People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

IBN KHALDOUN UNIVERSITY OF TIARET

Dissertation

Presented to:

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
DEPARTEMENT OF COMPUTER SCIENCE

in order to obtain the degree of :

MASTER

Specialty: software engineer

Presented by:

REBOUH Rosa

On the theme:

Web Service composition Model-Cheking

Defended publicly on 11 / 06 / 2024 in Tiaret in front the jury composed of:

Mr BEKKI Khadir	MAA	Tiaret University	Chairman
Mme HAMDANI Abdia	MCB	Tiaret University	Supervisor
Mr HATTAB Noureddine	MAA	Tiaret University	Examiner

2023-2024


بسم الله الرحمن الرحيم

Acknowledgements

The first and foremost gratitude and thanks go to Allah, the Almighty, who granted me the guidance and success to complete this humble work.

I extend my sincere thanks and appreciation to my supervisor, HAMDANI Abdia, who offered me assistance and support and did not spare her time and appreciated effort. I also thank Mr. BEKKI Khadir and Mr. HATTAB Noureddine for their approval and evaluation of this work.

I then express my heartfelt thanks to all the individuals who helped me in finishing this work. My gratitude especially goes to my mother, brother, and sisters, who never stopped encouraging me.

Finally, I express my appreciation to all my teachers in the Computer Science Department, Ibn Khaldoun University of Tiaret, for their dedication and support throughout these years, Thank you.

Dedication

To my mother, my brother Ali (Mohammed), my sisters, and all their children,
I thank you for everything.
But I apologize, as I dedicate the fruit of my humble work to all the children
and martyrs of Gaza, the capital of Palestine.

REBOUH Rosa

Abstract

Analyzing and verifying a complex web service composition permit to give answers about their correctness, reliability and safety. Indeed, (*Model-Cheking*: "check on the model"), is a very well known method to verify the qualitative (e.g.: stat accessibility, liveness, blocking, etc...) as well as the quantitative properties (e.g.: time response) on the model. The Petri net formalism and its extensions are widely used in the literature for their important number of advantages as: (graphical and intuitive representations, an aboundants analysis techniques...etc.). We are interested on the Petri net formalism as a modelling tool, extended to time constraints. Because of the infite state space of the model, many standards methods for the state space abstraction are used, gethering a set of states sharing some properties and based on the *State Class* formalism. (*TINA*: "TIme Network Analyzer") is one of the most known tool for this end. Finally, analysing and verifying a qualitative and quantitative properties on the model as well as on the system itself became possible.

Keywords: Analysis and verification, model-cheking, proprerties, Time Petri Nets (TPN), state space abstraction, TINA.

Résumé

L'analyse et la vérification de la composition des services web permettent de répondre aux questions liées, essentiellement, à la fiabilité et à la sûreté de fonctionnement de ces derniers. En effet, (le *Model-Cheking*: "vérification du/sur le modèle"), est une méthode très connue permettant de vérifier les propriétés qualitatives (e.g: accessibilité d'un état, vivacité, bloquage, etc...) et/ou quantitatives (e.g: la réponse du temps borné) sur le modèle. Le formalisme des réseaux de Petri et ses extensions sont largement utilisés dans la littérature, offrant un nombre considérables d'avantages citons: (sa représentation graphique et intuitive, ses techniques abondantes d'analyses...etc.). Nous nous intéressons aux réseaux de Petri comme outil de modélisation, étendu au temps. Cependant, l'espace d'état est généralement infinit et de nombreuses méthdes sont utilisées pour l'abstraction de l'espace d'états du modèle ceci consiste à regrouper un ensemble d'états partageant les même propriétés basé sur le formalisme de *classe d'état*. (*TINA*: "TIme Network Analyzer") est l'un des outils les plus connu pour cela. Enfin, l'analyse et la vérification des propriétés *qualitatives* et *quantitatives* sur le modèle ,et bien évidement, sur le système en question devient possibles.

Mots Clé: Analyse et vérification, model-cheking, propriétés, réseaux de Petri Temporels (RdpT), abstraction de l'espace d'états, TINA.

Contents

List of Figures

List of Tables

Acronyms

PN Petri Net

TINA Time Petri Net Analyzer

WSC Web Service Composition

XML Extensible Markup Language

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

J2EE Java 2 Platform, Enterprise Edition

CORBA Common Object Request Broker Architecture

.NET Microsoft's software framework

UDDI Universal Description, Discovery, and Integration

SOA Service-Oriented Architecture

API Application Programming Interface

SOAP Simple Object Access Protocol

XHTML Extensible Hypertext Markup Language

XSLT Extensible Stylesheet Language Transformations

RSS Rich Site Summary or Really Simple Syndication

SVG Scalable Vector Graphics

WSDL Web Service Description Language

REST Representational State Transfer

CRUD Create, Retrieve, Update, Delete

BPEL Business Process Execution Language

BPMN Business Process Model and Notation

WSFL Web Service Flow Language

XPDL XML Process Definition Language

WS-CDL Web Services Choreography Description Language

ACID Atomicity, Consistency, Isolation, and Durability

BPML Business Process Modeling Language

WSCI Web Service Choreography Interface

WSCL Web Service Conversation Language

BPEL4WS Business Process Execution Language for Web Services

BPM Business Process Management

TPN Timed Petri Nets

GNT General Net Theory

ACD Automatic Cash Dispensers

ESAM Enhanced Stacked Automata Model

EC.WSC E-commerce Web Service Composition

IoT Internet of Things

Chapter 1

Introduction

1.1 Context

In the modern web, web services have become an essential reality, exerting a tremendous and escalating impact on daily computing tasks. They have transformed the web into the largest, most accepted, and most dynamic distributed computing platform ever. [1]
One of the main benefits of using web service technologies is the ability to compose web services for added value. Web service composition involves mechanisms that encourage collaboration among individual web services to develop integrated functionalities, potentially reducing time and effort in development. [2]
Service composition encompasses all those processes that create added-value services, called composite or aggregated services, from existing services. [1]

1.2 Background

In today's digital world, web services have become a common and crucial aspect of information technology and communications. They provide access to information and facilitate interaction over the internet. Web services encompass a set of functions or operations available online, accessible through specific protocols. The significance of web service composition lies in the ability to integrate and connect these services together to achieve a specific function or goal. This requires a deep understanding of how these services integrate seamlessly and effectively, ensuring their compatibility with each other.
The first step to star with doing Model cheking is *to model*, we find so many formalism for this end and we choose the (**PN**: Petri Net) formalism as *Van Der Aalast* said [3]: "Three Good Reasons for Using A Petri-Net.... ". Modeling web service composition using Petri nets provides a powerful means to understand and analyze the processes involved in web service composition. Petri nets enable the representation of interactions between different services and components visually and effectively. These models allow for the analysis of data flow, process control, and identification of critical points and potential risks.
Petri nets are used to represent various components of web service composition, such as individual services and the different operations occurring between them.

The key elements in modeling web service composition using Petri nets include:
1- Places: Represent the current state of the system, such as the availability of services and available data.
2- Transitions: Represent the operations or events that occur in the system, such as service requests or process execution.
3- Arcs: Represent the relationships between places and transitions, indicating how places affect transitions and vice versa.
Using these elements, a model of the web service composition process can be created, providing a deeper understanding of system interactions, performance analysis, and identifying potential issues. These models contribute to improving the process of developing and composing web services and ensuring the delivery of high-quality services to users. The goal is to ensure error-free system operation and deliver the expected results, thereby guaranteeing the system's proper functioning.
Modeling web service composition aids in understanding the complex interactions between services and various components involved in the composition process. By providing visual representation, these models facilitate system understanding and clarify how different elements interact with each other. We focused on Petri nets because they are a powerful analytical tool used in various fields to model and analyze processes. Petri nets enable the representation of system processes using basic elements like places, transitions, and arcs, facilitating the understanding of system interactions, performance analysis, and identification of potential risks. Modeling web services using Timed Petri nets allows for the visual representation of temporal processes and variables. Through this modeling, performance analysis can be conducted, bottlenecks identified, and the behavior of composite services optimized under various scenarios. Analyzing web service composition through Timed Petri nets provides a precise understanding of service interactions and the impact of temporal factors on process flow. This analysis helps in identifying bottlenecks and improving the performance of composite services. As for the TINA (Time Petri Net Analyzer) tool, it is a comprehensive toolkit designed for the analysis and enhancement of Web Service Composition (WSC) and related web systems. TINA offers a diverse range of tools, including a graphical editor for Petri nets, both time-based and standard, as well as machines. Additionally, it provides tools for building graphical representations and time models, along with structural and path analysis tools, and verification tools for specified linguistic formulations of states and events. These tools empower users to gain a deep understanding of service interactions and the impact of temporal factors on WSC performance, enabling the improvement of complex processes and service compatibility.

1.3 Research Objectives:

As part of the final project, our main objective is to show how modeling web service composition could be done using Petri nets. thus, uncover potential system errors and ensure the delivery of expected results. These goals aim to enhance the quality and efficiency of web service composition processes, ensuring accurate and effective service delivery. The project aims to enhance understanding, improve performance, and address challenges in web service composition. We work throw the following questions (objectives):

1. How modeling web service composition using Petri net formalism
2. How TINA tool can be used to detect the main properties of such model(case study) and
3. How can we understand and compare the different form of graphs generated by TINA tool and theirs propertises
4. Doing that, we accomplish our aim that must give some answers such that: is our model correct? witch refers to MODEL CHEKING.

1.4 Organization Of The Report

This thesis is structured in five chapters besides a general introduction and a general conclusion:

- Chapter 1: "Introduction" , this chapter is a brief introduction of what is web services and web service composition to meet users' needs with, a general context, background, research objectives and report organization.
- Chapter 2: "Web Services Composition", in this chapter, we provide an overview of the concept of web service, the concept of composition web services, their types, etc.
- Chapter 3: "Petri Nets Formalism", genaralities definition and history of the Petri net are presented in this chapter, syntax and semantics of the model are also giving.
- Chapter 4: " Modeling Web Services Composition ", in this chapter we present the most known formalisms used to model WSC using Peti Net and extensions of it.
- Chapter 5 "Case Study", we apply the technique of ModelCheking on a real case study "BOOKIG", thus modeling and analysing with experiment results of the following case are presented in detail throw this chapter using TINA tool.
- Chapter 6: Conclusion, The last part of this work summarizes the main part of our work and draws the perspectives in general conclusion.

Our report ends with two importants ANNEXES :**TINA** tool used in the modeling and analyzing case study and **LATEX** for editing chapters.

Chapter 2

Web Services Composition

2.1 Introduction

A web service is a self-contained, self-describing, modular software component that is accessible over the web. It can be published by service providers and invoked by service requesters over the Internet. In recent years, there has been growing interest in web services due to their significant potential in real-world applications [4].

Although each individual web service holds value for users, its functionality alone is limited. The true potential of web services is realized when multiple services are combined to create more complex and powerful applications with advanced features. This process is known as Web Service Composition (WSC) [5].

Service composition is a fundamental technology in service-oriented computing, where existing services are reused as basic components to build new services with advanced functionalities. This enables rapid software development and effective reuse of development outcomes [6].

2.2 Web services (WS)

A Web service is an interface that describes a collection of operations that are networkaccessible through standardized XML messaging. A WS is described using a standard, formal XML notion, called its service description. It covers all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols and location.
The interface hides the implementation details of the service, allowing it to be used independently of the hardware or software platform on which it is implemented and also independently of the programming language in which it is written. This allows and encourages Web Services-based applications to be loosely coupled, component-oriented, cross-technology implementations. WS fulfill a specific task or a set of tasks.
They can be used alone or with other Web Services to carry out a complex aggregation or a business transaction. [7]

2.2.1 Why use web services?

Web services facilitate relatively easy reuse and sharing of common logic with clients as diverse as mobile, desktop, and web applications.
The broad scope of web services is possible because they rely on open standards that are ubiquitous, interoperable across different computing platforms, and independent of underlying execution technologies.
All web services, at a minimum, use HTTP and leverage data exchange standards such as XML and JSON, as well as common media types. Beyond that, web services use HTTP in two distinct ways. Some use it as an application protocol to define standard service behaviors. Others simply use HTTP as a transport mechanism to transmit data.
In any case, web services facilitate rapid application integration because, compared to their predecessors, they are generally much easier to learn and implement. Due to their inherent interoperability and simplicity, web services facilitate the creation of complex business processes through service composition. This involves a practice where composite services can be created by assembling simpler services into workflows. [8]

2.2.2 The basic characteristics of a Web services

- Web services utilize XML messaging, defining data exchanges between providers and users. They facilitate cross-platform integration of business applications over the Internet.
- Developers can employ various programming languages and existing components to build them. Unlike HTML-focused presentations, Web services generate XML for universal accessibility.
- They consist of loosely coupled components, each exposing unique functionality as a service. Utilizing industry-standard protocols like HTTP, they're easily accessible through corporate firewalls
- They're compatible with diverse client types, ranging from simple requests to complex transactions.
- Supported by platforms like J2EE, CORBA, and Microsoft .NET, they offer extensive creation and deployment capabilities.
- They're dynamically located and invoked through standardized registries like UDDI and ebXML.

2.2.3 Types of Web Services

Topologically, web services can come in two flavours. Informational, or type I, web services support only inbound operations. As such they always wait for a request, process it and respond. This type is very common and is generally stateless. Complex, or type II web services implement some form of coordination between inbound and outbound operations and are almost always statefull, see Figure 1.

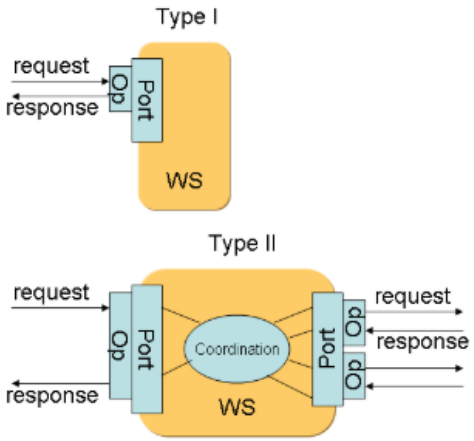


Figure 2.1: High-level view of informational and complex services [9]

1. Informational Services: These services are characterized by their simplicity and involve a series of simple interactions between services that provide access to content (such as content services) or expose backend business applications to other applications outside the firewall (such as business process services). These simple services perform business tasks of the "request-response" type and yield tangible results. Clients of these services can assemble them to build new applications.

2. Complex Services: These services require business partnerships between companies and comprehensive collaboration between operations. This includes exchanging long-term documents and transactions involving advanced security techniques and business process management. Business-to-business collaboration relies on describing business agreements and mutual service execution to complete multi-step business interactions. [9]

2.2.4 Architecture of Web Services

The architecture of web services aims to provide a standards-based platform for SOA (Service-Oriented Architecture). It characterizes a set of specifications that support an open platform based on XML for the description, discovery, and interoperability of distributed and heterogeneous applications in the form of services. [10] A typical Web Services architecture consists of three entities Figure 2: service providers, service users and service brokers (or service registries).

- Service Registry enables an enterprise to describe its businesses, services and rules. Through a registry, businesses describe how they wish to undertake transactions, search for other businesses that provide desired services and integrate with these to undertake a transaction. The API (Application Programming Interface) for registering services is called Universal Discovery and Description Interface (UDDI).

- Service Providers publish their services through brokers who maintain registries that clients can look up.

- Service Users (Human users or agents) search services in registries and invoke these services using a Web Interface. Simple Object Access Protocol (SOAP) is used to pass object information between applications. [11]

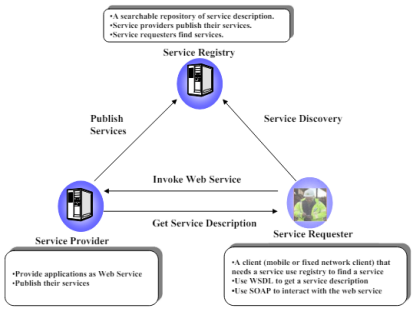


Figure 2.2: Three entities of Web Services architecture [11]

Standards Linked to Web Service Technologies and Protocols

1. Extensible Markup Language (XML):
   is a general-purpose encoding language that organizes data in a structured and machine-readable format. XML's structure allows for extensibility, enabling the definition of different languages with their own vocabularies and rules, such as XHTML, XSLT, RSS, SVG, and more. The primary goal of XML is to facilitate automated exchange of complex contents between diverse information systems or across various locations. [12]

2. The Simple Object Access Protocol (SOAP):
   is a messaging protocol that allows applications to communicate using HTTP and XML. It represents a fundamentally stateless paradigm of one-way message exchange between nodes. By combining one-way exchanges with the functionalities provided by the underlying transport protocol and/or application-specific information, SOAP can be used to create more complex interactions such as request/response, multiple request/responses, etc.
   There are three main types of SOAP nodes:
   SOAP Sender: Generates and transmits a SOAP message.
   SOAP Receiver: Receives and processes the SOAP message and can also generate a SOAP response, message, or error accordingly.
   SOAP Intermediary (Relay or Active): It acts as both a SOAP receiver and sender. It receives and processes SOAP header blocks intended for it and forwards the SOAP message to another SOAP receiver. This process is illustrated in the figure below: [13]

3. Web Service Description Language (WSDL):
   is a standard description language. This is the interface presented to users. It indicates how to use the Web service and how to interact with it. WSDL

Figure 2.3: SOAP nodes [13]

is based on XML and makes it possible to precisely describe details concerning the Web service such as protocols, ports used, operations that can be performed, formats of input messages and output and the exceptions that can be sent.

4. Universal Description, Discovery, and Integration (UDDI) of Web Services:
   is a directory of services. It provides the basic infrastructure for publishing and discovering web services.
   UDDI allows suppliers (companies) to present their services to customers. For example, suppose there are two companies A and B. Company B publishes the services it offers in the directory using a WSDL file.
   A customer of Company A searches the UDDI directory for available services and then downloads WSDL files from that registry. Depending on the information collected in the WSDL files, the client can invoke the web service to obtain the desired information.
   This information can be of three types, as shown in the figure:
   1-The white pages which contain all the information (contact details, contact details or even the description of a company) linked to the supplier (company).
   2-The yellow pages contain the Web services that the company offers with the WSDL standard.
   3-The green pages contain technical and precise information for a particular web service. [12]

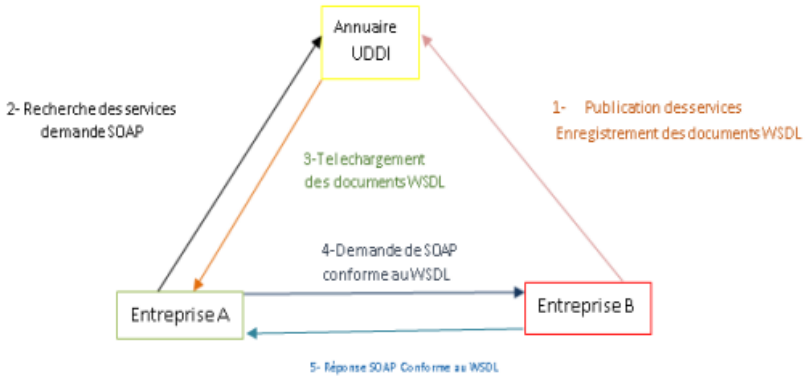


Figure 2.4: Types of the UDDI directory [12]

5. Representational state transfer (REST)
   REST or Representational State Transfer is based on a small set of widely-accepted standards such as Hypertext Transfer Protocol (HTTP), Uniform Resource Identifier (URI), and Extensible Markup Language (XML). REST requires far fewer development steps, toolkits, and execution engines than SOAP. Conceptual model of RESTful shows in Figure 5.

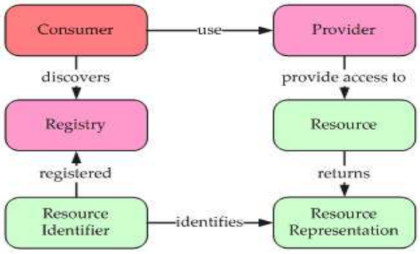


Figure 2.5: RESTful Conceptual Model

A RESTful web service provides access to a resource, identified by UR using HTTP. A resource is an abstraction of information. The HTTP methods, PUT, GET, POST, and DELETE define a uniform interface for accessing resources (i.e. Create, Retrieve, Update, Delete or CRUD). HTTP is stateless request-response application protocol. Request and response messages are comprised of a command (method), a header, and a body. [14]
REST is most often used as a management API for CRUD to implement interaction with resources in lightweight scalable services. A resource is usually a data model object or a database table.
Benefits of REST:
• openness of interaction.
• simple implementation.
• data caching at the HTTP level.
• work with several formats of data presentation.

• stability due to the high level of abstraction.
Disadvantages of REST.
• lack of a unified standardized structure.
• high load on the network.
• excessive or insufficient information, which can lead to the need to send an additional query.
Of all the specifications, REST implements the highest level of abstraction and is best suited for developing simpler CRUD API. It maintains a balance of reliability and ease of use. [15]

6. APIs
The framework security APIs are called explicitly by security-aware applications and implicitly by security-unaware applications via interceptors. Security APIs provide interfaces for access to the framework security services. The framework supports standard, custom, and vendor security APIs.

- Standard security APIs We encourage support for APIs based on open standards or industry de facto standards. Examples of such standards are the J2EE and COM+ security APIs These standards should be used whenever possible because they are likely to provide the most stability and the most portability across many different vendors' products.

- Custom security APIs. Custom APIs may be implemented when an enterprise's needs cannot be met by existing standard APIs. Custom APIs are required especially when an enterprise uses a security service that is tailored to its business, for example, a custom-rule-based entitlements engine developed internally by an investment bank.

- Vendor security APIs. As a last resort, vendor-specific proprietary APIs may be used where open standards have not yet been defined. We recommend avoiding the use of proprietary security APIs in applications if at all possible. Proprietary APIs make it very difficult for the developer or administrator to switch security products. Although vendors may think this is a great idea, we believe that security technology is changing much too rapidly for an enterprise to be confined to any one product. As an alternative, we recommend wrapping a vendor's proprietary API with a standard or custom API. [16]

Comparison of responses and requests from the use of SOAP and REST

To compare the use of SOAP and REST, access to the backend built with SOAP and REST will be carried out. The access used consists of POST, GET and PUT. Testing is done by measuring the response and request from each architecture by accessing the terminal, then the time given to serve the response and request from the client to the server is calculated. Figure 6, shows that when the client requests a request to the server using the terminal.
This Comparison uses a sample of 10 responses and requests with access used consisting of POST, GET and PUT architectures on SOAP and REST. Table II below are the results of the comparison tests carried out.

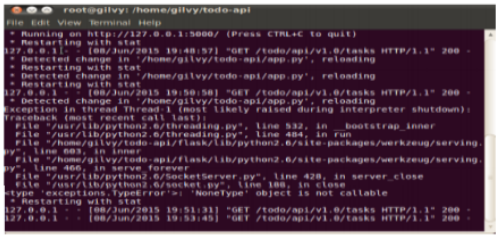


Figure 2.6: GET Response for REST Server [17]

Case Sample	Response And Request	
	SOAP	REST
1	2209	146
2	1809	154
3	1421	267
4	1749	128
5	2311	224
6	1852	265
7	1723	433
8	1982	198
9	2134	566
10	1876	432

Table 2.1: COMPARISON OF RESPONSE AND REQUEST IN SOAP AND REST [17]

From the data obtained in Table 5.1, it shows that for web services that use SOAP and REST, it can be seen that REST has a better performance than SOAP, because it requires faster request and response time for web services. [17]

2.2.5 Advantages and Disadvantages of Web services

Web services offer several key advantages:

- Reusability: Web services allow wrapping existing functionalities and exposing them as reusable services, thus promoting code and resource reuse.

- Location transparency: Web services utilize a registry to store the location of services, enabling clients to find and bind to a service without concern for its physical location, thus offering flexibility in deployment.

- Composition: Developers can assemble applications from reusable, independent services, regardless of the applications in which they are used, thus promoting modularity and component reuse.

- Scalability and availability: Web services can be deployed in scalable and highly available environments, leveraging techniques such as cluster deployment and load balancing, enabling efficient handling of increasing demand.

- Maintenance of investments in existing applications: Web services enable the maintenance of investments in existing applications by wrapping these applications in services, facilitating their replacement or upgrade in the future.

- Reduced vendor dependency: By using open standards, web services allow organizations to choose platforms based on their merits rather than relying on a specific vendor, thus offering greater freedom of choice and flexibility. [18]

Disadvantages:
All new technologies have problems and disadvantages that should be taken into consideration before they are used. To try and assist you with identifying if Web services is appropriate or not, the following list gives some issues you should consider when selecting to use Web services:

- Binding to Web services dynamically requires that the contents of the UDDI registry be trusted. Currently, only private UDDI networks can provide such control over the contents.
- The SOAP server footprint is significant and the technology is relatively new, so adding the Web service provider stack to existing enterprise systems can be a problem.
- Standards for integration of business processes, management of transactions, and the awareness of the policies of interchanging partners are all still under development. To realize the promise of Web services, these types of standards should be available in implementation products. [19]

2.3 web services composition

2.3.1 Definition of web services composition

Indeed, a service composition refers to the process of combining the functionalities of multiple services, either simple or compounded, within a single business process to meet the complex demands that a single service cannot satisfy them. [20] WSC requires a computer program to automatically select, integrate, and invoke multiple Web services in order to achieve a user-defined objective. [21]

2.3.2 The WSC lifecycle

The service composition life-cycle consists of five phases, each contributing to the process of describing, defining, scheduling, constructing, and executing composite services:

1. Planning Phase: In this phase, a formal description of desired service attributes and functionality is created using a service request language. This description includes temporal and non-temporal constraints between services, scheduling preferences, and information from a domain model.

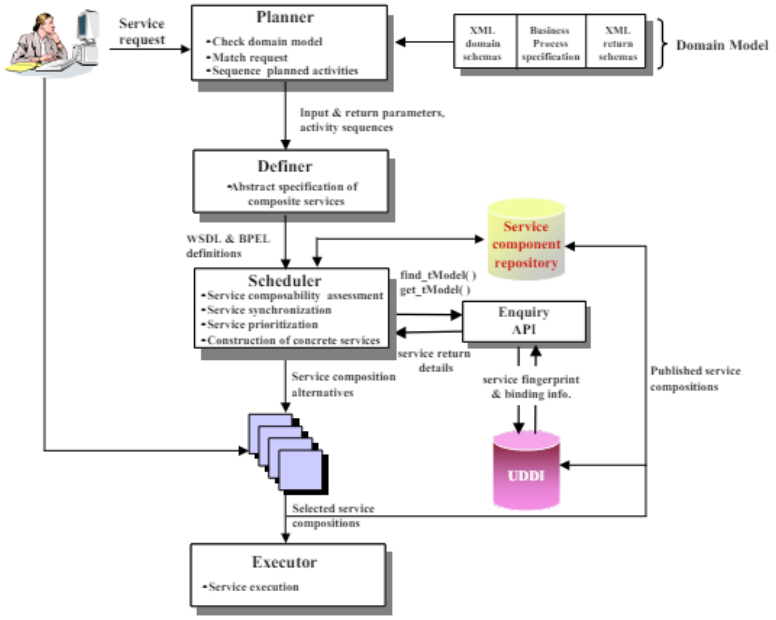


Figure 2.7: Phases involved in an explorative service composition

2. Definition Phase: Composite services are defined abstractly using Web Service Description Language (WSDL) and Business Process Execution Language (BPEL). Abstract service component classes may also be employed during this phase.

3. Scheduling Phase: This phase determines how and when services will run, composing abstract services, assessing their composability and conformance capabilities, correlating messages and operations, and synchronizing and prioritizing their execution.

4. Construction Phase: The outcome of this phase is the construction of a concrete composition of services ready for execution, based on a set of desirable or available constituent services.

5. Execution Phase: Composite service bindings are implemented based on scheduled composition specifications, and the services are executed accordingly.

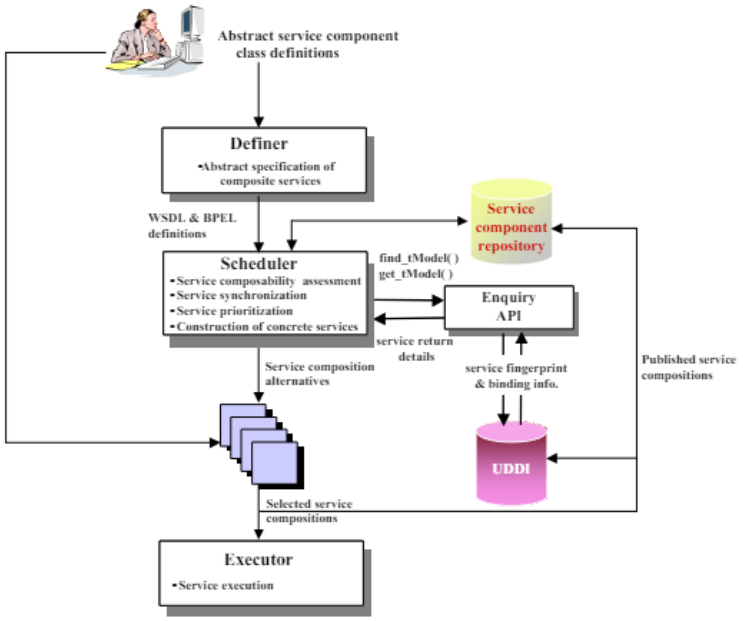


Figure 2.8: Phases involved in semi-fixed and fixed service composition

Figure 8 depicts the service life-cycle phases required for an explorative service composition. We assume that this procedure is initiated by a client who is aware of the format of the input and the output parameters defined in the XML schemas (document model) and the standard business process specifications for a particular domain. The planner module checks the consistency of the request with respect to the business process specification by finding appropriate paths of activities (service operations) potentially satisfying the client request.
If the request is consistent with the domain model specifications, a set of activities is returned for further processing. The planner returns activity paths along with the business process specification that could potentially satisfy the client request. Subsequently, the service definition (definer) module constructs abstract WSDL and BPEL service definitions for the planned activity sequences. These are then passed to the scheduler module. The scheduler needs to interact first with the service providers to be able invoke the service operations specified in the abstract definitions. The scheduler makes the abstract definitions concrete by first appropriately invoking the UDDI enquiry API. For this purpose the scheduler calls the enquiry UDDI operations to find and get detail of the UDDI API to retrieve detailed information about the service port-types, elements and bindings of the services.
The scheduler searches the UDDI for the services required and uses the information found in the UDDI registry to establish the particular invocation pattern needed for the specific service being employed.
Subsequently, the scheduler correlates the constituent services and checks for compatibility. Alternative service compositions, based on non-functional service characteristics, such as performance, security and pricing models, are then

proposed to the client for selection and approval. Finally, once the selected services are made concrete, they are stored in a service repository for future use and passed to the executor module for execution.
Figure 8 depicts the phases required for semi-fixed and fixed service composition that are the two service composition schemes used in conjunction with service components. Semi-fixed and fixed service composition is a much simpler affair when compared explorative service composition. Semifixed and fixed service composition does not require any planning activities as the client provides composite service definitions in the form of abstract service component classes that are further processed by the scheduler. [22]

2.3.3 Orchestration and Choreography

Service orchestration can be seen as the implementation of a behavioral interface. Both consider service composition from the viewpoint of a participant/service.
However, service orchestrations differ in their business objective, as they not only focus on the observable behavior of a web service but also specify the internal steps required for executing the service.
BPEL and BPMN are reference languages for describing executable service orchestrations. Other languages for executable process languages include XLANG, WSFL, XPDL, and BPML. [23]. Service orchestration always represents control from the perspective of a single party (see Figure. 9(a)).
This differs from service choreography, which is more collaborative and allows each involved party to describe its role in the interaction (see Figure. 9(b)).

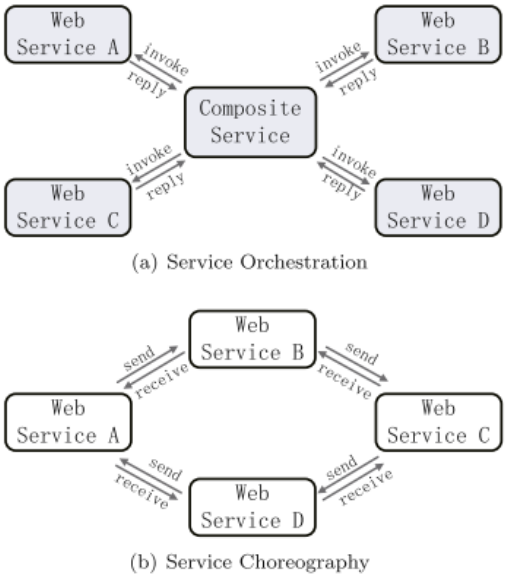


Figure 2.9: Orchestration de service et chorégraphie de service [24]

Choreography represents a global description of the observable behavior of

each participating service in the interaction, defined by the public exchange of messages, interaction rules, and agreements between two or more endpoints of business processes.
Choreography is typically associated with interactions occurring between multiple web services rather than a specific business process executed by a single party. The choreography mechanism is supported by the WS-CDL (Web Services Choreography Description Language) standard. [24].

2.3.4 Types of WSC

Web service composition contains three methods are:

Manual/Static Composition

Building a conceptual work model is required before starting the composition process. This model includes a set of tasks and their data dependencies. Each task is linked to a real web service using a query. This type of composition requires programming expertise and entails high costs, lacking flexibility and consuming significant time.

Automatic/Dynamic Composition

A work model is automatically created and atomic services are selected. This composition requires specifying various constraints including atomic service dependencies and user preferences. Semantic web is utilized to give precise meaning. This approach offers greater flexibility, rapid applicability, and can automatically produce work models.

Semi-automatic/Dynamic or Static Composition

Operates similarly to automatic composition but selects different processes based on varying conditions. Fixed processes and process implementations may be variable, reflecting semi-automatic characteristics. [25]

2.3.5 Modeling service composition

Given the complexity of the web service composition process, it's essential to undergo modeling that addresses the various aspects of this process. The use of tools, techniques, and concepts from task planning domain, such as workflows, is valuable in this perspective. This is justified by the fact that a composed web service can be perceived as a trans-organizational workflow, where processing is distributed across the web and involves multiple complex operations and/or several services. Indeed, the term "process definition" is employed to refer to a composition schema, while "process example" denotes an individual and specific execution (or instance) of a process definition.
Meanwhile, the term "orchestration schema" (or simply orchestration) pertains to the portion of the composition schema specifying the order in which the different composing services must be invoked. To characterize the service composition model, the following aspects need to be defined:

- Component model: This defines the nature of the element to be composed. It includes specifications such as XML messages and web service standards like SOAP and WSDL.
- Orchestration model: This defines the abstractions and languages used to specify the order in which services will be invoked and composed as a whole. There are various possibilities with variations in the formalism used, including: Activity diagrams ,Petri nets ,Pi calculus ,Activity hierarchy ,Rule-based orchestration.
  Each of these aspects plays a crucial role in modeling the service composition process, providing a structured approach to the composition of services while considering their interactions and dependencies.
- Data Model and Data Access Model: It defines the types of data manipulated (application-specific data, flow control data) and how these data are exchanged between different components (Blackboard, explicit data flow).
- Service Selection Model: Defines how links (bindings) are established (static, dynamic by reference, dynamic by query or dynamic selection of operations) and how a specific service is chosen as a component.
- Transactions: The approach used by web services to handle transactions involves allowing the definition of atomic regions in the orchestration schema. An atomic region encompasses a set of activities that adhere to the "all or nothing" property, as depicted in Figure 1.5, where the ACID principles (Atomicity, Consistency, Isolation, and Durability) of traditional transactions are relaxed.
  In the context of web services, some processes are long-running, so resources cannot be blocked for too long. Defining a specific compensation activity for each atomic region will enable the release of the resource at the end of the region. Additionally, each activity is linked to a compensation activity that will be executed in case of problems.

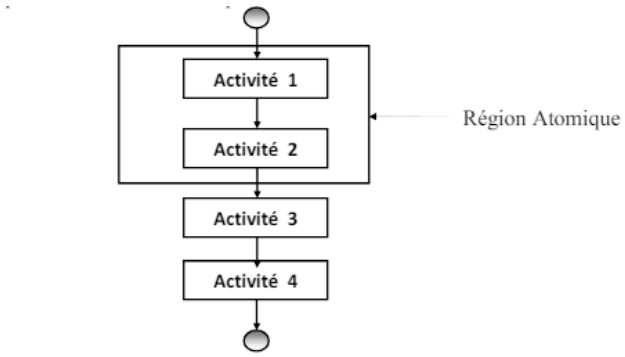


Figure 2.10: Simple Example of Atomic Region

- Exception Handling: Defines how exceptional situations that may arise during service execution are handled without abandoning the composed

service. Exceptions can have different causes such as system errors, errors during application invocation, and rare situations that, despite the clear semantics of the composed service, have not been addressed.
Transactions are a primary solution for exception management, but other modeling and exception handling techniques exist such as: flow-based approach, Try-catch-throw approach, and rule-based approach. [26]

2.3.6 Web service composition standards

Business Process Modeling Language (BPML)

Is a language for the modeling of business processes and was designed to support processes that a business process management system could execute. BPML and WSCI share the same underlying process execution model; therefore developers can use WSCI to describe public interactions among business processes and reserve, for example, BPML for developing private implementations. However, other coordination protocols than WSCI can be adopted.

Web Service Choreography Interface (WSCI)

It is an XML-based interface description language that describes the flow of messages exchanged by a Web service participating in choreographed interactions with other services. WSCI is a coordination protocol, in that it does not address the definition and the implementation of the internal processes that actually drive the message exchange. [27]

Web Service Conversation Language (WSCL)

WSCL is a light-weight interface specification language, with the goal "to define the minimal set of concepts necessary to specify conversations". Like XLANG, WSCL is specifically targeted at public workflow types. The minimalism of WSCL certainly makes the language and any implementation of it very simple, but at the same time restricts expressiveness of WSCL specifications. For instance, concerning the organizational perspective, WSDL limits the number of participants in an interorganizational workflow to two. Regarding the behavioral perspective, WSCL does not support parallel activities nor timing constraints, and transition conditions can only use the result type of a preceding activity for decisions. Despite its limitations, the simplicity of WSCL qualifies the language for a combination with WSDL in order to define stateful services. [28]

XLANG

XLANG is an XML block-structured specification which offers a set of flow control primitives in order to define the process model of the Web service. The flow control primitives organize the operation execution exactly like the different primitives that we meet in programming languages. An XLANG description is always built on one or more WSDL description which supplies a set of operations. It uses their operations as the basic elements in order to construct the processes. An XLANG process is built by applying control primitives on operations and XLANG subprocesses. Every flow control primitive represents

a specific execution order model to the XLANG processes and the WSDL operations according to a specific semantic. In addition to flow control primitive XLANG offers a set of primitives to structure the processes organization by defining an execution context for a set of processes or transactions. [29]

Web Service Flow Language (WSFL)

The Web Services Flow Language (WSFL) is an XML language for the description of Web Services compositions. WSFL considers two types of Web Services compositions:
• The first type specifies the appropriate usage pattern of a collection of Web Services, in such a way that the resulting composition describes how to achieve a particular business goal; typically, the result is a description of a business process.
• The second type specifies the interaction pattern of a collection of Web Services; in this case, the result is a description of the overall partner interactions. [30]

Business Process Execution Language for Web Services (BPEL4WS)

The recently released business process management language BPEL4WS is a high-level XMLbased language for stringing web services calls and other things together. It specifies the behaviour of business processes that make use of web services and of business processes that externalize their functionality as web services. Business Process Management (BPM) even though draws roots from workflow technologies is a standard that attempts to solve a much more general class of problems, allowing also collaborations to be established among multiple independently-managed processes. Therefore, BPEL4WS is a standard-based language and its role is to provide process orchestration, meaning the combination of structured activities, as flow chart of the process to express arbitrarily complex algorithms that represent the implementation of the service. This language aims to the implementation of a process-oriented information system, which is flexible and agile to changes and allows the reengineering of business processes. [31]

2.3.7 Web Service Composition and BPEL

Web service composition involves combining existing web services to create a value-added service. It encompasses two key aspects: specification using a composition language and execution through a suitable runtime environment. The specification phase defines interactions between the composition and composed services, as well as control and data flow around these interactions.
Business Process Execution Language (BPEL) is a process-oriented language for web service composition, where a composite service is implemented using a workflow process. BPEL, recently adopted as an OASIS standard, builds upon BPEL 1.1 and is widely embraced by both research and industry. BPEL processes are deployed on workflow engines, orchestrating invocations of partner services according to the process specification.
In BPEL, key concepts include partners, variables, and activities. Partners represent entities interacting with the composite service, such as clients or other

web services. Partner links define connections between WSDL port types, specifying roles for the composition and its partners. Variables store data exchanged between partners and process-specific data.
Activities are the units of work in BPEL, categorized into primitive (atomic) and structured (composite) activities.
Primitive activities like receive, reply, and invoke handle basic messaging interactions. The assign activity modifies variable content. Structured activities, like sequence and flow, group other activities, organizing them based on control flow patterns such as sequential execution or concurrency. [32]

2.4 Conclusion

In this chapter, the concept of web services and their significance were reviewed, along with elucidating their key characteristics and types. Additionally, an overview of the architectural structure and associated standards was provided. Furthermore, the chapter also addressed the notion of web services composition, including its types and standards.

Chapter 3

Petri Nets

3.1 Introduction

An overview of Petri nets formalism (PN) [33] is presented in this chapter, starting with a basic introduction and then delving into their core concepts and notations. It also explores their applications, advantages, and limitations, along with various extensions. Moreover, we focus on the time extension of this formalism noted Time Petri Nets (TPN) [34] .

Time Petri nets are one of the most widely used model for the specification and verification of real-time systems. They extend Petri nets with temporal intervals associated with transitions, specifying firing delay ranges for the transitions. [35]

3.2 Petri Net model

3.2.1 informal presentation

A Petri net is a graphical tool for the description and analysis of concurrent processes which arise in systems with many components(distributed systems). The graphics, together with the rules for their coarsening and refinement, were invented in August 1939 by the German Carl Adam Petri – at the age of 13 – for the purpose of describing chemical processes, such as Figure 1. [36]

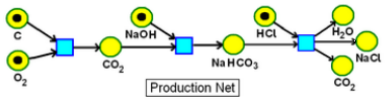


Figure 3.1: Example of a production net [36]

An example of a Petri net is given in Figure 1. Places are typically drawn as circles and transitions as bars or rectangular boxes. [37]

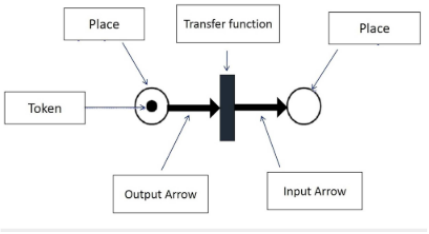


Figure 3.2: A simple example of a Petri net. It has four main parts: places, arrows, functions, and tokens. Tokens are located in places and move among them based on the directions of arrows and function rules [38]

3.2.2 History

The theory of Petri nets has developed from the work of Carl Adam Petri, A. W. Holt, Jack Dennis, and many others. In his thesis in Germany [39], Petri developed a new model of information flow in systems, based on the concepts of asynchronous and concurrent operations by the different parts of a system. Petri's ideas came to the attention of a group of researchers at Applied Data Research, Inc., who were working on the Information Systems Theory Project [40] led by Anatol Holt.
Holt developed the theory of "systemics" [41], which was concerned with the representation and analysis of systems and their behavior. This early theory helped provide the notation and representation for Petri nets and showed how Petri nets could be applied to the modeling and analysis of systems of concurrent processes.

3.2.3 formel presentation

A classical Petri net (PN) is a directed bipartite graph with two types of nodes, called places and transitions. The nodes are connected via directed arcs and connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by rectangles.

PN Syntax

Formally, the syntax of the PN model is defined as follows:

Definition 1 *An PN is given by the tuple (P, T, B, F, M_0) where:*

- *P and T are respectively two non empty disjoint sets of places and transitions;*
- *; B and F are respectively the backward and the forward incidence functions $B : P \times T \longrightarrow N = \{0, 1, 2, ..\}$; $F : P \times T \longrightarrow N$;*
- *M_0 is the initial marking function that associates with each place a number of tokens $M_0 : P \longrightarrow N$;*

PN Semantics

Before laying down the formal semantics of the model. We introduce, first, some notations:
Let $R := (P, T, B, F, M_0)$ be a PN and

- We call a marking the function, noted M that associates with each place a number of tokens, $M : P \longrightarrow \mathbb{N}$;

- A transition t is said to be *enabled* for the marking M, if $\forall p \in P, B(p,t) \leq M(p)$; the number of tokens in each input place of t is greater or equal to the valuation of the arc connecting this place to the transition t. Thereafter, we denote by $E(M)$ the set of transitions *enabled* for the marking M.

- Let M be a marking; two transitions t_i and t_j enabled for M are said to be *conflicting* for M, if $\exists p \in P, \quad B(p,t_i) + B(p,t_j) > M(p)$. Hence, we note $Conf(M)$ the relation built on $E(M)^2$ such that $(t_i, t_j) \in Conf(M)$, iff t_i and t_j are in conflict for the marking M.

PN Marking Graph

If the Petri net is structurally limited and its state space is limited (i.e. its markings are finite) it is possible to construct the marking graph for the net. Different algorithms can be used to construct the marking graph. In simple terms the marking graph represents all the possible states of the Petri net. The reachability tree or better known coverability tree for a restricted Petri net is easily constructed. The directed graph obtained from the Petri net can be used for different forms of analysis, which is often overlooked. This type of graph can become quite large if the Petri net has over one hundred states. A possible solution is to reduce the Petri net model and eliminate ambiguity.
The marking graph or reachability tree is a simple directed graph or digraph where the nodes or vertices represent a marking whilst the directed edges represent the transitions used to reach a particular marking. The reachability tree can be drawn as a marked directed graph of the form G = (V, E), where E = edges representing transitions and V= States or markings. There are various forms of the marking graph having different names but in essence they are similar. The delay of a transition can be represented on the edges. For the marking graph an adjacency matrix can be constructed.
Examples:
Some simple examples illustrating the four conversion methods are given. A specific Petri net is taken and a corresponding directed graph is constructed using each one of the four methods. It can be assumed that the structural reduction rules previously defined have been applied as required. These are quite simple to comprehend and are self explanatory. Note that the resulting graph can obviously be drawn as required, i.e. the node or edge layout could be drawn aesthetically in different ways for visualization e.g. using rounded or flat edges, circles for nodes, etc. Figure 3.1(a) shows a Petri net, complete with its marking graph Figure 3.1(b) and a compacted or reduced form of marking graph Figure 3.1(c).
Below the reduced marking graph the adjacency matrix for the marking graph has been given. For the marking graph in Figure 3.1(a), the adjacency matrix is easily constructed. It is assumed that edges represent transitions. For

the marking graph in 3.1(a) the adjacency matrix can be constructed, given in 3.1(d).

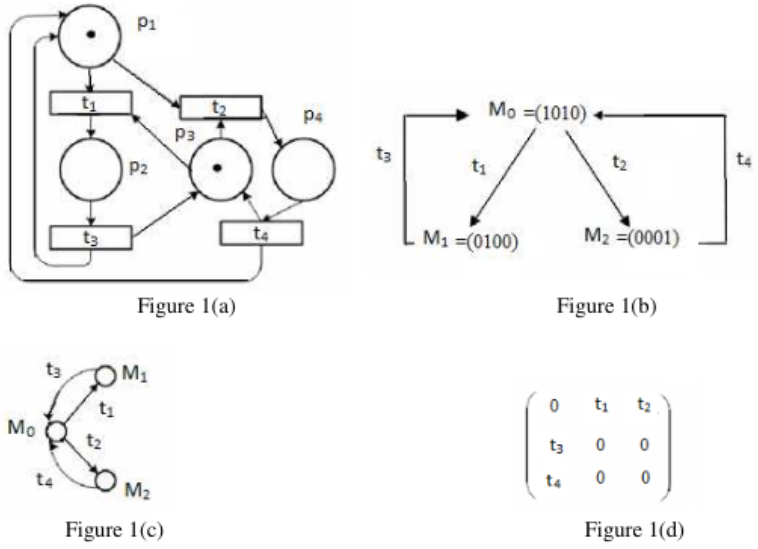


Figure 3.3: simple examples of marking graph

3.2.4 Petri Nets properties

Petri nets as mathematical tools possess a number of properties. These properties, when interpreted in the context of the modeled system, allow the system designer to identify the presence or absence of the application domain specific functional properties of the system under design. Two types of properties can be distinguished: behavioral and structural properties.

The behavioral properties are these which depend on the initial state, or marking, of a Petri net.
The structural properties, on the other hand, do not depend on the initial marking of a Petri net. These properties depend on the topology, or net structure, of a Petri net. In this section, we provide an overview of some of the most important, from practical point of view, behavioral properties. The focus on the behavioral properties is dictated by the space limitations of this tutorial. An extensive description of the structural properties, and the analysis methods can be found in [42]. The behavioral properties discussed in this section are reachability, boundedness, conservativeness(bounded), liveness, reversibility and home state. Descriptions of other properties such as coverability, persistence, synchronic distance, and faimess can also be found in [42].

Reachability

An important issue in designing distributed systems is whether a system can reach a specific state, or exhibit a particular functional behavior. In general, the question is whether the system modeled with Petri nets exhibits all desirable properties, as specified in the requirements specification, and no undesirable ones.
In order to find out whether the modeled system can reach a specific state as a result of a required functional behavior, it is necessary to find such a sequence of firings of transitions which would result in transforming a marking M_0 to M_i, where M_i represents the specific state, and the sequence of firings represents the required functional behavior. It should be noted that real systems may reach a given state as a result of exhibiting different permissible patterns of functional behavior.
In a Petri net model, this should be reflected in the existence of specific sequences of transitions firings, representing the required functional behavior, which would transform a marking M_0 to the required marking M_i. The existence in the Petri net model of additional sequences of transitions firings which transform M_0 to M_i indicates that the Petri net model may not be reflecting exactly the structure and dynamics of the underlying system.
This may also indicate the presence of unanticipated facets of the functional behavior of the real system, provided that the Petri net model accurately reflects the underlying system requirements specification. A marking M_i, is said to be reachable from a marking M_0 if there exists a sequence of transitions firings which transforms a marking M_0 to M_i. A marking M_1 is said to be immediately reachable from a marking M_0 if a firing of an enabled transition in M_0 results in marking M_1. For instance, in the Petri net model of the multirobot assembly system, the state in which robot arm R1 performs tasks in the common workspace, with robot arm R2 waiting outside, is represented by the marking vector $M_1 = (0, 1, 0, 1, 0, 0, 1, 3, 0)^T$. M_i can be reached from the initial marking M_0, where $M_0 = (1, O, 0, 1.0, 0, 1, 3)$ by the following sequence of transitions firings–$t_1t_2t_4$. The marking $M_1 = (0, 1, 0, 1, 0, 0, 1, 3, 0)^T$, which represents the state of the system in which robot arm R1 waits for the access to the common workspace and robot arm R2 performs tasks outside the commom workspace, is immediately reachable from the initial marking M_0 when transition t_1 fires. It should be noted that in M_0 transitions t_1, and t_4 are both enabled. The set of all possible markings reachable from M_0 is called the reachability set. and denoted by R(M_0). Thus the problem of identifying the existence of a specific state M_i, the system can take on, can be redefined as the problem of finding if $M_i \in R(M_0)$. [42]

Boundedness and Safeness
In a Petri net, places are often used to represent information storage areas in communication and computer systems, product and tool storage areas in manufacturing systems, etc. It is important to be able to determine whether proposed control strategies prevent from the overflows of these storage areas. The Petri net property which helps to identify the existence of overflows in the modeled system is the concept of boundedness.
Example:
The petri net in example is 2-bounded show figure 4. [**?**]

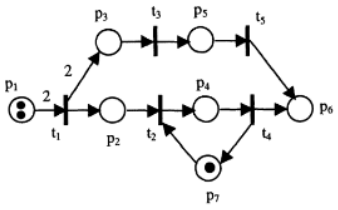


Figure 3.4: example is 2-bounded [?]

Conservativeness

Tokens in a Petri net may represent resources. The number of which in a real system is typically fixed, then the number of tokens in a Petri net model of this system should remain unchanged irrespective of the marking the net takes on. When Petri nets are used to represent resource allocation systems, conservation is an important property.

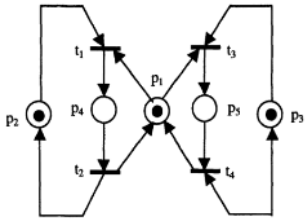


Figure 3.5: A manufacturing system's Petri net model which is not conservative [?]

Liveness

The concept of liveness is closely related to the deadlock situation, which has been situated extensively in the context of computer operating systems.
A Petri net modeling a deadlock-free system must be live. This implies that for any reachable marking M, it is ultimately possible to fire any transition in the net by progressing through some firing sequence. This requirement, however, might be too strict to represent some real systems or scenarios that exhibit deadlock-free behavior. For instance, the initialization of a system can be modeled by a transition (or a set of transitions) which fire a finite number of mes. After initialization, the system may exhibit a deadlock-free behavior, although the Petri net representing this system is no longer live as specified above. For this reason, different levels of liveness for transition t, and marking Mo, were introduced.

Example :
The Petri net shown in Figure 6 is strictly L1-live since each transition can be fired exactly once in the order of t_2, t_4, t_5, t_1 and t_3. The transitions t_1, t_2, t_3 and t_4 in Figure 7 are LO-live (dead), L1-live, L2-live and L3-live, respectively. [?]

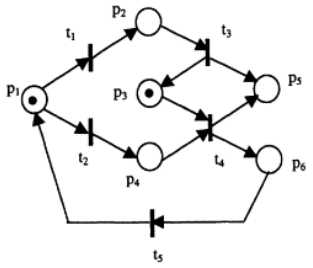


Figure 3.6: A nonlive Petri net. But it is strictly L1-live [?]

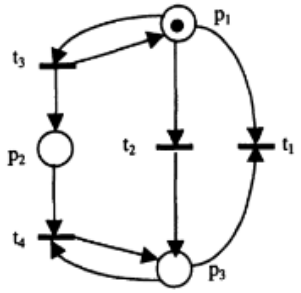


Figure 3.7: Transitions t_1, t_2, t_3 and t_4 are dead (LO-live), LI-live,L2-live, and L3-live, respectively [?]

Reversibiliiy and Home State

An important issue in the operation of real systems, such as manufacturing systems, process control systems, etc., is the ability of these systems for an error recovery. These systems are required to return from the failure states to the preceding correct states. This requirement is closely related to the reversibility and home state properties of a Petri net. A Petri net, for the initial marking M_0, is said to be reversible if for each marking M in $R(M_0)$, M_0 is reachable from M. The home state property is less restrictive, and more. practical, then the reversibility property of ;I Petri net. A Petri net state M; is said to be a home state if for each marking M in $R(M_0)$, $R(M_i)$ is reachable from M. The Petri net shown in Figure 8 is reversible. The Petri net shown in Figure 9 is nonreversible. [42]

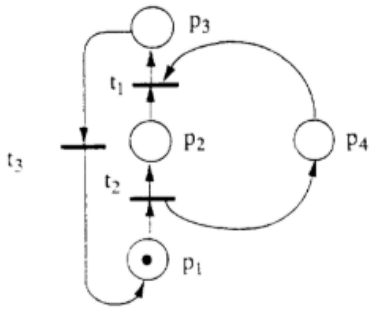


Figure 3.8: The Petri net is reversible [42]

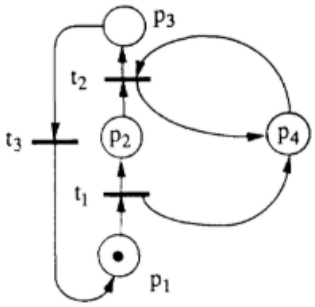


Figure 3.9: The Petri net is nonreversible [42]

3.3 Time Petri Net model [43]

Time Petri Net (TPN) is a temporel extension of the standard (PN) where transitions are labelled by two temporal constraints that indicate their earliest and latest firing times.

3.3.1 TPN Syntaxe

Definition 2 *An TPN is given by the tuple (R, I_s) where: R is a Peti Net;*

I_s is the delay interval mapping function; $I_s : T \longrightarrow Q^+ \times Q^+ \cup \{+\infty\}$, where Q^+ is a set of null or positive rational values. We write $I_s(t) = [x_0(t), y_0(t)]$. This gives the static time interval within which the transition t can fire, such that $0 \leq x_0(t) \leq y_0(t)$;

3.3.2 TPN Example

Let us consider the PN example shown in Figure 3, presented in [44, 45]

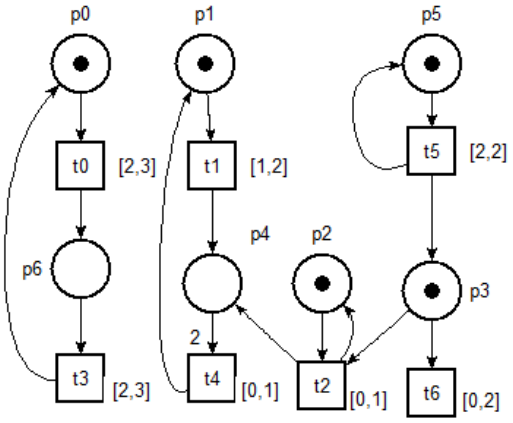


Figure 3.10: A TPN example [45]

3.3.3 TPN Formal Semantics

Definition 3: *A reachable state, noted e, of a TPN is a tuple* $e = (M, V)$, *where*
M *is the reachable marking and*
V *is the function that associates with each enabled transition its dynamic firing interval* $V(t) := [x(t), y(t)]$. *We note* $e_0 = [M_0, V_0]$ *the initial state, where:* $V_0(t) := [x_0(t), y_0(t)]$.

The initial state of our TPN example is given by (M_0, V_0) such that $V_0 : t_0 \mapsto [2, 3]; t_1 \mapsto [1, 2]; t_2 \mapsto [0, 1]; t_5 \mapsto [2, 2]; t_6 \mapsto [0, 2]$.

Important: le reachability graph for a TPN model needs more complex algorithms then the standard Marking graph becacuse of the temporal constraints added to transitions. Moreover, the obtained graph is in general infinit (combinatory explosion). An important number of methods were proposed in the litterature to investigate this problem, for more detail reed [46], [47]. In our work, we are not investigated each algorithm, we are only using them throw TINA TOOL and discuss the advantages and desavantages and make comparaison (see chapter case study).

3.4 PN and TPN APPLICATIONS

The scope of using Petri nets is broad, spanning various domains and applications due to their versatility and expressive power. Here are some key scopes of use for Petri nets:

3.4.1 Computer Science and Software Engineering

- Modeling and analyzing concurrent and distributed systems.
- Designing and verifying communication protocols.
- Specifying and simulating software workflows and business processes.
- Detecting deadlocks and race conditions in concurrent programs.
- Modeling and analyzing real-time systems.

3.4.2 Manufacturing and Operations Management

- Modeling and optimizing manufacturing processes, such as assembly lines and production systems.
- Scheduling and resource allocation in manufacturing environments.
- Analyzing workflow and logistics in supply chain management. Designing and controlling automated systems and robotics.

3.4.3 Biological Systems and Bioinformatics

Modeling biochemical reactions and metabolic pathways. .Analyzing gene regulatory networks and signal transduction pathways. .Studying cellular processes, such as cell division and apoptosis. .Predicting the behavior of biological systems under different conditions.

3.4.4 Telecommunications and Networking

- Modeling and analyzing network protocols and architectures.
- Evaluating the performance and reliability of communication networks.
- Designing and optimizing routing algorithms.
- Simulating and testing network congestion control mechanisms.

3.4.5 Business Process Management

- Modeling and optimizing business processes and workflows.
- Analyzing resource utilization and bottlenecks in organizational systems.
- Identifying opportunities for process improvement and automation.
- Supporting decision-making in workflow management systems.

3.4.6 Embedded Systems and Cyber-Physical Systems

- Modeling and verifying embedded control systems.
- Designing and analyzing cyber-physical systems, such as smart grids and autonomous vehicles.
- Ensuring safety and reliability in safety-critical systems.
- Simulating and testing control algorithms and feedback loops.

3.4.7 Education and Research

- Teaching concepts of concurrency,
- formal methods, and system modeling.
- Conducting research in the fields of formal verification,
- discrete event systems, and concurrent programming.
- Developing new methodologies and extensions for Petri nets to address specific application domains.
- Collaborating with interdisciplinary teams to apply Petri nets in novel contexts and problem domains.

3.5 Advantages Timed Petri Nets

Timed Petri nets offer the following advantages:
Suitable and precise framework: Provides a suitable and precise framework for accurately describing dynamic systems and generating basic statistical processes reliably.
Simplicity and power: Relies on few but powerful concepts, making it easy to learn and develop strong analytical methods.
Automation capability: Analysis of timed Petri nets can be automated, with several software tools available for this purpose.
Modeling accuracy: Allows for precise representation of non-product form features such as priorities, synchronization, branching, and blocking.
Logical and quantitative models: Can be used for both logical and quantitative modeling, allowing for the specification and validation of functional/logical properties as well as performance properties using the same modeling language.
Simulation readiness: Serves as a ready simulation model when state explosion problems occur or when underlying statistical models are not amenable to tractable mathematical analysis, facilitating simulation for both logical and quantitative property analysis.

3.6 Conclusion

In conclusion, we reaffirm the importance of Petri nets as an effective tool for modeling and analyzing dynamic systems, where we provided their definition, practical examples of their usage, and analyzed the areas of their application. We also explored the versatile and robust nature of these networks, shedding light on their computational and analytical capabilities.

Petri nets have constituted a powerful tool enabling researchers, engineers, and practitioners to tackle complex challenges and design efficient and reliable systems. Additionally, we briefly introduced timed Petri nets, providing an explanation along with some examples and characteristics, thus deepening our understanding of this valuable tool and its diverse applications across various fields. We will see in next chapter the importance of TPN for web service modelling.

Chapter 4

Web Services Composition Modeling

4.1 Introduction

The idea behind web service composition is that many sub tasks, already defined as web services, can be used together to accomplish a larger task. Realization of this larger task will be the resulting composite web service. [48] There are various approaches to modeling the processes of web service composition, including thoses based on Petri nets formalism. In this chapter, we present the modelling methods used for web service composition, the first part is dedicated to methods based on PN model, the second part present other methods found in the literature. Finally we give a formal approach based on TPN model modeling web service composition used in the case study.

4.2 WSC modeling based on PN formalism

In this section we give a related work on WSC modeling based on PN formalism and its extensions.

4.2.1 Time petri net based modeling of web service architecture

This model represents an architecture comprising M servers Ser_i and one federation server (Ser) for handling composite web service requests. Requests arrive periodically within a defined time interval. Each request invokes M elementary web services (Req_i), with each service being handled by its corresponding server. The response time of each elementary service is within a specified range. After all sub-requests are executed, the results are merged by the federation server to generate the final response within a defined time frame.

Definition

Time Petri Nets (TPNs) are used in computer science to model systems such as parallel processing, distributed systems, and computer networks due to their intuitive and efficient nature. TPNs consist of places (circles) representing events or resources, and transitions (rectangles) representing actions or processes. Each place holds tokens indicating its current state or availability.
In TPNs with multi-server semantics, multiple instances of a transition can be enabled simultaneously, reflecting concurrent activity. This extends the modeling capability to scenarios where parallelism or concurrency is significant.
TPNs can include read arcs and priority arcs for enhanced modeling. Read arcs, depicted with a full disk, allow modeling of priorities by inhibiting a transition until a specific place has sufficient tokens. Priority arcs from one transition to another enforce sequencing constraints, ensuring a transition cannot occur until another has completed.
Firing a transition in TPNs consumes tokens from its input places connected by regular arcs, reflecting the consumption of resources or fulfillment of prerequisites for the action.
Overall, TPNs leverage these features to provide a powerful framework for analyzing system behaviors and verifying properties such as timing constraints and resource allocation in complex systems.

Example

The basic architectural modeling is presented through a Time Petri net as shown in Figure 1. For clarity, only two servers are depicted among the M servers available.
The place "Req," marked with one token, represents the number of composite requests generated in each period. The transition "Arrival" signifies the arrival of a composite request into the system. The places $AdmSer_1$,.., $AdmSer_m$ represent the admission of the elementary queries of the composite request. Elementary queries are modeled here by tokens produced following the firing of the transition Arrival. The execution of each elementary request is performed in mutual exclusion by the related server. We use the place Me_1, ..Me_m to design this mechanism. Each firing of respectively the transitions Ser_1,..,Ser_m needs to consume the oldest token in respectively the places $AdmSer_1$,.., $AdmSer_m$ and produces a token in respectively the place Res_1, ..Res_m; this denotes the results of handling the m sub-request. Finally, the firing of the transition merge denotes the execution of the federation component which requires to consume the oldest token in each place of Res_1, ..Res_m once they are all provided. The marking of the place NReQSuc denotes the number of composite requests that have been successfully handled by the system. [49]

4.2.2 Modeling web services using G-nets

A model for designing web services using G-Net utilizes Petri Nets to represent the behavior of Discrete Event Dynamic Systems. G-Net consists of autonomous modules interconnected in a loosely coupled manner, resembling a distributed system. Each operation within the service corresponds to a method in G-Net,

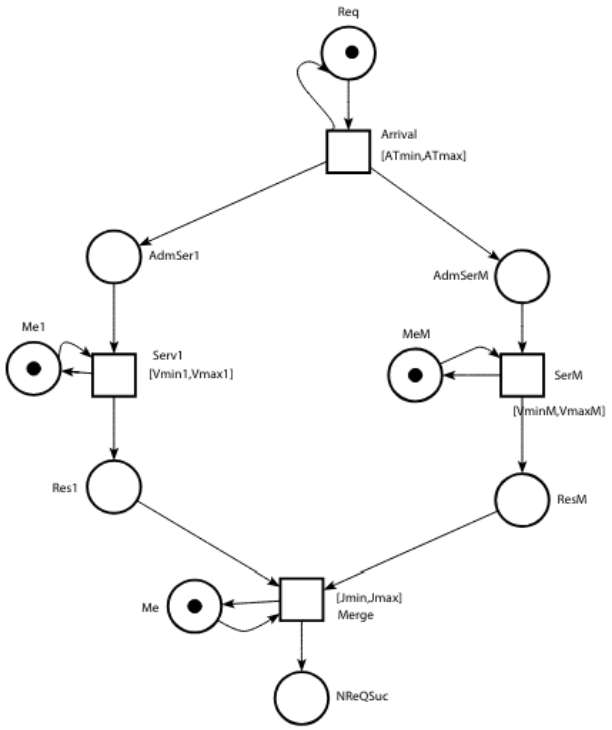


Figure 4.1: TPN specification modeling the basic architecture [49]

linked to a Petri-Net segment in its Information System (IS). Consequently, the state of the service is represented by the positions of tokens within G-Net.

Definition

4.2.3 Definition 1. (G-net Service)

A G-net service is a G-Net S (GSP,IS) where:

- GSP (MS, AS) is a special place that represents the abstraction of the service where:

  - MS is a set of executable methods in the form of `< MtdName > < description >={[P1: description; ...; Pn: description]` (`< InitPL >`) (`< GoalPls >)}` where < MtdName > and < description > are the name and the description of the method respectively. < P1 : description; .....; Pn : description > is a set of arguments for the method, < InitPL > is the name of the initial place for the method and < GoalPls > is(are) the name(s) of the goal place(s) for the method.
  - AS is a set of attributes in the form of <attribute-name >= < type > where <attribute-name> is the name of the attribute and <type>is the type of the attribute.

- IS (P,T,W,F,Trc, Tra, L) is the internal structure of the service, a modified predicate/transition net, where:
  - $P = NP \cup ISP \cup GP$ is a finite and non empty set of places where NP is a set of normal places denoted by circles, ISP is the set of instantiated switch places denoted by ellipses used to interconnect G-Nets, GP is the set of goal places denoted by double circles used to represent final state of method's execution.
  - T is a set of transitions.
  - W is a set of directed ares $W \subseteq (P \times T) \cup (T \times P)$.
  - F is an application that associates a description to certain elements of W.
  - Trc is an application that associates a condition to certain transitions (called selectors of transition), this condition is a logical formula constructed from variables which appeared in the inscriptions of adjacent input-ares.
  - Tra is an application that associates an action to certain transitions, this action is a sequence of affectations of values to variables.
  - $L : P \rightarrow O \cup \{\tau\}$ is a labeling function where O is a set of operation names and τ is a silent operation.

Definition 2. (Web Service)

A Web service is a tuple where:

- NameS is the name of the service used as its unique identifier.
- Desc is the description of the provided service. It summarizes what functionalities the service offers.
- Loc is the server in which the service is located.
- URL is the invocation of the Web service.
- CS is a set of the component services of the Web service, if CS = {NameS} then S is a basic service, otherwise S is a Composite service.
- SGN = (GSP,IS) is the G-Net modeling the dynamic behavior of the service.

The concept of G-Net service and Web service being presented, we show in the next section how Web services

can be incrementally composed. We recall that we use G- Nets as a means to offer a flexible and powerful algebra.

Example

In Figure 2, there is an illustrative example of two services: "Customer" and "Automatic Cash Dispensers (ACD)," represented by G-Nets. The "Customer" represents the behavior of an individual who wishes to use the automatic cash dispenser for operations such as balance inquiries and cash withdrawals. This requires a valid magnetic card and a secret code. Two cases to consider:

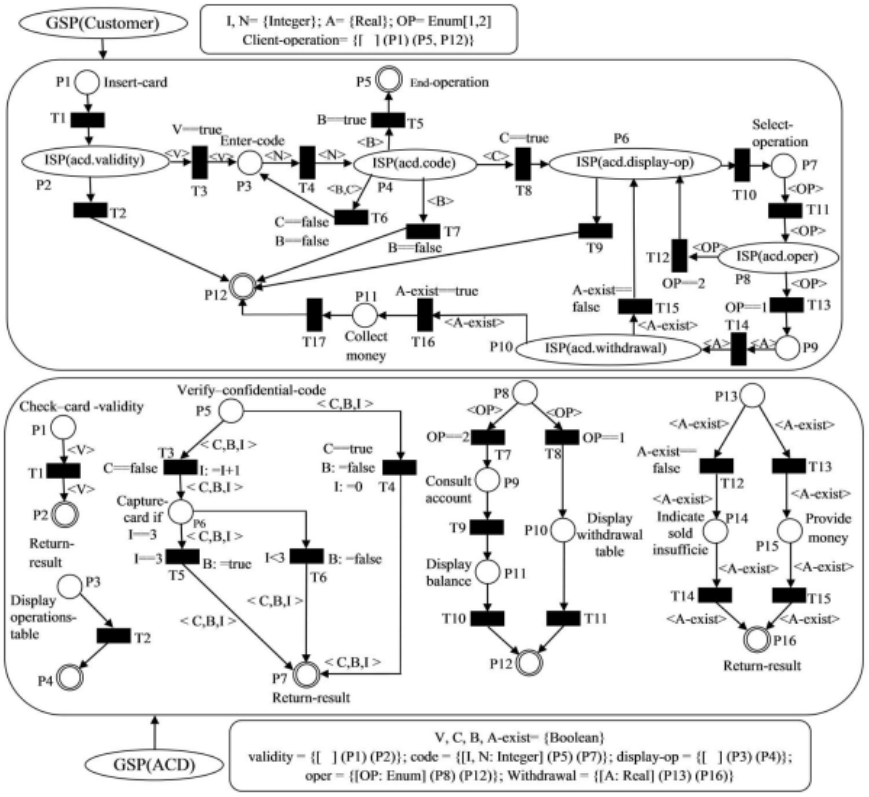


Figure 4.2: Example of G-net Services [50]

- If it concerns an application for credit, the machine displays the account balance, whereupon it redisplays the operations table.
- If it concerns funds withdrawal application, the machine displays a window related to this operation, which allows the client to introduce the sum he wants to withdraw. After the validation, the validity of the sum is automatically checked by the machine. If the sum is not sufficient; the machine would make an indication on the screen, it redisplays the operations table. In the case of the availability of funds, it will distribute the banknotes representing the inserted sum.

However, If the confidential code is entered incorrectly three times consecutively, the ACD will capture the magnetic card for security reasons, as it may indicate that the card user is unauthorized. The retrieval of the card involves special processing not detailed in this example. Additionally, the customer has the option to cancel any operation at any time before validation.
The ISP notation facilitates specifying the interconnections between different G-Nets. In Figure 1's example, ISP integrates ACD's methods (validity, code, display-op, oper, withdrawal) into Customer's IS to define a client-server relationship.

The significance of the using attributes:

- V: indicates either the card is valid or not.
- C: indicates either the confidential code is valid or not.
- B: indicates either the card is blocked or not.
- A-exist: indicates either the sum to withdraw is available in the account or not.
- I: represents the number of successive errors during the introduction of the confidential code (at each incorrect attempt the machine increment by 1 the number I, if the introduced code is accepted the machine affect the value 0 to I).
- N: represents the confidential code.
- A: represents the sum to withdraw.
- OP: represents the type of the operation (withdraw if OP==1 or credit consultation if OP==2). [50]

4.2.4 Petri nets model Timed Mop-ECATNet

The elementary Mop-ECATNets model integrates time constraints, but solely in the form of timeouts. Elapse of timers triggers adaptation processes modifying the structure of the lower-level controlled net.
Furthermore, Mop-ECATNets do not address the fact that the controlled net may contain alternative executions that fulfill the required functionality.

Definition

Mop-ECATNets enhance Petri nets by incorporating complex data structures and synchronization constraints through distinct condition-action distinctions in firing transitions. Timed Mop-ECATNets extend this further with timed patterns, where Meta places control and monitor transitions of lower-level nets. These nets associate waiting durations with Meta place tokens and firing durations with lower-level transitions.
The essential meaning of these patterns is as follows:
• The pattern of Figure 3.(a) Meta transitions fire only when all input Meta places' tokens are unavailable, controlling alternative tasks.
• The second pattern of Figure 3.(b) Meta transitions control parallel tasks, firing when at least one input Meta place token becomes unavailable.
• The pattern of Figure 3.(c) Sequential task control: Meta transitions fire if an input Meta place token becomes unavailable before completing all tasks.
• In the pattern of Figure 3.(d), Iterative task control: Meta transitions fire if an input place token becomes unavailable, regenerating another token in the same input Meta place.

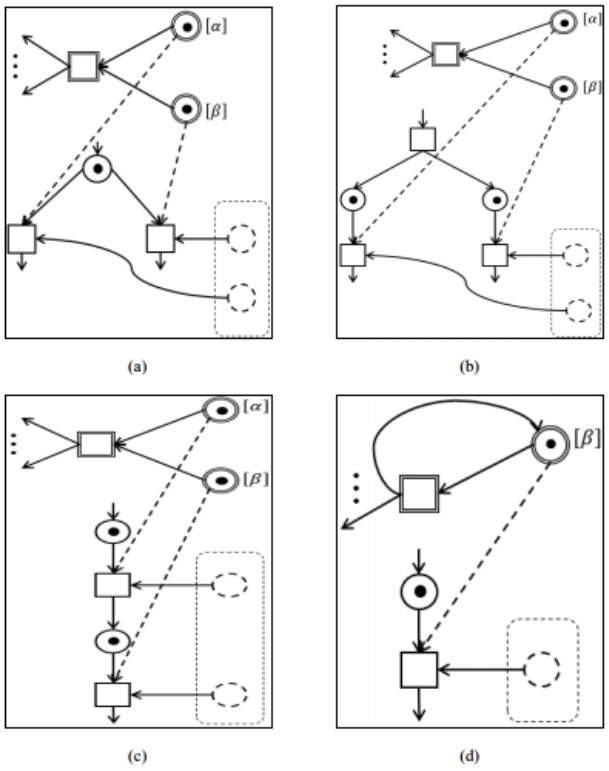


Figure 4.3: Timed Mop-ECATNets flexibility patterns, (a) timed Mop-ECATNet control pattern of alternative tasks (b) timed Mop-ECATNet control pattern for parallel tasks (c) timed Mop-ECATNet control pattern of sequential tasks (d) timed Mop-ECATNet control pattern of iterative tasks [51]

Example

Let's consider the GPS navigation web service, its Timed Mop-ECATNet model is depicted in Figure 4. Meta places are annotated with waiting durations and transition of invoked web services are annotated with treatment duration. Send and receive operations are modelled by the regular transitions of the lower level component. The execution of GPS service starts by firing the transition t1 that represents the reception of client request, then the firing of the transition t2 launches two parallel execution threads to calculate, the GPS coordinates and the kind of travel information. In case of failure, the meta transition mt is fired and a third alternative web service is invoked to determine GPS coordinates and the kind of travel information (the transition ti4). Finally, the process is concluded by invoking the service that returns travel information after receiving the GPS coordinates and the kind of travel information. [51]

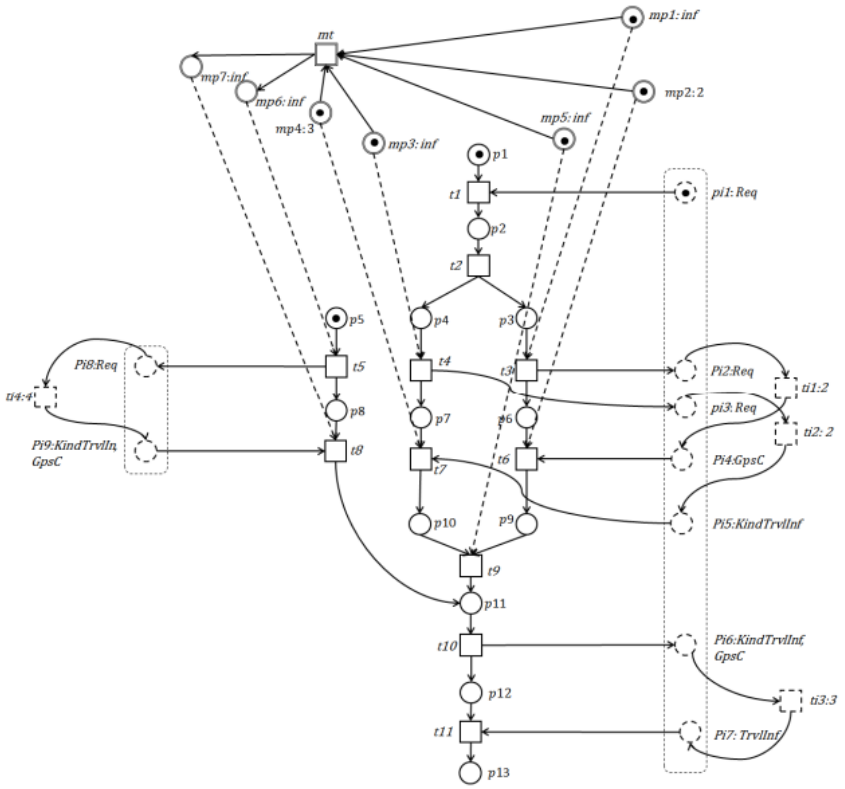


Figure 4.4: The GPS navigation as a timed Mop-ECATNet [51]

4.2.5 Petri net-based algebra for modeling Web services

The presented algebra provides a systematic framework for generating enhanced Web services by leveraging pre-existing ones as foundational components.

Definition

They describe the syntax and informal semantics of service algebras below. These architectures were chosen to enable compilations of common and advanced web services. The service set can be defined as follows in a BNF-like technique:

$$\begin{aligned} S ::= \quad & \epsilon \mid X \mid S(\odot\cdot)S \mid S \oplus S \mid S \diamond S \mid \mu S \\ & \mid S \mid\mid c\ S \mid (S \mid S) \rightsquigarrow S \\ & [S(p,q) : S(p,q)] \mid \mathrm{Ref}(S,a,S)\mathrm{Ref}(S,a,S) \end{aligned}$$

where:

ϵ represents an empty service, i.e, a service which performs no operation.

• X represents a service constant, used as an atomic or basic service in this context.

• S1 $\odot\cdot$ S2 represents a composite service that performs the service S1 followed by the service S2, i.e.,

is an operator of sequence.

$$S1 \oplus S2$$

• S1 $\oplus S2$ represents a composite service that behaves as either service S1 or service S2. Once one of them executes its first operation the second service is discarded, i.e., is an alternative (or a choice) operator.
$S \diamond S$ represents a composite service that performs either the service S1 followed by the service S2, or S2 followed by S1, i.e., $\diamond$ is an unordered sequence (or an arbitrary sequence) operator.
• µS represents a service that performs a certain number of times the service S, i.e., µ represents an iteration operator.
• S1 ||c S2 represents a composite service that performs the services S1 and S2 independently from each other with possibilities of communication over the set C of pairs of operations, that is, kC is a parallel operator with communication.
•(S1|S2) $\rightsquigarrow S3$represents a composite service that waits for the execution of one service (among the services S1 and S2) before activating the subsequent service S3, i.e., $\rightsquigarrow$ is a discriminator operator.
Note that S1 and S2 are performed in parallel and without communication.
$[S1(p1, q1) : Sn(pn, qn)]$ is a composite service that dynamically selects one service provider among n available services $S1, ..., Sn$ and executes it. It behaves as follows: first a request is sent by a composer to n available service providers of a given trading community through their entry access points $p1, ..., pn$. Then based on the received responses, from their exit access points $q1, ..., qn$, and according to given ranking criteria (e.g. price, delivery date/time, or a combination of both) the best service provider is chosen. Finally the needed operations are performed. [:] is an operator of selection.
• Ref(S1, a, S2) represents a composite service that behaves as S1 except for operations in S1 with label a that are replaced by the non empty service S2. Ref is a refinement operator. The proposed algebra verifies the closure property. It guarantees that each result of an operation on services is a service to which we can again apply algebra operators. We are thus able to build more complex services by aggregating and reusing existing services through declarative expressions of service algebra.

Example

Figure 5 shows a Web service composed of three basic services, OCS representing an Online Computer Store and SM and IP, representing respectively the Sony Monitors and the Intel Processors. Upon reception of an order rec ord PC for a computer from a customer, OCS starts, in parallel, the outsourced services SM to order a monitor and IP to order a processor by performing the operations send ord mon and send ord pr respectively. The set of communication elements are C1 = f(send ord mon, rec ord mon), (send del mon, rec del mon)g and C2 = f(send ord pr, rec ord pr), (send del pr, rec del pr)g. Once the requested items are received, OCS performs the assemble PC operation. Note that, for the sake of simplicity and clarity, not all the operations of the scenario (e.g, delivery and billing) are represented and labels are used instead of names for the transitions. [52]

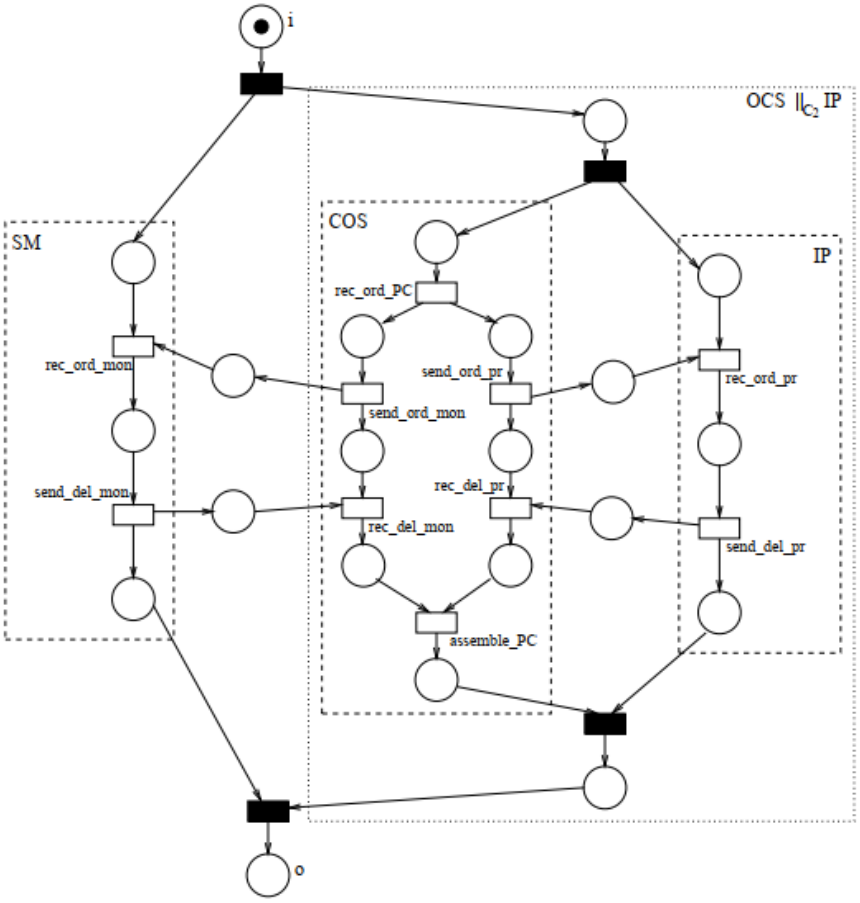


Figure 4.5: Service SM ||C1 (OCS ||C2 IP) [52]

4.3 Others

4.3.1 Enhanced Stacked Automata Model (ESAM)

A new model was proposed to verify composed web services using ESAM. ESAM is a combination of Amend Muller Automata and Pushdown Automata, making it suitable for both deterministic and non-deterministic systems.

Definition

An automaton is a mathematical model that represents system behavior using a discrete number of inputs and outputs. In deterministic systems, upon receiving an input, it transitions to a single state, whereas in nondeterministic systems, the same input can lead to multiple states.

Example

Figure 6 shows the example of deterministic finite automata .it contains six states namely Q0, Q1, Q2, Q3, Q4 and Q5. Giving input "a" to Q0 it goes to Q1state only. Double circle is represented as final state. On receiving an input "a", it does not go to more than one state.

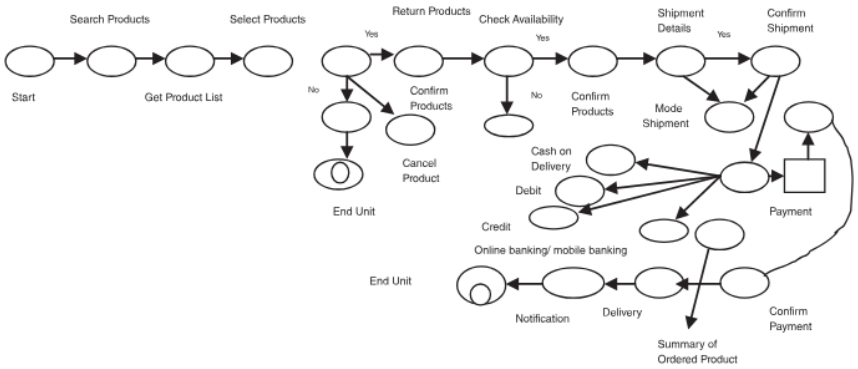


Figure 4.6: Deterministic finite automata [53]

The Figure 7 shows the transition diagram for the non - deterministic System. It contains six states namely Q0, Q1, Q2, Q3, Q4 and Q5. Giving an input a to Q0 it goes to Q1, Q3 and Q4states. Double circle is represented as final state. Deterministic system avoids the problem like reachability and emptiness to an extend of 40 to 50Timed automata and Interface Automata. Whereas non-deterministic System avoids the problems like Reachability, Emptiness, Dead Transition, and Deadlock at 95 to 99

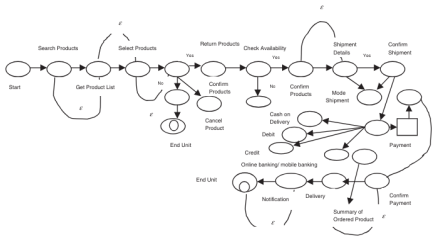


Figure 4.7: Non-deterministic finite automata [53]

4.3.2 model IMWSC

IMWSC is a framework used to represent the process of invoking web services, where web services are software applications accessible over the network.

Definition

Formally, an IMWSC is a septuple $\langle Service, Proc, Activity, L, Message, Ra, F\rangle$, where:

- $Service$ denotes a set of web services;
- $Proc$ is a set of processes;
- $Activity$ is a set of activities;
- L is a set of sequences of activities;
- $Message$ is a set of messages that are exchanged by services;
- $Ra \subseteq Activity \times Activity$ is a binary relation;
- F is a sextuple $\langle f_p, f_{pS}, f_{pU}, f_{aP}, f_{aT}, f_{mA}\rangle$, where:
  - $f_{pT} : Proc \to \{c, b\}$ is a mapping that describes the type of each process (composite or basic);
  - $f_{pS} : Proc \to \{\}$ Service is a mapping that describes the type of each process (composite or atomic);
  - $f_{pU} : Proc \to Proc$ is a mapping that associates a process with a composite process;
  - $f_{aP} : Activity \to Proc$ is a mapping that associates each activity with a process;
  - $f_{aT} : Activity \to Proc$ {ii, io, ei, eo, ex}is a mapping that describes the type of each activity (internal input, internal output, environmental input, environmental output, execute).f_mA: Message $\to$ Activity is a mapping that associates each message with an activity;
  - f_{mA}: Message $\to$ Activity is a mapping that associates each message with an activity;

We let $proc = \{proc\}$ for $Proc$. Let con $= \{a \in Activity | a \wedge a \in aP$ and $f_p(a) \in pT\}$. $c \subseteq Activity \times Activity$ be a partial order relation over $Activity$, defined as: $c = \{(a_1, a_2) | a_1, a_2 \in Activity \wedge aP_1 = aP_2 \wedge (a_1$ happens earlier than $a_2)\}$. An element $proc$ in $Proc$ is constructed by the following g. $proc = \alpha \,|\, 1 \,||\, proc_1\, proc_2 \,|\, proc_1\, p\, proc_2$, where: $\alpha \in Activity$; $proc_1$, $proc_2 \in Proc$.

- α is a new process that performs $proc_1$ and $proc_2$ independently;
- $1 \,||\, proc_1\, proc_2$ is a new process that performs $proc_1$ and $proc_2$ sequentially.
- $proc_1\, p\, proc_2$.

Figure. 8 presents an illustration of the structure of IMWSC. In Figure. 8, a service is visualized by a circle. interaction of services is visualized by a pair of parallel arrows (with opposite directions); the interaction process Definition, i.e., the definition of an instance of IMWSC, is visualized by a rectangle.

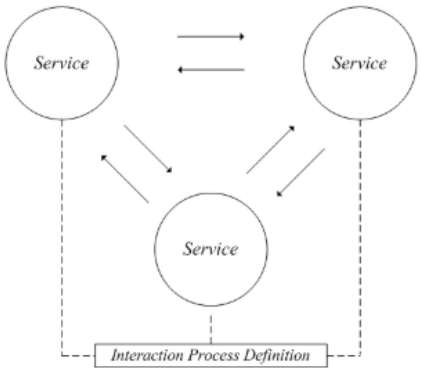


Figure 4.8: Structure of IMWSC [54]

Example

They investigated the application of IMWSC in a simple scenario. There are three services involved in this scenario:

1. The Client Service, which need to find out some useful information (for convenience, client here is considered as a service).
2. The Response Service, which is responsible for dealing with information inquiry requests.
3. The Information Service, which acts as a database and providing the useful information.

The business process of this scenario is introduced briefly as follows:
1. The Response Service receives a request from the Client Service which need to find out some useful information;
2. The Response Service contacts the Information Service and relay the

information inquiry request;
3. The Response Service answers the questions to the Client Service.
Fig. 3 presents an illustration of the structure of this scenario, where

- A service is visualized by a rectangle (with round angles);
- A state of a service is visualized by a circle (the initial and the terminative states of a service are visualized by icons, respectively);
- A transition between states is visualized by an arrow (with curve line), from the source state to the target state;
- The supply channels of services in this scenario is visualized by a pair of parallel arrows (with opposite directions) [54].

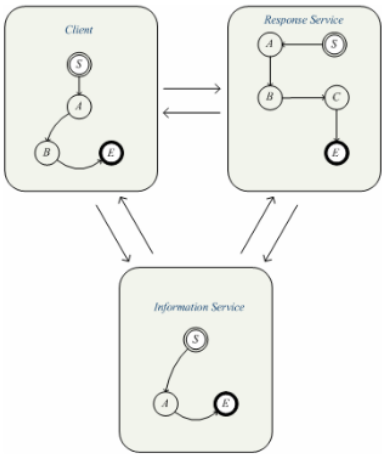


Figure 4.9: A Scenario of Interaction of Services [54]

4.3.3 Comparison of Modeling Approaches: Petri Nets and Others

Approaches	Modeling Scope	Complexity Handling	Semantic Clarity	Expressiveness
Petri net	A broader application scope beyond web service composition, covering areas like parallel processing, distributed systems, etc.	May suffer from state explosion	Might be lacking.	Offer expressive modeling capabilities.
ESAM	Are more tailored towards specific domains.	Offer methods to handle complexity through precise representations or hierarchical compositions.	Might be lacking .	Offer expressive modeling capabilities.
IMWSC	Are more tailored towards specific domains.	Offer methods to handle complexity through precise representations or hierarchical compositions.	Stands out for its emphasis on semantic clarity, ensuring correct preservation of behavior-related information.	Provides a specialized and potentially more focused approach.

Table 4.1: Comparison of Petri Nets and Others

Through comparison, we can conclude that each model has its own distinct properties that make it suitable for modeling web services in different ways.

4.4 Modelling approach base on TPN model: WSCTPN

We introduce, in the following, the WSCTPN model which is a particular case of an TPN.

Definition *A WSCTPN is a tuple* (RT, p_b, p_e)*, such that:*
- $RT = (P, T, B, F, M_0, I_s)$ *is a Time Petri Net .*
- p_b *is a special place of* P *called the beginning place of the WSCTPN , and we have:* $\bullet p_b = \emptyset$ *and* $M_0(p_b) \neq 0$*;*
- p_e *is a special place of* P *called the ending place of the workflow, and we have:* $p_e\bullet = \emptyset$ *and* $M_0(p_e) = 0$ *; where:* $\bullet x$ *denotes the set of input transitions connected to the place* x *while* $x\bullet$*: gives the set of output transitions connected to* x *.*

The place p_b denotes the source of the net while the place p_e the sink of the net. The WSCTPN should verify that there exists a run from the initial marking including the place p_b to a final marking including the place p_e ; we say that the net is *strongly connected*.

The WSCTPN model of the whole WSC can be obtained by the following approach:

1. First we create the places p_e and p_b.
2. *A single WS:* Each elementary unit:(*task*) is mapped into a transition $t \in T$ and an input place $p \in P$. For time constraints a time interval is associated with each transition's task $I(t) = [x(t), y(t)]$, thus, defining the earliest and the latest time delay of the task. If no time constraint are imposed, we have $I(t) = [0, +\infty]$. Otherwise with $I(t) = [0, 0]$ the task t cannot be delayed and must occur as soon as the input place is marked (See Fig.5.(x)). If the task is the first in the process then its input place is p_b. If it is the last in the workflow then its output place is p_e.
3. *Sequence*: In the example of Figure 5.a, tasks t_1 and t_2 are executed sequentially, representing precedence constraints of task (WS) execution;
4. *Choice*: In Figure 5.b, t_1 and t_2 are in conflict and can never occur both;
5. *Concurrency*: In Figure 5.c, tasks are in concurrency; they occur in parallel and are not in conflict. Their execution can be governed by synchronization rules that are expressed in the form of rendezvous.

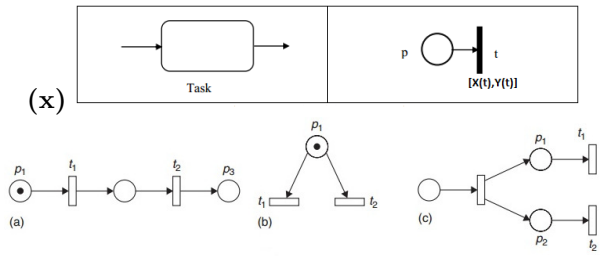


Figure 4.10: Modelling approach with WSCTPN

In the next chapter, we present a case-study to highlight the expression power of the the WSCTPN model in modelling web service composition systems.

4.5 Conclusion

In this chapter, we examined the modeling of web service composition using Petri nets, as well as other models such as ESAM and IMWSC. Examples were provided for each model to illustrate how they are used in representing the processes of web service composition. Finally, a comprehensive comparison was conducted between these models.

Chapter 5

Case Study: (EC.WSC)

5.1 Introduction

In this concrete chapter, we delve into a case study revolving around the E.COMMERCE WEB SERVICE COMPOSITION: (EC.WSC) system, a well-known example in the literature. This study aims to illustrate the "MODELCHEKING" process of (EC.WSC) system using Time Petri Net (TPN) models. E-commerce stands as one of the most significant web applications, encompassing online sales, booking, and payment processes. Through this study, we seek to analyze complex operations, ensuring the integration and efficiency of services using advanced analytical tools like the TINA framework.

5.2 MODELLING of (EC.WSC)

5.2.1 Description

E-commerce (EC) is the product/service sale for articles of different catégories via the Internet. Moreover, (EC) is one of the most known web service in hole world (exp: Aliexpress, Amazon, ...etc). (EC) web service may include ordering, booking, payment and delivery of goods/services. We call (EC.WSC) the e- commerce web service composition of component and is presented in Figure 1.

We now discuss our (EC.WSC) system where seller (S1) and (S2) may trade with one (in our case) or a several buyers (B) at the same time (see experiments later).
Our seller (S1) may have more than one home/appartement/ car for sale, wheras the seller (S2) may have more than one home/appartement/car for booking. We call product the set : home/appartement/car. Once the booking process finished the product became free and can be a part of the next process (either booking or buying), witch is not possible in the case of selling products.

We use the theoretical process of modelling based on Time Petri Net (TPN) formalism described in the previous chapter. First, the process start at transition start-EC.WSC at $[0, 2]$, three parallel sub-process are ready to begin: seller (S1) seller (S2) and buyer (B). The seller (S1) presents a selling offer represented by transitions (selling-offer) at $[0, 3]$ and the seller (S2) presents a booking offer presented by (booking-offer) at $[0, 3]$. In parallel the client reads the offers and choose either buying or booking process: booking-process and buying-process transitions.

If the client choose the booking process with transition :booking, the payment is requiered with a credit card : transition: booking-payement. Otherwise, he chooses selling with transition: selling, then the payement of buying is requiered transition: selling-payement. In both sub process the payment can be refuse if any problem with the credit card (exp : no significant account balance) palces booking-refuse and selling-refuse, otherwise, both process (booking or selling) finishe and the EC.WSC finishes as well : transition Finish-EC.WSC.

5.2.2 TPN model of EC.WSC

Using the modeling process of the last chapter we can extract some basic structures :

- First we create the places p_e and p_b of our EC.WSC.
- *A single WS:* elementary units:(*task*) S1 and seller-offer, S2 and bookig-offer...etc.
- *Sequence:* In our case study of Figure 1, tasks seller-offer and selling-process resp (booking-offer and booking-process) are executed sequentially, representing precedence constraints of task (WS) execution.
- *Choice:* In Figure 1, accept and refuse payments (s-payment or b-payments) are in conflict and can never occur both.
- *Concurrency:* In Figure 1, tasks seller-offer and bookig-offer are in concurrency; they occur in parallel and are not in conflict.

First, we represent the system for the case study using a Time Petri Net as illustrated in the following figure 1:

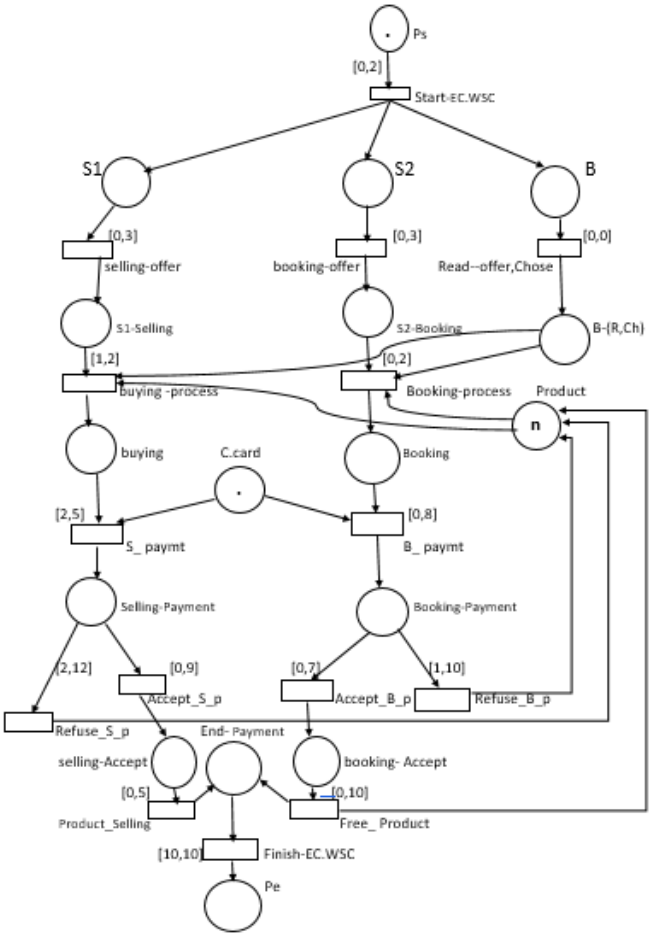


Figure 5.1: TPN of EC.WSC

In the following tables, we extract some of the places and transitions found in Figure 1 with their description within the system:

number places	descriptions places
p1	Beginning of the e-commerce web service composition process.
p4	The customer reads the offers and chooses.
p12	Payment for booking using a credit card.
p17	Completion of the e-commerce web service composition process.

Table 5.1: Description of some places in the EC.WSC system

number transitions	descriptions transitions
t2	The seller presents a sales offer
t3	The customer presents a booking offer.
t10	The client chooses the booking process.
t11	Payment for the purchase using a credit card.

Table 5.2: Description of some Transitions in the EC.WSC system

5.2.3 Input TINA

To start modeling the EC.WSC system case study using the TINA (Time Petri Net Analyzer) tool, we will first enter the various components of the system into TINA as illustrated in the following figure :

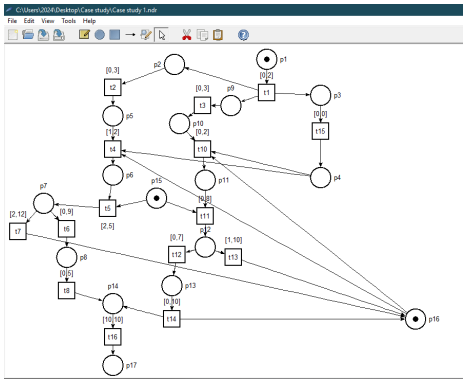


Figure 5.2: Case Study.ndr

This Figure 2 represents a graphical description of the case study and Figure 3 represents a textual description.

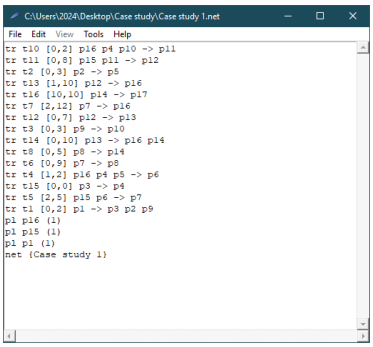


Figure 5.3: Case Study.net

5.3 CHECKING of (EC.WSC)

CHECKING (EC.WSC), means that we use output TINA TOOL and resd the (graphical/textual) generated files. The graphical description for the case study (in .ndr format for Time Petri Nets) in TINA by selecting tools->state space analysis, where the following window appears:

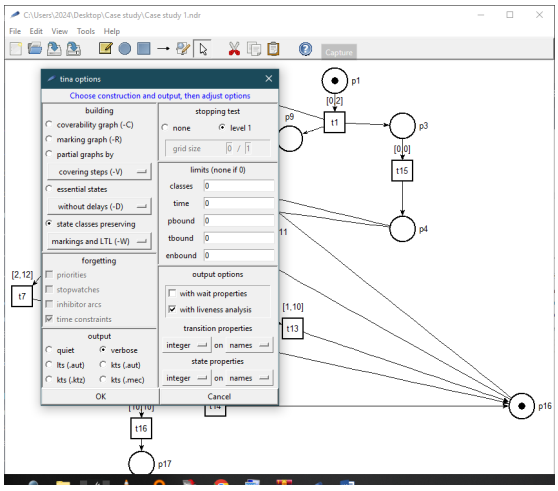


Figure 5.4: Tina options

In this window, we select in the elements building->state classes preserving and also output->verbose. In our study, we'll concentrate on six state classes preserving:

1. Markings (M).
2. Markings and LTL (W).
3. States E.
4. States and LTL (S).

5. States and CTL* (A).
6. States and CTL* (U).

Remark:
The equivalence between states is different when considering the CTL* (A) construction or the CTL* (U) construction (inclusion or/and equality).

5.3.1 The generated graphs

Markings (-M)

First, we select "Markings (-M)" and "verbose".

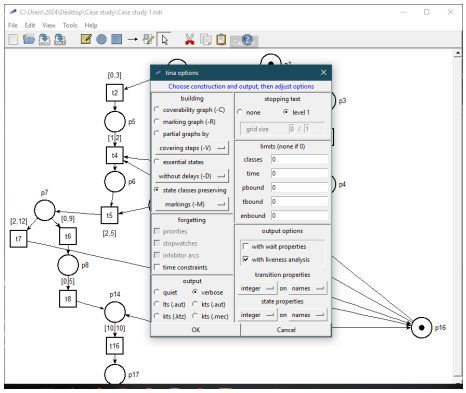


Figure 5.5: Markings (-M)

When we click on OK, we obtain the following:

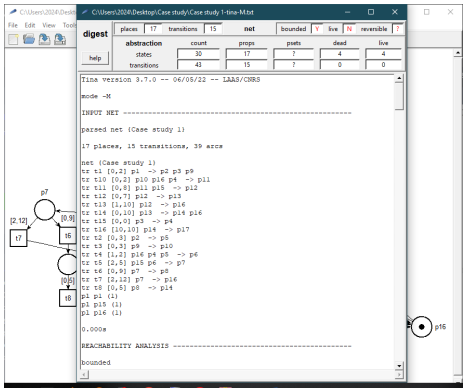


Figure 5.6: The output (-M)

And here is the mark of each class:

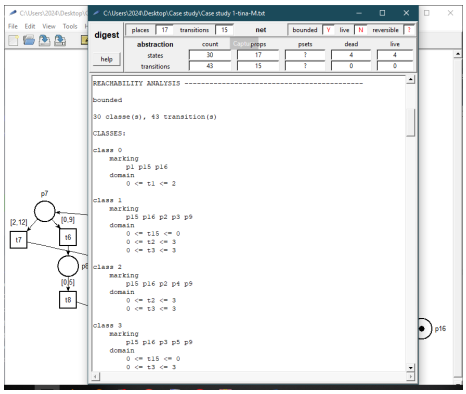


Figure 5.7: The mark of each class mode -M

The properties of markings (-M):

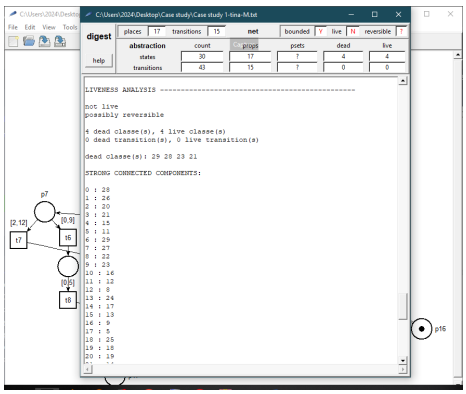


Figure 5.8: The properties of markings (-M)

Markings and LTL (-W)

First, we select "Markings and LTL (-W)" and "verbose" .

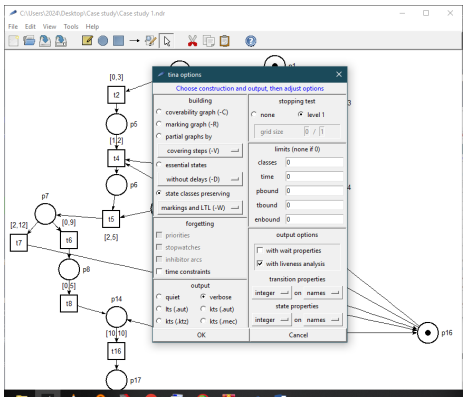


Figure 5.9: Markings and LTL (-W)

When we click on OK, we obtain the following:

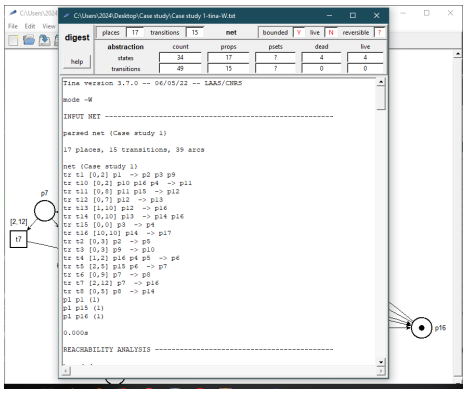


Figure 5.10: The output (-W)

And here is the mark of each class:

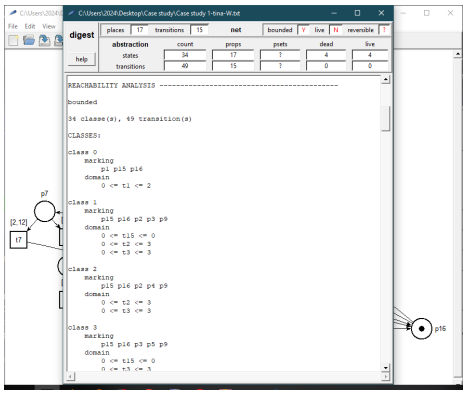


Figure 5.11: The mark of each class mode -W

The properties of markings and LTL (-W):

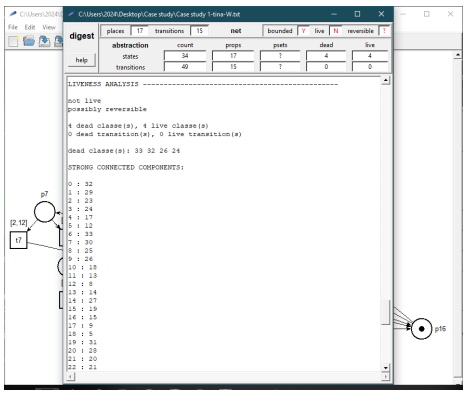


Figure 5.12: Properties of markings and LTL (-W)

States E

First, we select "States E." and "verbose" .

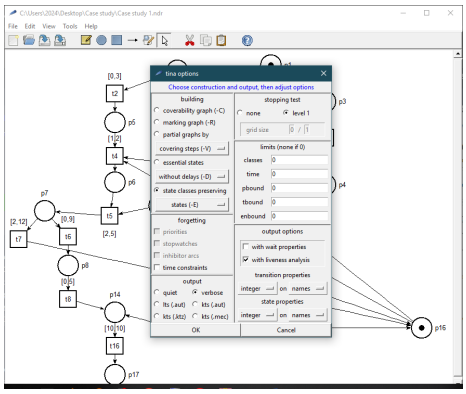


Figure 5.13: States E

When we click on OK, we obtain the following:

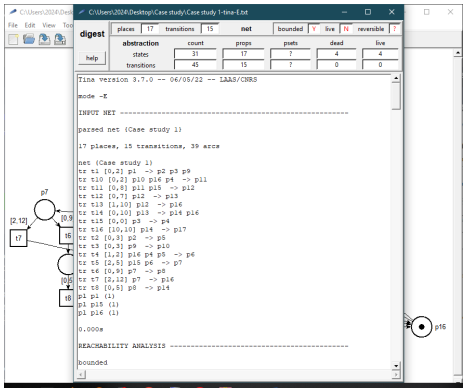


Figure 5.14: The output E

And here is the mark of each class:

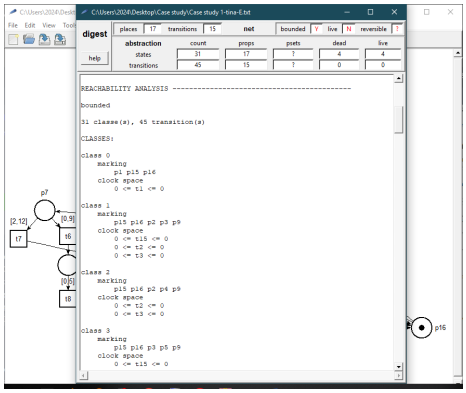


Figure 5.15: The mark of each class mode -E

The properties of States E:

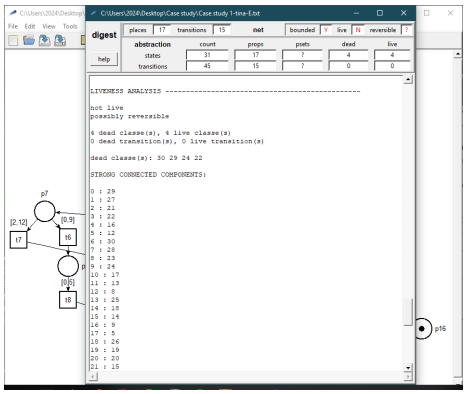


Figure 5.16: The properties of States E

States and LTL (S)

First, we select "States and LTL (S)" and "verbose" .

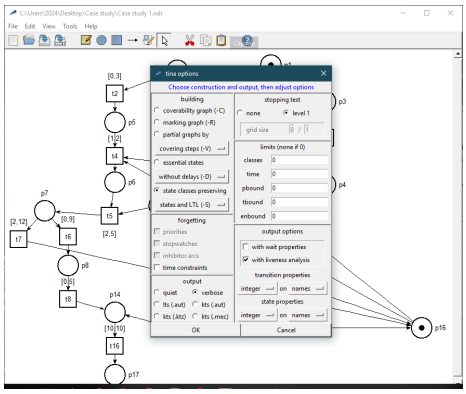


Figure 5.17: States and LTL (S)

When we click on OK, we obtain the following:

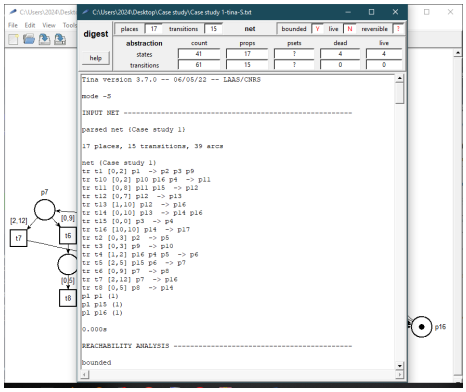


Figure 5.18: The output States and LTL (S)

And here is the mark of each class:

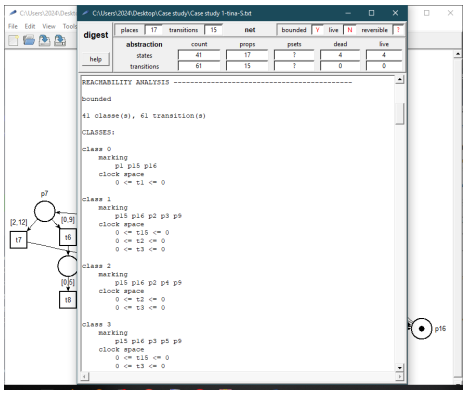


Figure 5.19: the mark of each class mode -S

The properties of States and LTL (S):

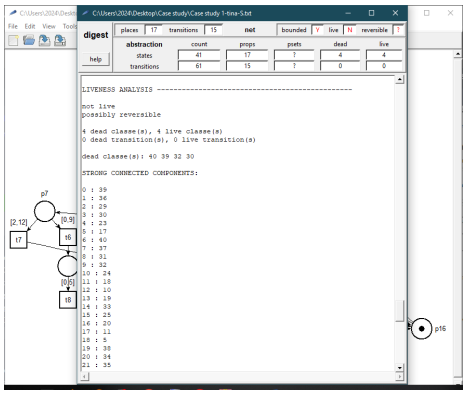


Figure 5.20: The properties of mode -S

States and CTL* (A)

First, we select "States and CTL* (A)" and "verbose" .

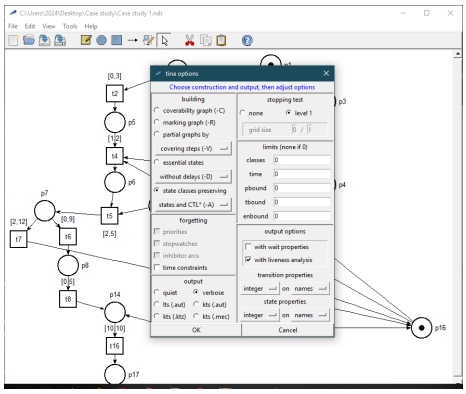


Figure 5.21: States and CTL* (A)

When we click on OK, we obtain the following:

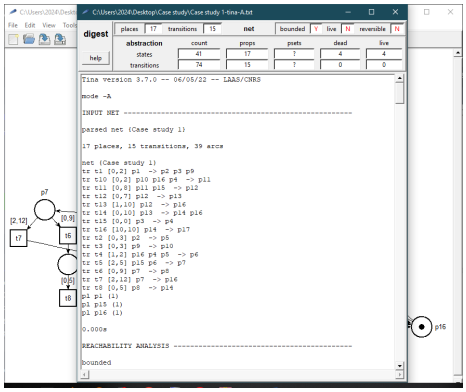


Figure 5.22: The output States and CTL* (A)

And here is the mark of each class:

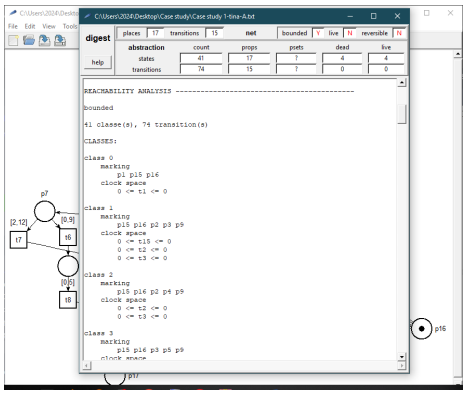


Figure 5.23: The mark of each class mode -A

The properties of States and CTL* (A):

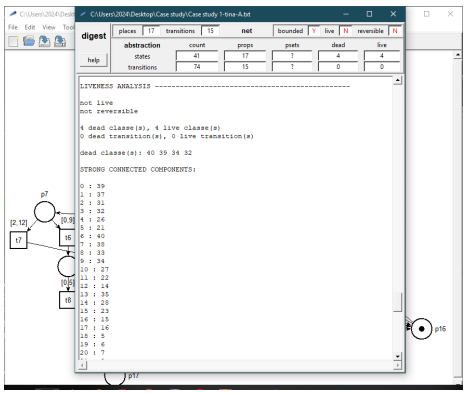


Figure 5.24: The properties of States and CTL* (A)

States and CTL* (U)

First, we select "Markings and CTL* (U)" and "verbose" .

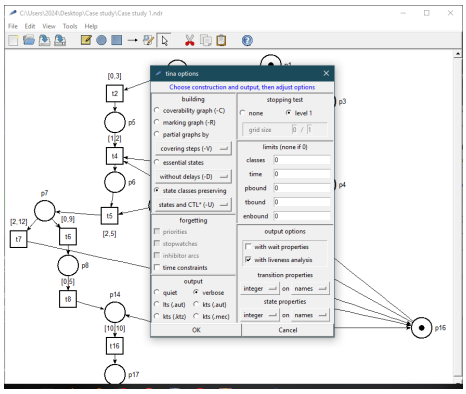


Figure 5.25: States and CTL* (U)

When we click on OK, we obtain the following:

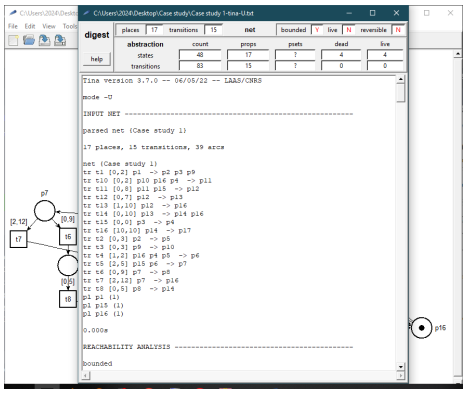


Figure 5.26: The output states and CTL* (U)

And here is the mark of each class:

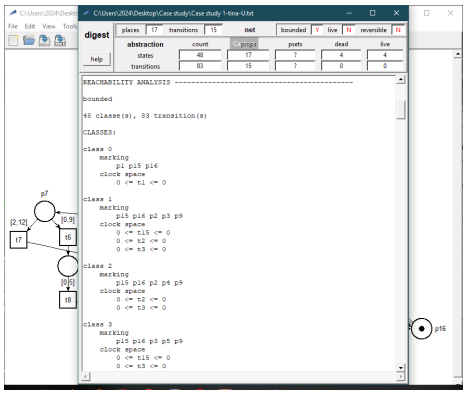


Figure 5.27: The mark of each class mode -U

The properties of States and CTL* (U):

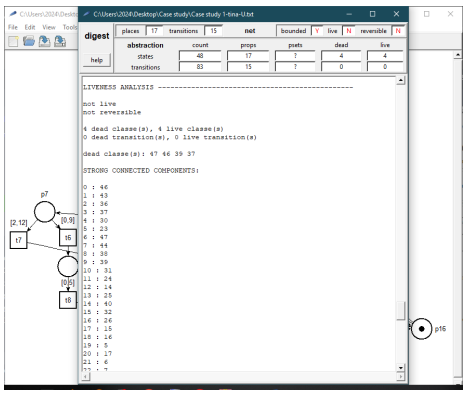


Figure 5.28: properties of States and CTL* (U)

5.3.2 Output of the case study with mode A:

1. Graphical description

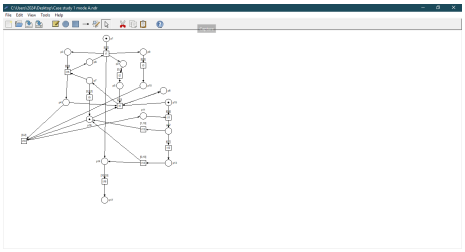


Figure 5.29: Graphical description of the case study with mode A

2. Text description

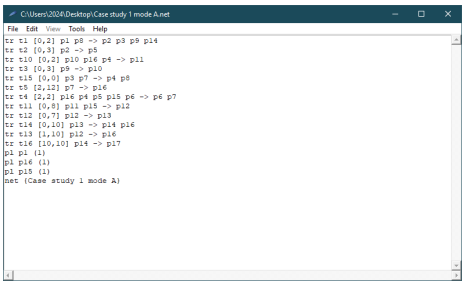


Figure 5.30: Text description of the case study with mode A

In the same way, we can obtain different construction implemented in TINA in the next table.

5.3.3 Computing experiments

TINA tool generate a many graphs, the most known and importants one for our case (cheking) are: (M) mode, (E) mode, (W) mode, (S) mode, and (A) and (U) modes. We are not investigated the algorithms in this experiments, we are comparing results of computing of the number of classes, edges (arcs) and time CPU.

Examples	computing	(M)	(E)	(W)	(S)	(A)	(U)
Case 1:	Classes	39	39	39	39	39	39
	Arcs	18	18	18	18	18	18
	CPU	0.00s	0.00s	0.00s	0.00s	0.00s	0.00s
Case 2:	Classes	30	31	34	41	41	48
	Arcs	39	39	39	39	39	39
	CPU	0.00s	0.00s	0.00s	0.00s	0.00s	0.00s

Table 5.3: Results of computing of the number of classes, edges (arcs) and time CPU

Remark:
Case:1 means the following values of transitions $t_1 = [0, 1]$; $t_2 = [0, 0]$; $t_3 = [0, 0]$. Whereas, the Case:2 means the values $t_1 = [0, 1]$; $t_2 = [0, 3]$; $t_3 = [0, 3]$ of transitions.

5.3.4 Discussion and comparaison

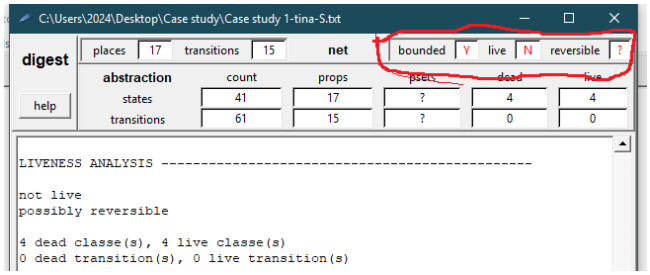


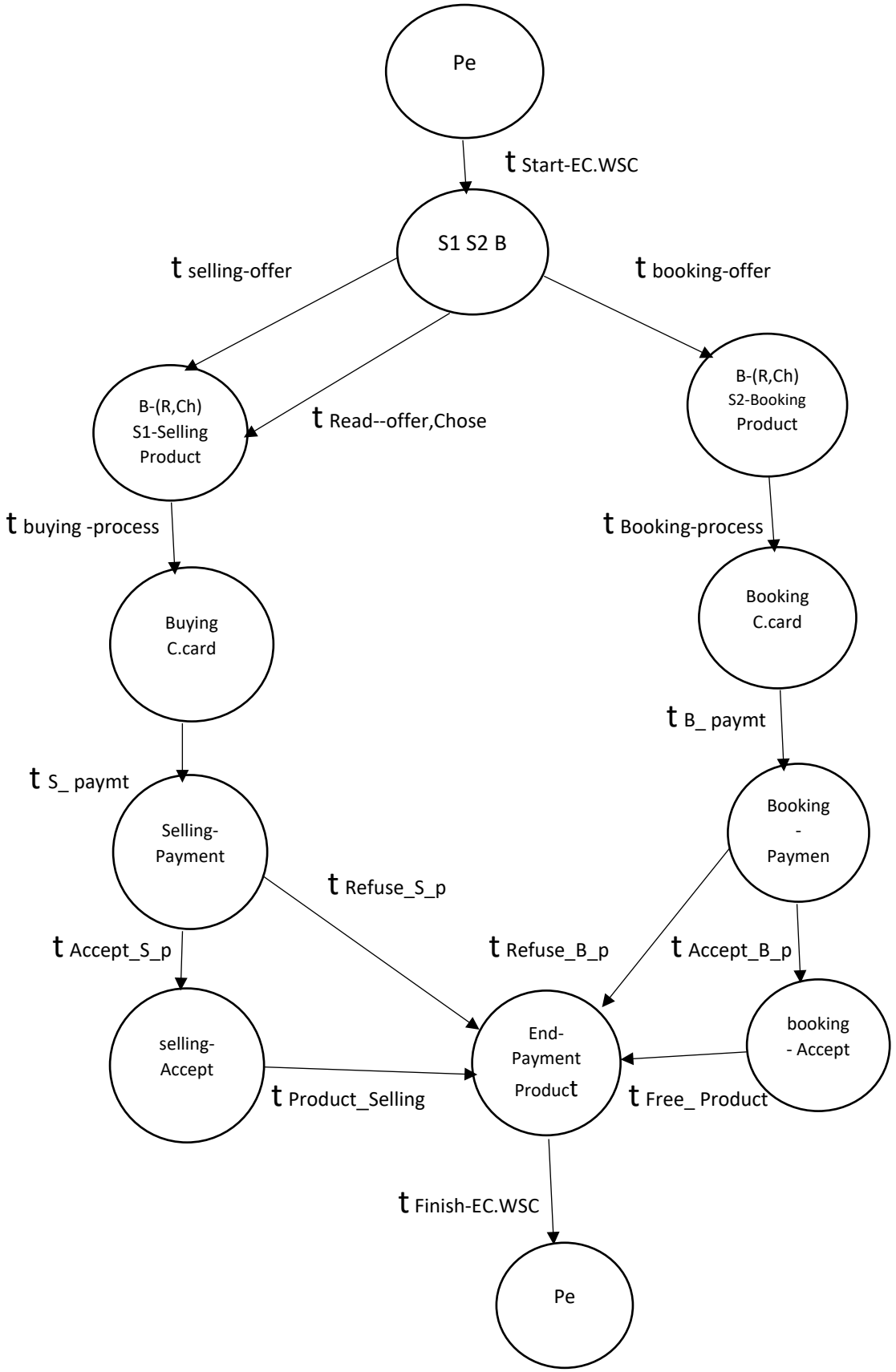
Figure 5.31: Checking properties

All constructions have the same number of classes, arcs and CPU time for case 1. In case 2 (when transitions values are modified), the number of classes change and became bigger, the reason is that the A mode some classes can split to more then one class due to the atomic properties.

When the example is not too important(2 seller and 1 buyer), we can see easily that it takes a tiny time execution (0.00s). For reachabiliy analysis, using the generated graphs, we can easily verify any linear or atomic properties. For reachability, we can see that the marking (stat) S1S2B1 is reachable. For safty, we can see that the graphs have no deadlock.

Finaly, the properties using TINA can be checked directly see figure on top right.

Marking EC.WSC :



The EC.WSC marking graph

5.4 Conclusion

In this chapter we choose the EC.WSC case study, a topical and current system nowadays. We use TPN formalism for modelling and TINA tool for checking properties (linear or atomics). TINA can generate a many graphs a cording to different algorithms and needs. We use the 8 most important modes in our work.

Chapter 6

Conclusion

6.1 Conclusion General

Web services and web service composition are powerful technologies with the potential to transform applications, hardware, and software resources into standardized, reusable, and dynamically integrated software components [1]. As web services are increasingly recognized as key drivers of online activities, their significance in achieving seamless and efficient integration between various systems and software components becomes ever more apparent.

The growing challenges in the technology world demand innovative solutions to meet the increasingly complex user needs. Since a single web service may not suffice to meet all user needs, web service composition has been developed to integrate multiple individual services into a composite service that addresses complex user requirements [55]. This composition should prioritize efficiently and effectively meeting users' functional requests.

Web services play a crucial role in the IT world, contributing to the integration between different systems and facilitating interaction between applications through standard web protocols. Web service composition can be performed in various ways using different models to ensure effective integration and high performance. Our study was comprehensive, reviewing various models used in web service composition, including Petri nets and time Petri nets.

To analyze and model web service composition, we utilized Petri nets, which are powerful tools for modeling and analyzing complex processes. Petri nets provide a robust mathematical framework to represent flows and concurrent processes in distributed systems, helping to detect errors and improve performance. Additionally, we employed time Petri nets (TPNs), which extend traditional Petri nets by incorporating the time dimension. This extension allows us to model time-dependent systems and analyze the temporal performance of processes, providing a more comprehensive understanding of system behavior in various operational environments.

Using the TINA tool for analyzing time Petri net models, we examined

and analyzed web service composition in our case study. Our study focused on the EC.WSC system where the seller provides both sales and booking services to the buyer. The results demonstrated the significant effectiveness of this approach in ensuring the correctness and performance of composite services. The use of the TINA tool proved highly effective in analyzing time Petri net models, ensuring high and accurate performance of integrated services.
In this work, we relied on two main tools: TINA and LaTeX. We used TINA, the Time Petri Net Analyzer, for modeling and analyzing the composition of web services. TINA provides powerful capabilities for constructing and verifying time Petri nets, which is essential for ensuring the correctness and performance of web services. Additionally, we used LaTeX, a high-quality typesetting system, for creating and editing PDF documents. LaTeX is particularly useful for producing technical and scientific documentation with professionalism and precision.
The use of time Petri nets and the TINA tool provides a solid framework for analyzing and developing composite web services, paving the way for further research and development in this vital field. By delving deeper into the use of these tools, we can achieve a more comprehensive and detailed understanding of the web service composition process, ensuring effective integration and high performance in various operational environments.

Future Work:
For future work, several directions can be explored to expand the research and deepen our understanding of web service composition and the utilization of available tools:
Expand Application Scope: Investigate the usage and applications of web services in other domains such as workflow, task scheduling, and the Internet of Things (IoT). This direction can provide a deeper understanding of how service compositions can be adapted to meet the requirements of these new areas.
Select Additional Tools: Choose and explore additional tools for system analysis and modeling, such as Romero, RT Studio, and Oris, to broaden the analytical capabilities and leverage more features and available technologies.
Expand Modeling: Extend the work to explore other extensions of the Petri net model, such as Timed Petri Nets (TPN) with Rendezvous technique, to better understand complex service compositions and interactions between them.
By directing work towards these directions, we can contribute to the development of understanding and knowledge in the field of web service composition, providing a strong foundation for future research endeavors. This expansion can open new horizons for innovation and improvement in the design and analysis of web services, ensuring that user needs are met more efficiently and effectively.

Bibliography

[1] Angel Lagares Lemos, Florian Daniel, Boualem Benatallah. *Web service composition: a survey of techniques and tools. ACM Computing Surveys (CSUR)*, vol. 48, no. 3, pp. 1–41, 2015. ACM New York, NY, USA.

[2] Hongxia Tong, Jian Cao, Shensheng Zhang, Minglu Li. *A distributed algorithm for web service composition based on service agent model. IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 12, pp. 2008–2021, 2011. IEEE.

[3] Wil Aalst. *Three Good Reasons for Using A Petri-Net-Based Workflow Management System.* In *Engineering and Computer Science*, vol. 11, pp. 161–182, 1998. ISBN 978-1-4613-7512-8. doi: 10.1007/978-1-4615-5499-810.

[4] Guobing Zou, Yixin Chen, Y. Yang, Ruoyun Huang, You Xu. *AI planning and combinatorial optimization for web service composition in cloud computing.* In *Proc international conference on cloud computing and virtualization*, pp. 1–8, 2010.

[5] Mahboobeh Moghaddam, Joseph G. Davis. *Service selection in web service composition: A comparative review of existing approaches. Web services foundations*, pp. 321–346, 2013. Springer.

[6] Antonio Bucchiarone, Annapaola Marconi, Marco Pistore, Heorhi Raik. *A context-aware framework for dynamic composition of process fragments in the internet of services. Journal of Internet Services and Applications*, vol. 8, pp. 1–23, 2017. Springer.

[7] Heather Kreger et al. *Web services conceptual architecture (WSCA 1.0). IBM software group*, vol. 5, no. 1, pp. 6–7, 2001.

[8] Robert Daigneau. *Service Design Patterns: fundamental design solutions for SOAP/WSDL and restful Web Services.* Addison-Wesley, 2012.

[9] Michael P. Papazoglou, Jean-jacques Dubray. *A survey of web service technologies.* University of Trento, 2004.

[10] Mohammad Hammoudeh, Ajlan Al-Ajlan. *Implementing web services using PHP soap approach.* International Association of Online Engineering, 2020.

[11] Z. Aziz, C. J. Anumba, Darshan Ruikar, P. Carrillo, D. Bouchlaghem. *Intelligent wireless web services for construction—A review of the*

enabling technologies. Automation in Construction, vol. 15, no. 2, pp. 113–123, 2006. Elsevier.

[12] Bery Leouro MBAIOSSOUM, Adoum Haroun ADOUM, Lang DIONLAR. *Conception d'une plateforme numérique de suivi des dossiers. Afrique SCIENCE*, vol. 19, no. 1, pp. 107–117, 2021.

[13] Festim Halili, Erenis Ramadani et al. *Web services: a comparison of soap and rest services. Modern Applied Science*, vol. 12, no. 3, pp. 175, 2018. Canadian Center of Science and Education.

[14] Digvijaysinh Rathod. *Performance evaluation of restful web services and soap/wsdl web services. International Journal of Advanced Research in Computer Science*, vol. 8, no. 7, pp. 415–420, 2017.

[15] D.V. Kornienko, S.V. Mishina, S.V. Shcherbatykh, M.O. Melnikov. *Principles of securing RESTful API web services developed with python frameworks.* In *Journal of Physics: Conference Series*, vol. 2094, no. 3, pp. 032016, 2021. IOP Publishing.

[16] Bret Hartman, Donald J. Flinn, Konstantin Beznosov, Shirley Kawamoto. *Mastering web services security.* John Wiley & Sons, 2003.

[17] Imam Ahmad, Emi Suwarni, Rohmat Indra Borman, Farli Rossi, Yessi Jusman et al. *Implementation of RESTful API Web Services Architecture in Takeaway Application Development.* In *2021 1st International Conference on Electronic and Electrical Engineering and Intelligent System (ICE3IS)*, pp. 132–137, 2021. IEEE.

[18] James McGovern, Sameer Tyagi, Michael Stevens, Sunil Mathew. *Java web services architecture.* Elsevier, 2003.

[19] Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pål Krogdahl, Min Luo, Tony Newling. *Patterns: service-oriented architecture and web services.* IBM Corporation, International Technical Support Organization, 2004.

[20] Zaki Brahmi, Afef Selmi. *Coordinate system-based trust-aware web services composition in edge and cloud environment. The Computer Journal*, vol. 66, no. 9, pp. 2102–2117, 2023. Oxford University Press.

[21] Guobing Zou, Qiang Lu, Yixin Chen, Ruoyun Huang, You Xu, Yang Xiang. *QoS-aware dynamic composition of web services using numerical temporal planning. IEEE Transactions on Services Computing*, vol. 7, no. 1, pp. 18–31, 2012. IEEE.

[22] Jian Yang, Mike P. Papazoglou. *Service components for managing the life-cycle of service compositions. Information Systems*, vol. 29, no. 2, pp. 97–125, 2004. Elsevier.

[23] Yasmine Charif, Nicolas Sabouret. *Dynamic service composition enabled by introspective agent coordination. Autonomous agents and multi-agent systems*, vol. 26, pp. 54–85, 2013. Springer.

[24] Quan Z. Sheng, Xiaoqiang Qiao, Athanasios V. Vasilakos, Claudia Szabo, Scott Bourne, Xiaofei Xu. *Web services composition: A decade's overview. Information Sciences*, vol. 280, pp. 218–238, 2014. Elsevier.

[25] Dr. K. Jayarajan. *WEB SERVICES COMPOSITION METHODS AND TECHNIQUES: A REVIEW.*

[26] Ali KHEBIZI. *Prise en Compte des Contraintes lors de la Découverte et de l'Orchestration des Services Web.* 2009.

[27] Florian Daniel, Barbara Pernici.

[28] Martin Bernauer, Gerti Kappel, Gerhard Kramler, Werner Retschitzegger. *Specification of interorganizational workflows-A comparison of approaches. Interaction*, vol. 2, pp. c1, 2003.

[29] Serge Haddad, Tarek Melliti, Patrice Moreaux, Sylvain Rampacek. *Modelling Web Services Interoperability.* In *ICEIS (4)*, pp. 287–295, 2004.

[30] Frank Leymann et al. *Web services flow language (WSFL 1.0)*, 2001.

[31] D. Papakonstantinou, V. Koufi, G. Vassilacopoulos. *A SERVICE-ORIENTED ELECTRONIC MEDICAL RECORD ARCHITECTURE. INFORMATION COMMUNICATION TECHNOLOGIES IN HEALTH.*

[32] Anis Charfi, Rainer Berbner, Mira Mezini, Ralf Steinmetz. *On the Management Requirements of Web Service Compositions. Emerging Web Services Technology, Volume II*, pp. 97–109, 2008. Springer.

[33] Carl Adam Petri. *Communication with Automata [Kommunikation mit Automaten]*, 1962. University of Bonn.

[34] Philip Meir Merlin. *A study of the recoverability of computing systems*, 1975.

[35] Bernard Berthomieu, François Vernadat. *Time Petri Nets Analysis with TINA.* In *QEST*, vol. 6, pp. 123–124, 2006.

[36] Carl Adam Petri, Wolfgang Reisig. *Petri net. Scholarpedia*, vol. 3, no. 4, pp. 6477, 2008.

[37] Richard Johnsonbaugh. *Discrete Mathematics and Its Applications, Sixth Edition: Petri Nets (Chapter 11)*, 2007. `https://condor.depaul.edu/~rjohnson/dm6th/petri.pdf`.

[38] Alireza Bahramian, Fatemeh Parastesh, Viet-Thanh Pham, Tomasz Kapitaniak, Sajad Jafari, Matjaž Perc. *Collective behavior in a two-layer neuronal network with time-varying chemical connections that are controlled by a Petri net. Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 31, no. 3, 2021. AIP Publishing.

[39] P. I., C. A. *Kommunikation mit Automaton. Schriften des Rheinlsch-Westfal-ischen Instituts für Instrumentelle Mathematik an der Universitat Bonn*, Heft 2, Bonn, W. Germany, 1962. Translation: C. F. Greene, Supplement 1 to Tech. Rep. RADC-TR-65-337, Vol. 1, Rome Air Development Center, Griffiss Air Force Base, N.Y., 1965.

[40] A. W. Holt, H. Saint, R. M. Shapiro, S. Warshall. *Final report of the information system theory project.* Tech. Rep. RADC-TR68-305, Rome Air Development Center, Griffiss Air Force Base, N. Y., Sept 1968.

[41] A. W. Holt, F. Commoner. *Events and condition. Applied Data Research*, N.Y., 1970. Also in Record Project MAC Conf. Concurrent Systems and Parallel Computatmn, (Chapters I, II, IV, and VI) ACM, N.Y., 1970, pp. 3-52.

[42] Richard Zurawski, MengChu Zhou. *Petri Nets and Industrial Applications: A Tutorial*. Pennsylvania State Univ., University Park, PA, USA, 1994.

[43] Chander Ramchandani. *Analysis of asynchronous concurrent systems by timed petri nets*. PhD thesis, Massachusetts Institute of Technology, USA, 1973. `http://hdl.handle.net/1721.1/13739`.

[44] Abdia Hamdani, Abdelkrim Abdelli. *Time Petri net with rendezvous*. In *4th International Conference on Control, Decision and Information Technologies, CoDIT 2017, Barcelona, Spain, April 5-7, 2017*, pp. 126–131, IEEE, 2017. `https://doi.org/10.1109/CoDIT.2017.8102578`.

[45] Abdia Hamdani, Abdelkrim Abdelli. *Towards modelling and analyzing timed workflow systems with complex synchronizations. J. King Saud Univ. Comput. Inf. Sci.*, vol. 32, no. 4, pp. 491–504, 2020. `https://doi.org/10.1016/j.jksuci.2019.08.007`.

[46] Bernard Berthomieu, François Vernadat. *State class constructions for branching analysis of time Petri nets*. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 442–457, Springer, 2003.

[47] Hanifa Boucheneb, U. Alger, Gérard Berthelot. *Towards a simplified building of time Petri Nets reachability graph*. In *Proceedings of the 5th International Workshop on Petri Nets and Performance Models, PNPM 1993, Toulouse, France, October 19-22, 1993*, pp. 46–47, IEEE Computer Society, 1993. `https://doi.org/10.1109/PNPM.1993.393436`.

[48] Roy Grønmo, Ida Solheim. *Towards Modeling Web Service Composition in UML. WSMAI*, vol. 4, pp. 72–86, 2004.

[49] Abdelkrim Abdelli, Walid Serrai, Lynda Mokdad, Youcef Hammal. *Time Petri Nets for performance evaluation of composite web services architectures*. In *2015 IEEE Symposium on Computers and Communication (ISCC)*, pp. 122–127, IEEE, 2015.

[50] Sofiane Chemaa, Raida Elmansouri, Allaoua Chaoui. *Web services modeling and composition approach using object-oriented Petri nets. arXiv preprint arXiv:1304.2080*, 2013.

[51] Fateh Latreche, Faiza Belala. *A layered Petri net model to formally analyse time critical web service composition. International Journal of Critical Computer-Based Systems*, vol. 7, no. 2, pp. 119–137, 2017. Inderscience Publishers (IEL).

[52] Rachid Hamadi, Boualem Benatallah. *A Petri net-based model for web service composition*. In *Proceedings of the 14th Australasian database conference-Volume 17*, pp. 191–200, 2003.

[53] Danapaquiame Nagamouttou, Ilavarasan Egambaram, Muthumanickam Krishnan, Poonkuzhali Narasingam. *A verification strategy for web services composition using enhanced stacked automata model. SpringerPlus*, vol. 4, pp. 1–13, 2015. Springer.

[54] Li Bao, Weishi Zhang, Xiong Xie. *A Formal Model for Abstracting the Interaction of Web Services. J. Comput.*, vol. 5, no. 1, pp. 91–98, 2010. Publisher: Citeseer.

[55] Dongjin Yu, Lei Zhang, Chengfei Liu, Rui Zhou, Dengwei Xu. *Automatic Web service composition driven by keyword query. World Wide Web*, vol. 23, pp. 1665–1692, 2020. Publisher: Springer.

[56] *TINA TOOL*. LAAS/CNRS. `http://projects.laas.fr/tina//home.php`. Date: July 5, 2021.

[57] Peter Jansson. *Writing Science with LATEX*. Stockholm University, 2019. Book.

[58] Swetha Priyanka Katta and others. *A Comparative Study of Overleaf and Cocalc using Usability Heuristics*, 2022.

Chapter 7

Annex A: TINA TOOL

TINA (TIme petri Net Analyzer) is a toolbox for the editing and analysis of Petri Nets, with possibly inhibitor and read arcs, Time Petri Nets, with possibly priorities and stopwatches, and an extension of Time Petri Nets with data handling called Time Transition Systems. TINA has been developed in the OLC, then VerTICS, research groups of LAAS/CNRS. General Petri nets information can be found on the Petri Nets World site.

The Tina application interface appears like this:

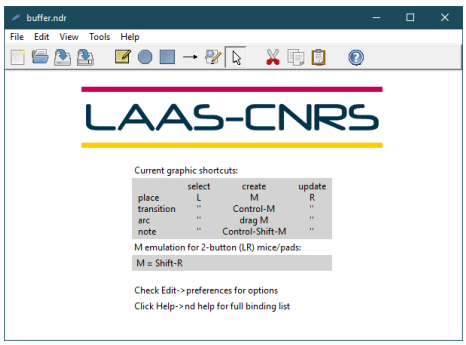


Figure 7.1: TINA application interface

The TINA toolbox includes the tools:
nd (NetDraw): Editor and GUI for Petri nets, Time Petri Nets and Automata.
Handles graphically or textually described nets or automata. Interfaced with analysis tools below. Includes drawing facilities for nets and automata and a stepper simulator for nets.
tina: Construction of reachability graphs.
From nets described in textual or graphical form, produces transition systems abstracting their behavior in human readable form or in various formats for available model checkers and equivalence checkers.depending on options retained, it builds: The coverability graph of a Petri net, by

the Karp and Miller technique.
The marking graph of a bounded Petri net.
Partial marking graphs of a Petri net.
Various state space abstractions for Time Petri nets (state class graphs). Depending on the option selected, the construction perserves markings, states, LTL properties, or CTL* properties of the concrete state space of the Time Petri net.

Operating modes: nd edits either Time Petri nets or automata, depending on command line flags, the file loaded, or options selected. The tools button provides specific analysis tools. In each case, descriptions may be textual or graphical:

1. Textual descriptions are in .net (resp .aut) format. They are converted into graphical form by Edit->draw. Text editing bindings are those of the tk text widget.

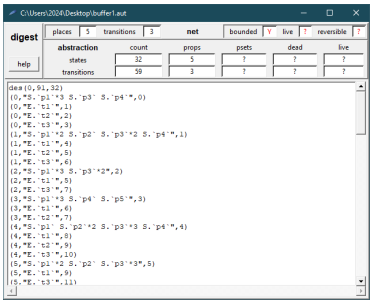


Figure 7.2: Textual description

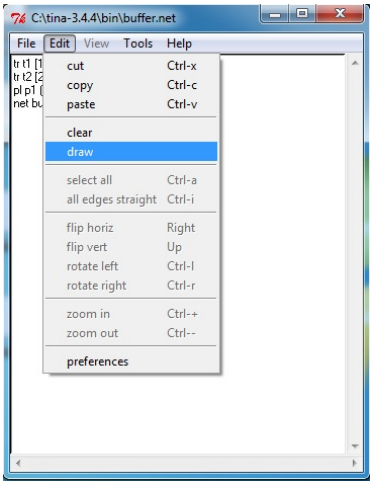


Figure 7.3: Edit->draw

2. Graphical descriptions are in .ndr (resp .adr) format. They are converted into textual form by Edit->textify. [56]

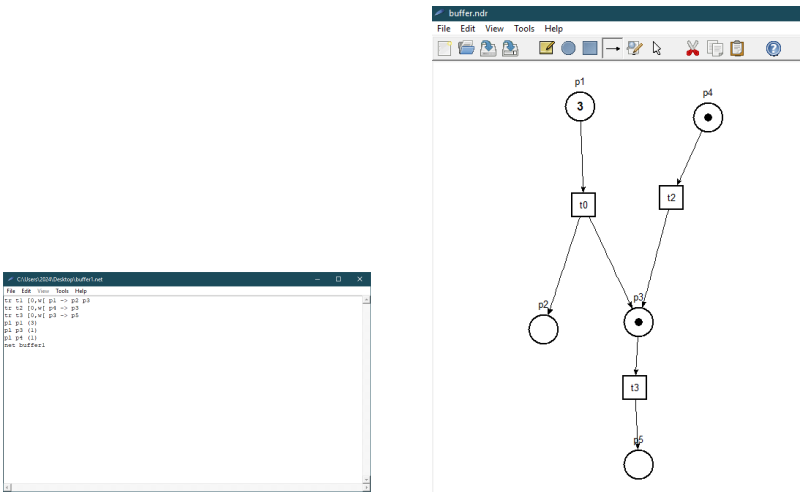


Figure 7.4: Textual and graphical description of net

tina options:
Choose "Construction" and "Output," then adjust the options. Set the parameters as shown in the following figure and click "OK" to generate the output.

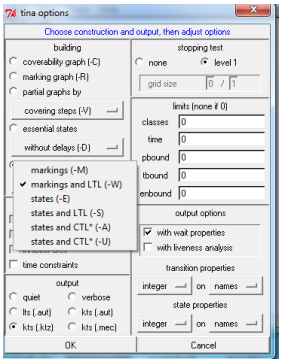


Figure 7.5: Tina options

Chapter 8

Annex B: LATEX TOOL

8.1 Latex

LaTeX was created by scientists for scientists, and it is widely used by publishers for journals, books, and even theses. Having practical knowledge of LaTeX should be part of every scientist's toolkit. [57] The LaTeX working environment is shown in the following image:

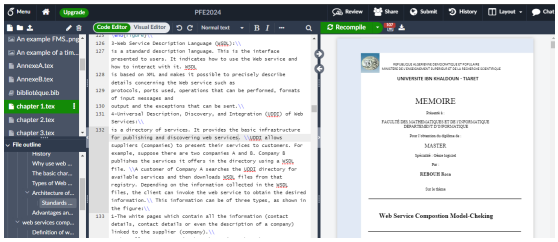


Figure 8.1: The LaTeX and Overleaf working environment

8.2 Overleaf

Overleaf is an online LaTeX editor "https://www.overleaf.com/" that is used to create and edit scientific documentation. Overleaf has a built-in rich-text editor, so it does not require any coding experience. The creation and modification of the document's content don't require any considerable skills.
By maintaining the document in a single location throughout its entire lifespan, it optimizes and facilitates the research writing and publishing processes. The document is safely stored in the cloud so that whenever it's their turn, the writers, editors, auditors, and users can all read, change, or provide feedback on the article using just a browser. With the help

of Overleaf, the writing and publication of scientific research are being moved into cloud, which allows it to be done more quickly, easily, and publicly. Many people can work together on a single task simultaneously, and documents are synchronized so that this is possible. [58]
There are two options for running LaTeX tools: online or offline. These options are not mutually exclusive; you might need to work in both ways. In this thesis, we chose to work with the online platform Overleaf, which I also recommend to anyone, whether beginner or experienced, as all you need is a web browser and an internet connection. [57] As shown in the following pictures:

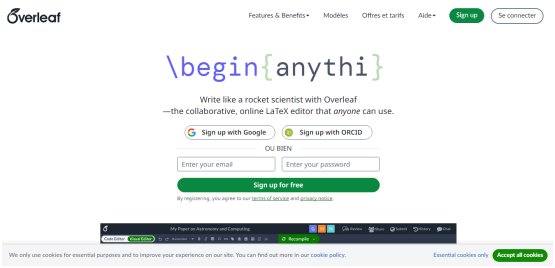


Figure 8.2: Sign up with google

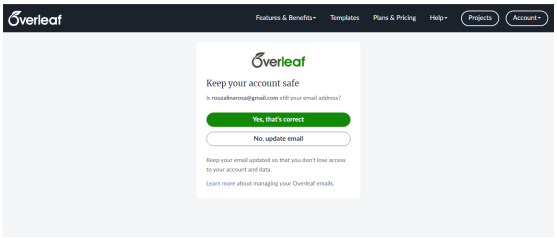


Figure 8.3: Continue registering

After registering for LaTeX, you create a New Project as shown in the following image:

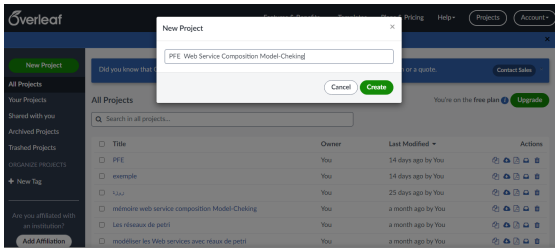


Figure 8.4: Create New project

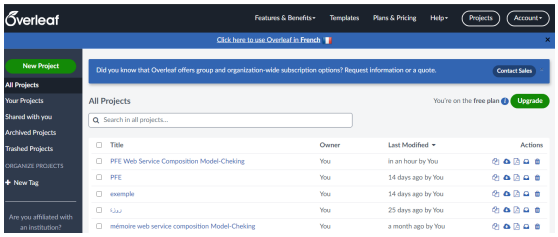


Figure 8.5: Project created

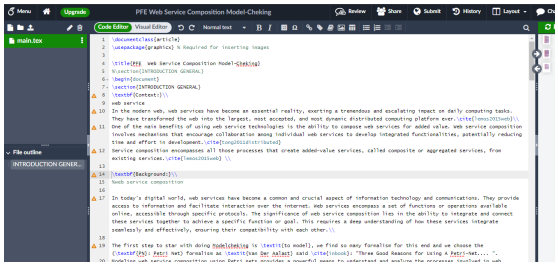


Figure 8.6: Created code in Latex

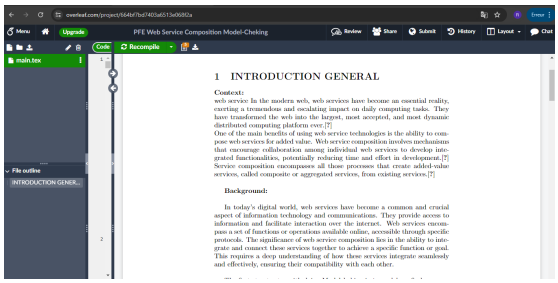


Figure 8.7: Recompile code in Latex

One of the advantages of Overleaf is that it allows multiple people to collaborate interactively on a document. [57] as shown in the following image:

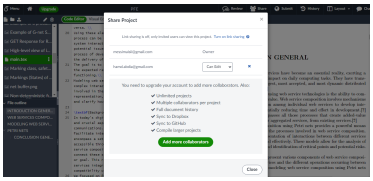


Figure 8.8: Example of collaborate interactively on a document

The disadvantage, of course, is that you cannot work on your documents without internet access. Given how much it has simplified many tasks for me, I highly recommend beginners to register and use Overleaf. [57]