PETRI NET MODELING OF

WEB SERVICES

BY

MUHAMMAD ASIF JAVED

Master of Science

Punjab University

Lahore, Pakistan

1990

Submitted to the Faculty of the
Graduate College of he
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
August 2003

PETRI NET MODELING OF WEB SERVICES

Thesis Approved:

[illegible]

Thesis Advisor

[illegible]

[illegible]

Timothy J. Pettibone

Dean of the Graduate College

ACKNOWLEDGEMENTS

I feel highly privileged in taking the opportunity to thank my worthy advisor Dr. Johnson P. Thomas, under whose auspices I took a stride in the completion of this work. I can never forget his incessant meticulous criticism, affectionate supervision, distinguished inspiring behavior and valuable knowledge, which he contributed to this work in a multitude of ways.

My heartiest thanks to committee members Dr. Marcin Paprzycki and Dr. Debao Chen for their valuable criticism and suggestions that helped me to cover the gaps. I love to thank Dr. Adjith Abraham for accepting my request to be a committee member while the time was very short.

I am in deep dept of gratitude to Mr. Khalique Rehman for his cooperation throughout the course of my study.

I acknowledge from the core of my heart the Chaudhary family for their kind help in all ways and loving behavior. And especially words are inadequate to thank Mr. Zahir Chaudhary and Mr. Nasir Chaudhary.

I would like to say special thank to my wife Saima Waheed who stood beside me all the time with her unfailing and indispensable support.

A heart-felt thanks goes to my parents and brother for their encouragement and emotional support throughout my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

d	Destination
HTTP	Hyper Text Transfer Protocol
IDE	Integrated Development Environment
M	Marking
M_{fin}	Final Marking
MIME	Multipurpose Internet Mail Extensions
M_{ini}	Initial Marking
P	Place
P_{inp}	Input Place
PKI	Public Key Infrastructure
PN	Petri Net
P_{out}	Output Place
R	Reachability
RPC	Remote Procedure Call
T	Transition
t_m	Time Stamp
t_{reset}	Reset Transition
SAML	Security Assertion Markup Language
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
W	Weight
WSDL	Web Services Description Language

WSPN	Web Service Petri Net
XACML	Extensible Access Control Markup Language
XML	Extensible Markup Language
XSD	XML Schema Definition
UDDI	Universal Description, Discovery and Integration

Chapter 1

Introduction to Web Services

1.1 Web services

In today's e-commerce environment, various commercial and other organizations provide their services through the web. These services range from educational services, which provide distance learning to commercial services such as credit rating services. For example, a car purchase transaction requires accessing a number of web services. Car dealer uses bank's web service to get financing for their customers while bank uses credit bureau's web service to check customer's credit history to decide for approval. A single business transaction therefore invokes a number of web services.

Web services are defined as "internet-based applications fulfilling a specific task or a set of tasks, that can be combined with other web services to maintain workflow or business transactions" [13].

Web Services, which is a distributed computing environment, is based on Extensible Markup Language (XML) technology. The concept of business communication is certainly not new, but web services bring a new perspective to the cross-domain interaction [11]. Web services [10] provide a conceptual foundation and a technology infrastructure for service-oriented computing. In web services interoperability is at highest priority. It allows program written in different languages on different platforms to communicate with each other in standard-based way. Web services are considered as reusable software components [14] over the Internet by wrapping the interface with XML and publishing it over the Internet. Web services are in fact a standardized integration approach. They are not limited to one environment, but can be integrated into every software-system that is web service-aware [9]. Web services have different layers. XML Schema (XSD)-used to define the message format which describes the type and structure of XML document, Simple Object Access Protocol (SOAP)-message envelope also defines a standard representation for errors and binding to HTTP or other open protocols and RPC, Web Services Description Language (WSDL)-explains how operations can be invoked using particular transport protocol bindings and Universal Description, Discovery and Integration (UDDI)- to get the technical details.

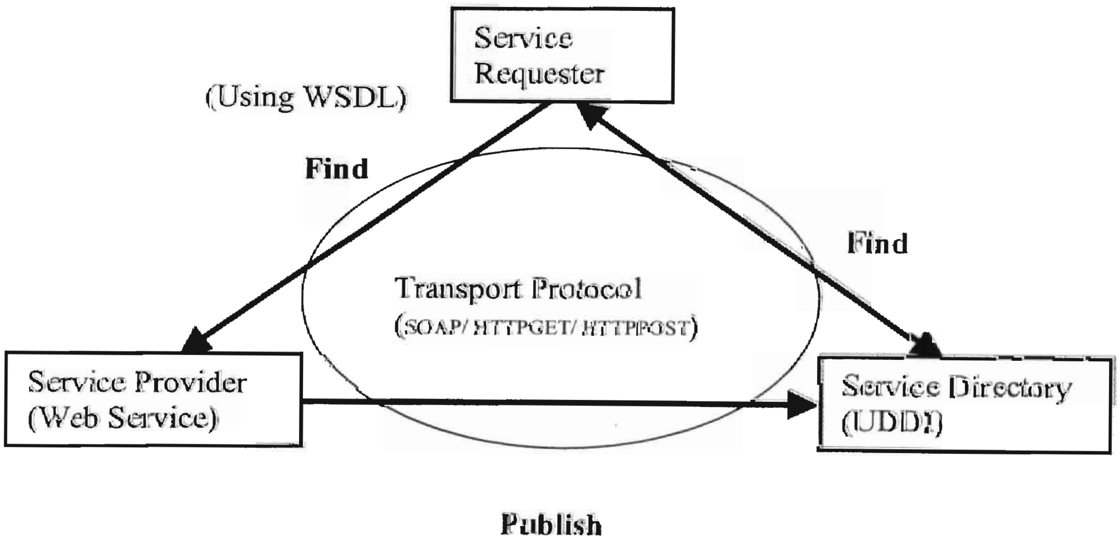


Fig. 1 General Architecture of Web Service

The scope of web services applications goes beyond organizational boundaries, such as e-Business. Web-based e-commerce began with stand-alone Web servers, but these Web servers soon began to be inter-connected with other systems [15]. Web services are rapidly evolving and are expected to change the paradigms of both software development and use [12].

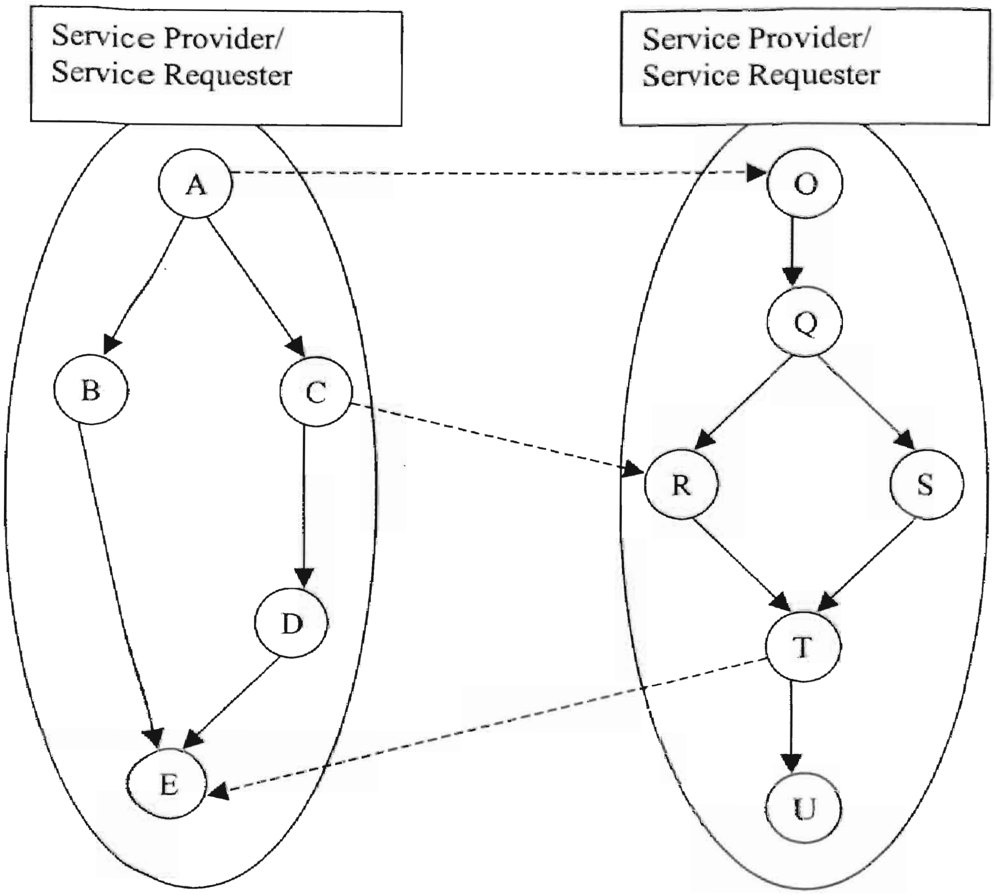


Fig. 2 Two interacting business process

As shown in Fig. 2, to make the business process work the requesting operation of one process (A, C, and T) is associated with matching performing operations (O, R, and E) in the other process [20].

Microsoft, SUN Microsystems, IBM, HP and some others have recently developed tools for building the Web Services.

Microsoft Visual Studio.Net has made it very easy to create, deploy and access an XML Web service. Visual Studio. Net have friendly interface and is very powerful tool to develop Web Services. So we used Microsoft's Visual Studio.Net to develop Web Services and to test our developed software.

1.2 Visual Studio.NET is a powerful set of tools for building web applications. It simplifies the development of XML Web Services and generates required codes. Different web references can be added to access those sites automatically and it generates WSDL which have the information that how operations will be completed. The Visual Basic, C++, C#, Jscript and XML languages create a mixed language solution by using same Integrated Development Environment (IDE) and leverage the functionality of .NET Framework.

Web services involve interaction between different distributed sites on the Internet. New web services are continuously being added and linked to existing web services. In such a dynamic distributed environment, there is enormous potential for deadlock and other problems. It is therefore essential to model web services and reason about them to verify the correctness of the services. Although tools exist for developing web services, tools for the abstract modeling of these web services do not exist. In this thesis we model Web services using Petri Nets.

Other companies besides Microsoft have developed their Web Services and are offering their web services development tools in the market.

1.3 WebSphere SDK for Web Services (WSDK): IBM has built Web Services support in WSDK. It offers tools in WSDK to build Web Services for Java programmers. WebSphere offers wide interoperability with other software and systems.

1.4 Java Web Services Developer Pack 1.0 01 (Java WSDP): Sun Microsystems is supporting Web Services in "iPlanet" and developed (Java WSDP), which in conjunction with Java platform allows to built, test and deploy Web Services. Sun ONE Studio 4 provides the latest tools for Java to develop Web Services.

1.5 HP Web Services Platform 2.0: developed by HP, which provides a standard based architecture and toolset for creating and deploying Web Services.

Oracle is moving to support Web Services standards in its 9i database management software, application service and development tools.

Microsoft have developed Visual Studio.Net, which is very easy to use, have friendly interface and is very powerful tool to develop Web Services. So we used Microsoft's Visual Studio.Net to develop Web Services and to test our developed software.

1.6 Petri net is a graphical and mathematical tool [17] used for modeling and analyzing systems with concurrency. The mathematical framework behind Petri Nets facilitates reasoning about the model generated. For example, Petri net analysis of the model will detect deadlock in the model. Petri net consist of Places, Transitions and Arcs. Transitions are active components and on firing change the state of the system. Places can have tokens, which represent the current state of the system. Both (places and transitions) are connected with input (from places to transition) and output (from transition to places) Arcs. It can be used for variety of purposes.

Software is developed for Petri net modeling of WSDL and Methods, being used during and after the development of Web Services by the Visual studio. We use reachability analysis to detect deadlock. This thesis also proposes a Petri Net merging method to model the integration of distributed web services. The software development process begins by XML'S parsing of the generated WSDL. It gets input and output data from types elements, operation name from portType element, network protocol from binding element, and operation address from service element. Having these information Petri net will be drawn and to detect deadlock reachability is analyzed by using C/C++. The reachability analysis ends in final states if there is no deadlock otherwise there is deadlock.

Chapter 2
Objectives

The main objective of this thesis is to verify the correctness of message flows in web services. This is achieved by generating a model of the web services. This model is then analyzed to determine if the web services contain deadlocks, will terminate correctly and will function as desired. To realize this main objective we aim to:

1. Generate a Petri Net model of web services. This is achieved by parsing the WSDL and collecting all required information about input/output data, network protocol, operation name and destination.
2. Propose a mechanism for linking different distributed web services. As stated earlier, a single business transaction invokes a number of web services.
3. Reason about the generated Petri Net model. Software will be written to generate the reachability tree of the Petri Net model. This will allow us to detect deadlock, check for correct termination of the business transaction and detect other such potential problems

We will be using the Visual Studio.NET, Visual Basic, C and C++.

Chapter 3

.NET AND WEB SERVICES

3.1. Web Services

The emergence of Web Services represents the next evolution of e-business [1]. Kreger [2] defines a web service as an interface that describes a collection of operations that are network accessible through standardized XML messaging.

A formal definition of a web service may be borrowed from IBM [3].

> *Web services are a new breed of Web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes...Once a Web service is deployed; other applications (and other Web services) can discover and invoke the deployed service.*

Viewed from an n-tier application architecture perspective, the web service is a veneer for programmatic *access* to a service, which is then *implemented* by other kinds of middleware. Access consists of service-agnostic request handling (a listener) and a facade that exposes the operations supported by the business logic. The logic itself is implemented by a traditional middleware platform. Lambros et al [7] state the interactions between the service registry, service requestor and service provider in the context of the establishment and commencement of a web service relationship.

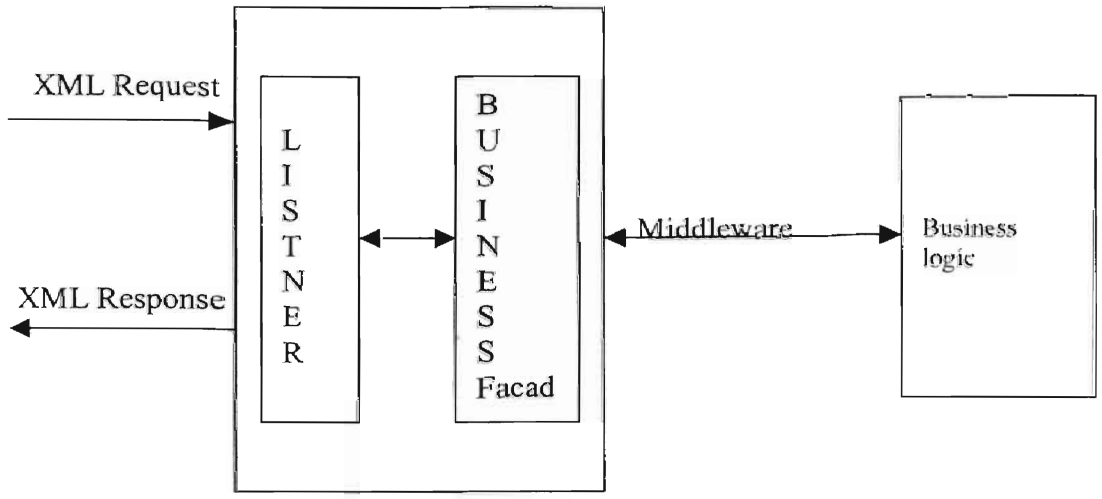


Fig. 3 Generic Web Service Architecture

Extensible Markup Language (XML) provides a metalanguage in which you can write specialized languages to express complex interactions between clients and services or between components of a composite service. Behind the facade of a web server, the XML message gets converted to a middleware request and the results converted back to XML. The full-function web services platform can be thought of as *XML plus* Hyper Text Transfer Protocol *(HTTP) plus* Simple Object Access Protocol *(SOAP)* [4] *plus* Web Services Definition Language (*WSDL*) [5] *plus* Universal Discovery Description and Integration (*UDDI*) [6].

The ability of Web Services to reach beyond the firewall, the loose coupling between applications encouraged by Web Service interfaces, and the wide support for core Web Service standards by major enterprise software vendors are the key reasons why Web Services technology promises to make integration of applications both within the enterprise and between different enterprises significantly easier and cheaper than before. Web Services are in essence a collection of standards and protocols that allow us to make processing requests to remote systems by speaking a common, non-proprietary language and using common transport protocols (HTTP, SMTP).
The introduction of the web services approach has increased efficiency however; business knowledge regarding relationships with suppliers and customers is now buried within the internal processing of the web services [8].

3.1.1 Idea behind Web services:

The basic idea behind Web services is to adapt the loosely coupled Web programming model for use in applications that are not browser-based.

3.1.2 Web Services Goal:

The goal is to provide a platform for building distributed applications using software running on different operating systems and devices, written using different programming languages and tools from multiple vendors, all potentially developed and deployed independently.

3.1.3 Major differences between Web services and traditional Web applications:

There are three major differences. Web services use SOAP messages instead of MIME messages, Web services are not HTTP-specific, and Web services provide metadata describing the messages they produce and consume.

Browsers usually just render HTML pages (or other MIME-typed data, like images) and leave interpretation of the information they display up to the user. Web service clients, on the other hand, typically need to interpret the data they receive and do something meaningful with it they may not even have a user interface.

The second major difference between Web services and traditional Web applications is that Web services are not transport protocol specific. While the SOAP specification only defines how to send SOAP messages over HTTP and that's what the vast majority of today's Web services do, other transport protocols can also be used. SOAP messages can be sent using SMTP, raw TCP, an instant messaging protocol like Jabber, or any other protocol you like.

The SOAP specification defines the notion of intermediaries, nodes that a message passes through on its way to its final destination. Using intermediaries, you can "virtualize" physical network topology so that messages can be sent to Web services using whatever path and whatever combination of transport protocols is most appropriate.

The third major difference between Web services and traditional Web applications is that Web services are self-describing; they provide metadata describing the messages they produce and consume, the message exchange patterns they use to expose behaviors, the physical transport protocols they use, and logical addressing information required to invoke them. A Web service's message formats are defined using XML Schema (XSD).

3.1.4 Security

Without doubt, the main problem facing Web Services is security. This is true with all software, but since the rapidly growing technology area that is Web Services doesn't yet have dedicated security protocols and standards, it is a more glaring hole. It is, though, a hole that is being filled, with proposals to secure all levels of the communication being presented and submitted; Security Assertion Markup Language (SAML, **http://www.saml.org/**, or **http://www.oasis-open.org/committees/security/**), for transferring the security level (authorization and authentication), along with XML encryption and Public Key Infrastructure (PKI), which can be managed by XML Key Management Specification; the security of storage of documents in repositories such as UDDI registries is targeted by Extensible Access Control Markup Language (XACML, **http://www.oasis-open.org/committees/xacml/**).

3.2 Analysis Of Web Services

Web services enable the exchange of data and the remote invocation of application logic using XML messaging to move data through firewalls and between heterogeneous systems. The programs written in any language, using any component model, and running on any operating system can access XML Web services.

It can be explained with the help of a Car Sale example, in which three different web sites are communicating automatically with each other. Graphically their communication can be shown as:

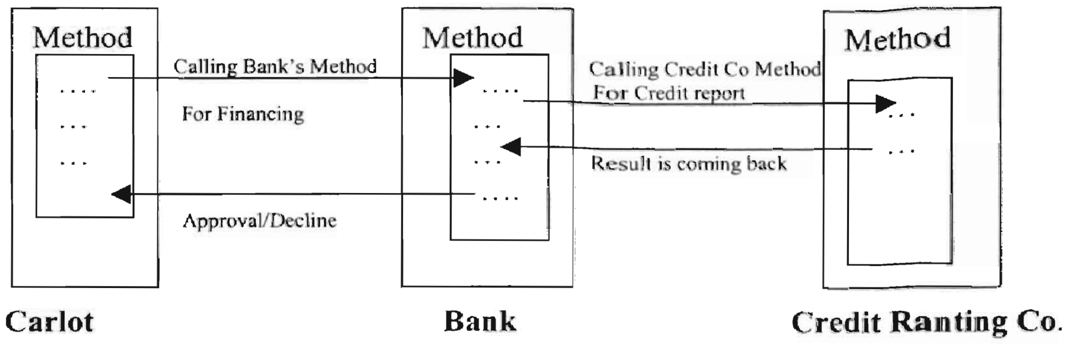


Fig. 4 A scenario of Interaction among Carlot, Bank and Credit Rating Co.

If a customer at Car Dealer needs a bank financing, then customer's data is needed by the bank to decide about loan approval or rejection. To send customer's data from Car Dealer to Bank, we need to invoke bank's method over Internet. To invoke bank's method, required parameters and its data type are encoded which are checked from the WSDL file, published by the bank.

When Bank receives customer's data, it checks if that is its active customer. If so then it approves otherwise Bank sends data to Credit Rating Co to check customer's credit history. Here bank is invoking another website's method through Internet, after meeting the requirements given in WSDL file published by the Credit Rating Co.

Through Internet bank invokes the method of Credit Rating Co. This method is calculating credit score, and after calculations it returns the result to bank.

When bank receives credit rating of the customer then it calculates interest rate and sends its decision back to car dealer. This whole flow is automatic, which is saving labor cost and avoiding delay due to unavailability of any concerned person at any point.

During building or consuming Web Services there are few key specifications and technologies that need to be encountered.

1- Extensible Markup Language (XML) – A standard way to represent data. It provides a significant advance in how data is described and exchanged by Web-based applications using SOAP. The Hypertext markup language (HTML) enables universal methods for viewing data while XML provides universal methods for working directly with data. XML facilitates the transfer of structured data between servers themselves.

2- Simple Object Access Protocol (SOAP) - A common, extensible, message format. SOAP is a lightweight XML based protocol for exchange of information in a decentralized, distributed environment. SOAP has three parts, i.e. envelope, header and body. SOAP codifies the use of XML as an encoding scheme for request and response parameters using HTTP as a transport.

3- Web Services Description Language (WSDL) - A common, extensible, service description language. It is explained in detail later.

4- Universal Discovery Description and Integration (UDDI) – To discover service providers. UDDI provides a mechanism for clients to dynamically find other web services. Using a UDDI interface, businesses can dynamically connect to services provided by external business partners. A UDDI registry is similar to a CORBA trader, or it can be thought of as a DNS service for business applications. A UDDI registry has two kinds of clients: businesses that want to publish a service (and its usage interfaces), and clients who want to obtain services of a certain kind and bind programmatically to them.

3.3 Anatomy Of WSDL

WSDL is an XML document. WSDL is used to describe *what* a web service can do, *where* it resides, and *how* to invoke it. It provides critical information about the Web Service that both the developers and programming tools need. In a compact, concrete way, this document describes everything, including:

- Messages that the Web Service understands and the format of its responses to those messages
- Protocols that the service supports
- Where to send messages

WSDL focuses on describing wire formats, not on describing implementation details of an endpoint.

A WSDL document always has a **<definitions>** element as its root.

Here we declare the WSDL namespace as the default namespace for the document so all elements belong to this namespace unless they have another namespace prefix.

The WSDL-specific elements are:

1- **types**: Describes the types used by **messages**

2- **message**: Defines the data passed from one point to another in a call

3- **portType**: Defines a collection of **operations**.

operation: Defines a combination of **input**, **output**, and **fault** messages

input: A **message** that is sent to the server

output: A **message** that is sent to the client

fault: An error value returned as a result of a problem processing a **message**

There are two main classes of faults: client faults and server faults. If a client fault occurs, the client should not resend the request until it fixes the input data in some fashion. A server fault indicates there was a failure at the server that was no fault of the client. Clients might wait a short time and try resending the request.

4- **binding**: Describes the protocol being used to carry the Web Service communication; bindings currently exist for SOAP, HTTP GET, HTTP POST, and MIME.

5- **Service**: Defines a collection of **ports** (end points); port specifies an address for a binding, thus defining a single communication endpoint. Each **service** should map to one **portType** and represent different ways of accessing the **operations** in that **portType**.

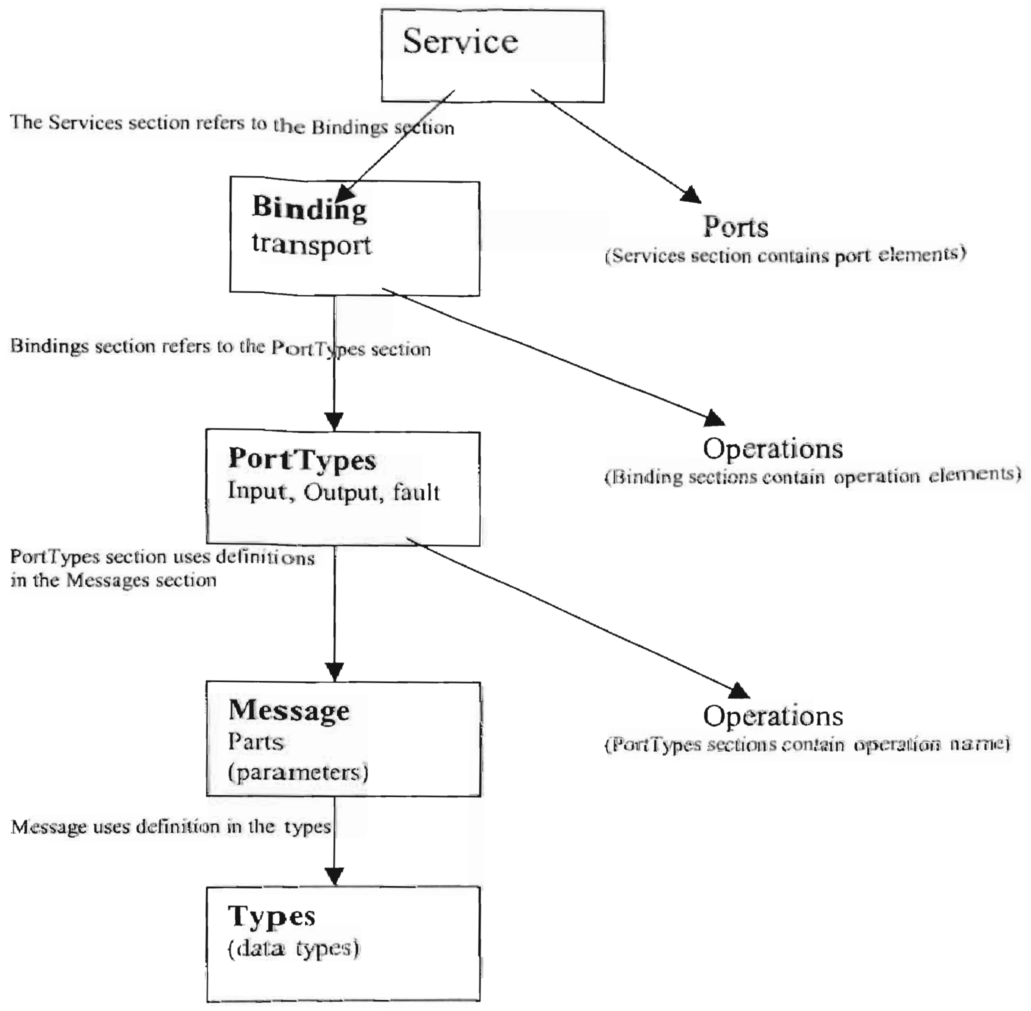

Service
The Services section refers to the Bindings section
Binding
transport
Ports
(Services section contains port elements)
Bindings section refers to the PortTypes section
Operations
(Binding sections contain operation elements)
PortTypes
Input, Output, fault
PortTypes section uses definitions in the Messages section
Message
Parts
(parameters)
Operations
(PortTypes sections contain operation name)
Message uses definition in the types
Types
(data types)

Fig. 5 Abstract Definition of WSDL

By using Bank and Credit Rating CO's example, the WSDL's each element can be explained.

3.3.1 TYPES

It defines the data types used by the messages. It has the information about both messages- input and output. Output message name ends with Response. The

<complexType> can express more than just the equivalent of a struct in C. For example, a data type for bank will contain strings.

3.3.2 MESSAGE

Messages consist of one or more logical parts. Each part is associated with a type from some type system using a message-typing attribute. The part name attribute provides a unique name among all the parts of the enclosing message.

In message each part defines the input and output parameters. There incoming and outgoing both messages are described along with their protocol of transfer. In our example all three ways SOAP, HTTPGET, and HTTPPOST for input and output messages are described.

3.3.3 PORT TYPES

It is collection of operations. The name of operation is name of the method that is being called. The operation specifies two messages, input message (sent to the web services) and output message (from web services to the client). PortType have information for operations, which can use any one out of three network protocols, i.e. SOAP, HTTPGET or HTTPPOST. Input and output message of one operation use one type of protocol.

WSDL has four transmission primitives that an endpoint can support:

- **One-way**. The endpoint receives a message.
- **Request-response**. The endpoint receives a message, and sends a correlated message.
- **Solicit-response**. The endpoint sends a message, and receives a correlated message.

- **Notification**. The endpoint sends a message.

WSDL refers to these primitives as **operations.**

3.3.4 BINDINGS

A binding defines message format and protocol details for operations and messages defined by a particular portType. There may be any number of bindings for a given portType. The **binding** element can be used to define how each **operation** within the **portType** maps to a particular protocol. A binding MUST specify exactly one protocol. The **binding** element states that the operations can travel using SOAP over HTTP, HTTPGET, or HTTPPOST. So it binds the operation and transfer protocols. In input and output use is equal to literal. "Literal" means that the resulting SOAP message contains data formatted exactly as specified in the abstract definitions (Types, Messages, and PortTypes sections). Message the each parts define the concrete schema of the message. The <binding> element is given a name (in this case " CreditRatingsSoap ") so that the <port> element in the Services section can refer to it. It has a "type" attribute that refers to a <portType>, which in this case is " CreditRatingsSoap ".

3.3.5 SERVICES

A service is a set of **<port>** elements. A port defines an individual endpoint by specifying a single address for a binding.

Each <port> element associates a location with a <binding> in a one-to-one fashion. Location has the address of the file, that file have the method need to be invoked. There can be more than one <service> element in a WSDL document. Within one WSDL document, the <service> "**name**" attribute distinguishes one service from another. The

binding attribute refers to the binding using the linking rules defined by WSDL. Because there can be several ports in a service. Client can search for the <service> that matches the protocol that it can deal with. A port MUST NOT specifies more than one address. Ports within a service have the following relationship:

None of the ports communicate with each other (e.g. the output of one port is not the input of another).

If a service has several ports that share a port type, but employ different bindings or addresses, the ports are alternatives. Each port provides semantically equivalent behavior (within the transport and message format limitations imposed by each binding). This allows a consumer of a WSDL document to choose particular port(s) to communicate with based on some criteria (protocol, distance, etc.).

By examining it's ports, we can determine a service's port types. This allows a consumer of a WSDL document to determine if it wishes to communicate to a particular service based whether or not it supports several port types. This is useful if there is some implied relationship between the operations of the port types, and that the entire set of port types must be present in order to accomplish a particular task.

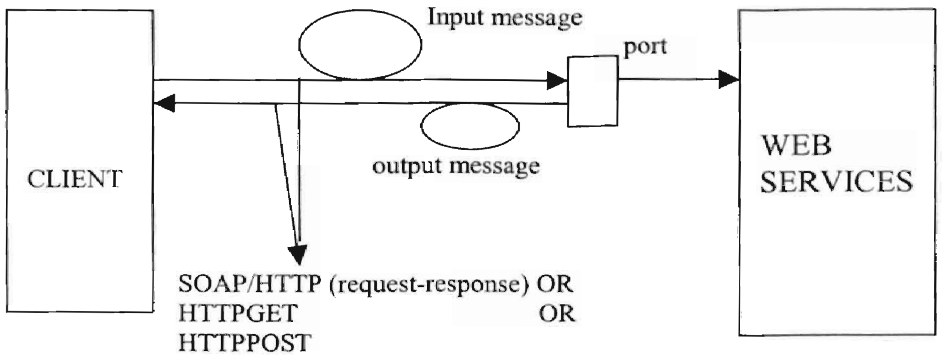


Fig. 6 A client invoking a Web Service

UDDI have all information about the Web services providers. Using UDDI the Web services provider(s) is selected which meet the required criteria. After choosing Web Services and having it's WSDL the method is developed accordingly. More than one Web services requiring different transfer protocol can be selected. Different clients can use the same Web service. One Web service also can use another web service to fulfill it requirements. In this way a big web like structure is developed. A small part of that is shown in Fig 7.

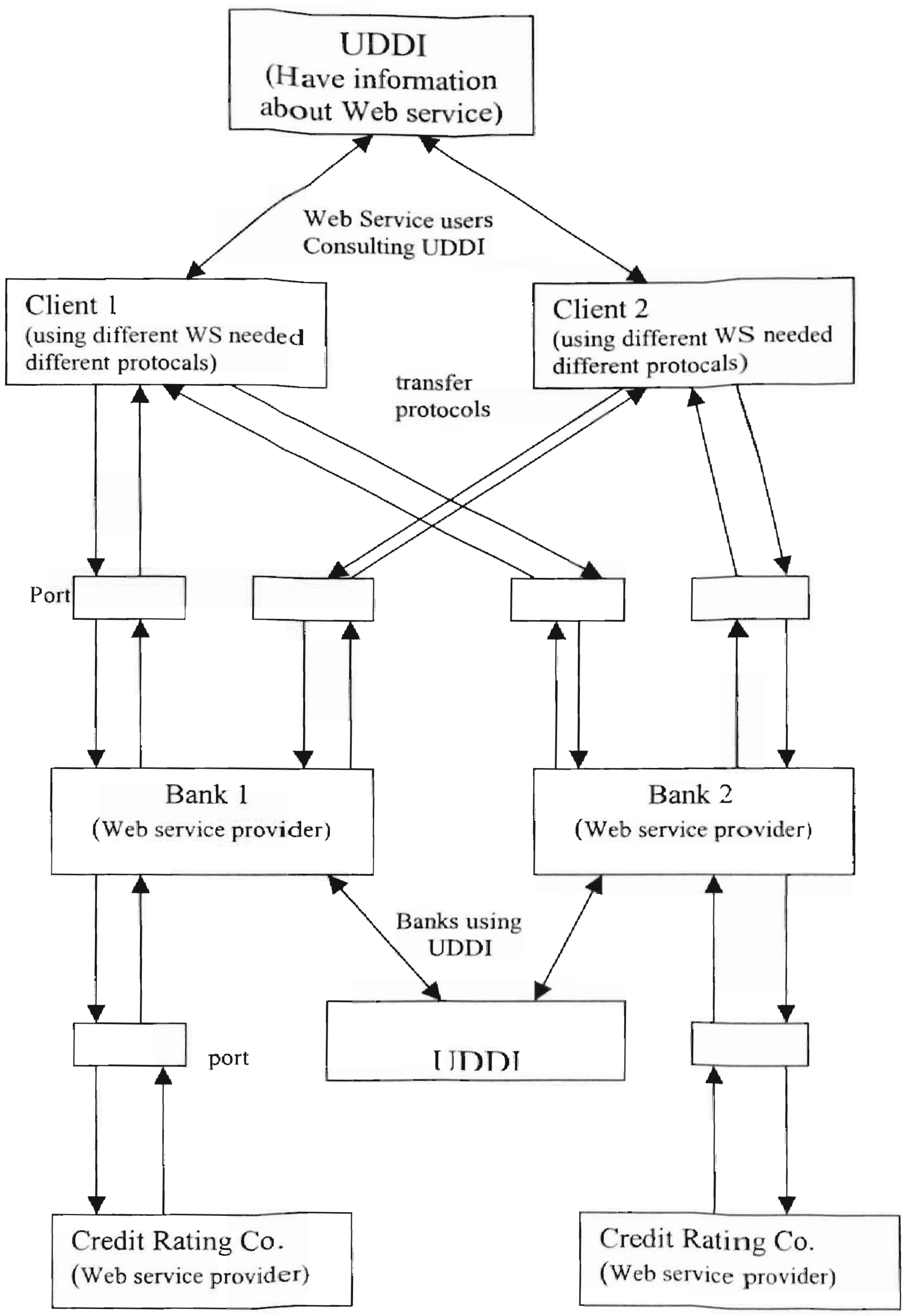


Fig. 7 A scenario showing the usage of UDDI and Web Services

After establishing contact with UDDI, a WSDL of a Carlot can be modeled as:

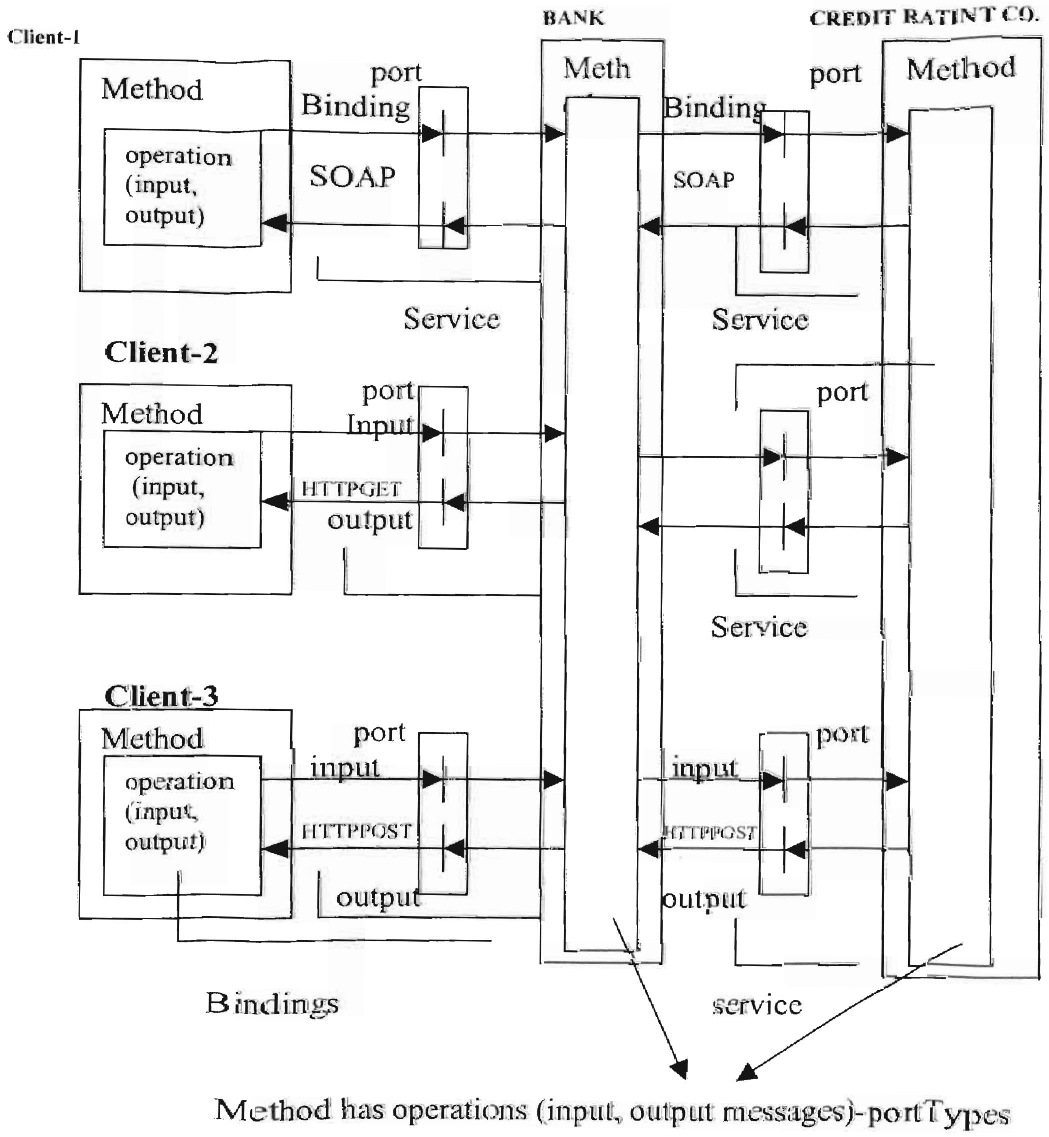


Fig. 8 WSDL model for carport transaction

Chapter 4
Petri Nets

Carl Adam Petri originally gave the concept of Petri Net, in his dissertation [16] in 1962. Petri nets are graphical and mathematical modeling tool applicable to many systems. They are very useful tool for describing and studying information processing systems, which are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic [17]. They can be used both by practitioners and theoreticians. Petri nets representation can also be used to analyze the performance and throughput [18].

4.1 Introduction

Petri nets are directed graphs. They include Places, Transition, Token, and Arcs.

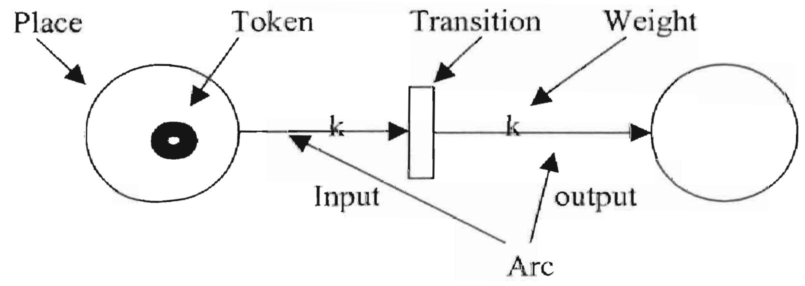


Fig. 9 Structure of Petri Net

Places may represent states, before transition represents pre-firing state and right of the transition represents post-firing state. Places contain tokens. Token could be data or truth of condition. Transitions are active components, transitions are allowed to fire only when they are enabled. Enabled means that pre-firing conditions are met successfully. Arcs represent the flow of system and carry weight where k-weight represents the k parallel arcs.

The formal definition of Petri net [17] is as:

A Petri net is a 5-tuple, PN = (P, T, F, W, M) where: $P = \{p_1, p_2, \ldots p_m\}$ is a finite set of places, $T = \{t_1, t_2, \ldots t_n\}$ is a finite set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is a finite set of arcs, $W: F \rightarrow \{1,2,3,\ldots\}$ is a weight function, $M: P \rightarrow \{0,1,2,3,\ldots\}$ is the initial marking, $P \cap T = \varnothing$ and $P \cup T \neq \varnothing$. A Petri net structure $N = \{P, T, F, W\}$ without any specific initial marking is denoted by N. A Petri net with given initial marking is denoted by (N, M).

Table –I

The complete absence of deadlock is closely related to the liveness. So a transition t in a Petri net N with initial marking M is said to be live if and only if for any M^1 in $R(N, M)$, there exists a marking reachable from M^1 in which t is enabled. A Petri net is live if all its transitions are live. Petri net is safe if and only if all its places are safe, while place is safe if it is one-bounded ($M(p) \leq 1$). A path in a Petri net is a finite alternating sequence of transitions and places. A circuit in a Petri net is a path that begins and ends at the same transition such that no transition or place appears more than once in the circuit. A live and save path is a path in a Petri net such that all the transitions in the path are live and the places in the path are safe, given an appropriate initial marking of the net [18].

4.2 Reachability

A fundamental basis for studying the dynamic properties of any system is reachability. In a net the token distribution (marking) is changed on firing the enabled transition. A sequence of firings will result in a sequence of markings. A marking M^{1} is said to be reachable from a marking M if there exists a sequence of firings that transforms M to M^{1}. A firing or occurrence sequence of is denoted by $\sigma = M\ t\ M_1\ t_2\ M_2 \ldots\ldots M_n$ or simply $\sigma = t_1\ t_2 \ldots t_n$. Here M_n is reachable from M by σ and we write $M[\sigma > M_n$. The set of all possible markings reachable from M in a net (N, M) is denoted by $R(N, M)$ or simply $L(M)$. The reachability problem for Petri nets is the problem of finding if $M_n \varepsilon (R(N, M)$ for a given marking M_n in a net (N, M) [17].

4.3 Petri Net model of Web Services

In our model the mapping of WSDL's parts is as below:

Place →	PortType (Operations – input, output messages)
Transition →	Service - Port (Name, Binding name, Location)
Token →	Message (Data)
Arc →	Binding (PortType, Protocol)

Table-II

The mapping for methods is defined as:

Place →	data storage
Transition →	computational primitives
Token →	Message (Data)

Table-III

The Web Services Petri Net (WSPN) representation of web service flows or WSDL is an ordinary Timed Petri net N:

$N = (P, T, F, W, M, t_m, d)$

Where the following applies.

- $P = \{ p_1, p_2, \ldots, p_m\}$ is a finite set of places representing the set of assembly objects.

- $T = \{ t_1, t_2, \ldots, t_n\} \cup \{ t_{reset}\}$ is a finite set of transitions; $\{ t_1, t_2, \ldots, t_n\}$

Transitions represent Service-Port in the WSDL and operations or computation in the methods of the web service. The transition t_{reset} models the resetting of the web service for the next transaction.

- F and W are same as mentioned above.

- There exists a set of places $P_{ip} \subseteq P$ where P_{ip} represents the initial state of the Web service transaction

- There exists a set of places $P_{op} \subseteq P$ where P_{op} represents the final state of the Web service transaction

- $P_{ip} \cap P_{op} = \varnothing$.

- If $p \in P_{ip}$, $M(p) = 1$ else 0. This is the initial marking.

- tm: $P \rightarrow$ time is the time stamp associated with a token at a place.

- d: $T \rightarrow$ time is the time duration associated with a transition.

In the initial marking of the net, there will be a token in each of the places P_{ip} representing the initial state of the web service transaction Attached to each place with a token is a time value specifying the time of data initialization for the transaction This is given by the function t_m. It can be seen that the WSPN modeling Web Services are live and safe.

4.3.1 Properties of WSPN

The formal basis of Petri nets allows the derivation and deduction of properties of the WSPN. Properties of the WSPN are listed below:

Property 1: Given the WSPN N representing web services, a subnet N^{1} representing one plan in N is conflict-free and persistent.

It is not sufficient to show that the WSPN is live and safe. The unique initial and terminating conditions of the transaction must be satisfied. This leads to the next two properties.

Property 2: The property of clean termination. This property states that the web services transaction terminates only when the transaction has been completed. A web services transaction should not terminate with a partial transaction.

Theorem 1- Property of Clean Termination: Given an WSPN

$N = (P, T, F, W, M, t_m, d)$ with initial marking M, there exists a final marking M_{fin} such that

$M_{\text{fin}} \in R(N, M)$ and $M_{\text{fin}} = M_{\text{op}}$(that is, $M(P_{op}) = 1$).

Property 3: The property of complete initialization. This property states that a web services transaction an assembly cannot terminate cleanly (Theorem 1) if all the data or information required for the transaction are not received from the external environment.

Theorem 2 – Property of Complete Initialization. Given an WSPN N = (P, T, F, W, M, t_m, d) with initial marking M, where only a subset of the input places are marked, that is, if $\exists M^{1} \subset M$, then the marking M where $M(P_{\text{op}}) = 1$ cannot be reached.

Proofs for these properties are simple and therefore omitted. The liveness property of the net guarantees that the plan will not deadlock and the safeness property ensure that each stage in the transaction is unique. The WSPN representation therefore provides for a precise and accurate framework for the description of web services, and furthermore, captures desirable properties of web services.

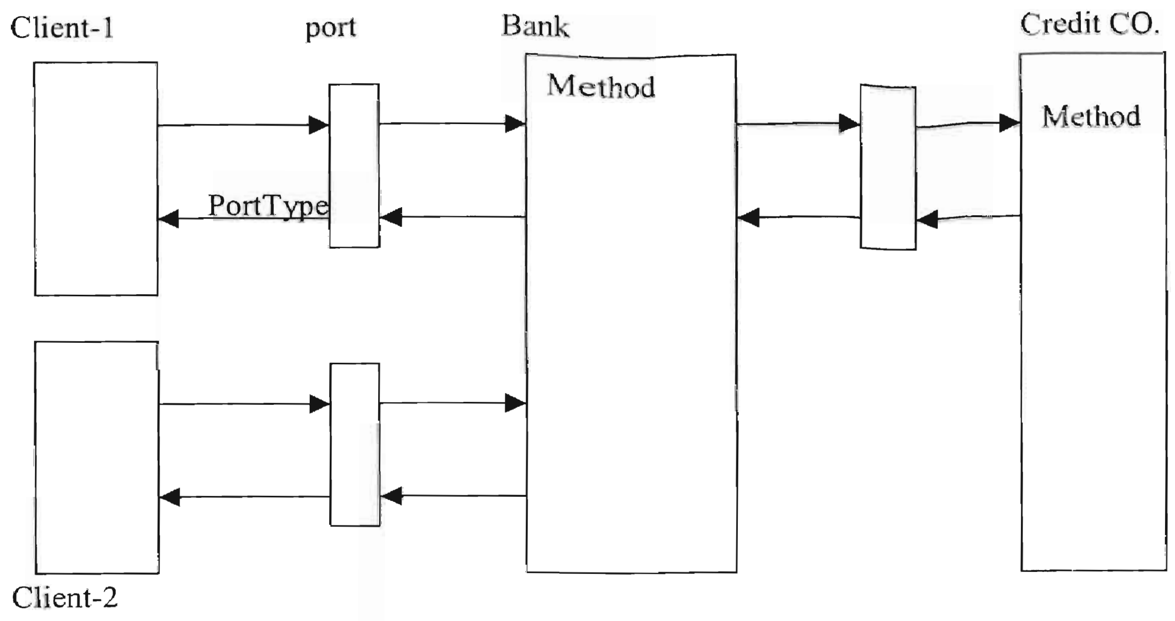


Fig. 10 WSDL flow model of Carlot

The Petri net model of WSDLs is below Fig. 10:

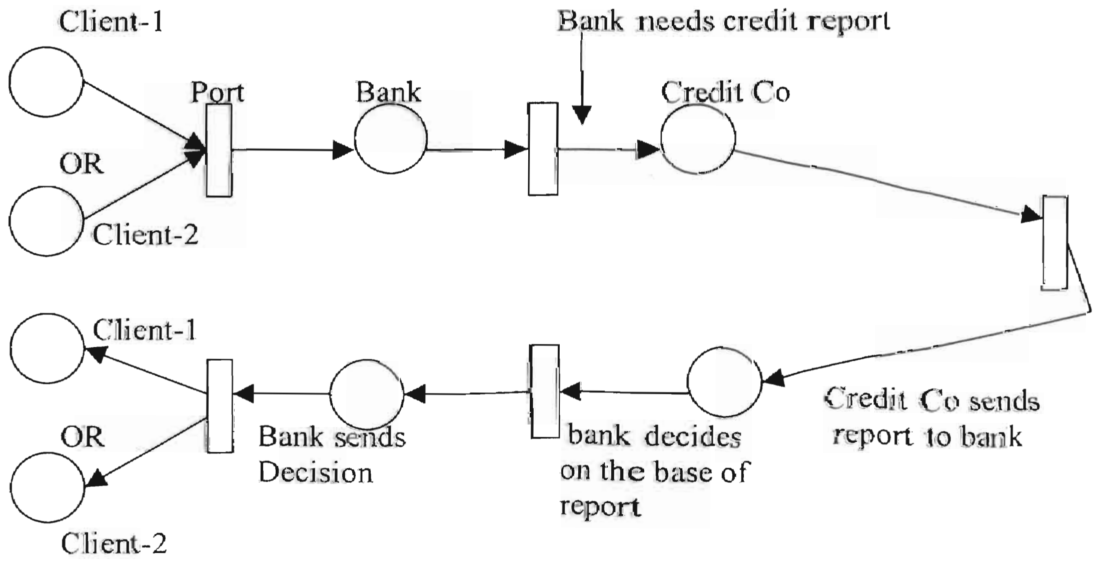


Fig. 11 Petri Net model of Carlot

4.3.2 Modeling of Methods

It is beyond the scope of this work to model every kind of computation found in a method. The bank decides whether to approve the loan request from the Carlot or to use the credit company's web services to obtain a credit history score for the customer. Decisions based on input conditions are modeled. It is also necessary to model the merging of two or more WSPNs. For example, the bank web service is modeled as two WSPNs. One net will model the situation where the bank makes the decision on the customer without recourse to the credit card company's services (the customer is along-standing customer of the bank for example). Another net will model the situation where the credit card company's services are needed. The merging of these two nets represents the bank's web services.

The Joining construction, which could be "OR" or " AND" form, is used for merging. In the OR form, the transition will fire if there is a token in any one of its input places and in the AND form there me be tokens in all of the input places before the transition can fire.

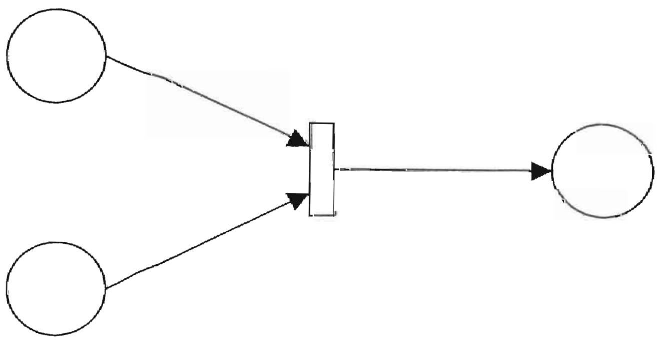


Fig. 12 Joining Model of Petri Net

PN splitting construction which could be "OR" or "AND". It is called conflict, choice or decision structure. In our case only "OR" model is used.

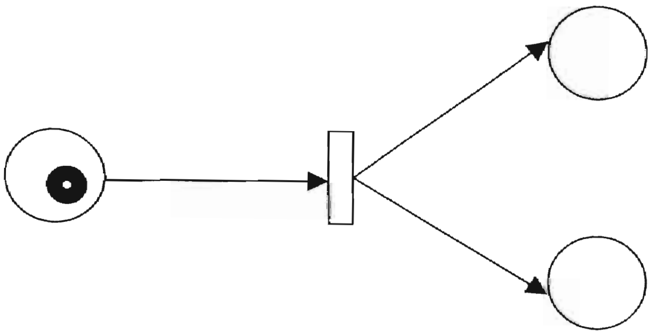


Fig. 13 Split Model of Petri Net

Methods may also contain loops. Looping (iteration) and merging construction is shown below:

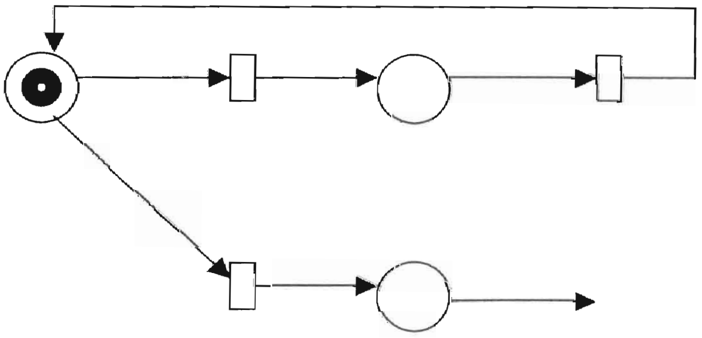


Fig. 14 Iteration Model of Petri Net

4.3.3 Petri Net Merging

Models of each individual web service are merged to obtain a system-wide view of a web business transaction. Although the individual models may display the desired properties of liveness, safeness and complete termination, the merged net may not display such properties. See the below example

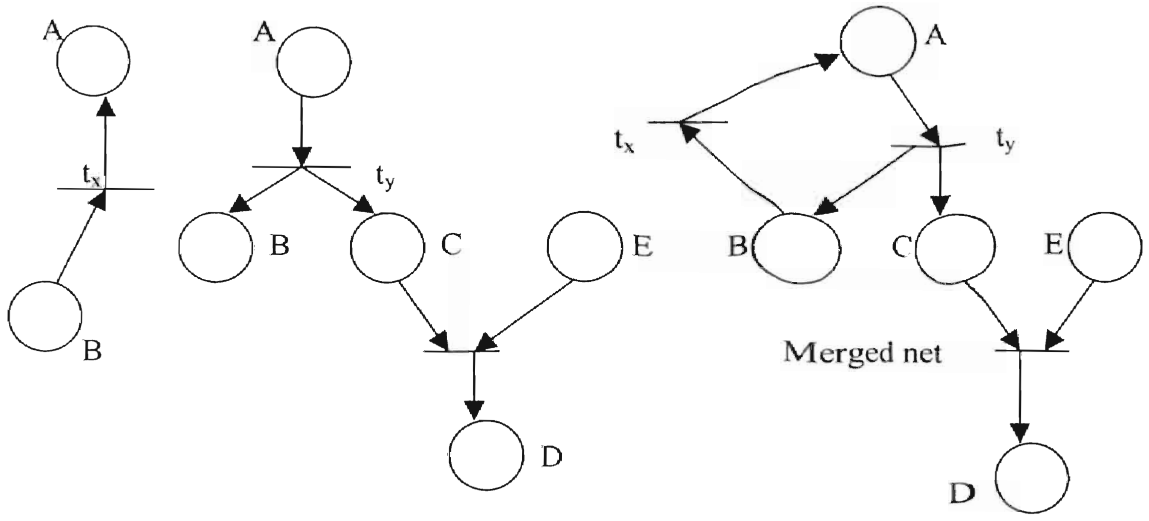


Fig. 15 Unsafe Merging

Both nets may be safe before merging, but after merging place C is not safe.

Merging Procedure

The WSPNs of each web service are merged to form a single control model of the entire transaction N_c. The WSPNs of each web service are merged such that the following applies.

- For each WSPN of a web service N'_C, with set of output places $P'_{oc} = m(O(t'))$ $\cup P'_{OR} = P'_{out} \cup P'_{OR}$, there exits a net N''_C with set of input places $P''_{IC} = m(I(t'')) \cup P''_{IR} = P''_{inp} \cup P''_{IR}$, such that there is a mapping from P'_{out} to a place $p \in P''_{inp}$ of net N''_c, unless $P'_{out} \subseteq m(P_{OA})$ or $P''_{inp} \subseteq m(P_{IA})$ of the plan net N_p. That is, there is a function

  $z: P'_{out} \rightarrow P''_{inp}$

- for each WSPN subnet $N^{|}_{c}$, with a set of output places $P^{|}_{OR}$ representing the final state of the transaction, there may be corresponding input production places $P^{||}_{IR}$ in net $N^{||}_{c}$ to which they are mapped.

Given a net $N^{|}$ with initial marking $M(p) = 1$, for all $p \in P^{|}_{IC}$ where $P^{|}_{IC} = P^{|}_{inp} \cup P^{|}_{IR}$ and a net $N^{||}_{c}$ with initial marking $M(p) = 1$, for all $p \in P^{||}_{IC}$ where $P^{||}_{IC} = P^{||}_{inp} \cup P^{||}_{IR}$, the initial marking M_{ini} of the merged net such that $P^{|}_{out}$ is merged with a place $p \in P^{||}_{inp}$ is defined as:

$$M_{ini}(P) = \{P^{|}_{inp} \cup P^{||}_{inp} - z(P^{|}_{out}) \cup P^{|}_{IR} \cup P^{||}_{IR} - P^{|}_{OR}\}.$$

The initial marking of the $N^{|}_{c}$ component of the merged net is as if $N^{|}_{c}$ were isolated. The initial marking of $N^{||}_{c}$ component of the merged net is partly derived from the output marking of net $N^{|}_{c}$.

Lemma 1: The merging of two live and safe refined WSPN's does not always result in a net, which is also safe and live.

Proof: If a loop is introduced in the merged net, this net may not be safe and live. In the simple example in Fig. 15 place C is not safe.

The types of nets that can be merged are therefore restricted so that the merged net is safe and live.

Merging Structure Constraint: Any circuit introduced into a net as a result of merging, must be a live and sage circuit, given and initial marking M_{ini} for the merged net.

To determine if a circuit is live and safe, it is necessary to extract the subnet associated with all transitions and places in the circuit. The subnet $N_{\text{sub}} = (P_{\text{sub}}, T_{\text{sub}}, F_{\text{sub}}, W_{\text{sub}}, M_{\text{sub}})$ associated with the circuit can be determined recursively as follows.

For transition t in the circuit, let p_{sub} be an input place of the transition:

a) if $p_{\text{sub}} \in O(t)$, then $P_{\text{sub}} = P_{\text{sub}} \cup I(t)$, $T_{\text{sub}} = T_{\text{sub}} \cup t$;

b) if a place $p \in P_{\text{sub}}$ such that $p \in O(t_i)$ and $t_i \neq t$, then $P_{\text{sub}} = P_{\text{sub}} \cup I(t_i)$, $T_{\text{sub}} = T_{\text{sub}} \cup t_i$, $M_{\text{sub}} = M_{\text{ini}}$ for $p \in P_{\text{sub}}$.

If $O(t) \subseteq P_{\text{sub}}$, this implies that the circuit is a live and safe circuit. In Fig.12 For transition t_y, $P_{\text{sub}} + \{ A, B\}$, $T_{\text{sub}} = t_x$, $O(t_y) = \{B, C\}$, and therefore, $O(t_y) \subseteq P_{\text{sub}}$ is not true. This circuit therefore does not satisfy the merging structure constraint given above.

Theorem 3: An WSPN N_c representing individual web services formed by merging two live and safe control WSPN's $N^{'}_{c}$ and $N^{''}_{c}$ such that the merging structure constraint is satisfied in the merged net, will be a safe and live WSPN given that the initial marking of the merged net is M_{ini}.

Proof Outline: Each net $N^{'}_{c}$ and $N^{''}_{c}$ is live by definition. Moreover, given the initial marking of the merged net and by the appropriate introduction of transition t_{reset}, it can be deduced that the merged net is live. The $N^{''}_{c}$ component of the merged net N_c can be made unsafe only if the $N^{'}_{c}$ component provides net $N^{'}_{c}$ more that one token at the merged places. Show that this is not possible and therefore the merged net is safe.

Theorem 3 is now extended to cover a complete web service transaction that includes all the individual web services. A complete transaction is a sequence with commences in the initial state of the transaction and terminates in the final state of the transaction with the

Initiator of the transaction getting the requested web services. By a recursive application of Theorem 3 we get Lemma 2.

Lemma 2: A merged complete web services transaction is a live and safe net.

Proof Outline: Follows from Theorem 3.

This leads to the following conclusion.

Theorem 4: given an WSPN N_c composed of merged nets that represent a complete transaction such that Theorem 3 is satisfied, then given an initial marking M_{init} where $M_{init}(p) = 1$ for all $p \in P_{IC}$ where $P_{IC} = P_{inp} \cup P_{IR}$, there exists a final marking M_{final} such that $M_{final}(p) = 1$ for all $p \in P_{oc}$ where $P_{oc} = P_{out} \cup P_{IR}$. The merged net obeys the properties of complete initialization and clean termination with respect to assembly parts and the production system.

Proof Outline: First we prove the property of clean termination. From Lemma 2, transition t_{reset} is live in the merged net N_c. Therefore, we deduce that $M(p) = 1$ for all $p \in P_{out}$. Next we show that in the final marking there does not exist a place p_y such that p_y is some place excluding places $\{ P_{inp} \cup P_{out} \cup P_{IR}\}$. From this result that p_y cannot also be one of places P_{inp}. We can therefore conclude that there exists $M \in R(N_c, M_{init})$ such that $M(p) = 1$ for all $p \in \{P_{out} \cup P_{IR}\}$ and $M(p) = 0$ for all $p \notin \{ P_{out} \cup P_{IR}\}$. However, a situation may arise whereby transition t_{reset} could fire before the net has reached marking $M(P_{IR})$. In other words, there could exist a marking $M(p_x)$ where $p_x = \{ P_{out} \cup P_x\}$ and $P_x \neq P_{ir}$. We next show that there exists a marking $P_{out} \cup P_{IR}$ that is reachable from $P_{out} \cup P_x$. A similar approach is used to prove the property of complete initialization.

The use of computer-aided tools is necessary for practical application of Petri nets. Safely modifying workflow logic in real-time requires validation of system changes before actual system deployment [19].

Petri net modeling of the WSDL gives very descriptive information of the web service. With the help of Petri nets possible deadlocks and safeness in the system can be checked. Moreover, the WSPN models both WSDL and the methods in a web service. The detailed picture of the entire system can therefore be analyzed.

4.3.4 Petri Net Modeling of Methods:

At Carlot we have choice that if payment is cash then done otherwise send to Bank for financing.

The Carlot transaction is modeled with OR model to represent two choices. One arc shows that payment is made by cash and if this is not the case then customer's data will go to Bank for loan approval and this is modeled with second arc of OR model.

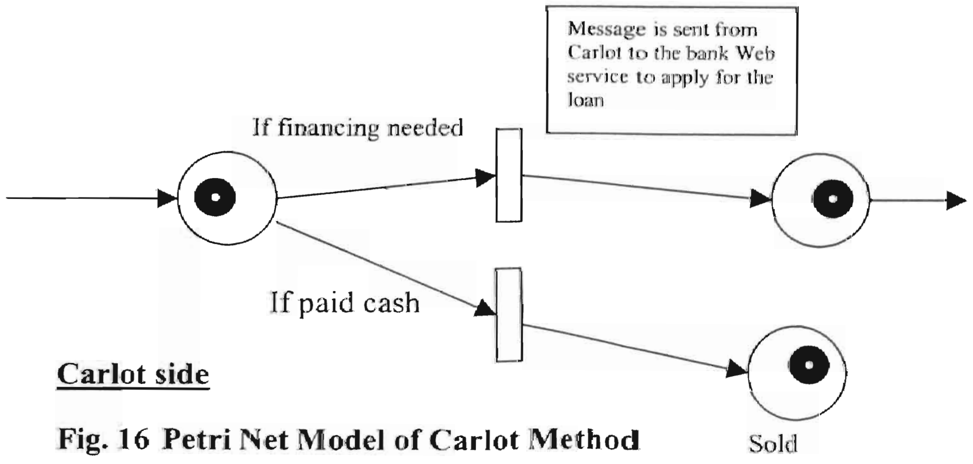

Message is sent from Carlot to the bank Web service to apply for the loan
If financing needed
If paid cash
Carlot side
Sold

Fig. 16 Petri Net Model of Carlot Method

Bank can send two kinds of replies i.e. approved or declined. Similar to Carlot these both replies are modeled with the help of OR model. As is shown below:

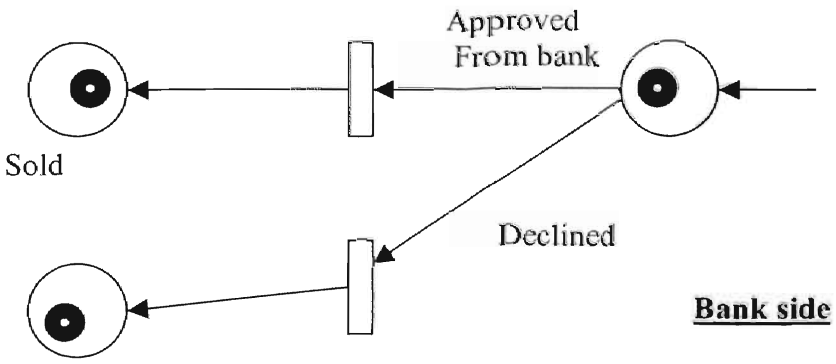


Fig. 17 Petri Net Model of Bank Method

To model the complete set of transactions at the carlot, the outgoing and incoming messages are merged. After merging of both we get following model:

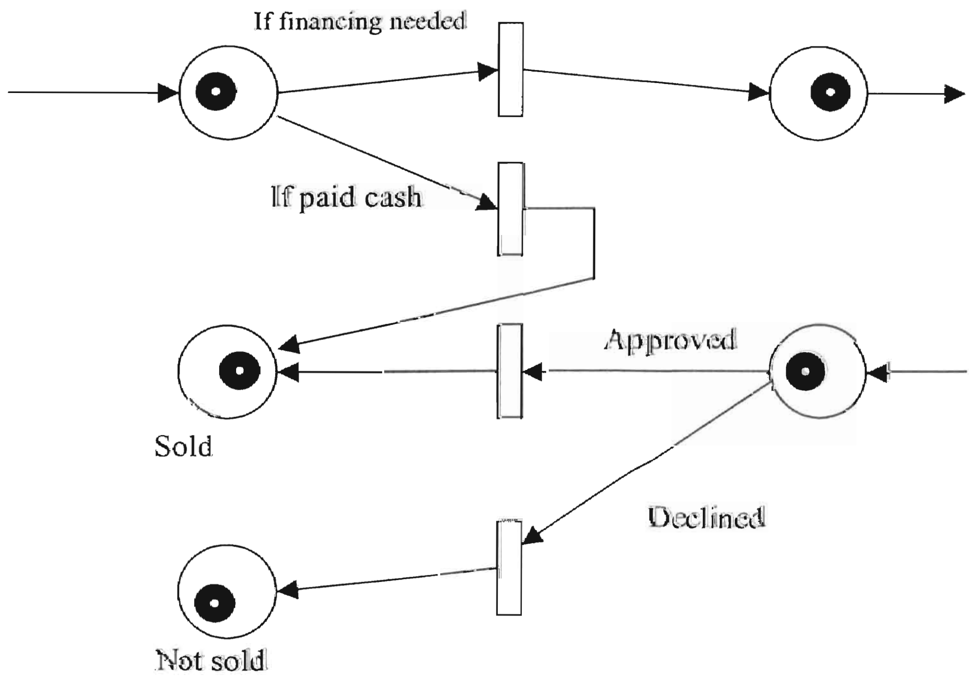


Fig. 18 After merging of both Carlot and Bank methods

Similarly the methods in Bank and in Credit Co. can also be modeled.

However, these only represent the individual web services. To get a view of the entire transactions requires a model of the web transactions at all three sites. This model will be generated and presented as part of the implementation.

4.4 Limitation to this approach

A major problem with formal modeling is scalability. As the number of sites increases, the modeling does not scale well. The usual approach to this problem is to hierarchical modeling of systems. In our case as shown above, we refine transitions for scalability.

Chapter 5

Methodology and Implementation

5.1 Petri Net model

The Petri net modeling captures Web sites, which use Web Service, such as the Carlot website in our example. The Web Services provide services to the Carlot and WSDL is the description of the Web Services. Reachability analysis of the Petri Net yields properties of the Petri Net such as deadlocks and safeness.

One web site's method may use many Web Services. Before and after the calling of Web Services there is a computational work to complete the initiated business process. The called Web Service, if needed can also invoke another Web Service to get required information and so on. In this way the whole process of Web Services is like a big complicated web.

Methods are written to implement the company's business process computation. They include conditional statements, and calls for other Web Services methods. The conditional statements may include "if-else" statements. One "if-else" statement can have some other "if-else" statements nested in it. Although not considered in this thesis, other programming constructs such as loops may also be included.

To draw the Petri net of Web Services, global information of the web services is required. Each web site is only aware of its own methods and other web services it calls. The entire web service transaction can therefore be modeled by merging the models of individual transactions. Each line of a method file is scanned. The sequence of all Web Services calls is saved. All the possible paths from the beginning of the web services transaction are traced. As conditional statements increases the number of possible paths increase and Petri net becomes complex.

When a method of a Web Service needs to be invoked, its respective WSDL file is traced. A WSDL file has all the information required to communicate with Web Services. For example, the list of sending parameters, detail of returning response, allowed Internet protocol, name of Web Service and its method along with its address are all contained in a WSDL file. Like a method, each WSDL file is scanned. WSDL's information is enclosed in different fixed elements. It includes input and output messages which are collected from <types> element, method's name from <binding> element, transport protocols (SOAP, HTTPGET, HTTPPOST), Web Services name, and its address from <service> element. WSDL's format is fixed; the detail of WSDL is given in chapter 3.

The complete Petri net represents all the possible execution paths of the whole system, in which a web site's method is followed by the WSDL and then the Web Service's method. In a correct web transaction, the path will come back to the calling method in the same sequence.

A high level global view of all participating Web site, Web services and WSDL file is sketched in fig 19.

As each web site is aware only of its local computation, the Petri Net model of the web service at an individual site is first constructed fig 20.

The generated Petri Nets of individual web services are merged to create the Petri Net model of the entire web transaction fig 21. Our approach also allows for the hierarchical modeling of web transaction.

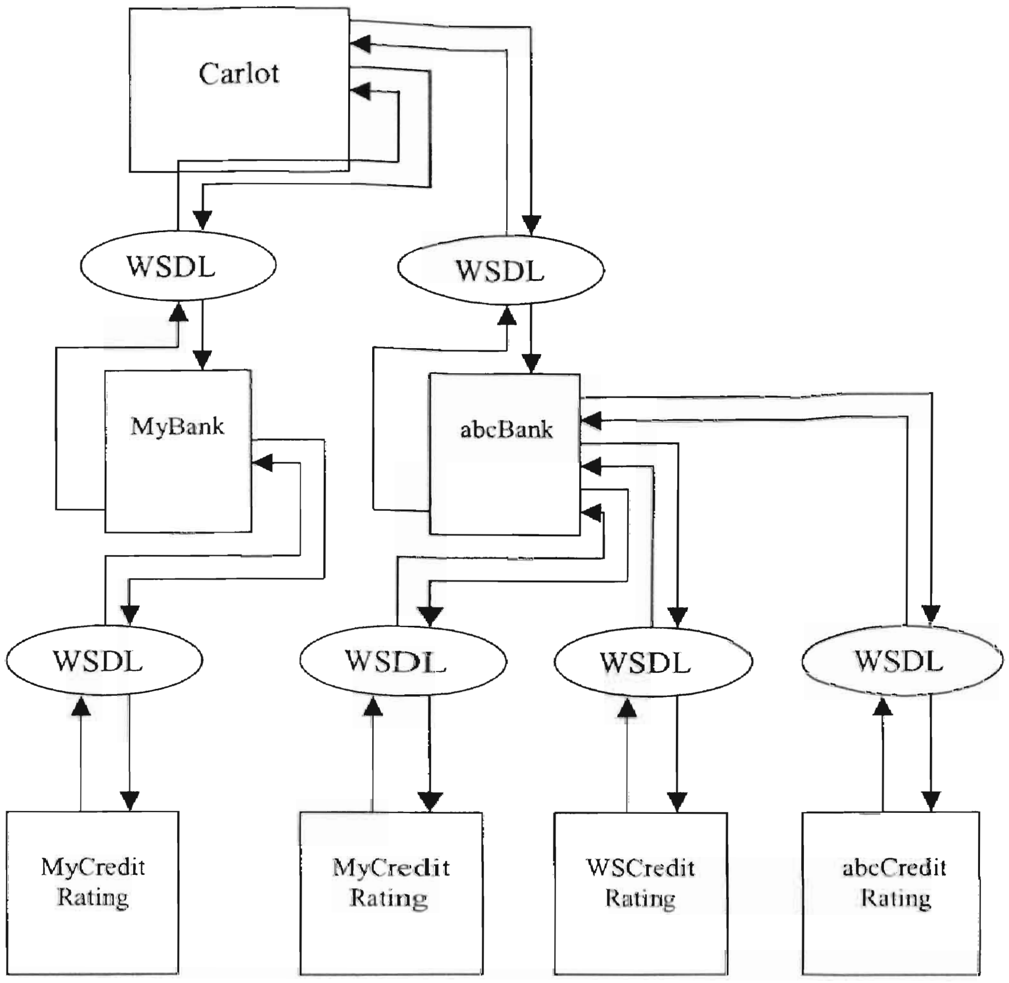


Fig. 19 Abstract diagram of example

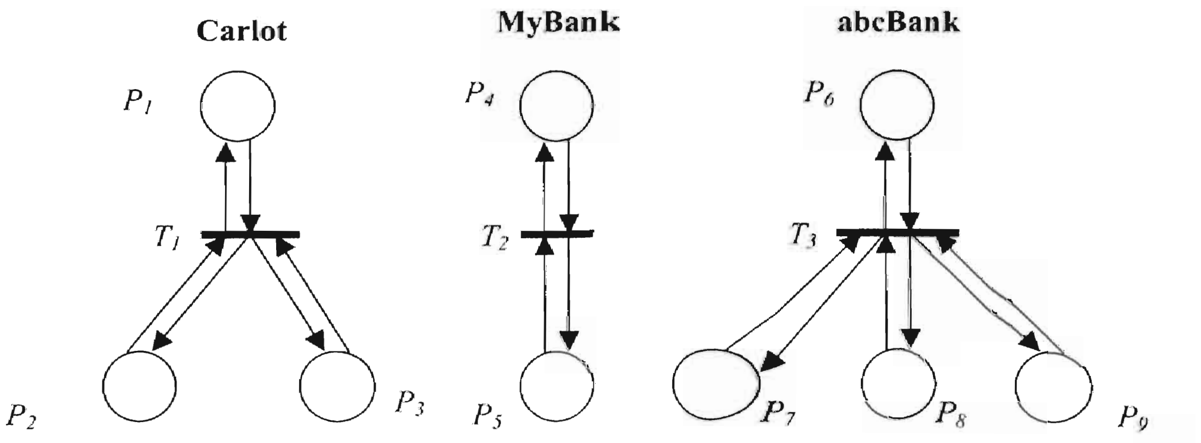
The results from the remote web site are turned to the Carlot method and finally to the user.

In Web Services a Carlot invokes the web services of two Banks to get finance for its customers and on the basis of the cheapest interest rate returned it selects one of them. Out of these two banks one calls one Credit Co's Web Services and the other bank calls three Credit Co's Web Services to check the credit history of the customers. After getting a credit score from the credit companies, the bank decides whether to approve the loan and the interest rate to be charged. The results from the banks are returned to the Carlot. This scenario is depicted in fig 23 .

Each Web Service is accessed through its particular WSDL file. The code for this Web services application is shown in the appendix A. Our modeling software reads all the Web Service's methods and their WSDL files, and derives the Petri net model of the entire business transaction.

In the example below, the individual higher level Petri net diagram of Carlot, Banks and Credit Rating Companies are shown fig 20. In the Carlot transaction, place P_1 represents the user who is applying for a loan. The transition represents the methods at the Carlot website. This method calls two web services, MyBank and abcBank whose ports are represented by place P_2 and P_3. The Two banks, the abcBank is invoking three credit companies while the MyBank Web Service is communicating with only one Credit Company. But the Credit companies are not invoking any further Web Services.

Fig. 20(a) Petri net modeling of Web Services



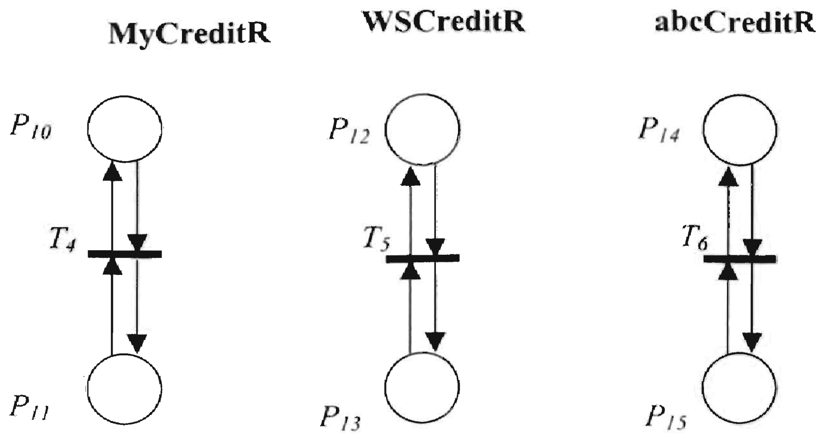


Fig. 20(b) Petri net modeling of Web Services

An abstract global view of the entire system is obtained by merging these Petri Nets as shown in figure 21.

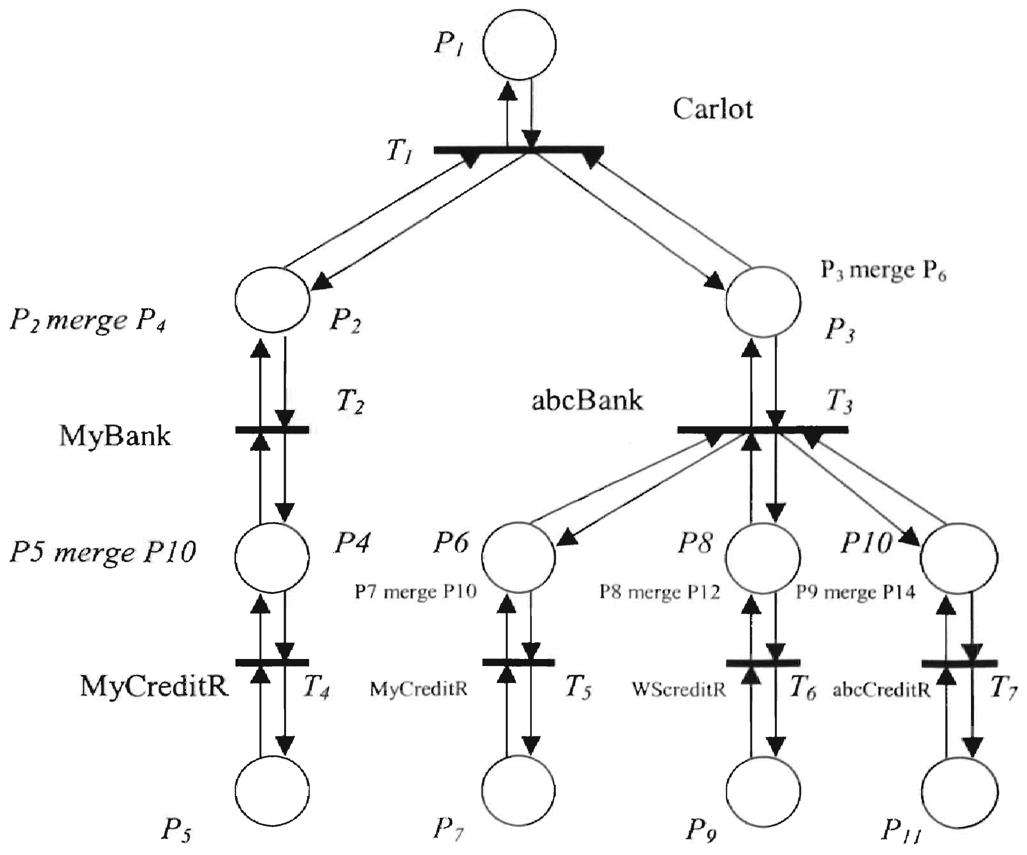
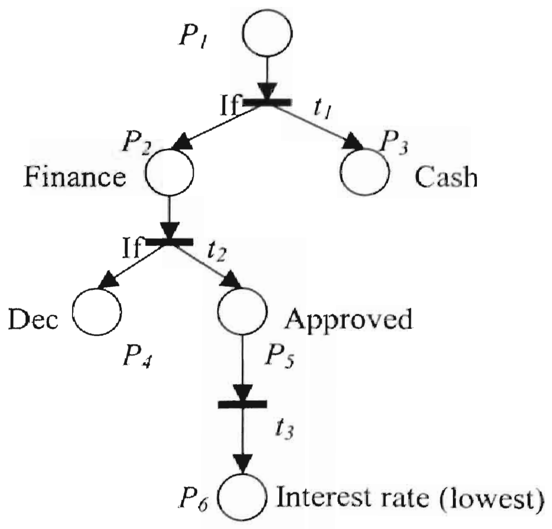


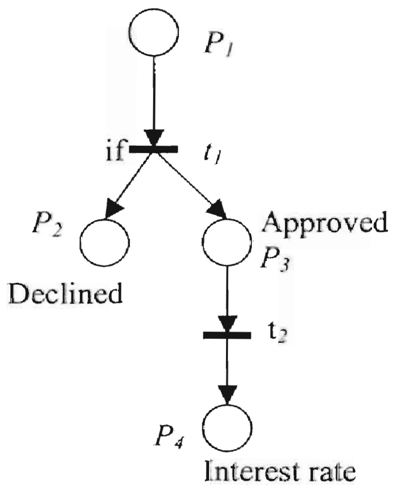
Fig. 21 Merged global view of the entire system

The detailed method of each site is then constructed. This is modeled as a Petri Net refinement. For example, the detailed model of the Carlot method and others are shown below. In Theorem 3 in chapter 4 we showed that Petri Nets can be merged to form a composite Petri Net that is safe and deadlock free. Based on this theorem these nests are then merged to create the model of the entire web transaction.

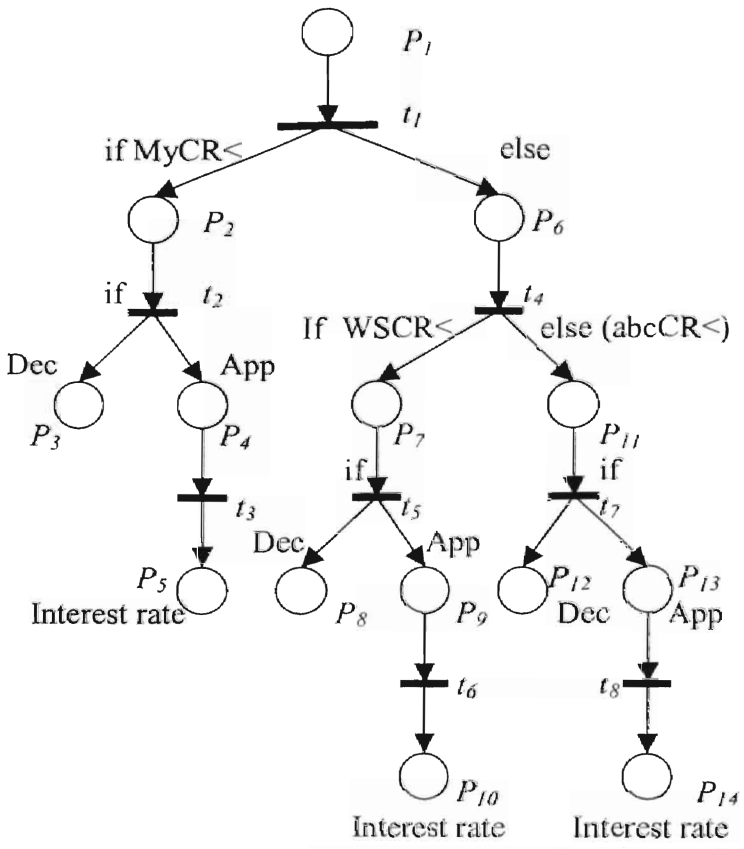
The Petri net of Web Services methods can also be sketched. In a method If-else statements provide a number of execution paths, and these are modeled in the Petri net. The Carlot's first if-else statement decided whether to invoke web services on the basis of payment mode. This is shown by transition t_1 below in fig 22(a). If the Web service is used then on the returned results, declining or approving shown by t_2, from banks it accepts the lowest interest rate offering bank mention by t_3. Similarly, the Petri net of the methods of Web Services are sketched. These Petri net models are shown in fig 22.



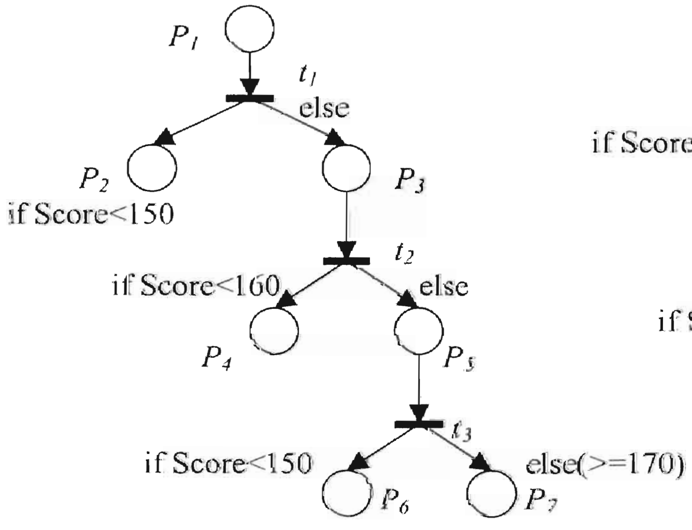
(a) Carlot Web site method



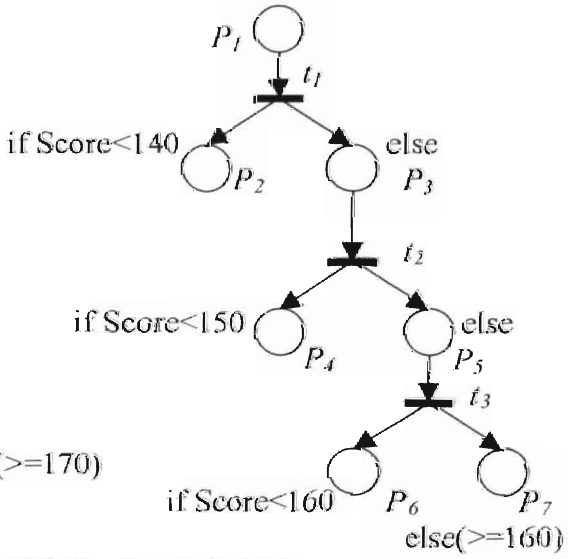
(b) MyBank Web Service method



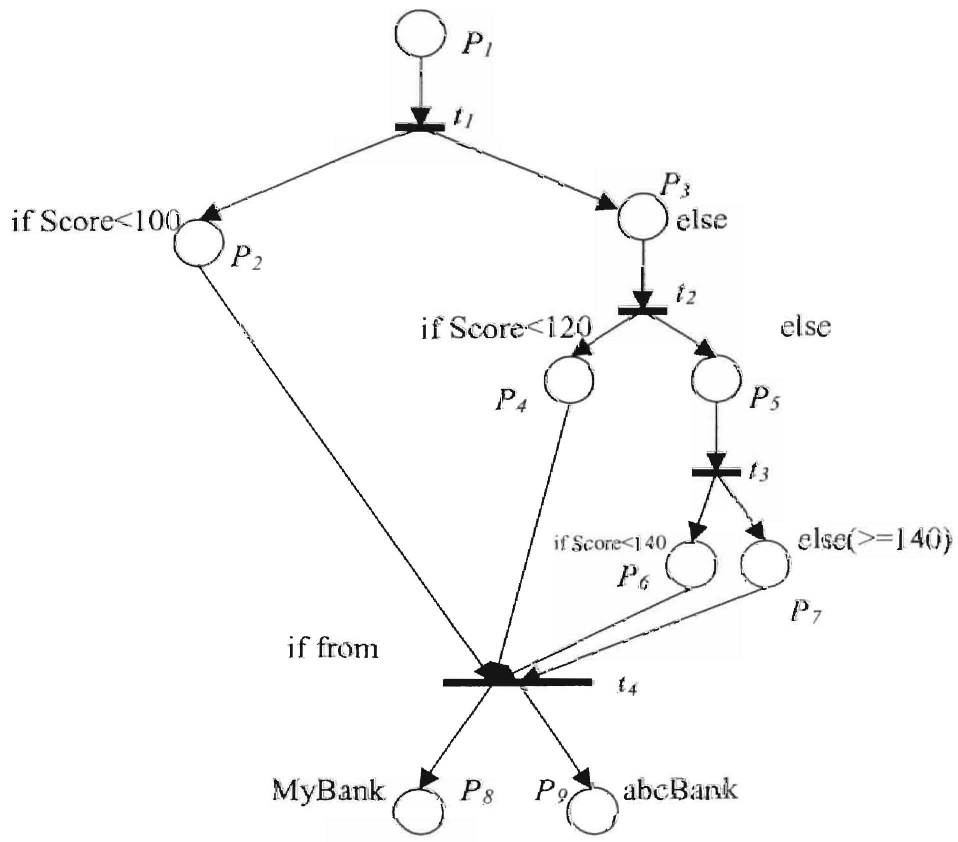
(c) abcBank Web Service method



(d) abcCreditR Web Service method



(e) abcCreditR Web Service method



(f) MyCreditRatings Web Service method

Fig. 22 Petri Net models of Web site and Web services Methods

The Petri Net derived by the tool is described as:

{P0,T0,P1},{P1,T1,P2},{P2,T2,P3},{P3,T3,P4},{P4,T4,P5},{P5,T5,P6},{P6,T6,P7},

{P7,T7,P8},{P8,T8,P9},{P1,T1,P10},{P10,T9,P11},{P11,T10,P12},{P12,T11,P13},

{P13,T12,P14},{P14,T13,P15},{P11,T10,P16},{P16,T14,P17},{P17,T15,P18},

{P18,T16,P19},{P11,T10,P20},{P20,T17,P21},{P21,T18,P22},{P22,T19,P23},

{P23,T20,P24},{P24,T21,P25},{P25,T22,P26},{P0,T0,P27}.

Our tool does not generate a graphical representation of the Petri Net model. The Petri net can also be shown graphically in which different abbreviations are used to represent the different states of methods and WSDL. As *P* is for Places, which is representing message data for methods and PortTypes for WSDL. *T* is for Transitions, which shows computational work in methods and Service (ports) in WSDL, and Arcs (→) are representing bindings (protocols) for WSDL while in methods incoming Arc to transition is incoming data for computation and outgoing Arc from transition is outgoing data after computational work to the next state. Tokens in both cases are data. Transition can have one or more input Arcs and similarly one or more than one output Arcs. In methods varying number of input and output Arcs depends upon the implementation of business strategy. Starting and final places are observed to implement the Reachability analysis. The graphical Petri net representation of above given example is shown in fig 23.

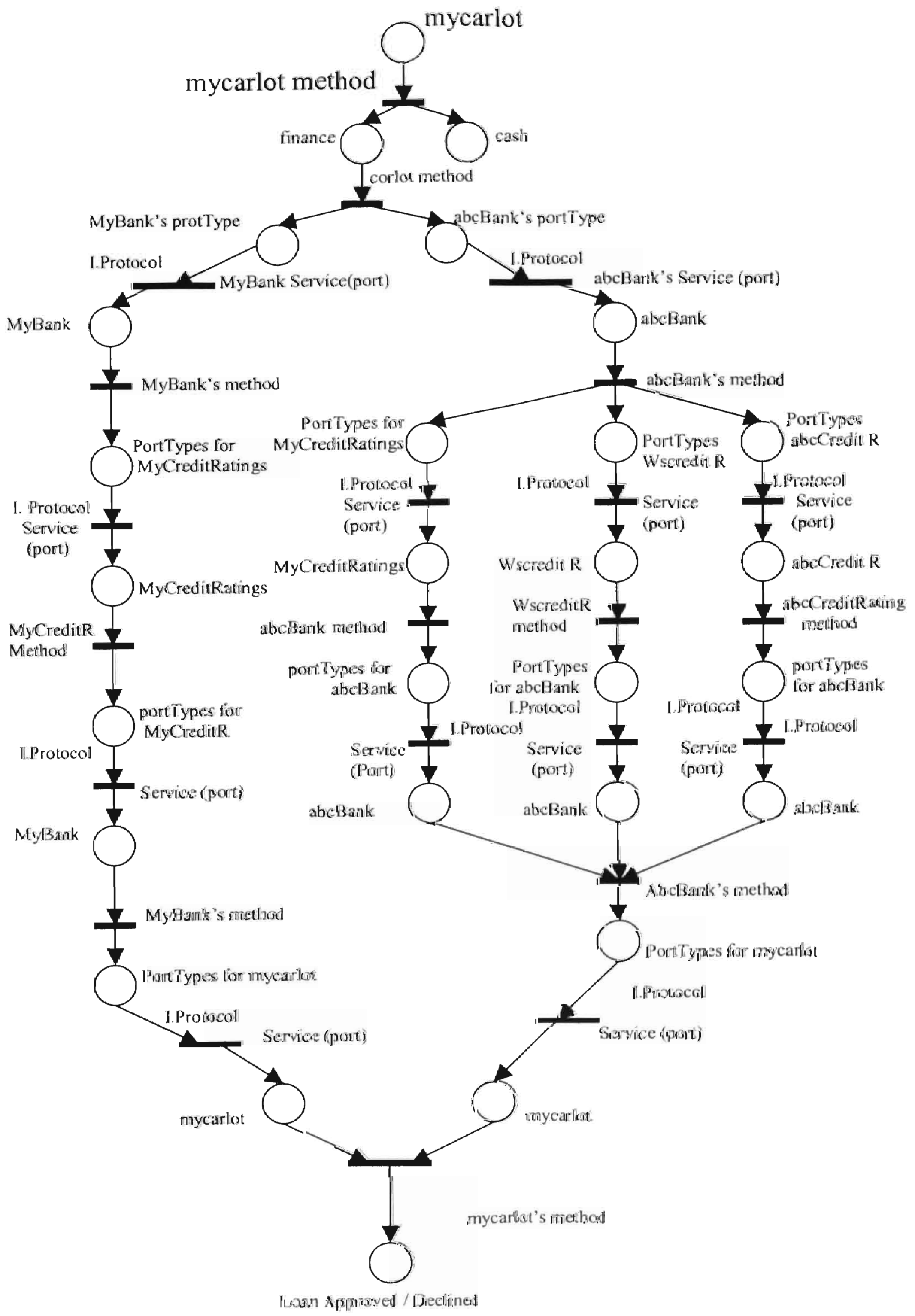


Fig. 23 Graphical Petri net representation of Carlot and invoked Web Services

A dynamic approach to sketch the Petri net is also implemented. In this approach on each step of program execution a token is passed to the next state of program and respective transition (T) number along with place (P) number is updated. As the program executes a token moves representing a new state of the system. At the end of program we get whole execution path, from beginning to end, in the form of a Petri net.

5.2 Modeling Tool for Web Services

To capture this whole scenario a software-modeling tool is developed. Its first input in the "*main( )*" is method's file name, which is invoking Web Services. This software checks each line of method, finds the "if-else" statements and calls for Web Services. Prior to access the Web Service's method it needs WSDL file name as another input to have communications information of that Web Services as mentioned above. To read WSDL file "*main( )*" calls "WSDL( )" function. "WSDL( )" reads a WSDL file line by line and saves the information of the Internet Protocol being used by that Web Service, name of Web Service and method, which is going to be invoked. Then it requires the respective Web Services method file. To read Web Service's method file "*WSDL( )*" calls "*ReadWSMethod( )*". Like first method's file this file is checked line by line, too. If Web Service is calling another Web Service then again it needed respective WSDL file and that called Web Services method, and so on. In this way "*WSDL( )*" and "*ReadWSMethod( )*" calls each other alternatively and required information is collected.

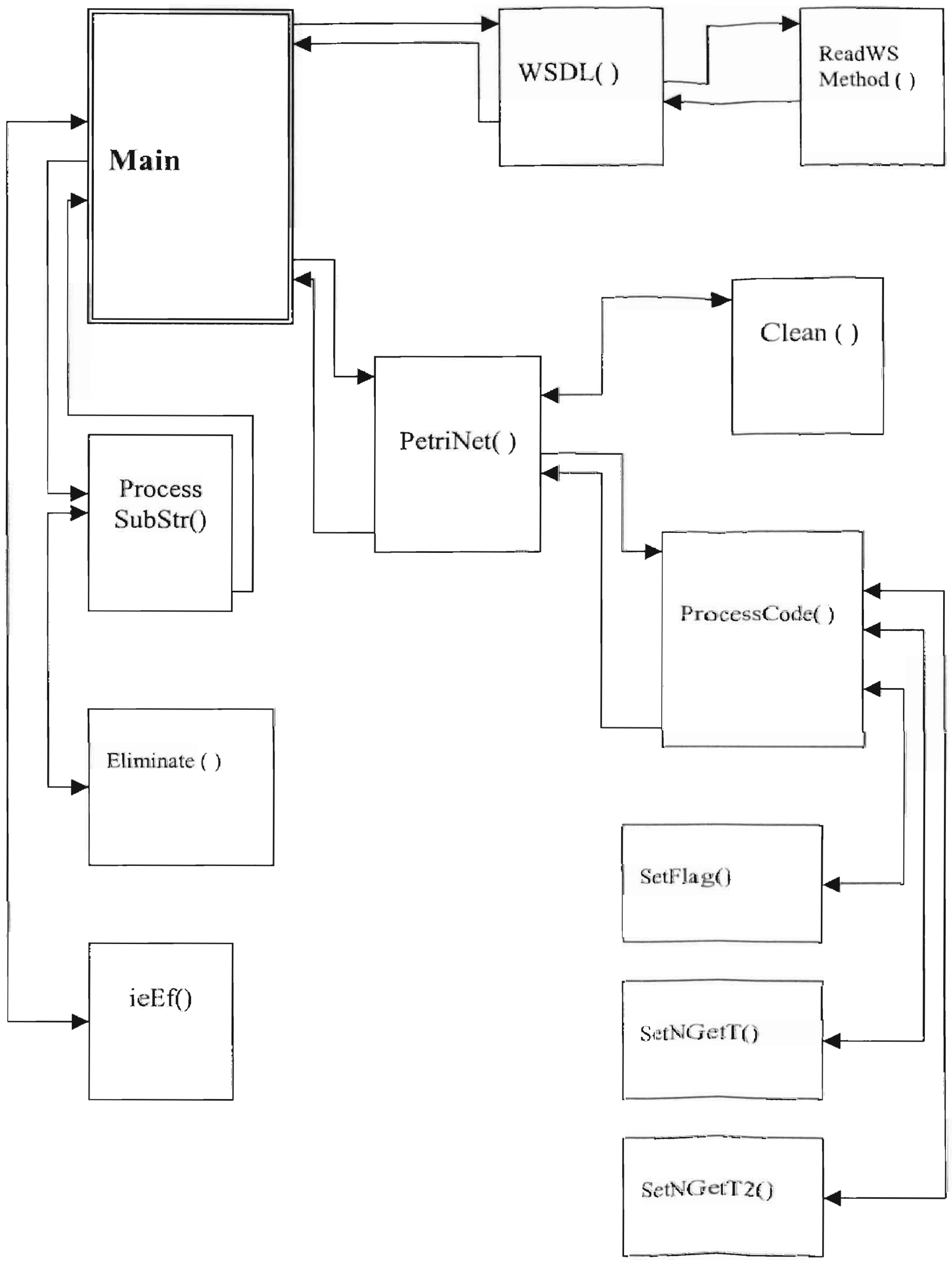


Fig. 24 Abstract Diagram of Software

All method files are checked for all their "if-else" statements. When no more Web Service is called then the program follows the return messages and finally comes back to its first method file. Then the rest of this method file is checked, for another Web Service call. If there is another web service call, then the required information is obtained as described above and the sequence of computation is saved. After reading all WSDL files and Method files, the program returns to "*main()*" for further processing towards Petri net sketching of Web Services. The sequence number of file reading, with their name, is saved. If the file method is calling other file the sequence number is labeled "C" but when this method is returning then the sequence number is labeled "R". Between two file numbers with their label a "-" sign is inserted to differentiate one file from the other. The different particular letter is assigned to the particular conditional statement. For "If" statement "i", for "Else" statement "e", for "Else If" statement "E" and for "End If" the "f" are saved in a global array. In this global array all required information are stored and ready for further processing. At this point control is in the "*main()*" which sends this global array's data to the "*ieEf()*" to check if it is holding only valid letters otherwise returns error message and program comes back in "*main()*".

After verifying the validity of letters in array we need to match the "if", "else", "else if" and "end if" statements in the methods which is needed to determine the flow of execution and necessary for Reachability analysis. To perform this matching the sub string having conditional statements data string of one method file is passed to the "*ProcessSubStr()*". This function searches first "f" moving pointer forward in the sub string which show "End If" statement in the method and then pointer moves backwards to find "i" which show "If" statement. Then portion of the sub string is passed to

"Eliminate( )" from the *"ProcessSubStr( )"* to replace them with equal number of "*" characters, which shows that this portion of the sub string of the method is perfectly matching. Then program comes back in *"ProcessSubStr( )"* to match next "If" and "End If" statement and goes in *"Eliminate( )"* to confirm the matching. As each "If" statement must match with "End If" statement, if there is any mismatch then program show error in the method. This mismatching also affects Reachability. At the end of *"ProcessSubStr( )"* all "i", "e", "" and "f" which are involved in invoking other Web Services are converted in to "*" and those which are involved in invoking other Web Services are left but their matching is verified. When program comes back from *"ProcessSubStr( )"* to the *"main( )"*, we get confirmation of matching of "If" and "End If" statements. Now the data string is ready to pass to *"PetriNet( )"* for further processing and representing in Petri Net form.

The resulting array from *"ProcessSubStr( )"* is passed to *"PetriNet( )"* from *"main( )"*. The *"PetriNet( )"* sends data array to *"Clean( )"* to remove all "*" characters from the string and after cleaning data string comes back in *"PetriNet( )"*, which sends this cleaned data string to *"ProcessCode( )"* for final processing. The *"ProcessCode( )"* with the help of three functions store the data in Petri net form. First of those three functions is *"SetNGetT2( )"*, which process the if-else statements which are involved in invoking Web Services. Second is *SetNGetT( )"* which handles calling (C) and returning (R) methods and third function *"SetFlag( )"* keep the record of forward and backward movement of execution. When program returns from the *"ProcessCode( )"* to the *"PetriNet( )"* then *"PetriNet( )"* have enough information to present the Petri net of Web Services and run Reachability analysis. Different abbreviations are used to represent the different states of methods and WSDL. As *P* is for Places, which is representing message data for methods and PortTypes for WSDL. *T* is for Transitions, which shows computational work in methods and Service (ports) in WSDL. In each curly bracket first *P* is starting place and second *P* is next place after performing transition.

5.3 Reachability

The Reachability analysis is part of our Petri net modeling software. It is applied at the end of Petri net sketching. To perform Reachability analysis, the program takes two inputs - one representing the beginning place number from where the Reachability analysis will start and other the ending place number at which Reachability analysis will end. Validity of starting place number and ending place number is checked, in other words, at the initial state of the transaction there is a token in place P_0 and at the end of the web services transaction, the final state will be defined by a token in place P_{26} only. On giving range of starting place number and ending place number for Reachability analysis, program reads a global array, which have the whole Petri net data and verifies the given ranges of places. If both beginning and ending ranges are valid, it fills one array with zeros (0) equal to all places in between both given ranges, which shows prior to execution state. Then it starts reading all Petri net steps (places) one by one and on reading each step it turns next zero into one which represents the movement of token from one place to another and previous one is turned into zero which shows that there is no token anymore. In this way it goes through all Petri net steps till reaches the last step. If the net is safe, at each step there will be only one token in at least one place while rest places will be showing zero (no token) and at the end the token will be in the last place only. If the net is live, all the places will eventually receive at least one token.

In our example, the Reachability analysis was applied on the whole Petri net and its result are shown in the next fig 25.

```
Reachability testing:

Enter starting state number (-1 to exit): 0

Enter final state number : 26


Start State : 0

Final State : 26

{0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 }

{1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 }

{0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 }

{0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 }

{0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 }

{0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 }

{0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 }

{0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 }

{0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 }

{0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 }

{0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 }

{0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 }

{0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 }

{0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 }
```

```
{0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 }
{0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 }
{0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 }
{0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 }
{0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 }
{0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 }
{0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 }
{0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 }
{0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 }
{0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 }
{0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 }
{0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 }
{0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 }
{0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 }
```

Fig. 25 An output of Reachability analysis

Chapter 6

Conclusion

In this thesis we present a model and a tool for representing Web services. Petri Net formalism has been used to model web services and a C++/C tool has been developed to generate a Petri Net representation of distributed web services and their WSDL. The large number of distributed web services and corresponding distributed clients make the modeling of web services a very complex task. The hierarchical modeling approach we present in this thesis is designed to deal with the complexity of such systems.

The developed tool generates for the user the sequence of all execution steps performed to complete a business process. This is achieved by invoking many Web services where each Web service is accessed through its WSDL file and each web service does some computational work. Such a model of the complete execution of tasks shows how the intermediate and final results in the business transaction are being compiled and what options are feasible during the execution of the business process at different steps. The model is obtained by merging the representations of the individual web services. Our approach allows the representation of web services in an abstract high-level form, which can then be refined to show more detail. The model generated using the software tool shows the flow of programs in which many programs are involved and invoking each other.

Reachability analysis allows the evaluation of the Petri Net model for system deadlock and enables the user to check if the Petri net is safe and live.

6.1 Future work

There are a number of areas for future work. The tool can be enhanced to provide a graphical representation of the Petri Net representation of Web services.

A graphical representation of the Reachability tree can also be incorporated in the tool.

Our tool currently generates Petri Net models of a method's conditional statements only. Future work will include Petri Net modeling of more complex programming constructs. Due to the complexity of the web, a more compact representation is desirable. Other Petri Net formalisms such as colored Petri Nets may result in a more compact Petri Net representation. Although, the work reported here forms the underlying basis of Web services modeling, to implement such a system in a real internet web-service environment would require a lot of additional work.

Other issue such as modeling security has not been touched in this work and is an area for future research.

The modeling of accessing the UDDI and then selection of Web Services out of hundreds, which will be thousands very soon, can also be added to this work.

Chapter 7

References

7.1 Bibliography

1 Web Services and UDDI, IBM Corporation. http://www.ibm.com/services/uddi.

2 Kreger, H., 2001, "Web Services Conceptual Architecture (WSCA 1.0)", *IBM Software Group*, www-4.ibm.com/software/solutions/webservices/resources.html

3 Web Services, IMB Corporation. http//www.xml.com/pub/a/2001/04/04/webservices/index.html?page=3#ibmtut

4 Technical Report: *SOAP Version 1.2 Working Draft.* http://www.w3.org/TR/soap12

5 Technical Report: Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl

6 Universal Description, Discovery and Integration. http://www.uddi.org

7 Lambros, P., Schmidt, M., Zentner, C., 2001, "Combine Business Process Management Technology and Business Services to Implement Complex Web Services", www- 4.ibm.com/software/solutions/webservices/resources.html

8 *McGregor, C.; Kumaran, S.* "Business process monitoring using web services in B2B e-commerce", Parallel and Distributed Processing Symposium, Proceedings International, IPDPS 2002, Abstracts and CD-ROM, 2002, pp. 219 -226

9 *Rettberg, A.; Thronicke, W.*, "Embedded system design based on web services", Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings, 2002, pp. 232 -236

10 IBM Web Services Architecture Tam. *"Web services architecture overview. The next stage of evolution for e-business"*, IBM Technical Document, Web Architecture Library, 2000.

11 D.G. Schwartz. "*Cooperating Heterogeneous Systems*". Kluwer Academic Publisher, 1995.

12 *Aoyama, M.; Weerawarana, S.; Maruyama, H.; Szyperski, C.; Sullivan, K.; Lea, D.* "Web services engineering: promises and challenges", Software Engineering, 2002. ICSE 2002. Proceedings of the 24th International Conference, 2002 pp. 647 –648

13 IBM, Web Services: Taking e-business to the Next Level, White Paper 2000, http://www-3.ibm.com/services/uddi/papers/e-businessj.pdf

14 Szyperski, C., Component Software, Addison Wesley, 1988.

15 Hatashima, T.; Yokozeki, D.; Suzuki, M.; Tokumaru, K.; Miyata, S.; Kawasaki, R.; Kato, J.," WebServices processing platform - eCo-Flow", Applications and the Internet (SAINT) Workshops, 2002. Proceedings. 2002 Symposium on , 2002, pp. 186 – 195

16 C.A.Petri, " Kommunikation mit Automaten." Bonn: Institute fur Instumentelle Mathematik, Schriften des IIM Nr. 3, 1962. Also, English Translation, "Communication with Automata." New York: Griffiss Air Force Base. Tech. Rep. RADCTR-65-377, Vol. 1, Suppl. 1, 1966.

17 Murata T, "Petri Nets: Properties, Analysis and Applications", Proceedings of the IEEE, Vol. 77, NO. 4, April 1989. pp. 541-580

18 Thomas J.P, Nissanke N, Baker K.D, " A Hierarchical Petri Net Framework for the Representation and Analysis of Assembly", IEEE Transactions on Robotic and Automation, Vol. 12, NO. 2, April 1996. pp. 268-279.

19 Faul B.M, "Using Petri net web services to build dynamic and adaptive service oriented workflows", Silver Falls Software, Inc.

20 Leymann F, Roller D, Schmidt M.T, " Web services and business process management", IBM Systems Journal- New Development in Web Services and E-Commerce, Vol. 41, NO. 2, 2002.

7.2 Appendix

(A) A Carlot program

```
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
'Carlot program takes customer's, vehicle's and payments information. If customer needs financing then it
'invokes two Bank's Web 'services to get financing and after getting response from the banks it checks
'which bank is charging less interest rate and forwards 'that to customer for finalizing the deal.
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Public Class WebForm1   'input boxes to collect required information
  Inherits System.Web.UI.Page
  Protected WithEvents TextBox13 As System.Web.UI.WebControls.TextBox
  Protected WithEvents TextBox19 As System.Web.UI.WebControls.TextBox
  Protected WithEvents Process As System.Web.UI.WebControls.Button
  Protected WithEvents Cash As System.Web.UI.WebControls.RadioButton
  Protected WithEvents Finance As System.Web.UI.WebControls.RadioButton
  Protected WithEvents lblBankReference As
System.Web.UI.WebControls.Label
  Protected WithEvents lblCustomerNo As System.Web.UI.WebControls.Label
  Protected WithEvents RadioButton1 As
System.Web.UI.WebControls.RadioButton
  Protected WithEvents RadioButton2 As
System.Web.UI.WebControls.RadioButton
  Protected WithEvents Button1 As System.Web.UI.WebControls.Button
  Protected WithEvents TextBox9 As System.Web.UI.WebControls.TextBox
  Protected WithEvents lblInterestRate As
System.Web.UI.WebControls.Label
  Protected WithEvents lblName As System.Web.UI.WebControls.Label
  Protected WithEvents txtFirstName As
System.Web.UI.WebControls.TextBox
  Protected WithEvents txtLastName As System.Web.UI.WebControls.TextBox
  Protected WithEvents txtStreet As System.Web.UI.WebControls.TextBox
  Protected WithEvents txtCity As System.Web.UI.WebControls.TextBox
  Protected WithEvents txtState As System.Web.UI.WebControls.TextBox
  Protected WithEvents txtZip As System.Web.UI.WebControls.TextBox
  Protected WithEvents txtSSN As System.Web.UI.WebControls.TextBox
  Protected WithEvents txtHomePhone As
System.Web.UI.WebControls.TextBox
  Protected WithEvents txtVehMake As System.Web.UI.WebControls.TextBox
  Protected WithEvents txtVehModel As System.Web.UI.WebControls.TextBox
  Protected WithEvents txtVehYear As System.Web.UI.WebControls.TextBox
  Protected WithEvents txtNoOfPayments As
System.Web.UI.WebControls.TextBox
  Protected WithEvents txtFinanceAmount As
System.Web.UI.WebControls.TextBox
  Protected WithEvents lblToken1 As System.Web.UI.WebControls.Label
  Protected WithEvents lblToken2 As System.Web.UI.WebControls.Label
  Protected WithEvents Label11 As System.Web.UI.WebControls.Label
  Protected WithEvents Label10 As System.Web.UI.WebControls.Label
  Protected WithEvents Label9 As System.Web.UI.WebControls.Label
  Protected WithEvents Label8 As System.Web.UI.WebControls.Label
  Protected WithEvents Label7 As System.Web.UI.WebControls.Label
  Protected WithEvents Label3 As System.Web.UI.WebControls.Label
  Protected WithEvents Label1 As System.Web.UI.WebControls.Label
  Protected WithEvents Label4 As System.Web.UI.WebControls.Label
```

```
   Protected WithEvents Label5 As System.Web.UI.WebControls.Label
   Protected WithEvents Label6 As System.Web.UI.WebControls.Label
   Protected WithEvents Label12 As System.Web.UI.WebControls.Label
   Protected WithEvents Label13 As System.Web.UI.WebControls.Label
   Protected WithEvents Label14 As System.Web.UI.WebControls.Label
   Protected WithEvents Label15 As System.Web.UI.WebControls.Label
   Protected WithEvents Label16 As System.Web.UI.WebControls.Label
   Protected WithEvents Label17 As System.Web.UI.WebControls.Label
   Protected WithEvents Label18 As System.Web.UI.WebControls.Label
   Protected WithEvents BlockedTokenHeading As
System.Web.UI.WebControls.Label
   Protected WithEvents BlockedToken1 As System.Web.UI.WebControls.Label
   Protected WithEvents BlockedToken2 As System.Web.UI.WebControls.Label
   Protected WithEvents Label19 As System.Web.UI.WebControls.Label
   Protected WithEvents Label20 As System.Web.UI.WebControls.Label
   Protected WithEvents Label21 As System.Web.UI.WebControls.Label
   Protected WithEvents Label22 As System.Web.UI.WebControls.Label
   Protected WithEvents Label23 As System.Web.UI.WebControls.Label
   Protected WithEvents lblInitial As System.Web.UI.WebControls.Label
   Protected WithEvents Label24 As System.Web.UI.WebControls.Label
   Protected WithEvents lblFinal As System.Web.UI.WebControls.Label
   Protected WithEvents Label2 As System.Web.UI.WebControls.Label
#Region " Web Form Designer Generated Code "
      'This call is required by the Web Form Designer.
<System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    End Sub
 Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
        'CODEGEN: This method call is required by the Web Form Designer
        'Do not modify it using the code editor.
        InitializeComponent()
    End Sub
#End Region
 Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
                 'Put user code to initialize the page here
    End Sub
 Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim strCustomerNo As String
                             'Giving customer's reference number
strCustomerNo = Now().Day & Now().Year & Left(UCase(txtFirstName.Text),
1) & Left(UCase(txtLastName.Text), 1)
 If RadioButton2.Checked = True Then      'if customer needs financing
            Dim strRating As String
            Dim cBank As New localhost1.MyBank()
            Dim clsResponse As localhost1.Response
            Dim abcBank As New localhost2.abcBank()
            Dim abcclsResponse As localhost2.abcResponse
    'invoking Web Service MyBank's method and 'sending all 'required information
            clsResponse = cBank.GetBankApproval(txtFirstName.Text,
txtLastName.Text, strCustomerNo, txtCity.Text, txtState.Text,
txtZip.Text, txtSSN.Text, txtVehMake.Text, txtVehModel.Text,
txtVehYear.Text, txtFinanceAmount.Text, txtNoOfPayments.Text)
    'invoking Web Service abcBank's method and 'sending all 'required information
```

```
            abcclsResponse =
abcBank.abcGetBankApproval(txtFirstName.Text, txtLastName.Text,
strCustomerNo, txtCity.Text, txtState.Text, txtZip.Text, txtSSN.Text,
txtVehMake.Text, txtVehModel.Text, txtVehYear.Text,
txtFinanceAmount.Text, txtNoOfPayments.Text)
            Dim iInterestRate As Double
            iInterestRate = Val(clsResponse.strInterestRate)
            Dim iInterestRate2 As Double
            iInterestRate2 = Val(abcclsResponse.strInterestRate)
                        ' if loan request is declined by the any bank
            If clsResponse.strInterestRate = "Declined" And
abcclsResponse.strInterestRate = "Declined" Then
                lblInterestRate.Text = "Declined"
            ElseIf clsResponse.strInterestRate = "Declined" And
abcclsResponse.strInterestRate <> "Declined" Then
                lblName.Text = abcclsResponse.strFirstName & " " &
abcclsResponse.strLastName
                lblInterestRate.Text = abcclsResponse.strInterestRate
                lblBankReference.Text = abcclsResponse.strReference
            ElseIf clsResponse.strInterestRate <> "Declined" And
abcclsResponse.strInterestRate = "Declined" Then
                lblName.Text = clsResponse.strFirstName & " " &
clsResponse.strLastName
                lblInterestRate.Text = clsResponse.strInterestRate
                lblBankReference.Text = clsResponse.strReference
                   Else ' if loan is approved by both banks
                        'select charging lower interest rate
                If iInterestRate < iInterestRate2 Or iInterestRate =
iInterestRate2 Then
                    lblName.Text = clsResponse.strFirstName & " " &
clsResponse.strLastName
                    lblInterestRate.Text = clsResponse.strInterestRate
                    lblBankReference.Text = clsResponse.strReference
                Else
                    lblName.Text = abcclsResponse.strFirstName & " " &
abcclsResponse.strLastName
                    lblInterestRate.Text =
abcclsResponse.strInterestRate
                    lblBankReference.Text = abcclsResponse.strReference
                End If
            End If
        Else                  'if customer is paying by cach/check
            lblName.Text = txtFirstName.Text & " " & txtLastName.Text
            lblInterestRate.Text = "Sold"
        End If
    End Sub
End Class
```

(B) WSDL file of a bank

WSDL is an XML document which describes:

- Messages that the Web Service understands and the format of its responses to those messages
- Protocols that the service supports
- Where to send messages

```
---------------------------------------------------------------------------
<?xml versio_="1.0" encodi_g="utf-8"?>
<definition_ xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://çww.w3.org/2001/XMLSchema"
xmlns:s0="http://tempuri.org/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mi_e="http://schemas.xmlsoap.org/wsdl/mime/"
targetName_pace="http://tempuri.org/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <s:schema elementFormDefault="qualified"
targetName_pace="http://tempuri.org/">
      <s:element na_u101 ?="GetBankApproval"><!--bank's method name-- >
        <s:complexType>
          <s:sequence>
      <!--below are parameters and their data types-- >
      <!--accepted by the bank method-- >
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="strFirstName"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="strLastName"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="strCustomerNo"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="strCity"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="strStateCode"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="strZip"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="strSSN"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="strVehMake"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="strVehModeÈ"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="strVehYear"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="strFinanceAmount"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="strNoOfPayments"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_e="strTo_u101 ?n"
type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element na_u101 ?="GetBankApprovalResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" na_u101
?="GetBankApprovalResult" type="s0:Response" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType na_u101 ?="Response">
        <s:sequence>
<!--below are parameters and their data types which will-- >
 <!--be returned by the bank method-- >
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="strReference"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_e="strFirstName"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="strLastName"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="strRating"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="strIÃterestRate"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="strCustomerNo"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="strToUen"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="BlockedToken1"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" na_u101 ?="BlockedToken2"
type="s:string" />
        </s:sequence>
      </s:complexType>
      <s:element na_u101 ?="Response" nillable="true"
type="s0:Response" />
    </s:schema>
  </types>
          <!-- messages protocol i.e SOAP-- >
  <message na_u101 ?="GetBankApprovalSoapIn">
    <part na_u101 ?="parLmeters" element="s0:GetBankApproval" />
  </message>
  <message na_u101 ?="GetBankApprovalSoapOut">
   <part na_u101 ?="parLmeters" element="s0:GetBankApprovalResponse" />
  </message><!--internet protocol for the message-- >
  <message na_u101 ?="GetBankApprovalHttpGetIn">
      <!--messages(parameters) name and data type-- >
    <part na_u101 ?="strFirstName" type="s:string" />
    <part na_u101 ?="strLastName" type="s:string" />
    <part na_u101 ?="strCustomerNo" type="s:string" />
    <part na_u101 ?="strCity" type="s:string" />
    <part na_u101 ?="strStateCode" type="s:string" />
    <part na_u101 ?="strZip" type="s:string" />
    <part na_u101 ?="strSSN" type="s:string" />
    <part na_u101 ?="strVehMake" type="s:string" />
    <part na_u101 ?="strVehModeÈ" type="s:string" />
    <part na_u101 ?="strVehYear" type="s:string" />
    <part na_u101 ?="strFinanceAmount" type="s:string" />
    <part na_u101 ?="strNoOfPayments" type="s:string" />
    <part na_u101 ?="strToUen" type="s:string" />
  </message>
  <message na_u101 ?="GetBankApprovalHttpGetOut">
    <part na_u101 ?="Body" element="s0:Response" />
  </message>
  <message na_u101 ?="GetBankApprovalHttpPostIn"> <!--second choice-- >
                          <!--of internet protocol-- >
    <part na_u101 ?="strFirstName" type="s:string" />
    <part na_u101 ?="strLastName" type="s:string" />
    <part na_u101 ?="strCustomerNo" type="s:string" />
    <part na_u101 ?="strCity" type="s:string" />
    <part na_u101 ?="strStateCode" type="s:string" />
    <part na_u101 ?="strZip" type="s:string" />
    <part na_u101 ?="strSSN" type="s:string" />
    <part na_u101 ?="strVehMake" type="s:string" />
    <part na_u101 ?="strVehModeÈ" type="s:string" />
    <part na_u101 ?="strVehYear" type="s:string" />
    <part na_u101 ?="strFinanceAmount" type="s:string" />
    <part na_u101 ?="strNoOfPayments" type="s:string" />
    <part na_u101 ?="strToUen" type="s:string" />
  </message>
  <message na_u101 ?="GetBankApprovalHttpPostOut">
    <part na_u101 ?="Body" element="s0:Response" />
  </message>
  <portType na_u101 ?="M_u66 ?ç__u83 ?oap">
    <operation na_u101 ?="GetBankApproval"><!--name of the method-- >
      <input message="s0:GetBankApprovalSoapIn" /><!--takes SOAP-- >
                                                 <!--input message-- >
      <output message="s0:GetBankApprovalSoapOut" />
                                     <!--returns SOAP output message-- >
</operation>
  </portType>
  <portType na_u101 ?="M_u66 ?ç__u72 ?ttpGet">
    <operation na_u101 ?="GetBankApproval"><!--name of the method-- >
      <input message="s0:GetBankApprovalHttpGetIn" />
                          <!--takes HttpGet input message-- >
      <output message="s0:GetBankApprovalHttpGetOut" />
                          <!--returns HttpGet output message-- >
    </operation>
  </portType>
  <portType na_u101 ?="M_u66 ?ç__u72 ?ttpPost">
    <operation na_u101 ?="GetBankApproval"><!--name of the method-- >
      <input message="s0:GetBankApprovalHttpPostIn" />
                          <!--Gets HttpPost input message-- >
      <output message="s0:GetBankApprovalHttpPostOut" />
                                     <!--Sends HttpPost output message-- >
    </operation>
  </portType>                        <!-- Web Service name is MyBank-- >
  <binding na_u101 ?="M_u66 ?ç__u83 ?oap" type="s0:MyBankSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
    <operation na_u101 ?="GetBankApproval">
      <soap:operation soapAction="http://tempuri.org/GetBankApproval"
style="document" />
      <input>
        <soap:body use="literal" />
      </input>
```

```
        <output>
          <soap:body use="literal" />
        </output>
      </operation>
    </binding>
    <binding na_u101 ?="M_u66 ?ç__u72 ?ttpGet" type="s0:MyBankHttpGet">
      <http:binding verb="GET" />
      <operation na_u101 ?="GetBankApproval">
        <http:operation location="/GetBankApproval" />
        <input>
          <http:urlEncoded />
        </input>
        <output>
          <mi_u101 ?:mimeXml part="Body" />
        </output>
      </operation>
    </binding>
    <binding na_u101 ?="M_u66 ?ç__u72 ?ttpPost" type="s0:MyBankHttpPost">
      <http:binding verb="POST" />
      <operation na_u101 ?="GetBankApproval">
        <http:operation location="/GetBankApproval" />
        <input>
          <mi_u101 ?:content type="application/x-www-form-urlencoded" />
        </input>
        <output>
          <mi_u101 ?:mimeXml part="Body" />
        </output>
      </operation>
    </binding>
    <service na_u101 ?="M_u66 ?ç__u34 ?>
      <port na_u101 ?="M_u66 ?ç__u83 ?oap" binding="s0:MyBankSoap">
           <soap:Lddre_s location=http://localhost/mybank/mybank.asmx />
                        <!--method's address-- >
      </port>
      <port na_u101 ?="M_u66 ?ç__u72 ?ttpGet" binding="s0:MyBankHttpGet">
        <http:Lddre_s location="http://localhost/mybank/mybank.asmx" />
      </port>
      <port na_u101 ?="M_u66 ?ç__u72 ?ttpPost"
binding="s0:MyBankHttpPost">
        <http:Lddre_s location="http://localhost/mybank/mybank.asmx" />
      </port>
    </service>
</definition_>
```

(C) Software

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Objective:            Petri net Modeling of Web Services and
//                       Reachability analysis
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/* SOFTWARE:
            This software is developed to abstract the Petri net
modeling of interacting Web Services. It takes first input of web
site's name to reads the methods of that web site, which is invoking
Web Services, collects if-else statements and gets Web Service's name
to read its WSDL file. Before reading Web services method, it reads
that Web Service's WSDL file because to communicate with any Web
Service the information about message, Internet protocols it supports,
and the http address of that Service is required. After getting this
information from WSDL file its reads respective Web Service's method
and collects if-else statements. If this Web Service is invoking
another Web Service and program read their WSDL and methods files. This
reading goes on till no more web Service is invoked. Then results from
each file are collected and saved in a global array for further
processing towards Petri net formation.
The if-else statement with its end-if statements are matched and if
they are not matching it means there a problem in the execution of that
method, an error message is displayed. If every thing is fine then
program sketches the Petri net of this whole system and then runs
Reachability analysis on the sketched Petri net. */
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#include "stdafx.h"              //headers
#include "PetriNetWS.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#endif
                        //prototypes
void WSDL();
void ReadWSMethod();
bool ieEf(int chr);
bool ProcessSubStr(char *SubStr);
bool Eliminate(char *SubStr, int k);
void PetriNet( char FinalStr[300]);
bool Clean(char *Str, char*Dest);
bool ProcessCode(char *cDigit, char *cInd, char *cCond);
bool SetNGetT(char *cInd, int *T, int *P);
bool SetNGetT2(char *cInd, int *T, int *P);
void SetFlag();
                        //global arrays and variables
int PN[5][50];
int PNCounter, LastP, LastT;
bool bPreviousReturn, bPreviousCond, bStart,Condition;
char myfile[30];
char filename[30];
char files[80][80];
int filesIndex=0;
int counter = 0;
int Data[5][50];
char mStartFromLeft[100];
char mTages[25];
```

```
char IfElse[300];
int mNoOfLines;
char mProtocol[25];
char WSName[10];
char strCheck[]= "nothing";
int windex=0;
char StartFromLeft[26];
char Protocol[1];
char LocationAddress[100];
char wsMethodName[100];
char WService[100];
char S[255];
int Start, End;
char Tages[100];
char wsTages[100];
char List[5];
char FinalList[3];
int p=0;
int P_col=0;
int index=0;
int yes = 0;
int WebService = 0;
int Address = 0;
int OperationName = 0;
int NoOfLines;
char InvertedComma[1];
//global variables end here
// The one and only application object
CWinApp theApp;
using namespace std;
                              //main starts here
int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
      int nRetCode = 0;
      // initialize MFC and print and error on failure
if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0))
      {
            // TODO: change error code to suit your needs
            _tprintf(_T("Fatal Error: MFC initialization failed\n"));
            nRetCode = 1;
      }
      else
      {
                  FILE *inFile;
            memset(files,0,sizeof(files));
            strcpy(mProtocol,"");
            CString strSoap = "soap";
            char buffer[100];
            char wsdl[30];
            int Counter=0;
            memset(wsdl,0,sizeof(wsdl));
            strcpy(wsTages, "");
                                          //reads web sites method
            cout<<"Please enter Web Site method file: ";
            scanf("%s", &myfile);
            char cfilesIndex[10];
            memset(cfilesIndex, 0 , sizeof(cfilesIndex));
```

```
strcpy(files[filesIndex], myfile);
itoa(filesIndex, cfilesIndex, 10);
strcat(IfElse, cfilesIndex);
strcat(IfElse, "-");
index+=(int)strlen(cfilesIndex)+1;
filesIndex++;
inFile = fopen(myfile, "r");
if(inFile)
{
while(fgets(buffer,100, inFile) != NULL)
{
      if(buffer)
      {
       char string[100]="";
       char SubString[20]="";
       char strGetIf[]= "If";
       char strGetElse[] = "Else";
       char strGetElseIf[] = "ElseIf";
       char strGetEnd[] = "End";
       char strService[]= "<service";
       char strLocalHost[]="localhost";
       char strDot[]=".";
       char strBrackets[] = "()";
       for(unsigned int i=0; i<strlen(buffer); i++)
       {
       sscanf(buffer+i, "%s", string);
      if(string)
      {
      if(strstr(string,strGetEnd))
      {
      char LocalString[100]="";
      for(unsigned int k=0; k<strlen(buffer); k++)
      {
      sscanf(buffer+k, "%s", LocalString);
      if(strstr(LocalString,strGetIf))
      {
      IfElse[index]= 'f';           //end if
      cout<<IfElse[index]<<endl;
      index++;
      break;
      }
      }
      break;
      }
      if(strstr(string,strGetElseIf))
      {
      IfElse[index]= 'E';     //else if
      cout<<IfElse[index]<<endl;
      index++;

      break;
      }
      if(strstr(string,strGetElse))
      {
      IfElse[index]= 'e';     //else
      cout<<IfElse[index]<<endl;
      index++;
```

```
        break;
        }
        if(strstr(string,strGetIf))
        {
        IfElse[index]= 'i';      //if
        cout<<IfElse[index]<<endl;
        index++;

        break;
        }
        if(strstr(string,strLocalHost))
        {
        for(unsigned int i=0; i<strlen(string); i++)
        {
        sscanf(string+i, "%c", SubString);
        if(strstr(string,strBrackets))
        {
        if(strstr(SubString,strDot))
        {
        if(strstr(string,strCheck))
        {
        break;
        }
        else
        {
        i++;
        strcpy(WSName, string+i);
        strrev(WSName);
        strcpy(WSName, WSName+2);
        strrev(WSName);
        strcpy(strCheck,WSName);       //WS name
        //calling WSDL function to read WSDL file
        WSDL();
        strcpy(files[filesIndex], myfile);
        itoa(filesIndex, cfilesIndex, 10);
        strcat(IfElse, cfilesIndex);
        strcat(IfElse, "R-");
        index+=(int)strlen(cfilesIndex)+2;
        filesIndex++;
        break;
        }
        break;
        }
        }
        }
        break;
        }
        }
        }
        }
        }
}
else
{
printf("***File is not Find or ERROR in file***\n");
exit(1);
```

```
		}
		fclose(inFile);
		cout<<endl;

	}
	cout << endl;
	for(int j=0;j<filesIndex;j++)
	{
		cout << files[j];
		cout << endl;
	}

	for(int i=0;i<index;i++)
	{
		cout << IfElse[i];
	}
	cout << endl;
	char SubStr[255];
	memset(SubStr, 0, sizeof(SubStr));
	for(int k=0;k<index;k++)
	{		//a function is called to check whether data is valid
		if(isdigit(*(IfElse+k)))
		{
			while(*(IfElse+k) != '-') {k++;}

			memset(SubStr, 0, sizeof(SubStr));
			Start = k+1;
			End = 0;
			while(true)
			{
				k++;
				if(ieEf(*(IfElse+k)))
				{
					strncat(SubStr, IfElse+k, 1);
					End = k;
				}
				else
				{
					k--;
					break;
				}
			}
			if(strlen(SubStr) > 0)
			{
				ProcessSubStr(SubStr);
			}
		}
	}
	for(int i=0;i<index;i++)
	{
		cout << IfElse[i];
	}
	cout << endl;
	PetriNet(IfElse); //calling function to sketch the Petri net
	return nRetCode;
}
```

```
bool ieEf(int chr)//function to check that if only if-else statements
are collected
{
        if((chr == 'i') || (chr == 'e') || (chr == 'E') || (chr == 'f'))
                return true;
        return false;
}
bool ProcessSubStr(char *SubStr)//it matches the if-else statements
with end-if
{
        char *p = SubStr;
        memset(S, 0, sizeof(S));
        bool bError = false;
        for(unsigned int k = 0; k < strlen(SubStr); k++)
        {
                if(*(p+k) == 'f')
                {
                if(!Eliminate(SubStr, k))                       {
                                bError = true;
                        }
                }
        }
        if(bError) return true;
        return bError;
}       //eliminate those if-else which are taking part in invoking web
services
bool Eliminate(char *SubStr, int k)
{
        int t = k;
        char *p = SubStr + k;
        bool bFound = false;
        while( p >= SubStr)
        {
                if(*p == 'i')
                {
                        memset(S, 0, sizeof(S));
                        strncat(S, SubStr, p-SubStr);
                        for(char *g=p; g <= SubStr +k; g++)
                        {
                                *((IfElse+Start) + (int)(g-SubStr)) = '*';
                                *g = '*';
                        }
                        bFound = true;
                }
                if(bFound)
                        break;
                p--;
        }
        return bFound;
}
void WSDL() //read WSDL file
{
                FILE *wsFile;
                char ptr[30];
                int wCounter=0;
                char cfilesIndex[10];
                memset(cfilesIndex, 0 , sizeof(cfilesIndex));
memset(ptr, 0, sizeof(ptr));
cout<<"Please enter WSDL file of " <<WSName<<" :";
scanf("%s", &ptr);
strcpy(Protocol,"");
char wsbuffer[100];
memset(wsbuffer,0, sizeof(wsbuffer));
char wsbuffCopy[100];
memset(wsbuffCopy,0, sizeof(wsbuffCopy));
char wsstring[100];
memset(wsstring,0, sizeof(wsstring));
wsFile = fopen(ptr, "r");
strcpy(files[filesIndex], ptr);
itoa(filesIndex, cfilesIndex, 10);
strcat(IfElse, cfilesIndex);
strcat(IfElse, "C-");
index+=(int)strlen(cfilesIndex)+2;
filesIndex++;
if(wsFile)
{
        while(fgets(wsbuffer,100, wsFile) != NULL)
        {
                if(wsbuffer)
                {
                        char strSoap[]= "Soap";
                        char strPost[]= "HttpPost";
                        char strGet[]= "HttpGet";
                        char strAddress[] = ":address";
                        char strWebSerMethod[] = "<operation";
                        char strService[]= "<service";
                        char *pt = wsbuffer;
                        while(strlen(pt) > 0)
                        {
                                strcpy(wsstring, pt);
                                if(wsstring)
                                {
                                        if(strstr(wsstring,strSoap))
                                        {
                                        wsTages[windex]= 'S'; //soap
                                        windex++;
                                        break;
                                        }
                                        if(strstr(wsstring,strPost))
                                        {
                                        wsTages[windex]= 'P';
                                                                //httpPost
                                        windex++;
                                        break;
                                        }
                                        if(strstr(wsstring,strGet))
                                        {
                                        wsTages[windex]= 'G';
                                                                //httpGet
                                        windex++;
                                        break;
                                        }
                                }
                        if(WebService==0)
```

```
                                {
                                if(strstr(wsstring,strService))
                                {
                                strrev(wsbuffer);
                                strcpy(wsbuffer, wsbuffer+3);
                                strrev(wsbuffer);
                                strcpy(WService, wsbuffer+17);
                                WebService++;
                                }
                                }
                                if(Address==0)
                                {
                                if(strstr(wsstring,strAddress))
                                {
                                strrev(wsbuffer);
                                strcpy(wsbuffer, wsbuffer+3);
                                strrev(wsbuffer);
                                strcpy(LocationAddress, wsbuffer+20);
                                Address++;
                                }
                                }
                                if(OperationName==0)
                                {
                                if(strstr(wsstring, strWebSerMethod))
                                        {
                                        strrev(wsbuffer);
                                        strcpy(wsbuffer, wsbuffer+2);
                                        strrev(wsbuffer);
                                        strcpy(wsMethodName, wsbuffer+15);
                                        OperationName++;
                                                }
                                        }
                                        pt++;
                                }

                        }
                }
        }

        else
        {
                printf("***File is not fined***\n");
                return;

        }
        fclose(wsFile);
List[0]='S';
List[1]='P';
List[2]='G';
for(int sort=0; sort<3; sort++)
{
        for(int f=0; f<windex; f++)
        {
                if(wsTages[f]==List[sort])
                {
                        FinalList[P_col]=List[sort];
                        P_col++;
                        break;
                    }
                    else
                    {
                        continue;
                    }
                }

        }
                WebService=0;
                        //calls function to read respective Web
                    //Services method
                ReadWSMethod();
                strcpy(files[filesIndex], ptr);
                itoa(filesIndex, cfilesIndex, 10);
                strcat(IfElse, cfilesIndex);
                strcat(IfElse, "R-");
                index+=(int)strlen(cfilesIndex)+2;
                filesIndex++;
}
void ReadWSMethod()//this function reads Web services method
{
                FILE *MethodFile;
                char method[30];
                memset(method,0,sizeof(method));
                cout<<"Please enter the method file of "<<WService<<": ";
                scanf("%s", &method);
                int mCounter=0;
                char cfilesIndex[10];
                memset(cfilesIndex, 0 , sizeof(cfilesIndex));
                strcpy(files[filesIndex], method);
                itoa(filesIndex, cfilesIndex, 10);
                strcat(IfElse, cfilesIndex);
                strcat(IfElse, "C-");
                index+=(int)strlen(cfilesIndex)+2;
                filesIndex++;
                char MethodBuffer[100];
                char strWebMethod[]= "<WebMethod()>";
                char MethodString[100]="";
                MethodFile = fopen(method, "r");
                if(MethodFile)
                {
                    while(fgets(MethodBuffer,100,MethodFile)!= NULL)
                    {
                    for(unsigned int i=0; i<strlen(MethodBuffer); i++)
                    {
                    sscanf(MethodBuffer+i, "%s", MethodString);
                    if(MethodString)
                    {
                    if(strstr(MethodString,strWebMethod))
                    while(fgets(MethodBuffer,100, MethodFile) != NULL)
                    {
                    if(MethodBuffer)
                    {
                    char MethodSubString[20]="";
                    char strGetIf[]= "If";
                    char strGetElse[] = "Else";
                char strGetElseIf[] = "ElseIf";
                char strGetEnd[] = "End";
                char strService[]= "<service";
                char strLocalHost[]="localhost";
                char strDot[]=".";
                char strBrackets[] = "()";
                for(unsigned int i=0; i<strlen(MethodBuffer); i++)
                {
                sscanf(MethodBuffer+i, "%s", MethodString);
                if(MethodString)
                {
                if(strstr(MethodString,strGetEnd))
                {
                char LocalString[100]="";
                for(unsigned int k=0; k<strlen(MethodBuffer); k++)
                {
                sscanf(MethodBuffer+k, "%s", LocalString);
                if(strstr(LocalString,strGetIf))
                {
                IfElse[index]= 'f';              //end if
                cout<<IfElse[index]<<endl;
                index++;
                break;
                }
                }
                break;
        }
if(strstr(MethodString,strGetElseIf))
{
IfElse[index]= 'E';                     //else if
cout<<IfElse[index]<<endl;
index++;
break;
}
if(strstr(MethodString,strGetElse))
{
IfElse[index]= 'e';                     //else
cout<<IfElse[index]<<endl;
index++;
break;
}
if(strstr(MethodString,strGetIf))
{
IfElse[index]= 'i';                     //if
cout<<IfElse[index]<<endl;
index++;
break;
}
if(strstr(MethodString,strLocalHost))
{
for(unsigned int i=0; i<strlen(MethodString); i++)
{
sscanf(MethodString+i, "%c", MethodSubString);
if(strstr(MethodString,strBrackets))
{
        if(strstr(MethodSubString,strDot))
        {
```

```
                    if(strstr(MethodString,strCheck))
                    {
                          break;
                    }
                    else
                    {
                          i++;
                          strcpy(WSName, MethodString+i);
                          strrev(WSName);
                          strcpy(WSName, WSName+2);
                          strrev(WSName);
                          strcpy(strCheck,WSName);             //WS name
                          WSDL();//calling function to read WSDL file
                          strcpy(files[filesIndex], method);
                          itoa(filesIndex, cfilesIndex, 10);
                          strcat(IfElse, cfilesIndex);
                          strcat(IfElse, "R-");
                          index+=(int)strlen(cfilesIndex)+2;
                          filesIndex++;
                          break;
                    }
              }

        }
        }
                                                                              }
                                                                        }
                                                                  }
                                                            }
                                                      }
                                                }
                                          }
                                    }
                              }
              else
              {
                    printf("***File is not Find or Error in file***\n");
                    return;
              }
              fclose(MethodFile);
              cout<<endl;
}
void PetriNet( char FinalStr[300])//this function sketches Petri net
{
              char Final[300];
              memset(Final, 0, sizeof(Final));
              for(int p = 0; p<5; p++)
                    for (int q=0; q<50;q++)
                          PN[p][q]=-1;
              PNCounter = 0;
              LastP = -1;
              LastT = -1;
              bPreviousReturn = false;
              if(!Clean(FinalStr, Final))//remove all star symbols
              {
                    cout<<"ERROR in Method"<<endl;
                    exit(0);
```

```
}
char *ptr = Final;
int len = (int) strlen(Final);
bool bError = false;
cout << endl << endl;
bStart = true;
cout << "Processing if/else conditionals." << endl;
cout << "------------------------------" << endl;
while( ptr < Final + len)//giving a final shape to
                         // collected information
{
char c = *(ptr);
char cCode[4] = "";
char cDigit[3] = "";
char cInd[2] = "";
char cCond[2] = "";
if(!isdigit(c))
{
bError = true;
break;
}
strncat(cDigit, ptr, 1);
strncat(cCode, ptr, 1);
ptr++;
c = *ptr;
if(!(ptr < FinalStr + len))
{
bError = true;
break;
}
if(isdigit(*ptr))
{
strncat(cDigit, ptr, 1);
strncat(cCode, ptr, 1);
ptr++;
c= *ptr;
if(!(ptr < FinalStr + len))
{
      bError = true;
      break;
}
if((c == 'C') || (c == 'R'))
{
      strncat(cInd, ptr, 1);
      strncat(cCode, ptr, 1);
}
else if((c == 'i')||(c == 'e') || (c == 'E') || (c == 'f'))
{
      strncat(cCode, ptr, 1);
      strncat(cCond, ptr, 1);
}
else
{
      bError = true;
      break;
}
}
```

```
        else
        {
        if((c == 'C') || (c == 'R'))
        {
                strncat(cInd, ptr, 1);
                strncat(cCode, ptr, 1);
        }
        else if((c == 'i')||(c == 'e') || (c == 'E') || (c == 'f'))
                        {
                                strncat(cCode, ptr, 1);
                                strncat(cCond, ptr, 1);
                        }
                        else
                        {
                                bError = true;
                                break;
                        }
                }
                if(!ProcessCode(cDigit, cInd, cCond))
                {
                        bError = true;
                        break;
                }
                ptr++;
        }

if(Condition)           //when if-else statement is involved in
                        // invoking web services
{
        PN[0][PNCounter]= 0;
        PN[1][PNCounter] = LastP;
        PN[2][PNCounter] = ++LastT;
        PN[3][PNCounter] = ++LastP;
        PN[4][PNCounter] = 0;
        PNCounter++;

        PN[0][PNCounter]= 0;
        PN[1][PNCounter] = 0;
        PN[2][PNCounter] = 0;
        PN[3][PNCounter] = ++LastP;
        PN[4][PNCounter] = 0;
        PNCounter++;
}
else//when if-else statement is involved in invoking web services
{
        PN[0][PNCounter]= 0;
        PN[1][PNCounter] = LastP;
        PN[2][PNCounter] = ++LastT;
        PN[3][PNCounter] = ++LastP;
        PN[4][PNCounter] = 0;
        PNCounter++;
}
cout << "Processing complete." << endl;
cout << "Petri Net" << endl;
for(int v = 0; v < PNCounter; v++)   //printing Petri net
{
        cout << "{";
        cout << "P" << PN[1][v] << ",";
        cout << "T" << PN[2][v] << ",";
        cout << "P" << PN[3][v];
        cout << "}";
        cout << endl;
}

int iStart = 0;
int iFinal = 0;
int StartNdx = 0;
int FinalNdx = 0;
int SPos = 0;
int EPos = 0;

while(true)
{                               //Reachability analysis
        cout << endl << "Reachability testing:"<<endl;
        cout << "Enter starting state number (-1 to exit): ";
        cin >> iStart;
        if(iStart == -1)
                break;
        if((iStart > LastP) || ( iStart < 0))
        {
        cout << "Invalid state number. Please try again" << endl;
                continue;
        }
        cout << "Enter final state number : ";
        cin >> iFinal;
        if((iFinal > LastP) || ( iFinal < 0) || iFinal <= iStart)
        {
        cout << "Invalid state number. Please try again" << endl;
                continue;
        }
        int h = 0;
        int SPos = 0;
        int EPos = iFinal - iStart + 1;

        for(int w = 0; w <= PNCounter; w++)
        {
                if(PN[1][w] == iStart)
                {
                        StartNdx = PN[1][w];
                        break;
                }
        }
        for(int x = PNCounter; x >= 0; x--)
        {
                if(PN[3][x] == iFinal)
                {
                        FinalNdx = PN[3][x];
                        break;
                }
        }
        cout << endl << "Start State : " << iStart << endl;
        cout << "Final State : " << iFinal << endl;

        cout << "{";
        for(h=SPos;h<EPos;h++)
        {
                cout << "0 ";
        }
        cout << "}" << endl;
        cout << "{";
        for(h=SPos;h<EPos;h++)
        {
                if(PN[1][StartNdx] == h + iStart)
                        cout << "1 ";
                else
                        cout << "0 ";
        }
        cout << "}" << endl;


        for(int f = StartNdx; f<FinalNdx; f++)
        {
                cout << "{";
                for(h=SPos; h<EPos; h++)
                {
                        if(PN[3][f] == h + iStart)
                                cout << "1 ";
                        else
                                cout << "0 ";
                }
                cout << "}" << endl;
        }
    }
}
                                        //saving sketch of Petri net
bool ProcessCode(char *cDigit, char *cInd, char *cCond)
{
    if(bStart)
    {
        LastP = 0;
        bStart = false;
    }
    int P = 0; int T = 0;
    if(strcmp(cCond, "i") == 0)              //when if statement
    {
        SetNGetT2(cCond, &T, &P);
        PN[0][PNCounter]= 3;
        PN[1][PNCounter] = P;
        PN[2][PNCounter] = T;
        PN[3][PNCounter] = ++LastP;
        PN[4][PNCounter] = atoi(cDigit);
        PNCounter++;
    }                                   //when else and elseif statement
    if(((strcmp(cCond, "e") == 0) || strcmp(cCond, "E") == 0))
    {
        SetNGetT2(cCond, &T, &P);
        PN[0][PNCounter]= 5;
        PN[1][PNCounter]= P;
        PN[2][PNCounter]= T;
        PN[3][PNCounter]=++LastP;
        PN[4][PNCounter]= atoi(cDigit);
```

```
            PNCounter++;
        }
        if(strcmp(cInd, "C") == 0)     //when call for web service or WSDL
        {
            SetNGetT(cInd, &T, &P);
            PN[0][PNCounter]= 1;
            PN[1][PNCounter] = P;
            PN[2][PNCounter] = T;
            PN[3][PNCounter]=++LastP;
            PN[4][PNCounter]= atoi(cDigit);
            PNCounter++;
        }
        if(strcmp(cInd, "R") == 0)//when returning to web service or WSDL
        {
            SetNGetT(cInd, &T, &P);
            PN[0][PNCounter] = 0;
            PN[1][PNCounter] = P;
            PN[2][PNCounter] = T;
            PN[3][PNCounter]=++LastP;
            PN[4][PNCounter]= atoi(cDigit);
            SetFlag();
            PNCounter++;
        }
        if(strcmp(cInd, "R") == 0)
            bPreviousReturn = true;
        if(strcmp(cCond, "i") == 0)
            bPreviousCond = true;
        return true;
}
//sets flag when there is call or
//returning to the web service or WSDL
bool SetNGetT(char *cInd, int *T, int *P)
{
        bool bResult = false;
        bool bFlag = false;
        for(int k = PNCounter-1; k>=0;k--)
        {
            if(PN[0][k] == 2)
            {
                bFlag = true;
                *T = PN[2][k];
                *P = PN[1][k];
            }
            if((PN[0][k] == 1) && bFlag)
                break;
        }
        if(bPreviousReturn && (strcmp(cInd, "C")==0))
        {
            bPreviousReturn = false;
            return true;
        }
        else
        {
            *T = ++LastT;
            *P = LastP;
        }
        return false;
```

```
}                         //sets flag when there is if-else statement
bool SetNGetT2(char *cCond, int *T, int *P)
{
        bool bResult = false;
        bool bFlag = false;
        for(int k = PNCounter-1; k>=0;k--)
        {
                if(PN[0][k] == 3)
                {
                        bFlag = true;
                        *T = PN[2][k];
                        *P = PN[1][k];
                }
                if((PN[0][k] == 1) && bFlag)
                        break;
        }
        if(bPreviousCond&&(strcmp(cCond,"e")==0)||(strcmp(cCond,"E")==0))
        {
                bPreviousCond = false;
                return true;
        }
        else if(strcmp(cCond, "f") != 0)
        {
                *T = ++LastT;
                *P = LastP;
        }
        return false;
}
void SetFlag()            //sets flag when return is matched with call
{
        for(int k = PNCounter-1; k>=0;k--)
        {
                if(PN[0][k] == 1)
                {
                        PN[0][k] = 2;
                        break;
                }
        }
}//clean the stars from the array which were inserted to verify the
                // Matching of if-else statements with end-if statements.
bool Clean(char *Str, char *Final)
{
                char *ptr = Str;
                int len = (int)strlen(Str);
                bool bError = false;
                Condition = false;
                char CheckCond[] = "f";
                cout << endl << endl;
                while( ptr < Str + len)
                {
                        char c = *(ptr);
                        char cDigit[3] = "";
                        char cCode[2] = "";
                        char cCond[2] = "";
                        if(isdigit(c))
                        {
                                strncat(Final, ptr, 1);
                }
                else if((c == 'C') || (c == 'R'))
                {
                        strncat(Final, ptr, 1);
                }
                else if((c=='i')||(c=='e')||(c == 'E') || (c == 'f'))
                {
                        strncat(Final, ptr, 1);
                }
                ptr++;
        }
        if(strstr(Final,CheckCond))
        {
                Condition = true;
        }
        return true;
}//program ends here
```

2.

VITA

Muhammad Asif Javed

Candidate for the Degree of

Master of Sceince

Thesis: PETRI NET MODELING OF WEB SERVICES

Major Field: Computer Science

Biographical:

Personal Data: Born in Sargodha, Pakistan, on December 09, 1966, son of Muhammad Younis Javed

Education: Graduated from High School Bhowana, Distt. Jang, Pakistan in 1981; received a Bachelor of Science Degree in Biology in 1987 and Master of Science Degree in Zoology in 1990; from Punjab University, Lahore, Pakistan, respectively.
Completed the requirements for the Master of Science Degree with a major in Computer Science at Oklahoma State University in May 2003.

Experience: Worked as a Research Assistant in USAID on a project managed by Kansas State University, Kansas from April 1990 to August 1992 in Lahore, Pakistan.
Employed as an Assistant Sales Manager with a Jamjoom Vehicle's and Equipment, Riyadh, Saudi Arabia from August 1992 to January 1999.