTSA Representation of Guard

**Composite Processes**

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

**Parallel Composition "||"**

The parallel structure of P and Q procedures is expressed ( P || Q). It builds an LTS that enables the activities of the two procedures to intersect as much as possible. Actions occurring in both P and Q alphabets restrict interleaving as these activities must be carried out simultaneously by both procedures. These shared activities synchronize the two processes ' implementation. If the processes contain no shared actions then the composite state machine will describe all interleaving. In the following example, x is an action shared by the processes A and B.

```
A = (a -> x -> A).
B = (b -> x -> B).
||SYS = (A || B).
```

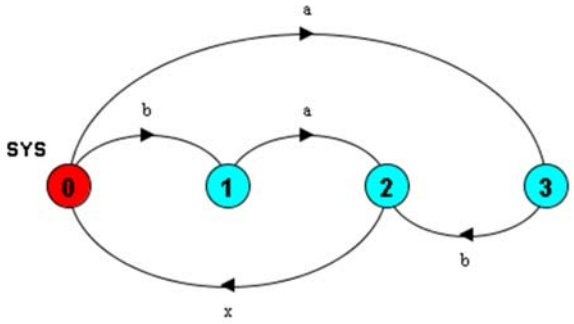



Figure 2.7 : LTSA Representation of Parallel process

**Process Labeling**

The garage system could be more elegantly expressed using process labelling as shown below:

```
CAR   = (outside -> enter -> ingarage -> exit -> CAR).

GARAGE(N=2)  =  (car[x:1..N].enter  ->  car[x].exit  ->
GARAGE).

||SHARE     = (car[1]:CAR || car[2]:CAR || GARAGE).
```

The construction car[1]:CAR prefixes all action labels within CAR with the label car[1]. Finally, we can generalise the description such that it describes a systems with N cars sharing the garage:

```
||SHARE(N=3) = (car[1..N]:CAR || GARAGE(N)).
```

The || operator is commutative `(P||Q ° Q||P).`

**Relabeling "/"**

Relabeling functions are applied to processes and the action label names are changed. Usually this is performed to guarantee that the right activities are synchronized by composed procedures. For both primitive and composite procedures, a relabeling feature can be implemented. In composition, however, it is usually used more. The overall form of the function of relabeling is / {newlabel_1/oldlabel_1,.... , newlabel_n / oldlabel_n}. The instance demonstrates the structure of two procedures of binary semaphore to produce a semaphore that can be increased twice (by up). The diagram demonstrates how the down action of the first SEMA method is linked to the up action of the next weekphore by relabeling both to mid. The alphabet of *SEMA2* is {*up, down, mid*} and its LTS is depicted below:[6]

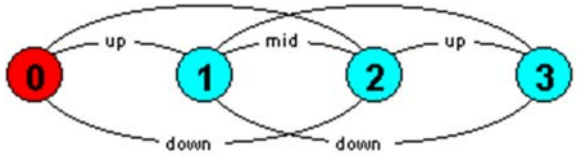



Figure 2.8 : LTSA Representation of relabeling

**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 }. Hiding expressions can be applied to both primitive and composite processes but are generally used in defining composites.

## 2.4 Property Processes to Verify the System

**Safety Properties**

LTSA specifies safety characteristics as deterministic primitive procedures that do not contain silent (tau) transitions (no hiding). The keyword property denotes security ownership processes. For the scheme with which it is comprised, a property method informally establishes a set of acceptable behaviors. A scheme S will fulfill a property P if S can only produce action sequences (traces) when restricted to the alphabet of P, are acceptable to P. For example, the following property specifies that only behavior in which knock occurs before enter is acceptable.

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

The systems specified below would generate property violations:

```
HESITANT = (knock->knock->enter->HESITANT).
      IMPATIENT = (enter -> IMPATIENT)
||SysA = (HESITANT || POLITE).
||SysB = (IMPATIENT || POLITE).
```

Property procedures do not limit the operation of the devices with which they are comprised. They are collected into LTS "picture," which accepts all possible alphabet interleaving. Violating action sequences, however, leads to a state of mistake (represented as -1) as shown in POLITE's LTS [5].

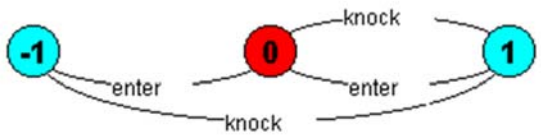



Figure 2.9 : LTSA Representation of Property verification

# Chapter 3

# Real Estate Service Composition

## 3.1 Real Estate Web Service

An online version of the real estate sector, web real estate is the notion of selling or renting residential properties and buying or renting a property for customers. Web real estate is often run by landlords themselves. There are several special cases, though, in which there would be an online real estate agent, still dealing through the web and often saying a flat fee rather than a commission based on the percentage of overall sales. Web real estate emerged around 1999 when sophisticated technology and facts show that the landlords themselves sold more than 1 million homes in America in 2000. Zillow, Trulia, Yahoo! Real Estate, Redfin and Realtor.com are some of the top web real estate sites[7].

The method of web real estate idea generally starts with landlords posting their properties on online sites with their quoted cost. The more website owners list their properties, the more data is disseminated. Search engines are generally their first pit-stop as clients searching for a piece of property. Client searches at the website. Once a prospective client contacts the landlord, if not indicated, they would go through the estate information–sizing, facilities, situation, and pricing. Agent can contact with landlord and bank. After which, an appointment for inspecting the property would normally be planned and in some cases, prospective clients may ask that certain facilities or sections of the estate be refurnished. If terms of the contract are completed between both sides, the client would normally deal for the best deal if the landlord could ask for a payment. Client can connect with bank through the website. Lastly, both sides will agree on a complete payment date, signing on formal payment, and passing on keys to the estate.

## 3.2 Client

In our proposed system client can search at website. Then he/she selects property according to his/her preference. Then he/ she has to select agent from the website list. After agent contacts with landlord, client confirms the deal. In mean time agent communicates with bank and reply to client. Client finally communicates with bank for loan. When everything is sorted out about deal, client gives payment and takes the property.

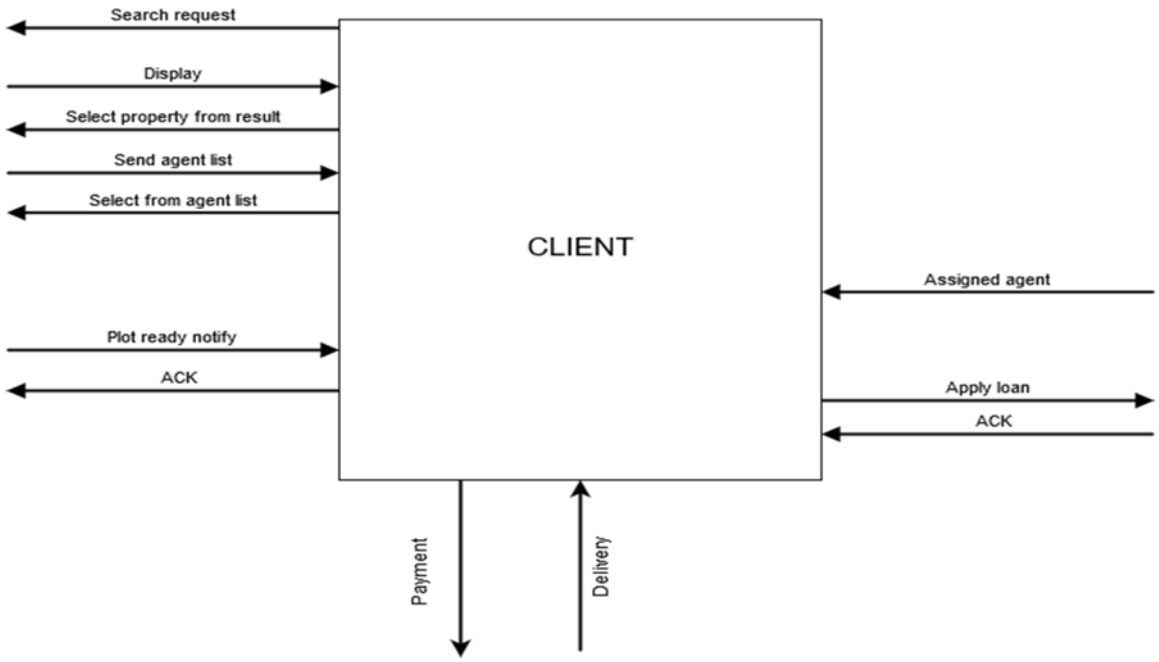



Figure 3.1 : Architecture view of Client

## 3.3 Website

Website tool is the medium of communication between client and other processes. When client search for property it connects with property database. And convey the found result to client. Then it also show the agent list to client when he/she selects any property. After assigning the agent it stops communicate before any need.

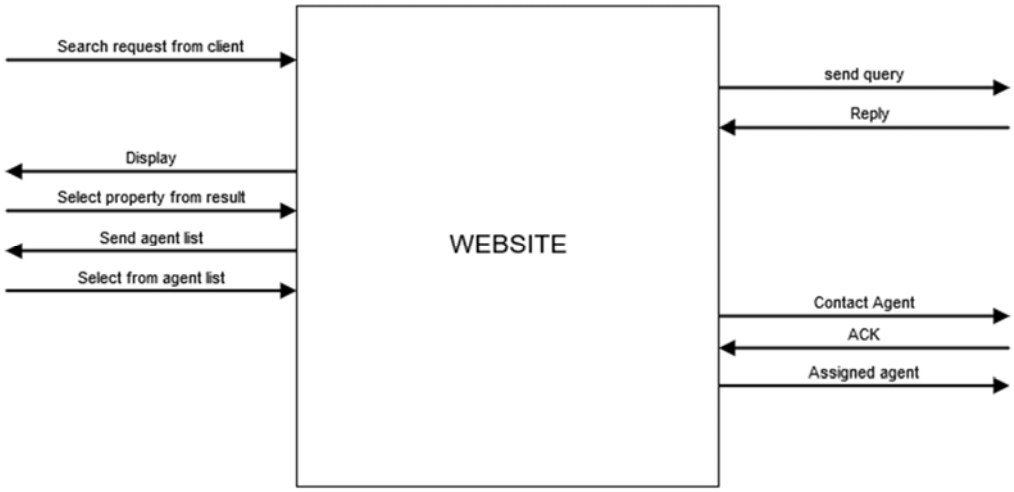



Figure 3.2 : Architecture view of Website

## 3.4 Plot Data

Plot Data is the database of this website. Here all the property information and landlord contacts can be found. When client searches for property, website asks the preferred property to database. It shows the necessary information through website.

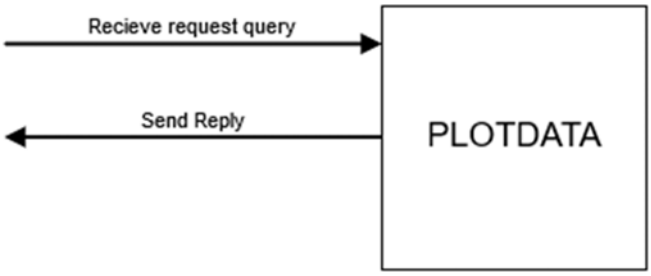



Figure 3.3 : Architecture view of Plot Data

## 3.5 Agent

Here Agent is the broker of the whole system. It first contacted with client through the website. Website checks the availability of agent. When it confirms, website assigns fitted agent for the client. Agent contacts with client and then with landlord. When landlord gives consent, agent gives this message to client. It also gives plot availability confirmation to client. Client finally confirm the deal.


Figure 3.4 : Architecture view of Agent

## 3.6 Landlord

Here landlord is the owner of the properties. They contact through the agent. When agent sends request for plot they replies through. Then they can contact with client with their own and make the deal. Finally they receive payment and deliver the property to client.


Figure 3.5 : Architecture view of Landlord

## 3.7 Bank

Bank is in here for the deposit and payment purpose. When Client asks for loan, it verifies for affordability of client and gives reply.

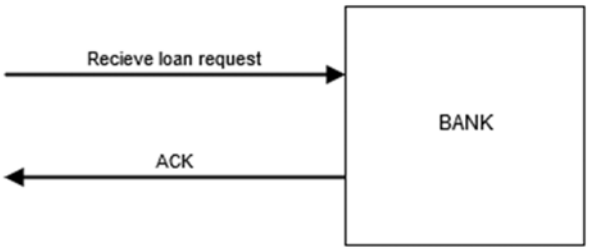



Figure 3.6 : Architecture view of Bank

## 3.8 The system architectural structure

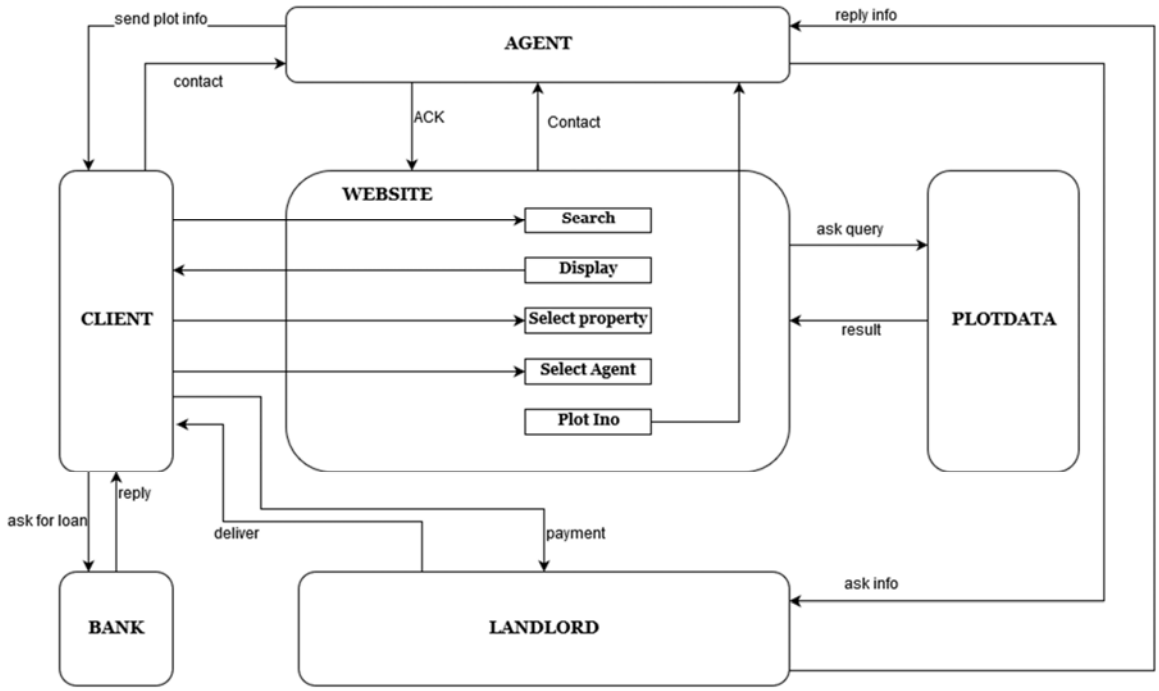



Figure3.7: Architectural View of Real state Webservice

## 3.9 Sequence Diagram of Overall System

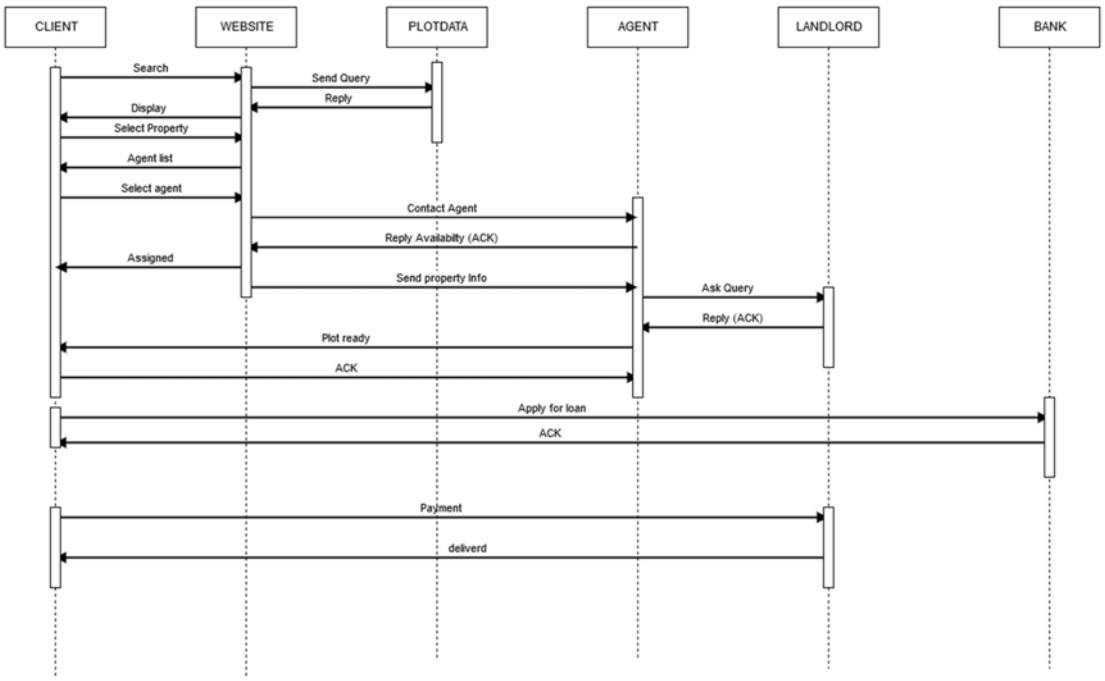



Figure 3.8 : Sequence diagram of overall System

# Chapter 4

# Service Composition in FSP

## 4.1 Code Representation

In our system we have six major processes which have their own safety property to ensure good composition. In FSP we modelled the system like that , Client will search at Website then it gives data through Plotdata. Website will display and will assign Agent with its consent. Agent will contact with Landlord. Landlord will give confirmation to Agent, it will convey this to client. Then Landlord will ready the property and give confirmation through Agent. Client will apply for loan to Bank. Bank will verify and reply to Client. Then Client can pay to Landlord and take property from it. [5]

## 4.2 Modelling Single Process in FSP

**CLIENT**

Client process consists with sequence of action and choice. The process starts with `search` , then receive display action which is `c.display` . Then Client select plot, get agent list, select agent and assigned Agent. The following actions are `c.sel_prp, c.aglist, c.sel_ag`. When Client will have confirmation which is `c.ay_ack`, it will apply for loan which is `c.apply_loan`. Then Client will get reply, pay to Landlord and take delivery which are `c.yb_ack, c.paymeny, c.deliver.` If Client get negative reply then it will get `c.nb_ack, c.an_ack.`

```
CLIENT = ( c.search->c.display->c.sel_prp->c.aglist->
     c.sel_ag -> c.assigned -> c.plot_ready ->
(c.ay_ack ->   c.apply_loan ->(c.yb_ack -> c.paymeny -
>c.deliver -> CLIENT     |c.nb_ack ->CLIENT) | c.an_ack
-> CLIENT)).
```

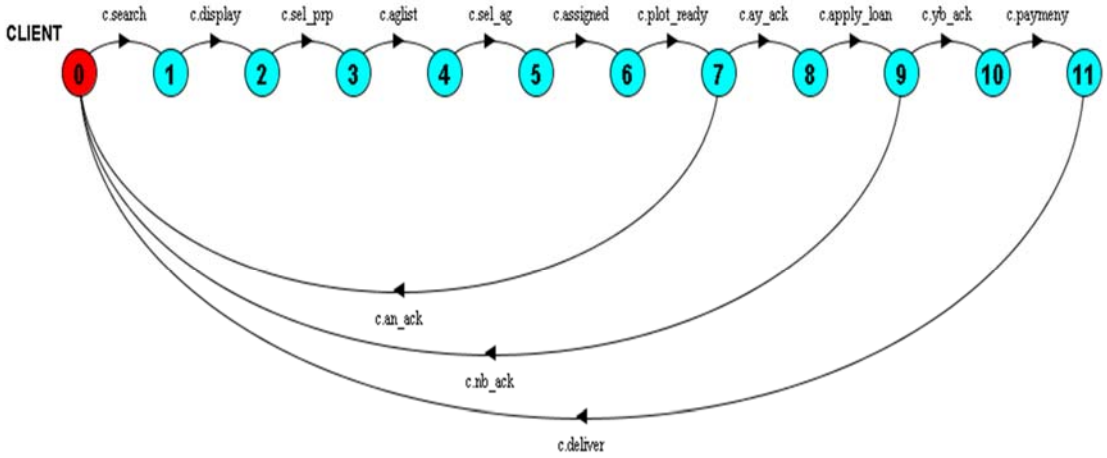



Figure 4.1 : LTSA Representation of Client Process

**WEBSITE**

Website also consists with sequence of action and choice. First it will search action from CLIENT, send query to PLOTDATA and receive reply, display to Client again; which are `w.search`, `w.send_qry`, `w.reply`, `w.display`. When Client select property it will show agentlist and verify agent availability; which are `w.sel_prop`, `w.aglist`, `w.sel_ag`, `w.contact_ag`. If Agent is not available it gets negative ack which is `w.an_ack`.

```
WEBSITE = ( w.search -> w.send_qry -> w.reply ->
w.display ->   w.sel_prop -> w.aglist -> w.sel_ag ->
w.contact_ag ->     (w.ay_ack -> w.assigned ->
w.prop_info -> WEBSITE | w.an_ack -> WEBSITE)).
```

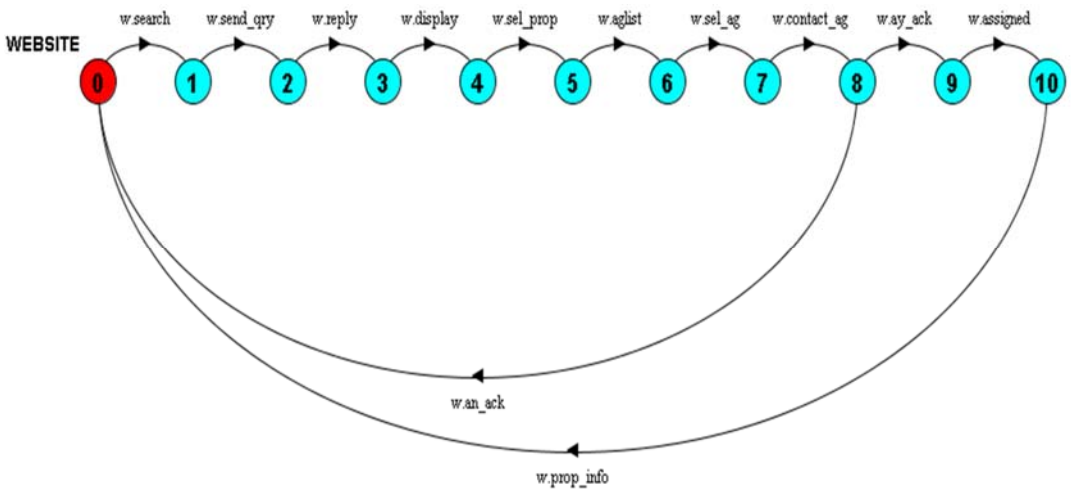



Figure 4.2 : LTSA Representation of Website Process

**PLOTDATA**

Plotdata has not complicated communication with processes. It only get query which is `p.send_qry` and send appropriate property list which is `p.reply.`

```
PLOTDATA = ( p.send_qry -> p.reply -> PLOTDATA).
```

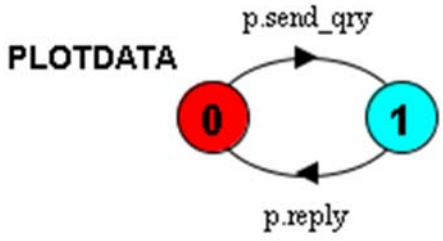



Figure 4.3 : LTSA Representation of Plot Data process

**AGENT**

Agent has lot of actions and choices. When Website contacts with it, it gives availability confirmation then start contacts with Landlord. Agent gets reply from Landlord which are l.y_ack, a.plot_ready . Sometimes it can get negative reply also which are c.n_ack, l.n_ack, a.n_ack. These all are defined by choices.

```
AGENT =( a.contact ->( a.y_ack->a.prop_info -> ask_qry
->   (l.y_ack -> a.plot_ready ->( c.y_ack-> AGENT |
c.n_ack ->     AGENT)| l.n_ack ->AGENT) | a.n_ack ->
AGENT)).
```

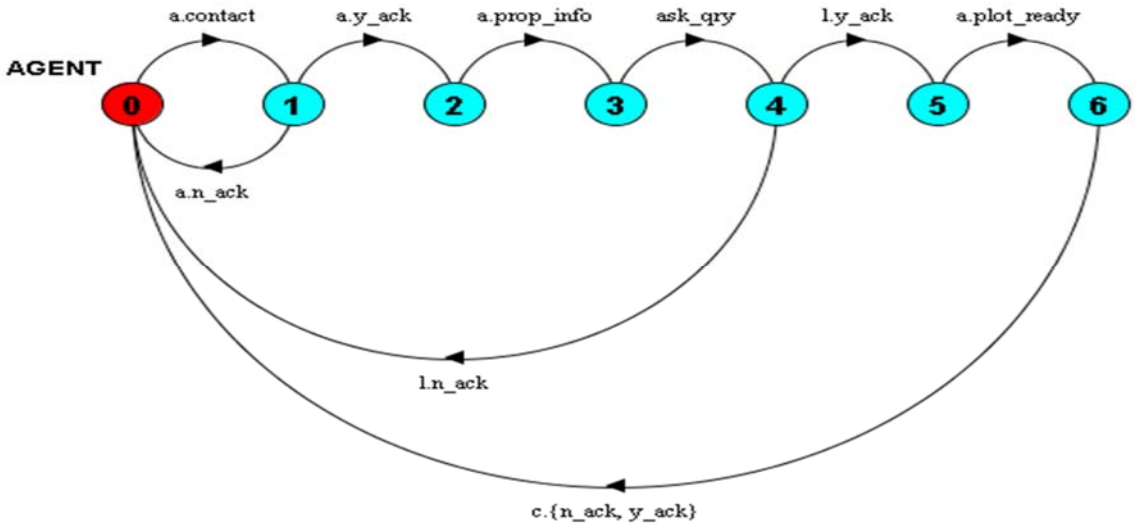



Figure 4.4 : LTSA Representation of Agent Process

**LANDLORD**

Landlord basically confirms deal and confirms the availability of property. It first contacts with Agent which is `l.ask_qry .` Then following actions like give confirmation for the plot which are `l.y_ack , rcv_payment, deliver`. It can gives negative reply too which is `l.n_ack.`

```
LANDLORD = ( l.ask_qry -> (l.y_ack -> rcv_payment ->
deliver ->          LANDLORD | l.n_ack -> LANDLORD)).
```

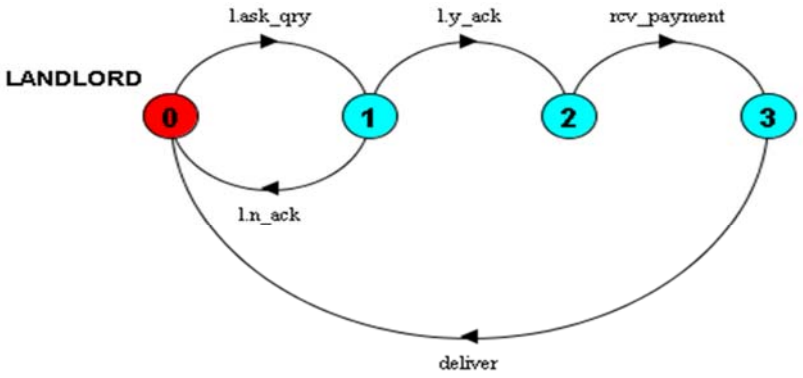



Figure 4.5 : LTSA Representation of Landlord

**BANK**

Bank actions are very simple. It gets request from Client `rcv_loan.req` then reply with ack. Both positive and negative reply can be get by the Client which are `b.pack, b.nack.`

```
BANK = (rcv_loan.req -> (b.pack-> BANK|b.nack ->BANK)).
```

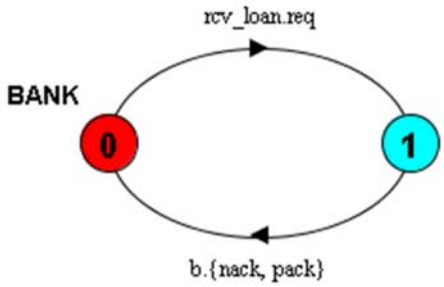



Figure 4.6 : LTSA Representation of Bank Process

## 4.3 Modeling Double Processes in FSP

### CLIENT and WEBSITE

```
CLIENT = ( c.search-> c.display -> c.sel_prp ->
c.aglist -> c.sel_ag -> contact -> (agpack -
>c.assigned -> webprop -> al.qry -> (lq.pack ->
c.plot_ready -> (c.ay_ack -> c.apply_loan -> (c.yb_ack
-> c.paymeny ->c.deliver -> CLIENT |c.nb_ack ->CLIENT)
| c.an_ack -> CLIENT)|lq.nack -> CLIENT)| agnack -
>CLIENT)).


WEBSITE = ( w.search -> w.send_qry -> w.reply ->
w.display -> w.sel_prop -> w.aglist -> w.sel_ag ->
w.contact_ag -> (w.ay_ack -> w.assigned ->
w.prop_info -> la.qry -> WEBSITE | w.an_ack ->
WEBSITE)).


||CW = ( CLIENT || WEBSITE ) /
            { c.search/w.search , c.display/w.display ,
             c.sel_prp/w.sel_prop , c.aglist/w.aglist ,
             c.sel_ag/w.sel_ag , contact/w.contact_ag ,
                    agpack/w.ay_ack , agnack/w.an_ack ,
         c.assigned/w.assigned , webprop/w.prop_info }.
```

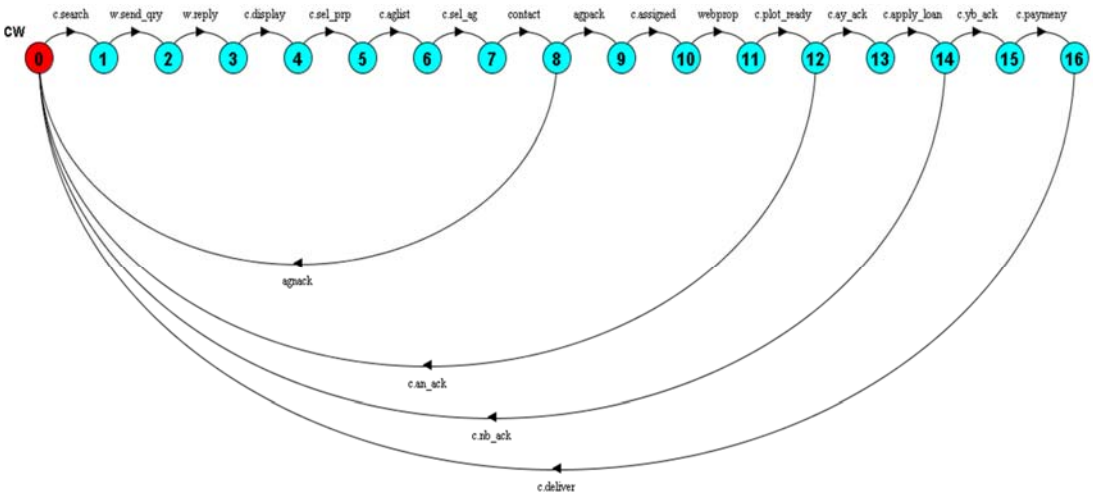



Figure 4.7 LTSA Representation of CW ( CLIENT and WEBSITE ) process

**CLIENT and AGENT**

```
CLIENT = ( c.search-> c.display -> c.sel_prp ->
c.aglist -> c.sel_ag -> contact -> (agpack ->
c.assigned -> webprop -> al.qry -> (lq.pack ->
c.plot_ready -> (c.ay_ack->c.apply_loan ->(c.yb_ack ->
c.paymeny ->c.deliver -> CLIENT |c.nb_ack-> CLIENT) |
c.an_ack -> CLIENT)|lq.nack -> CLIENT)| agnack ->
     CLIENT)).

AGENT = ( a.contact -> ( a.y_ack -> assigned ->
a.prop_info -> ask_qry -> (l.ay_ack -> a.plot_ready ->(
c.y_ack-> loan -> (apploan -> pay -> deli -> AGENT |
rejloan ->AGENT) | c.n_ack -> AGENT)| l.an_ack -
>AGENT) | a.n_ack -> AGENT)).

||CA = ( CLIENT || AGENT ) /
                        { agpack/a.y_ack, agnack/a.n_ack
             ,c.assigned/assigned, webprop/a.prop_info ,
     al.qry/ask_qry , lq.pack/l.ay_ack, lq.nack/l.an_ack
          ,c.plot_ready/a.plot_ready, c.ay_ack/c.y_ack ,
           c.an_ack/c.n_ack, c.plot_ready/a.plot_ready ,
 c.apply_loan/loan, c.yb_ack/apploan, c.nb_ack/rejloan ,
                        c.payment/pay , c.deliver/deli}.
```

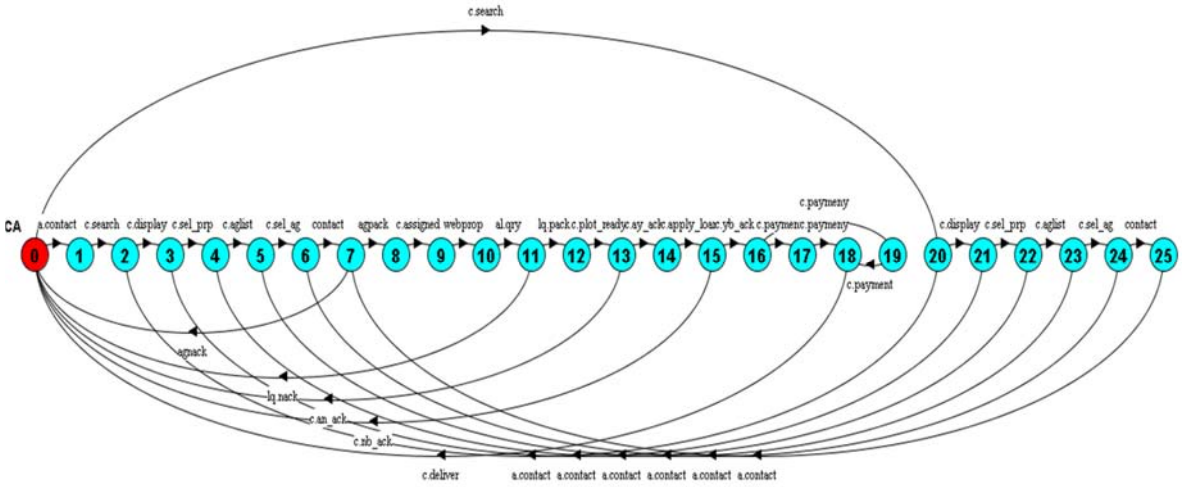



Figure 4.8 LTSA Representation of CA ( CLIENT and AGENT ) process

**WEBSITE and PLOTDATA**

```
WEBSITE = ( w.search -> w.send_qry -> w.reply ->
w.display ->   w.sel_prop -> w.aglist -> w.sel_ag ->
w.contact_ag ->     (w.ay_ack -> w.assigned ->
w.prop_info -> la.qry -> WEBSITE   | w.an_ack ->
WEBSITE)).

PLOTDATA = ( p.send_qry -> p.reply -> PLOTDATA).

||WP = ( WEBSITE || PLOTDATA ) / {w.send_qry/p.send_qry
     ,w.reply/p.reply }.
```

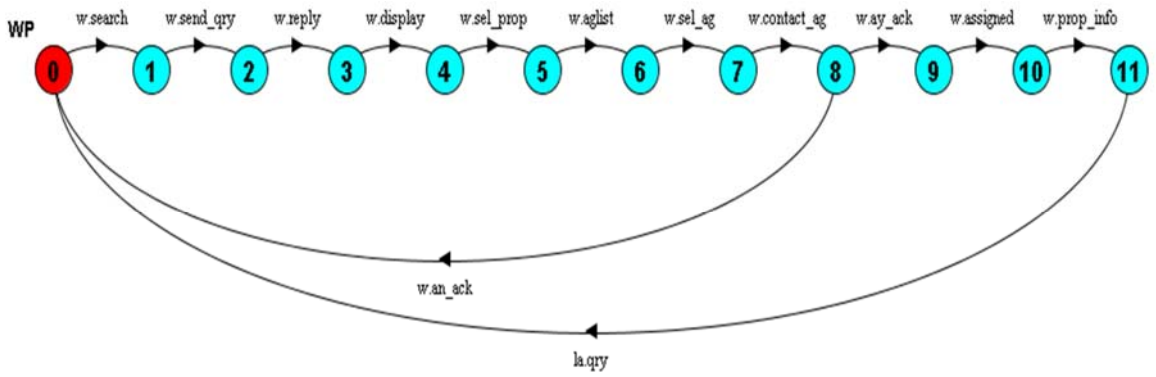



Figure 4.9 LTSA Representation of WP ( WEBSITE and PLOTDATA ) process

**CLIENT and LANDLORD**

```
CLIENT  =  (  c.search->  c.display  ->  c.sel_prp  ->
c.aglist ->    c.sel_ag  ->  contact  ->  (agpack  ->
c.assigned -> webprop -> al.qry    ->    (lq.pack    ->
c.plot_ready -> (c.ay_ack ->  c.apply_loan -> (c.yb_ack
-> c.paymeny ->c.deliver -> CLIENT |c.nb_ack   ->CLIENT)
| c.an_ack -> CLIENT)|lq.nack -> CLIENT)|    agnack   -
>CLIENT)).


LANDLORD  =  (  l.ask_qry  ->  (l.y_ack  ->  ready_plot  ->
(rd_pack ->    applyloan -> (b.confim -> rcv_payment ->
deliver ->     LANDLORD|b.reject ->LANDLORD)| rd_nack -
> LANDLORD)| l.n_ack     -> LANDLORD)).
```

```
||CL = ( CLIENT ||LANDLORD ) /
  {al.qry/l.ask_qry , lq.pack/l.y_ack, lq.nack/l.n_ack ,
            c.plot_ready/ready_plot , c.ay_ack/rd_pack ,
               c.an_ack/rd_nack , c.apply_loan/applyloa,
                 c.yb_ack/b.confirm ,c.nb_ack/b.reject ,
                                  c.payment/rcv_payment ,
                                      c.deliver/deliver}.
```

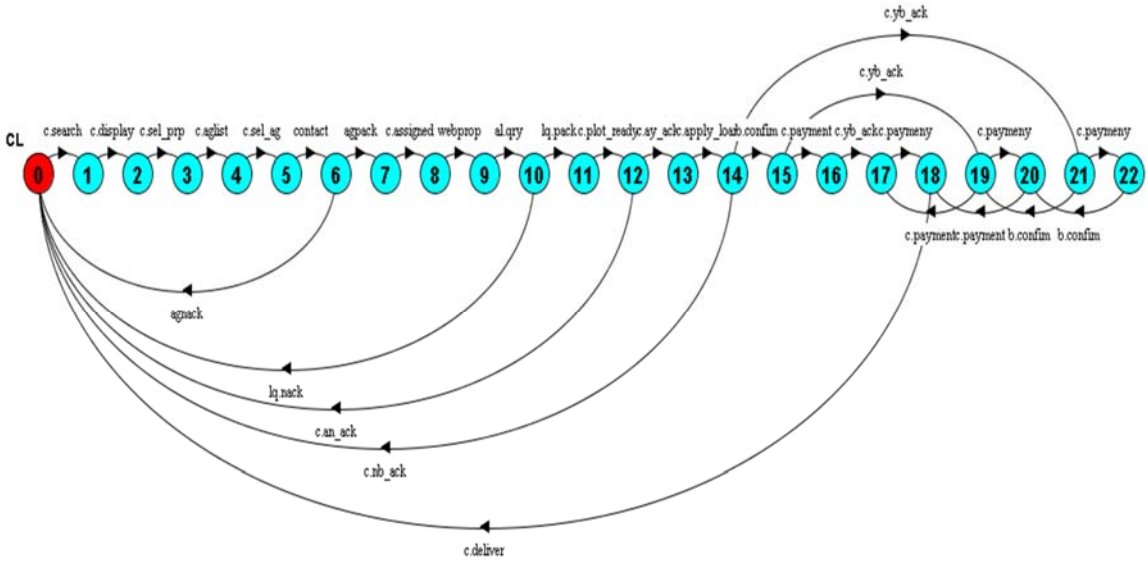



Figure 4.10 LTSA Representation of CL ( CLIENT and LANDLORD ) process

**CLIENT and BANK**

```
CLIENT = ( c.search-> c.display -> c.sel_prp ->
c.aglist -> c.sel_ag -> contact -> (agpack ->
c.assigned -> webprop -> al.qry -> (lq.pack ->
c.plot_ready -> (c.ay_ack -> c.apply_loan -> (c.yb_ack
-> c.paymeny ->c.deliver -> CLIENT |c.nb_ack ->CLIENT)
| c.an_ack -> CLIENT)|lq.nack -> CLIENT)| agnack -
>CLIENT)).
```

```
BANK = (rcv_loan.req -> (b.pack ->pays -> deliv ->
BANK|b.nack    ->BANK)).


||CB = ( CLIENT || BANK ) /
          {c.apply_loan/rcv_loan.req , c.yb_ack/b.pack ,
  c.nb_ack/b.nack , c.payment/pays  , c.deliver/deliv }.
```

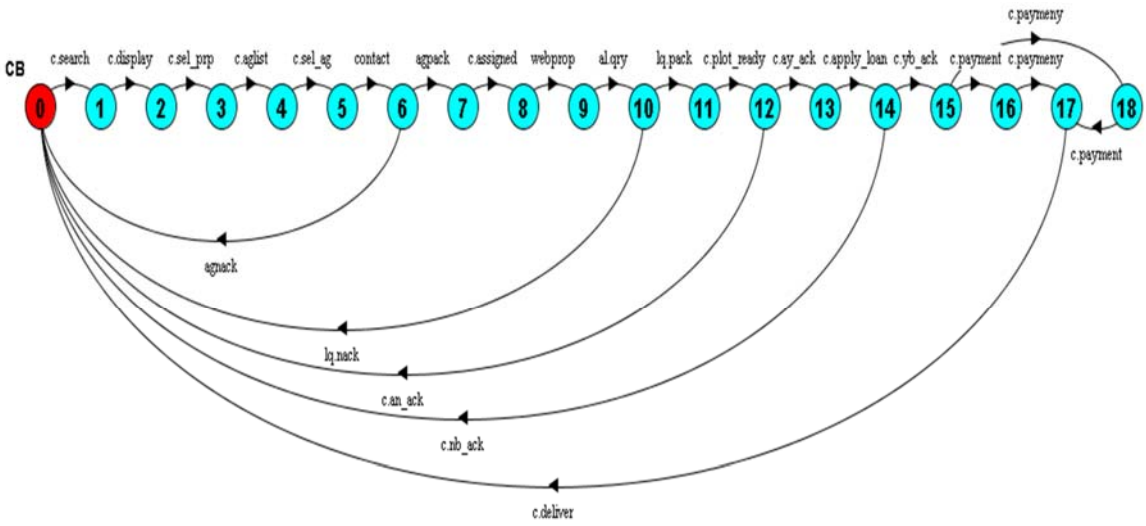



Figure 4.11 LTSA Representation of CB ( CLIENT and BANK ) process

**WEBSITE and AGENT**

```
WEBSITE = ( w.search -> w.send_qry -> w.reply ->
w.display ->  w.sel_prop -> w.aglist -> w.sel_ag ->
w.contact_ag ->    (w.ay_ack    ->    w.assigned    ->
w.prop_info -> WEBSITE | w.an_ack  -> WEBSITE)).
```

```
AGENT = ( a.contact -> ( a.y_ack ->assign ->
a.prop_info -> ask_qry -> (l.y_ack -> a.plot_ready -
>( c.y_ack-> AGENT | c.n_ack -> AGENT)| l.n_ack -
>AGENT) | a.n_ack -> AGENT)).


||WAG = (WEBSITE || AGENT )/
            {w.contact_ag/a.contact , w.ay_ack/a.y_ack ,
          w.assigned/assign , w.prop_info/a.prop_info ,
                                    w.an_ack/a.n_ack }.
```

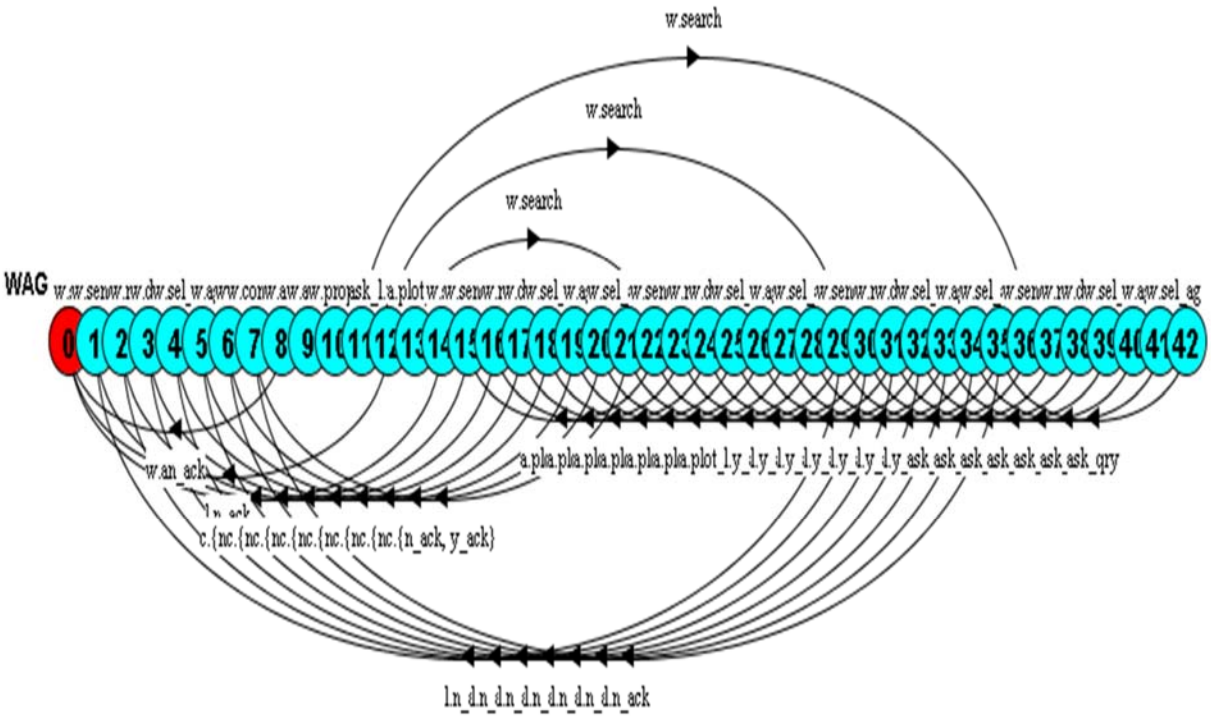



Figure 4.12 LTSA Representation of WAG (WEBSITE and AGENT) process

## 4.4 Modeling Triple Processes in FSP

### CLIENT, WEBSITE and AGENT

```
CLIENT = ( c.search-> c.display -> c.sel_prp -> 
c.aglist -> c.sel_ag -> contact -> (agpack -> 
c.assigned -> webprop -> al.qry -> (lq.pack -> 
c.plot_ready -> (c.ay_ack -> c.apply_loan -> (c.yb_ack 
-> c.paymeny ->c.deliver -> CLIENT |c.nb_ack -
>CLIENT) | c.an_ack -> CLIENT)|lq.nack -> CLIENT)|
     agnack ->CLIENT)).

WEBSITE = ( w.search -> w.send_qry -> w.reply -> 
w.display -> w.sel_prop -> w.aglist -> w.sel_ag -> 
w.contact_ag -> (w.ay_ack -> w.assigned -> 
w.prop_info -> la.qry -> WEBSITE | w.an_ack -> 
WEBSITE)).

AGENT = ( a.contact -> ( a.y_ack -> assigned -> 
a.prop_info -> ask_qry -> (l.ay_ack -> a.plot_ready ->( 
c.y_ack-> loan -> (apploan -> pay -> deli -> AGENT | 
rejloan -> AGENT)
     | c.n_ack -> AGENT)| l.an_ack ->AGENT) | a.n_ack -
>AGENT)).

||CWA = ( CLIENT || WEBSITE ||AGENT) /
             { c.search/w.search , c.display/w.display ,
              c.sel_prp/w.sel_prop , c.aglist/w.aglist ,
              c.sel_ag/w.sel_ag , contact/w.contact_ag ,
              w.contact_ag/a.contact , agpack/w.ay_ack ,
                      agnack/w.an_ack , agpack/a.y_ack ,
               agnack/a.n_ack , c.assigned/w.assigned ,
              c.assigned/assigned , webprop/w.prop_info
             ,w.prop_info/a.prop_info , al.qry/ask_qry ,
                   lq.pack/l.ay_ack , lq.nack/l.an_ack ,
          c.plot_ready/a.plot_ready , c.ay_ack/c.y_ack ,
                  c.an_ack/c.n_ack , al.qry/l.ask_qry ,
                     lq.pack/l.y_ack, lq.nack/l.n_ack ,
      c.payment/rcv_payment, c.plot_ready/a.plot_ready ,
            a.plot_ready/ready_plot , c.ay_ack/rd_pack ,
```

c.an_ack/rd_nack , c.apply_loan/applyloan ,
c.yb_ack/b.confirm ,c.nb_ack/b.reject ,
c.apply_loan/loan , c.yb_ack/apploan,
c.nb_ack/rejloan , c.payment/pay , c.deliver/deliver ,
c.deliver/deli }.

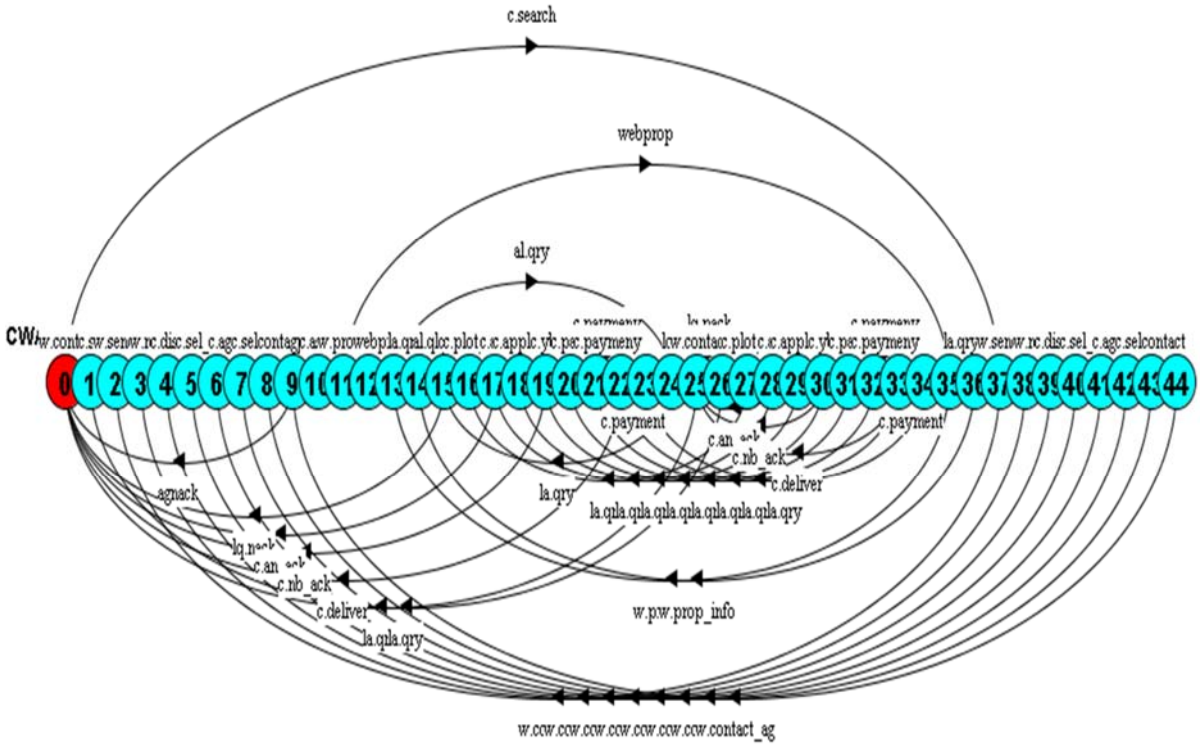



Figure 4.13 LTSA Representation of CWA (CLIENT ,WEBSITE and AGENT) process

## 4.5 Modelling four Processes together in FSP

### CLIENT, WEBSITE, AGENT and LANDLORD

```
CLIENT = ( c.search-> c.display -> c.sel_prp ->
c.aglist  c.sel_ag -> contact -> (agpack -> c.assigned
-> webprop ->  al.qry -> (lq.pack -> c.plot_ready ->
(c.ay_ack ->   c.apply_loan ->(c.yb_ack -> c.paymeny -
>c.deliver -> CLIENT     |c.nb_ack ->CLIENT) | c.an_ack
-> CLIENT)|lq.nack ->    CLIENT)|  agnack ->CLIENT)).

WEBSITE  =  (  w.search  ->  w.send_qry  ->  w.reply  ->
w.display ->   w.sel_prop  ->  w.aglist  ->  w.sel_ag  ->
w.contact_ag ->     (w.ay_ack    ->    w.assigned    ->
w.prop_info -> la.qry -> WEBSITE   |     w.an_ack     ->
WEBSITE)).


AGENT  =  (  a.contact  ->  (  a.y_ack  ->  assigned  ->
a.prop_info -> ask_qry -> (l.ay_ack -> a.plot_ready ->(
c.y_ack-> loan ->   (apploan -> pay -> deli -> AGENT |
rejloan -> AGENT) | c.n_ack   ->   AGENT)|   l.an_ack   -
>AGENT) | a.n_ack ->AGENT)).


LANDLORD  =  (  l.ask_qry  ->  (l.y_ack  ->  ready_plot  ->
(rd_pack ->    applyloan -> (b.confim -> rcv_payment ->
deliver ->     LANDLORD|b.reject ->LANDLORD)| rd_nack -
> LANDLORD) |  l.n_ack   -> LANDLORD)).


||CWAL = ( CLIENT || WEBSITE ||AGENT || LANDLORD ) /

            { c.search/w.search , c.display/w.display ,
             c.sel_prp/w.sel_prop , c.aglist/w.aglist ,
             c.sel_ag/w.sel_ag , contact/w.contact_ag ,
             w.contact_ag/a.contact , agpack/w.ay_ack ,
    agnack/w.an_ack , agpack/a.y_ack , agnack/a.n_ack ,
          c.assigned/w.assigned , c.assigned/assigned ,
        webprop/w.prop_info ,w.prop_info/a.prop_info ,
                    al.qry/ask_qry , lq.pack/l.ay_ack ,
        lq.nack/l.an_ack , c.plot_ready/a.plot_ready ,
```

```
c.ay_ack/c.y_ack , c.an_ack/c.n_ack , al.qry/l.ask_qry
                  , lq.pack/l.y_ack, lq.nack/l.n_ack ,
    c.payment/rcv_payment, c.plot_ready/a.plot_ready ,
          a.plot_ready/ready_plot , c.ay_ack/rd_pack ,
                                    c.an_ack/rd_nack ,
        c.apply_loan/applyloan , c.yb_ack/b.confirm ,
              c.nb_ack/b.reject , c.apply_loan/loan ,
                 c.yb_ack/apploan, c.nb_ack/rejloan ,
 c.payment/pay , c.deliver/deliver , c.deliver/deli }.
```

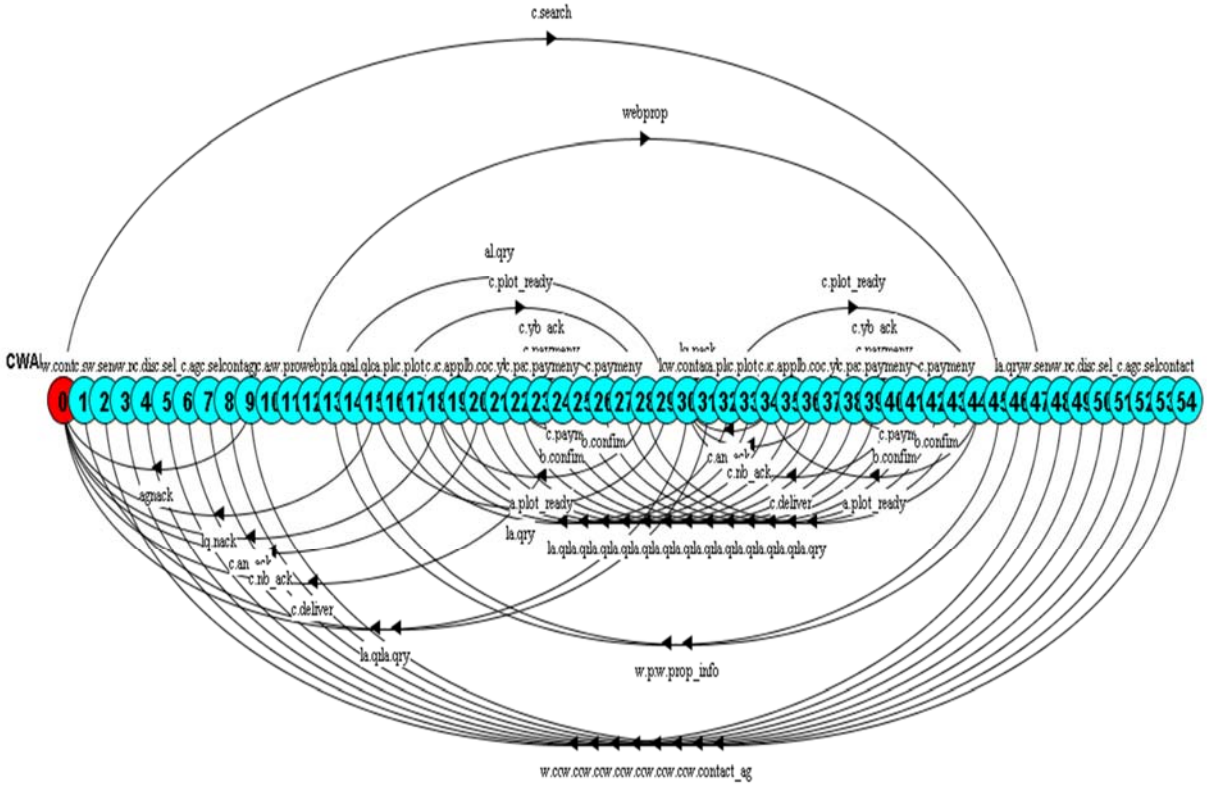



Figure 4.14 LTSA Representation of CWAL (CLIENT ,WEBSITE , AGENT and LANDLORD) process

## 4.6 Modelling All Processes in FSP

```
CLIENT = ( c.search-> c.display -> c.sel_prp ->
c.aglist ->    c.sel_ag -> contact -> (agpack ->
c.assigned -> webprop -> al.qry -> (lq.pack ->
c.plot_ready -> (c.ay_ack ->  c.apply_loan ->(c.yb_ack
-> c.paymeny ->c.deliver -> CLIENT |c.nb_ack ->CLIENT)
| c.an_ack -> CLIENT)|lq.nack ->   CLIENT)| agnack -
>CLIENT)).


WEBSITE = ( w.search -> w.send_qry -> w.reply ->
w.display ->   w.sel_prop -> w.aglist -> w.sel_ag ->
w.contact_ag ->     (w.ay_ack -> w.assigned ->
w.prop_info -> la.qry -> WEBSITE   | w.an_ack ->
WEBSITE)).

PLOTDATA = ( p.send_qry -> p.reply -> PLOTDATA).
AGENT = ( a.contact -> ( a.y_ack -> assigned ->
a.prop_info -> ask_qry -> (l.ay_ack -> a.plot_ready ->(
c.y_ack-> loan ->   (apploan -> pay -> deli -> AGENT |
rejloan -> AGENT) | c.n_ack -> AGENT)| l.an_ack -
>AGENT) | a.n_ack -> AGENT)).


LANDLORD = ( l.ask_qry -> (l.y_ack -> ready_plot ->
(rd_pack ->    applyloan -> (b.confim -> rcv_payment ->
deliver ->     LANDLORD|b.reject ->LANDLORD)| rd_nack -
> LANDLORD)|   l.n_ack -> LANDLORD)).

BANK = (rcv_loan.req -> (b.pack ->pays -> deliv ->
BANK|b.nack->  BANK)).


||CWALBP = ( CLIENT || WEBSITE || PLOTDATA ||AGENT ||
LANDLORD || BANK ) /
             { c.search/w.search , c.display/w.display ,
              c.sel_prp/w.sel_prop , c.aglist/w.aglist ,
              c.sel_ag/w.sel_ag , contact/w.contact_ag ,
              w.contact_ag/a.contact , agpack/w.ay_ack ,
                      agnack/w.an_ack , agpack/a.y_ack ,
                agnack/a.n_ack , c.assigned/w.assigned ,
```

```
c.assigned/assigned , webprop/w.prop_info
,w.prop_info/a.prop_info , al.qry/ask_qry ,
lq.pack/l.ay_ack , lq.nack/l.an_ack ,
c.plot_ready/a.plot_ready , c.ay_ack/c.y_ack ,
c.an_ack/c.n_ack ,al.qry/l.ask_qry ,
lq.pack/l.y_ack, lq.nack/l.n_ack ,
c.payment/rcv_payment, c.plot_ready/a.plot_ready ,
a.plot_ready/ready_plot , c.ay_ack/rd_pack ,
c.an_ack/rd_nack , c.apply_loan/applyloan ,
c.yb_ack/b.confirm ,c.nb_ack/b.reject ,
c.apply_loan/loan , c.yb_ack/apploan,
c.nb_ack/rejloan , c.payment/pay ,
c.deliver/deliver , c.deliver/deli ,
c.apply_loan/rcv_loan.req ,
c.yb_ack/b.pack , c.nb_ack/b.nack ,
c.payment/pays , c.deliver/deliv ,
w.send_qry/p.send_qry ,w.reply/p.reply }.
```

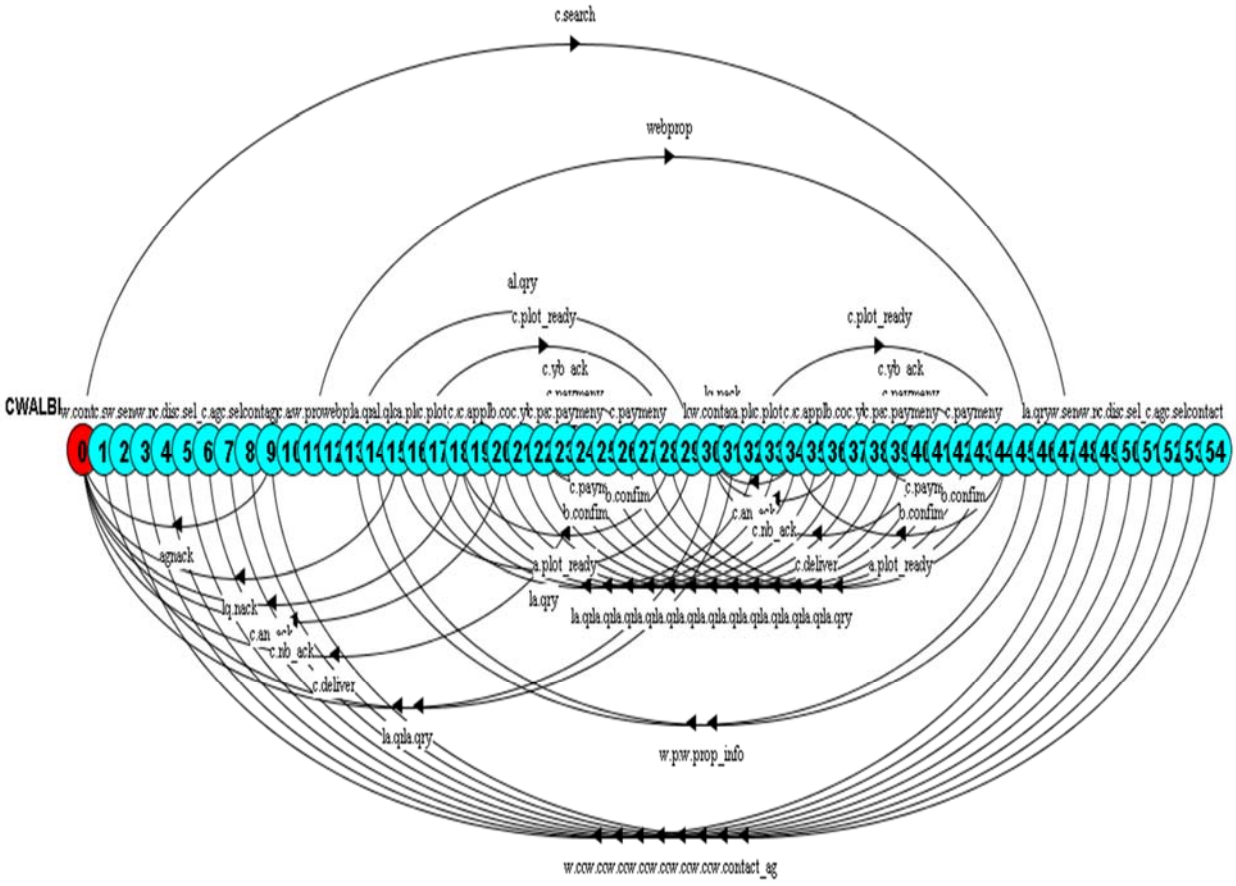



Figure 4.15 LTSA Representation of Real Estate Web Service.

# Chapter 5

## Property process for verification

A property is a quality of a program that is valid for each conceivable execution of that program. Normally there are two kinds of properties such as safety and liveness. A safety property states that nothing terrible occurs during execution. A liveness property states that something good occurs in execution such as execution of a program in a good state. If a safety property is composed in parallel with a process and no trace violation is generated after the composition we can share that the safety property verifies that process. If any trace violation occurs we can say that the safety property could not verify the process.[ report book ]

### 5.1 Verifying Client process

```
property SAFE_B = ( c.nb_ack -> cancel_order ->
                    SAFE_B).
```

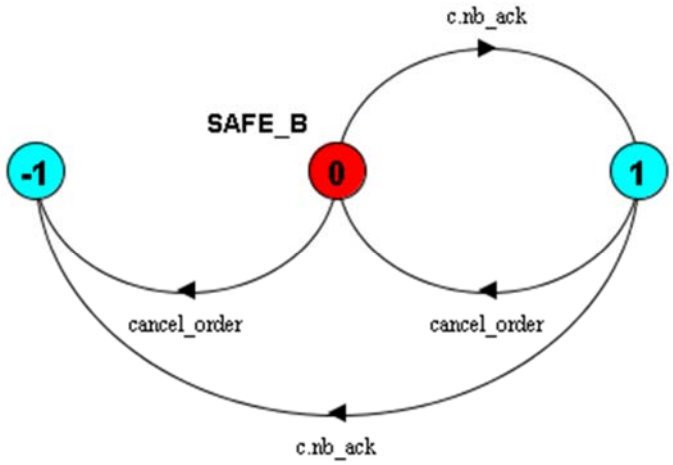



Figure 5.1: LTSA representation of safety property SAFE_B

The property process SAFE_B ensures that when a negative acknowledgement is received from client it actually synchronizes with the CLIENT process. The property is described in two actions `c.nb_ack` which is a negative acknowledgement from CLIENT process , which leads to `cancel_order`.

```
property SAFE_A = ( c.an_ack ->cancel_rcv_qt -> SAFE_A).
```

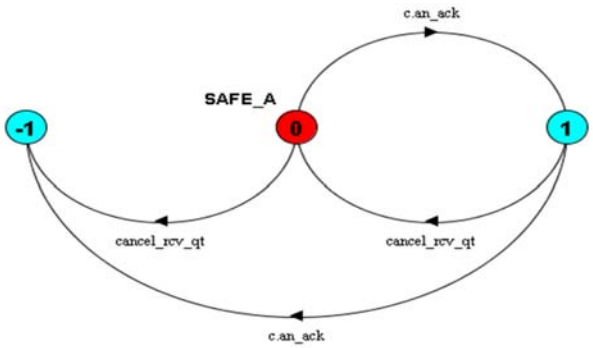



Figure 5.2 LTSA representation of safety property SAFE_A

As like as SAFE_B property process , this property also has two actions , ensuring that when a negative acknowledgement `c.an_ack` is thrown from CLIENT and `cancel_rcv_qt` is the next action.

When the property processes SAFE_B and SAFE_A is composed with CLIENT in SAFE_CLIENT, the actions of our property processes found in a sequential manner and does not show any trace violation in the LTS, we can say that there are no problem and they have been synchronized with each other successfully and satisfied the condition of our property process, otherwise not.

```
||SAFE_CLIENT = ( CLIENT || SAFE_B || SAFE_A ).
```

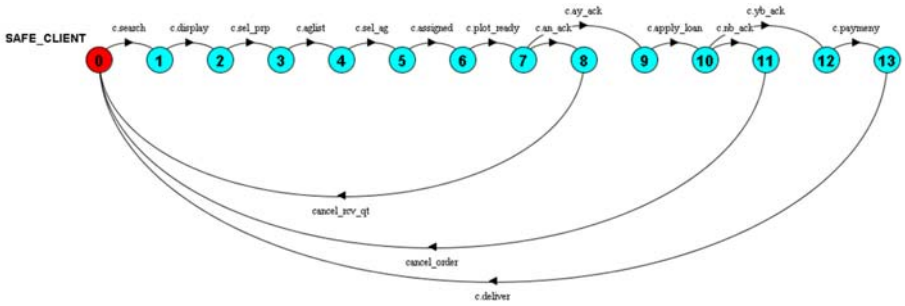
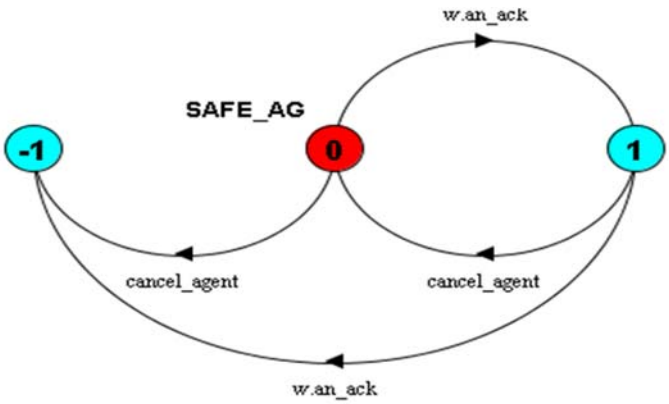



Figure 5.3 LTSA representation of SAFE_CLIENT process

## 5.2 Verifying Website process

```
property SAFE_AG = (w.an_ack -> cancel_agent ->
```




```
SAFE_AG).
```

Figure 5.4 LTSA representation of safety property SAFE_AG

The property process SAFE_AG is described by two actions that shows when `w.an_ack` which is a negative acknowledgement stating that no agent is available something like that , is thrown from website the next step would be to cancel the agent stating `cancel_agent`. When the property process SAFE_AG is composed with WEBSITE in SAFE_WEB, the actions of our property processes found in a sequential manner and does not show any trace violation in the LTS, we can say that there are no problem and they have been synchronized with each other successfully and satisfied the condition of our property process, otherwise not.

```
||SAFE_WEB = ( WEBSITE || SAFE_AG ).
```

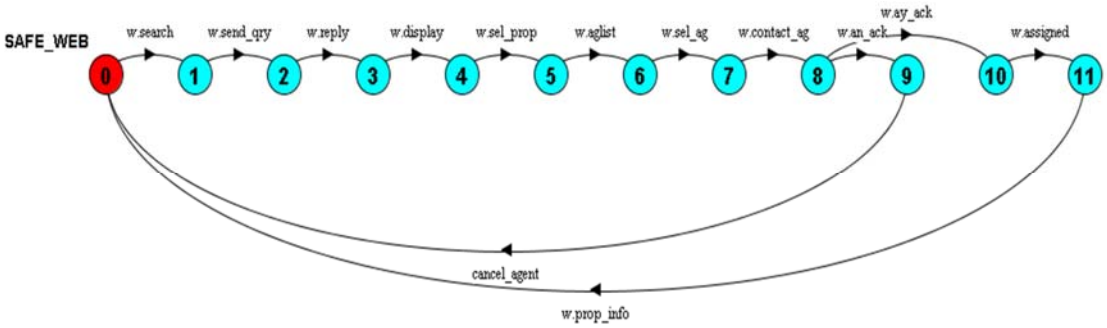



Figure 5.5 LTSA representation of SAFE_WEB Process

## 5.3 Verifying AGENT process

```
property SAFE_C = (c.n_ack -> cancel_order -> SAFE_C).
```

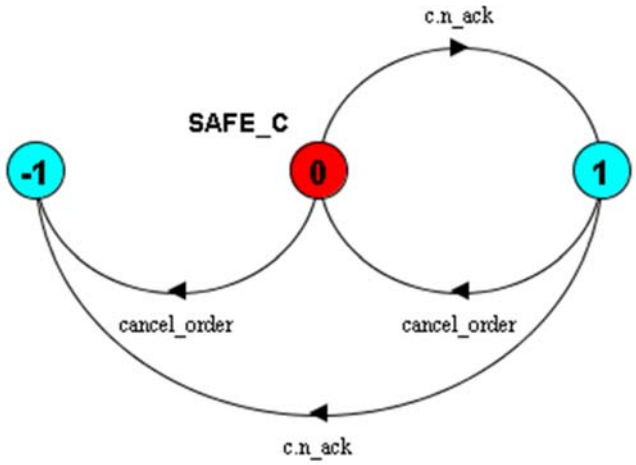



Figure 5.6 LTSA representation of safety property SAFE_C

```
property SAFE_LA = (l.n_ack -> not_available ->
SAFE_LA).
```

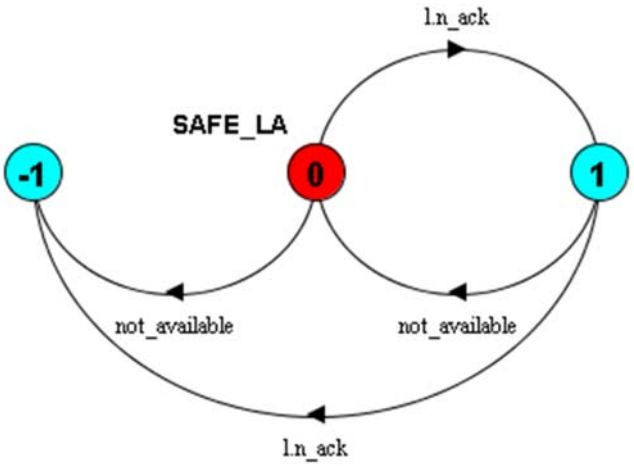



Figure 5.7 LTSA representation of safety property SAFE_LA

```
property SAFE_A2 = (a.n_ack -> cancel_ag -> SAFE_A2).
```

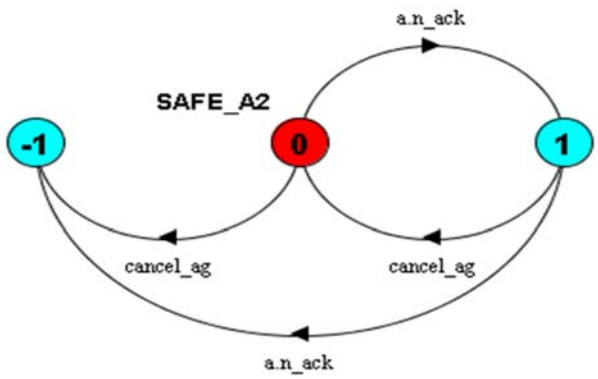



Figure 5.8 LTSA representation of safety property SAFE_A2

When the property processes SAFE_C, SAFE_LA and SAFE_A2 is composed with AGENT in SAFE_AGENT, the actions of our property processes found in a sequential manner and does not show any trace violation in the LTS, we can say that there are no problem and they have been synchronized with each other successfully and satisfied the condition of our property process, otherwise not.

```
||SAFE_AGENT = ( AGENT ||SAFE_C || SAFE_LA || SAFE_A2).
```

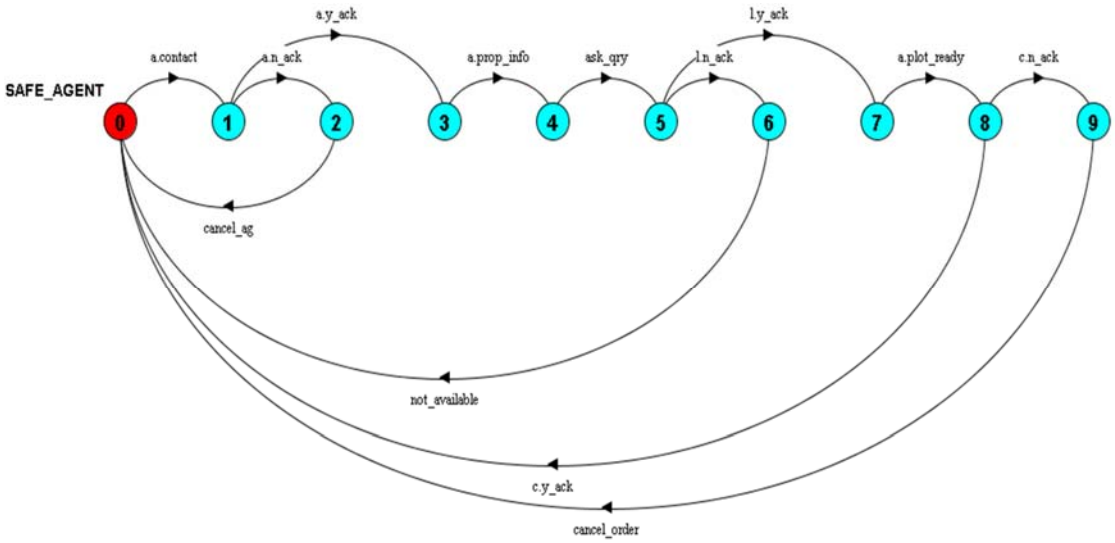



Figure 5.9 LTSA representation of SAFE_AGENT Process

## 5.4 Verifying LANDLORD process

```
property SAFE_LAN = (l.n_ack -> not_available ->
SAFE_LAN).
```

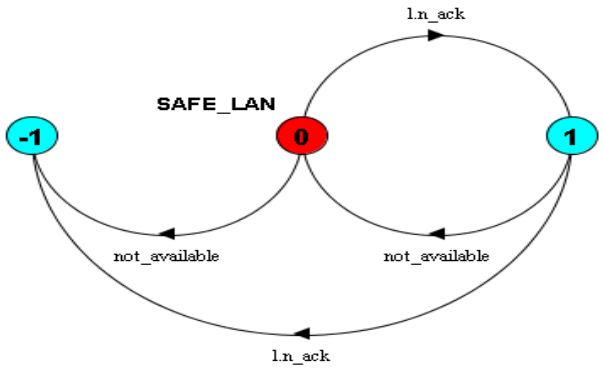



Figure 5.10 LTSA representation of safety property SAFE_LAN

When the property processes SAFE_LAN is composed with LANDLORD in SAFE_LAND, the actions of our property processes found in a sequential manner and does not show any trace violation in the LTS, we can say that there are no problem and they have been synchronized with each other successfully and satisfied the condition of our property process, otherwise not.

```
||SAFE_LAND = ( LANDLORD || SAFE_LAN ).
```

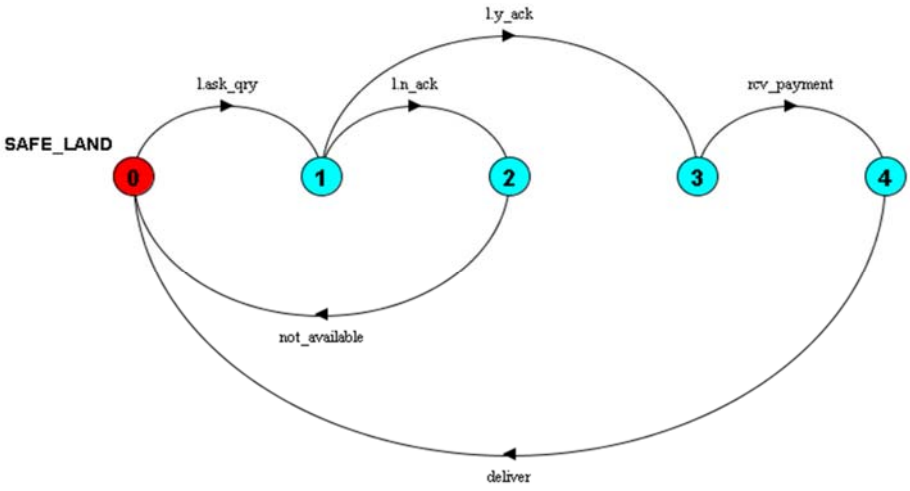



Figure 5.11 LTSA representation of SAFE_LAND process

## 5.5 Verifying BANK process

```
property SAFE_BA = (b.nack -> cancel_order -> SAFE_BA).
```

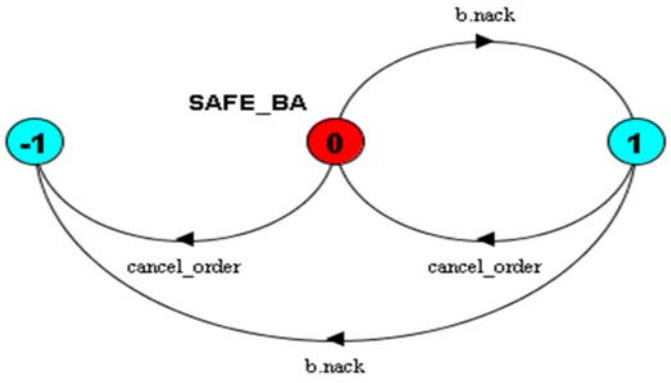



Figure 5.12 LTSA representation of safety property SAFE_BA

When the property processes SAFE_BA is composed with BANK in SAFE_Bank, the actions of our property processes found in a sequential manner and does not show any trace violation in the LTS, we can say that there are no problem and they have been synchronized with each other successfully and satisfied the condition of our property process, otherwise not.

```
||SAFE_Bank = ( BANK|| SAFE_BA).
```

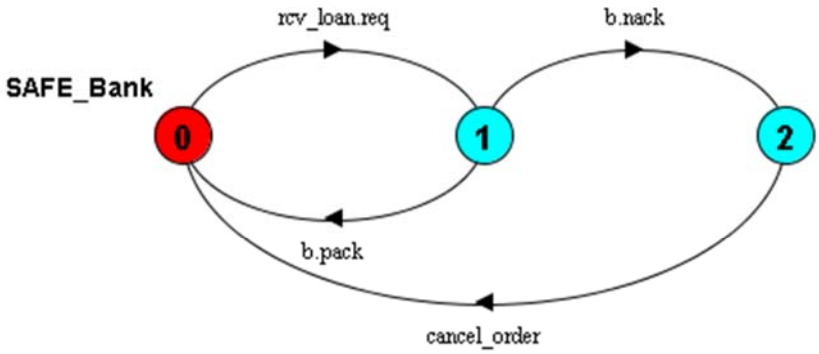



Figure 5.13 LTSA representation of SAFE_Bank process

# Chapter 6

# Conclusion

## 6.1 Summary

We have analyzed about Web Service and its Composition. We have modeled the Real Estate Web Service by composing several web services to create a composite web service in a choreographic manner. From our proposed model firstly we analyzed the connections between the processes. We have used FSP notations to model and verify our desired system. We modeled every process individually and two or more processes together with FSP notations. In order to verification we have verified safety property. We have verified the processes with negative property.

## 6.2 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 among the processes. We planned to include a recommendation system with this web service composition system. We will work on how we can get the best individual process comparing to other processes from different places with recommendation system.

# Appendix A

## A.1 CLIENT Web Service

```
CLIENT = ( c.search->c.display->c.sel_prp->c.aglist->
     c.sel_ag -> c.assigned -> c.plot_ready ->
(c.ay_ack ->   c.apply_loan ->(c.yb_ack -> c.paymeny -
>c.deliver -> CLIENT     |c.nb_ack ->CLIENT) | c.an_ack
-> CLIENT)).
```

## A.2 WEBSITE Web Service

```
WEBSITE = ( w.search -> w.send_qry -> w.reply ->
w.display ->   w.sel_prop -> w.aglist -> w.sel_ag ->
w.contact_ag ->     (w.ay_ack -> w.assigned ->
w.prop_info -> WEBSITE | w.an_ack -> WEBSITE)).
```

## A.3 PLOTDATA Web Service

```
PLOTDATA = ( p.send_qry -> p.reply -> PLOTDATA).
```

## A.4 Agent Web Service

```
AGENT =( a.contact ->( a.y_ack->a.prop_info -> ask_qry
->   (l.y_ack -> a.plot_ready ->( c.y_ack-> AGENT |
c.n_ack ->     AGENT)| l.n_ack ->AGENT) | a.n_ack ->
AGENT)).
```

## A.5 LANLORD Web Service

```
LANDLORD = ( l.ask_qry -> (l.y_ack -> rcv_payment ->
deliver ->          LANDLORD | l.n_ack -> LANDLORD))
```

## A.6 BANK Web Service

```
BANK = (rcv_loan.req -> (b.pack-> BANK|b.nack ->BANK)).
```

## A.7 CLIENT and WEBSITE

```
CLIENT = ( c.search-> c.display -> c.sel_prp ->
c.aglist ->    c.sel_ag   ->   contact   ->   (agpack   -
>c.assigned -> webprop ->     al.qry   ->   (lq.pack   ->
c.plot_ready -> (c.ay_ack ->  c.apply_loan -> (c.yb_ack
-> c.paymeny ->c.deliver ->   CLIENT    |c.nb_ack      -
>CLIENT) | c.an_ack -> CLIENT)|lq.nack  ->      CLIENT)|
     agnack ->CLIENT)).


WEBSITE  =  (  w.search  ->  w.send_qry  ->  w.reply  ->
w.display ->   w.sel_prop  ->  w.aglist  ->  w.sel_ag  ->
w.contact_ag ->     (w.ay_ack    ->    w.assigned    ->
w.prop_info -> la.qry -> WEBSITE   |     w.an_ack     ->
WEBSITE)).


||CW = ( CLIENT || WEBSITE ) /
            { c.search/w.search , c.display/w.display ,
             c.sel_prp/w.sel_prop , c.aglist/w.aglist ,
             c.sel_ag/w.sel_ag , contact/w.contact_ag ,
                    agpack/w.ay_ack , agnack/w.an_ack ,
         c.assigned/w.assigned , webprop/w.prop_info }.
```

## A.8 CLIENT and AGENT

```
CLIENT = ( c.search-> c.display -> c.sel_prp ->
c.aglist ->    c.sel_ag -> contact -> (agpack ->
c.assigned -> webprop -> al.qry -> (lq.pack ->
c.plot_ready -> (c.ay_ack->   c.apply_loan   -
>(c.yb_ack -> c.paymeny ->c.deliver ->  CLIENT
|c.nb_ack->    CLIENT) | c.an_ack -> CLIENT)|lq.nack->
     CLIENT)| agnack ->  CLIENT)).


AGENT = ( a.contact -> ( a.y_ack -> assigned ->
a.prop_info -> ask_qry -> (l.ay_ack -> a.plot_ready ->(
c.y_ack-> loan ->   (apploan -> pay -> deli -> AGENT |
rejloan ->AGENT) |  c.n_ack   -> AGENT)| l.an_ack -
>AGENT) | a.n_ack ->AGENT)).
```

```
||CA = ( CLIENT || AGENT ) /
                        { agpack/a.y_ack, agnack/a.n_ack
             ,c.assigned/assigned, webprop/a.prop_info ,
     al.qry/ask_qry , lq.pack/l.ay_ack, lq.nack/l.an_ack
          ,c.plot_ready/a.plot_ready, c.ay_ack/c.y_ack ,
           c.an_ack/c.n_ack, c.plot_ready/a.plot_ready ,
 c.apply_loan/loan, c.yb_ack/apploan, c.nb_ack/rejloan ,
                        c.payment/pay , c.deliver/deli}.
```

## A.9 WEBSITE and PLOTDATA

```
WEBSITE = ( w.search -> w.send_qry -> w.reply ->
w.display ->   w.sel_prop -> w.aglist -> w.sel_ag ->
w.contact_ag ->     (w.ay_ack -> w.assigned ->
w.prop_info -> la.qry -> WEBSITE   | w.an_ack ->
WEBSITE)).

PLOTDATA = ( p.send_qry -> p.reply -> PLOTDATA).

||WP = ( WEBSITE || PLOTDATA ) / {w.send_qry/p.send_qry
     ,w.reply/p.reply }.
```

## A.10 CLIENT and LANDLORD

```
CLIENT = ( c.search-> c.display -> c.sel_prp ->
c.aglist ->    c.sel_ag -> contact -> (agpack ->
c.assigned -> webprop -> al.qry -> (lq.pack ->
c.plot_ready -> (c.ay_ack ->  c.apply_loan -> (c.yb_ack
-> c.paymeny ->c.deliver ->   CLIENT    |c.nb_ack -
>CLIENT) | c.an_ack -> CLIENT)|lq.nack  -> CLIENT)|
     agnack ->CLIENT)).


LANDLORD = (  l.ask_qry  -> (l.y_ack  -> ready_plot  ->
(rd_pack ->    applyloan -> (b.confim -> rcv_payment ->
deliver ->     LANDLORD|b.reject ->LANDLORD)| rd_nack -
> LANDLORD)|   l.n_ack   -> LANDLORD)).

||CL = ( CLIENT ||LANDLORD ) /
  {al.qry/l.ask_qry , lq.pack/l.y_ack, lq.nack/l.n_ack ,
            c.plot_ready/ready_plot , c.ay_ack/rd_pack ,
```

```
                c.an_ack/rd_nack , c.apply_loan/applyloa,
                  c.yb_ack/b.confirm ,c.nb_ack/b.reject ,
              c.payment/rcv_payment , c.deliver/deliver}.
```

## A.11 CLIENT and BANK

```
CLIENT = ( c.search-> c.display -> c.sel_prp ->
c.aglist ->    c.sel_ag -> contact -> (agpack ->
c.assigned -> webprop -> al.qry -> (lq.pack ->
c.plot_ready -> (c.ay_ack ->  c.apply_loan -> (c.yb_ack
-> c.paymeny ->c.deliver ->   CLIENT    |c.nb_ack -
>CLIENT) | c.an_ack -> CLIENT)|lq.nack  -> CLIENT)|
     agnack ->CLIENT)).

BANK  =  (rcv_loan.req  ->  (b.pack  ->pays  ->  deliv  ->
BANK|b.nack    ->BANK)).


||CB = ( CLIENT || BANK ) /
          {c.apply_loan/rcv_loan.req , c.yb_ack/b.pack ,
  c.nb_ack/b.nack , c.payment/pays  , c.deliver/deliv }.
```

## A.12 WEBSITE and AGENT

```
WEBSITE = ( w.search -> w.send_qry -> w.reply ->
w.display ->   w.sel_prop -> w.aglist -> w.sel_ag ->
w.contact_ag ->     (w.ay_ack -> w.assigned ->
w.prop_info -> WEBSITE | w.an_ack  -> WEBSITE)).


AGENT = ( a.contact -> ( a.y_ack ->assign ->
a.prop_info -> ask_qry   -> (l.y_ack -> a.plot_ready -
>( c.y_ack-> AGENT |     c.n_ack ->AGENT)| l.n_ack -
>AGENT) | a.n_ack ->     AGENT)).

||WAG = (WEBSITE || AGENT )/
            {w.contact_ag/a.contact , w.ay_ack/a.y_ack ,
           w.assigned/assign , w.prop_info/a.prop_info ,
                                    w.an_ack/a.n_ack }.
```

## A.13 CLIENT , WEBSITE and AGENT

```
CLIENT = ( c.search-> c.display -> c.sel_prp -> c.aglist -> c.sel_ag -> contact -> (agpack -> c.assigned -> webprop -> al.qry -> (lq.pack -> c.plot_ready -> (c.ay_ack -> c.apply_loan -> (c.yb_ack -> c.paymeny ->c.deliver -> CLIENT |c.nb_ack ->CLIENT) | c.an_ack -> CLIENT)|lq.nack -> CLIENT)|
     agnack ->CLIENT)).


WEBSITE = ( w.search -> w.send_qry -> w.reply -> w.display -> w.sel_prop -> w.aglist -> w.sel_ag -> w.contact_ag -> (w.ay_ack -> w.assigned -> w.prop_info -> la.qry -> WEBSITE | w.an_ack -> WEBSITE)).


AGENT = ( a.contact -> ( a.y_ack -> assigned -> a.prop_info -> ask_qry -> (l.ay_ack -> a.plot_ready ->( c.y_ack-> loan -> (apploan -> pay -> deli -> AGENT | rejloan -> AGENT)
     | c.n_ack -> AGENT)| l.an_ack ->AGENT) | a.n_ack -> AGENT)).
||CWA = ( CLIENT || WEBSITE ||AGENT) /
             { c.search/w.search , c.display/w.display ,
              c.sel_prp/w.sel_prop , c.aglist/w.aglist ,
              c.sel_ag/w.sel_ag , contact/w.contact_ag ,
              w.contact_ag/a.contact , agpack/w.ay_ack ,
                      agnack/w.an_ack , agpack/a.y_ack ,
               agnack/a.n_ack , c.assigned/w.assigned ,
              c.assigned/assigned , webprop/w.prop_info
             ,w.prop_info/a.prop_info , al.qry/ask_qry ,
                   lq.pack/l.ay_ack , lq.nack/l.an_ack ,
         c.plot_ready/a.plot_ready , c.ay_ack/c.y_ack ,
                  c.an_ack/c.n_ack , al.qry/l.ask_qry ,
                     lq.pack/l.y_ack, lq.nack/l.n_ack ,
      c.payment/rcv_payment, c.plot_ready/a.plot_ready ,
            a.plot_ready/ready_plot , c.ay_ack/rd_pack ,
             c.an_ack/rd_nack , c.apply_loan/applyloan ,
                 c.yb_ack/b.confirm ,c.nb_ack/b.reject ,
                   c.apply_loan/loan , c.yb_ack/apploan,
```

```
                    c.nb_ack/rejloan , c.payment/pay ,
                       c.deliver/deliver ,
c.deliver/deli }.
```

## A.14 CLIENT, WEBSITE, AGENT and LANDLORD

```
CLIENT = ( c.search-> c.display -> c.sel_prp -> c.aglist -> c.sel_ag -> contact -> (agpack -> c.assigned -> webprop -> al.qry -> (lq.pack -> c.plot_ready -> (c.ay_ack -> c.apply_loan ->(c.yb_ack -> c.paymeny ->c.deliver -> CLIENT |c.nb_ack ->CLIENT) | c.an_ack -> CLIENT)|lq.nack -> CLIENT)| agnack ->CLIENT)).

WEBSITE = ( w.search -> w.send_qry -> w.reply -> w.display -> w.sel_prop -> w.aglist -> w.sel_ag -> w.contact_ag -> (w.ay_ack -> w.assigned -> w.prop_info -> la.qry -> WEBSITE | w.an_ack -> WEBSITE)).

AGENT = ( a.contact -> ( a.y_ack -> assigned -> a.prop_info -> ask_qry -> (l.ay_ack -> a.plot_ready ->( c.y_ack-> loan -> (apploan -> pay -> deli -> AGENT | rejloan -> AGENT) | c.n_ack -> AGENT)| l.an_ack ->AGENT) | a.n_ack ->AGENT)).

LANDLORD = ( l.ask_qry -> (l.y_ack -> ready_plot -> (rd_pack -> applyloan -> (b.confim -> rcv_payment -> deliver -> LANDLORD|b.reject ->LANDLORD)| rd_nack -> LANDLORD) | l.n_ack -> LANDLORD)).

||CWAL = ( CLIENT || WEBSITE ||AGENT || LANDLORD ) /

            { c.search/w.search , c.display/w.display ,
              c.sel_prp/w.sel_prop , c.aglist/w.aglist ,
              c.sel_ag/w.sel_ag , contact/w.contact_ag ,
              w.contact_ag/a.contact , agpack/w.ay_ack ,
    agnack/w.an_ack , agpack/a.y_ack , agnack/a.n_ack ,
          c.assigned/w.assigned , c.assigned/assigned ,
         webprop/w.prop_info ,w.prop_info/a.prop_info ,
                    al.qry/ask_qry , lq.pack/l.ay_ack ,
         lq.nack/l.an_ack , c.plot_ready/a.plot_ready ,
  c.ay_ack/c.y_ack , c.an_ack/c.n_ack , al.qry/l.ask_qry
                   , lq.pack/l.y_ack, lq.nack/l.n_ack ,
     c.payment/rcv_payment, c.plot_ready/a.plot_ready ,
```

```
          a.plot_ready/ready_plot , c.ay_ack/rd_pack ,
                                    c.an_ack/rd_nack ,
         c.apply_loan/applyloan , c.yb_ack/b.confirm ,
               c.nb_ack/b.reject , c.apply_loan/loan ,
                  c.yb_ack/apploan, c.nb_ack/rejloan ,
  c.payment/pay , c.deliver/deliver , c.deliver/deli }.
```

## A.15 Real-Estate Webservice

```
CLIENT = ( c.search-> c.display -> c.sel_prp ->
c.aglist ->    c.sel_ag -> contact -> (agpack ->
c.assigned -> webprop -> al.qry -> (lq.pack ->
c.plot_ready -> (c.ay_ack ->  c.apply_loan ->(c.yb_ack
-> c.paymeny ->c.deliver -> CLIENT |c.nb_ack ->CLIENT)
| c.an_ack -> CLIENT)|lq.nack ->   CLIENT)| agnack -
>CLIENT)).


WEBSITE = ( w.search -> w.send_qry -> w.reply ->
w.display ->   w.sel_prop -> w.aglist -> w.sel_ag ->
w.contact_ag ->     (w.ay_ack -> w.assigned ->
w.prop_info -> la.qry -> WEBSITE   | w.an_ack ->
WEBSITE)).


PLOTDATA = ( p.send_qry -> p.reply -> PLOTDATA).


AGENT = ( a.contact -> ( a.y_ack -> assigned ->
a.prop_info -> ask_qry -> (l.ay_ack -> a.plot_ready ->(
c.y_ack-> loan ->   (apploan -> pay -> deli -> AGENT |
rejloan -> AGENT) | c.n_ack -> AGENT)| l.an_ack -
>AGENT) | a.n_ack -> AGENT)).


LANDLORD = ( l.ask_qry -> (l.y_ack -> ready_plot ->
(rd_pack ->    applyloan -> (b.confim -> rcv_payment ->
deliver ->     LANDLORD|b.reject ->LANDLORD)| rd_nack -
> LANDLORD)|   l.n_ack -> LANDLORD)).
```

```
BANK = (rcv_loan.req -> (b.pack ->pays -> deliv ->
BANK|b.nack->  BANK)).


||CWALBP = ( CLIENT || WEBSITE || PLOTDATA ||AGENT ||
LANDLORD || BANK ) /
            { c.search/w.search , c.display/w.display ,
             c.sel_prp/w.sel_prop , c.aglist/w.aglist ,
             c.sel_ag/w.sel_ag , contact/w.contact_ag ,
             w.contact_ag/a.contact , agpack/w.ay_ack ,
                     agnack/w.an_ack , agpack/a.y_ack ,
               agnack/a.n_ack , c.assigned/w.assigned ,
              c.assigned/assigned , webprop/w.prop_info
            ,w.prop_info/a.prop_info , al.qry/ask_qry ,
                  lq.pack/l.ay_ack , lq.nack/l.an_ack ,
         c.plot_ready/a.plot_ready , c.ay_ack/c.y_ack ,
                  c.an_ack/c.n_ack ,al.qry/l.ask_qry ,
                    lq.pack/l.y_ack, lq.nack/l.n_ack ,
     c.payment/rcv_payment, c.plot_ready/a.plot_ready ,
           a.plot_ready/ready_plot , c.ay_ack/rd_pack ,
            c.an_ack/rd_nack , c.apply_loan/applyloan ,
                c.yb_ack/b.confirm ,c.nb_ack/b.reject ,
                  c.apply_loan/loan , c.yb_ack/apploan,
                     c.nb_ack/rejloan , c.payment/pay ,
                   c.deliver/deliver , c.deliver/deli ,
                             c.apply_loan/rcv_loan.req ,
                    c.yb_ack/b.pack , c.nb_ack/b.nack ,
                    c.payment/pays  , c.deliver/deliv ,
              w.send_qry/p.send_qry ,w.reply/p.reply }.
```

# Appendix B

## B.1 CLIENT process

```
property SAFE_B = ( c.nb_ack -> cancel_order ->
SAFE_B).

property SAFE_A = ( c.an_ack ->cancel_rcv_qt ->
SAFE_A).

||SAFE_CLIENT = ( CLIENT || SAFE_B || SAFE_A ).
```

## B.2 WEBSITE process

```
property SAFE_AG = (w.an_ack -> cancel_agent ->
SAFE_AG).

||SAFE_WEB = ( WEBSITE || SAFE_AG ).
```

## B.3 AGENT process

```
property SAFE_C = (c.n_ack -> cancel_order -> SAFE_C).

property SAFE_LA = (l.n_ack -> not_available ->
SAFE_LA).

property SAFE_A2 = (a.n_ack -> cancel_ag -> SAFE_A2).

||SAFE_AGENT = ( AGENT ||SAFE_C || SAFE_LA || SAFE_A2).
```

## B.4 LANDLORD process

```
property SAFE_LAN = (l.n_ack -> not_available ->
SAFE_LAN).

||SAFE_LAND = ( LANDLORD || SAFE_LAN ).
```

## B.5 BANK process

```
property SAFE_BA = (b.nack -> cancel_order -> SAFE_BA).

||SAFE_Bank = ( BANK|| SAFE_BA).
```

# References


[1] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Compatibility verification for web service choreography," *Proc. - IEEE Int. Conf. Web Serv.*, pp. 738–741, 2004.

[2] S. H. Ripon, "Process algebraic support for web service composition," *ACM SIGSOFT Softw. Eng. Notes*, vol. 35, no. 2, p. 1, Mar. 2010.

[3] "Web Service Composition - an overview | ScienceDirect Topics." [Online]. Available: https://www.sciencedirect.com/topics/computer-science/web-service-composition. [Accessed: 09-Sep-2019].

[4] F. Daniel, "Web Service Orchestration and Choreography," in *E-Business Models, Services and Communications*, 2011.

[5] *Concurrency: State Models & Java Programs, 2nd Edition byJeff MageeandJeff Kramer John Wiley & Sons 2006 (432*, vol. 2006. 2006.

[6] "FSP-notation." [Online]. Available: https://www.doc.ic.ac.uk/~jnm/LTSdocumention/FSP-notation.html. [Accessed: 09-Sep-2019].

[7] "Internet real estate - Wikipedia." [Online]. Available: https://en.m.wikipedia.org/wiki/Internet_real_estate. [Accessed: 09-Sep-2019].