Towards the distribution control of manufacturing systems : a component-based approach for taking into account the communication architecture in modeling

Aladdin Masri

N° d'ordre : 102

ÉCOLE CENTRALE DE LILLE

Thèse

Présentée en vue d'obtenir le grade de

DOCTEUR

En

Spécialité : Automatique et Informatique Industrielle

Par

Aladdin MASRI

Doctorat délivré par l'Ecole Centrale de Lille

Vers le Contrôle Commande Distribué des Systèmes de Production Manufacturiers: Approche Composant pour la prise en compte de l'Architecture de Communication dans la Modélisation

Soutenue le 10 Juillet 2009 devant le jury d'examen:

Président : Jean-Pierre BOUREY, Professeur à l'École Centrale de Lille

Rapporteur : Moncef TAGINA, Professeur à l'Université de Manouba, Tunisie

Rapporteur : Pascal BERRUET, Professeur à l'IUT de Lorient

Membre : Pavol BARGER, Maître de Conférences à l'Université de Technologie de Compiègne

Directeur de thèse : Armand TOGUYENI, Professeur à l'École Centrale de Lille

Co-directeur de thèse : Thomas BOURDEAUD'HUY, Maître de Conférences à l'École Centrale de Lille

Thèse préparée au sein du Laboratoire LAGIS
École Doctorale SPI 072

Order No: 102

ÉCOLE CENTRALE DE LILLE

PhD Thesis

Presented to obtain the degree of

Doctor of Philosophy

In

Speciality: Automation Control and Computer Engineering

By

Aladdin Masri

PhD degree delivered by École Centrale de Lille

Towards the Distributed Control of Manufacturing Systems: A Component-Based Approach for taking into account the Communication Architecture in Modeling

Defended on 10 July 2009 in presence of the Board of Examiners:

President: Jean-Pierre Bourey, Professor at École Centrale de Lille

Reviewer: Moncef Tagina, Professor at University of Manouba, Tunisia

Reviewer: Pascal Berruet, Professor at IUT of Lorient

Examiner: Pavol Barger, Assistant Professor at University of Technology of Compiègne

Thesis Supervisor: Armand Toguyeni, Professor at École Centrale de Lille

Co-supervisor: Thomas Bourdeaud'huy, Assistant Professor at École Centrale de Lille

PhD Thesis prepared in the LAGIS Laboratory

Science for engineers Doctoral School SPI 072

To the memorial of my parents,

To my Family,

To my wife and my two little daughters Hala and Dana

"يَرْفَعِ اللَّهُ الَّذِينَ آمَنُوا مِنكُمْ وَالَّذِينَ أُوتُوا الْعِلْمَ دَرَجَاتٍ ۚ وَاللَّهُ بِمَا تَعْمَلُونَ خَبِيرٌ" [المجادلة:11]

"God raises those among you who believe and those who acquire knowledge to higher ranks. God is fully Cognizant of everything you do". **[Al-Mujadila:11], The Holy Quran**

« Allah élèvera en degrés ceux d'entre vous qui auront cru et ceux qui auront reçu le savoir. Allah est parfaitement Connaisseur de ce que vous faites ». **[Al-Mujadila:11], Le Saint Coran**

Acknowledgment

First of all, I would like to thank Prof. Armand Toguyeni and Dr. Thomas Bourdeaud'huy with whom I had the opportunity and pleasure to work during these three years. I also thank them for their encouragement and rigorous advices at scientific and personal levels which have resulted in the production of this work.

I present my sincere thanks to Prof. Jean-Pierre Bourey who did me the honor of chairing the board of examiners.

Then, I wish to express my deep gratitude to Professor Moncef Tagina and Professor Pascal Berruet for the interest they have shown to my work by agreeing to be the reporters of this thesis.

I also wish to thank Dr Pavol Barger for agreeing to review this thesis and be part of the board of examiners.

I express my sincere thanks to my colleagues and all the members of Ecole Contrale de Lille and the OSSc team for their friendly humour and availability.

I want to express my thanks to my family, particularly *my wife* and my two little daughters *Hala* and *Dana*, who have always supported and encouraged me. Without them I would not be able to accomplish this work.

Finally, I would like to say that without the Grace of Allah, I would never arrive here. So, thanks to Allah first.

GENERAL INTRODUCTION

Manufacturing systems are a class of *discrete event systems* whose elements are interacting together to build products or to perform services. In order to improve the adaptability to the market and the quality of manufactured products and to allow their fast evolution, the implementation of *flexible manufacturing cells* is necessary. However, a large initial cost for the production resources and for the system control design is required.

In the eighties, the concept of *flexible manufacturing systems FMS* has been introduced to develop new systems of manufacturing production able to produce small or average series of products. An FMS is a *discrete event system (event-driven)* that includes the notion of flexibility. It is a production system that consists of a set of machines connected together via an automatic transportation system. Machines and transportation components such as *robots* are controlled by numerical controllers or *CNC*. In all cases, additional *computers or programmable logical controllers PLC* are used to coordinate the resources of the system.

Information and knowledge exchanges in FMS are *controlled communications* (with shorter messages but to be exchanged rapidly). The cell controllers or computers have a lot of functions and are used to control all the operations of an FMS. The control system manage most of the activities within an FMS like parts transportation, synchronising the connection between machine and transportation system, issuing commands to each machine...

Such systems are integrating the *modern communication and control functions in all levels of the system*. *Networking* is extensively applied in industrial applications (local industrial networks). These applications include production systems and more particularly manufacturing plants. The connection of the system components/elements through a network reduces the system complexity and the resources cost. Moreover, it allows sharing the data efficiently. *LANs and Internet* are the most appropriate and economical choices for many system-applications. However, networks performances affect the application services in terms of time-dependent and packet losses.

Thus, the control [Bubnicki05] of such systems is very important. Nowadays, a controlled system [Fadali09] [Paraskevopoulos02] [Burns01] is the combination of *sensors, controllers, actuators and other components/elements* distributed around media of communication, working together according to the user requirements. It is used to manage, command, direct or regulate the behaviour of devices or systems. Combining networks and control systems together reduces the cost and complexity of distributed systems greatly. It facilitates also the maintenance of the systems.

The resulting of this combination is referred to as the *networked control system NCS* [Zhang01] [FWang08]. NCS are one of the main focuses in the research and industrial applications. Networked control systems are entirely distributed and networked control system used to provide data transmission between devices and to provide resource sharing and coordinating management. These benefits have made many industrial companies to apply networking technologies to manufacturing systems applications.

However, there is a need to model such systems to verify some properties such as deadlocks, liveness, boundness and other performance issues. But, the classical modeling paradigm is generally based on a *centralized viewpoint*. Indeed, this kind of modeling does not take into account the fact that the system will be *distributed when implemented* over different machines, sensors, actors, etc. So, the properties that are obtained by the design stage are not necessary guaranteed at the implementation stage

Another issue in such models is that *reconfiguration* [Lejri08] process is not always considered. Today, the reconfiguration capability is a major problem to improve

the functioning of industrial processes. Indeed, a main objective is to adapt quickly the system changes and evaluations: such as the possibility to upgrade the systems or the modifications of the underlying network protocols.

A third problem is that these models does not take into account the underlying network and protocols in terms of performance and information exchange. Networks affect directly the manufacturing system. The behavior and design of manufacturing systems are affected by the underlying networks: particularly performance, mobility, availability and quality of service characteristics. For example, the use of Ethernet will not give the same results if we used Giga Ethernet for the same system.

One way to overcome such problems is to model these systems in a distributed way. A distributed system model offers means to describe precisely all interesting forms of inconsistency as they occur. It takes into account each part in the system, available resources, and system changes together with the underlying network. Once this model is made, its application and implementation are easier since it has the same characteristic as the desired system.

In this context, we propose in this work the modeling of manufacturing systems and their underling network protocols in a distributed model in the form of a client /server distributed system, figure 1. This approach has been originally proposed at the OSSc team to model the manufacturing system with colored Petri nets.

Nevertheless, these systems are complex: massive distribution, high dynamics, faults, and high heterogeneity. Therefore, it is very necessary to model these systems in a way that provides higher degree of confidence and rigorous solutions. One way to cope with this challenge is the use of the component-based methodology which is consistent with the principle of distributed systems in which components are reusable and composable units of code. The component-based approach uses hierarchical and modular means to design and analyze systems. It defines that the system model can be assembled from components working together and the designer needs only to identify the good components that offer suitable services with regard to applications requirements. This

methodology allows the reusability and genericity of the components which reduces the cost of the systems development.

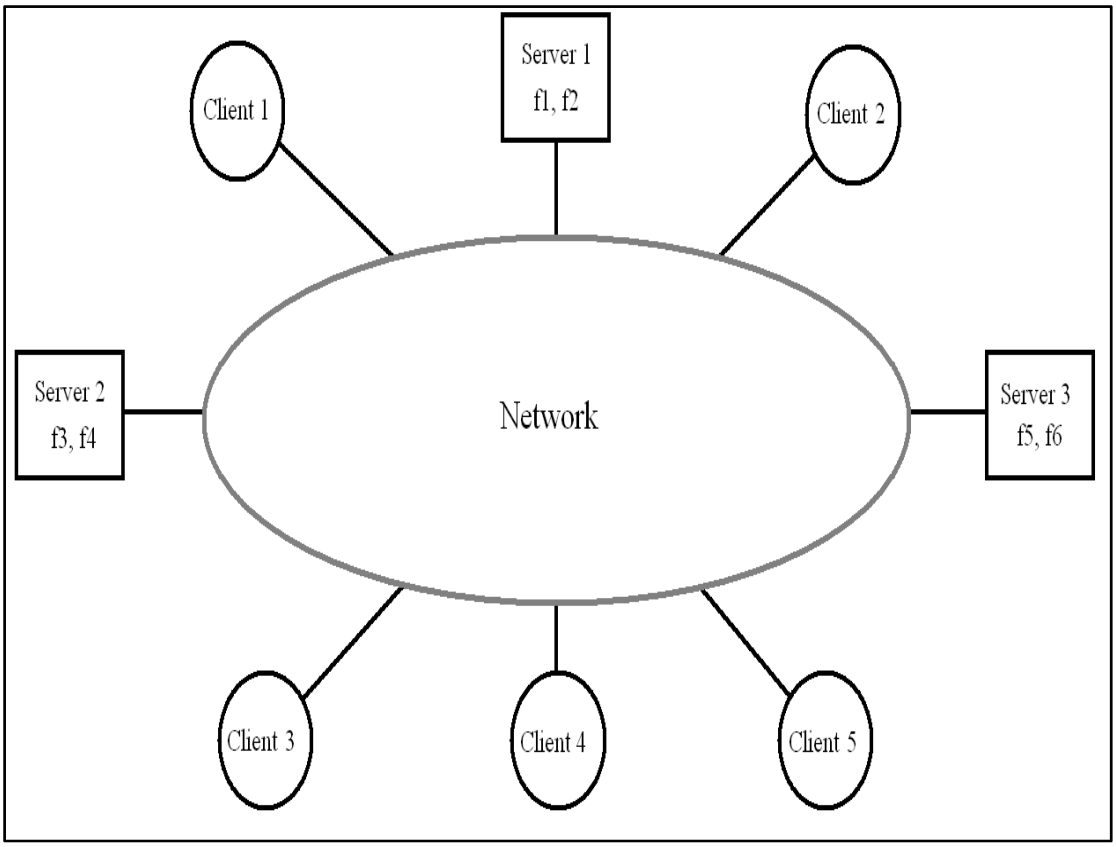


Figure 1, Client/Server Distributed System

Many methods and formalisms are used to model such systems. However, these formalisms are only used to model one view of the distributed systems. The main issue here is the ability to model both communication networks and their details and the distributed services of applications executed over these networks. In this thesis we propose to model them with *High-Level Petri Nets* which is a powerful formalism able to model both views. This ability comes from the ability of Petri nets formalism for modeling concurrent and distributed systems.

Figure 2 shows the composition of a distributed model. The higher level represents the services F1, F2 and F3offered by the different machines M1, M2 and M3 in the system. It represents any other resource on the system O1, O2 and O3 such as the robots

or a transfer unit. The low level represents the workstations that control the service or the resource. The messages exchanged over the network are made between these machines. The medium block represents the kind/class of network used by the system. The model is basically based on generic components (workstations, services, resources…). This can facilitate the upgrade of the system easily; each component is modified separately, and the impact of changing the communication protocol can be evaluated easily.

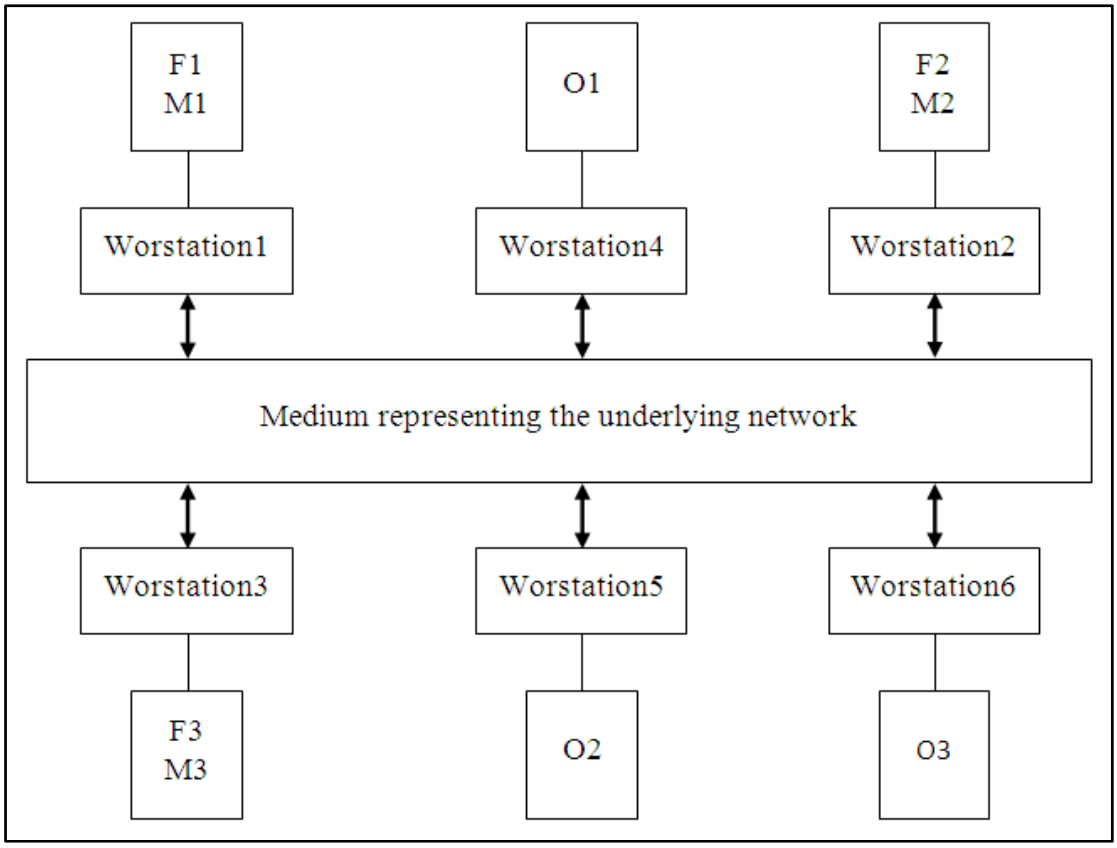


Figure 2, Component-Based Distributed Model

Yet, the presented work is not limited to manufacturing systems. On the contrary, it is valid for any type of DES distributed and networked control systems where services and applications are executed over the network. In this work we focus on the modeling of manufacturing systems controlled over a local industrial network. But since the communication protocols and services are modeled as *blocks*, this allows adapting the method to any type of protocol and any distributed service such as data base systems or transportation systems.

SCIENTIFIC WORKING TEAM

The research work presented in this thesis has been realized within the *Optimisation et Supervision des Systèmes complexes* OSSc team. The OSSc team is one of the *Laboratoire d'Automatique, Génie Informatique et Signal LAGIS* [LAGIS09] (*a research unit of the Centre National de la Recherche Scientifique CNRS*) teams, located at *l'Ecole Centrale de Lille*.

The team is working on two main themes of research:

1- *Quality of service for discrete event systems*: This work aims to build DES models in order to develop or adapt the services delivered by the system to the user requirements. Various viewpoints can be considered such as the diagnostic of the system status; the check of its properties or its performance. It involves several application areas: production systems, computer networks, embedded systems and transportation. The modeling is based on Petri Nets. This mathematical formalism and its multiple and semantic extensions are suitable for analyzing the properties of models and the study of their performances.

2- *The piloting and reconfiguration of production systems*: The work aims to optimize production systems behaviour in the presence of disturbances, both in the short term (piloting) and medium term (reconfiguration). Concerning the piloting, the work aims to maintain the system around a determined operating point by compensating disturbances. The problem addressed is related to the size of the production system. The reconfiguration process is triggered when the operating point can no longer be maintained. In this case, the knowledge of a new architecture helps to synthesize a new control to fill all or a part of the original objectives. The current works are based on the analysis on the fly of the reachability graph of the Petri net model of the system. To avoid the problem of combinational explosion, they relay on the use of solvers.

We believe that Petri nets can be used to unify previous approaches to develop a methodology for comprehensive and integrated design.

THESIS OUTLINE

The thesis is organized as it follows:

Chapter 1 introduces first the distributed control systems approaches, their application and architectures. This section shows the different types of distributed architectures that can be used to implement a distributed system-model. However, this part will not be covered in this thesis. In the second part we introduce the different models proposed to model the manufacturing systems. We focus more particularly on the approach proposed by the OSSc team. This approach will be used in the case study of chapter 4.

Chapter 2 is divided in two parts. The first part introduces the architecture of a communications system, communication networks and protocols models. The second part focuses on the different methods and techniques used to model the protocols and services. We mainly focus on the use of Petri nets and their advantages over other methods.

Chapter 3 first part summarises the different component-based modeling techniques, particularly formal methods. In the second part, we will describe the used modeling formalism: High-Level Petri Nets. In the third part, we specify the modeling technique used for building the patterns and components for communication networks. Finally, we apply our methodology on two illustrative examples: Ethernet and 802.11b DCF.

Chapter 4 will introduce the different models and methods used to evaluate the performance in communication and distributed systems. The second part will give the simulation results of the communication protocols models presented in chapter 3. The last part will combine the modeling of communication protocols and the manufacturing system presented in chapter 1. The impact on the system performances of using different communication protocols will be analyzed.

Chapter 1

MODELING DISTRIBUTED CONTROL SYSTEMS

1. INTRODUCTION

In our days, industrial and commercial systems are integrating the modern communication and control functions in all levels of the system. *Networking* is extensively applied in industrial applications. These applications include production systems and more particularly *manufacturing systems*. The connection of the system elements through a network reduces the system complexity and the assets cost. Moreover, it allows sharing the data efficiently. *LANs* and *Internet* are the most appropriate and economical choices for many system-applications. However, networks performances can affect the application services in terms of time-dependent and packet losses.

In the literature, many methods are proposed to model manufacturing systems. Schematically, most of these approaches distinguish two stages: a *design stage* where one generally uses a formal test to define the functionalities of the control software, and *implementation stage* where this software is translated into code depending on the language of the target industrial computer. But, the complexity of industrial manufacturing systems implies to split the original control software on different computers connected by one or several networks. In this case, as the properties of the original control software were checked with a centralized viewpoint in the design stage, they can no more be guaranteed after the implementation. An alternative way to model such systems is to use *distributed models* at the design stage. This approach allows the consideration of communication protocols and network used to exchange the messages and information between the different parts of the systems to perform a service.

In this chapter we will introduce the different applications that can be used to implement a distributed system (at the implementation stage). However, in this thesis, the implementation stage will not be covered. Only the classical client/server approach will be used in the design stage. The second part of the chapter introduces the different models proposed to model the manufacturing systems. We will mainly focus on the distributed model approach proposed by the LAGIS/OSSc team to model manufacturing systems. This approach will be used for the case study in chapter 4.

2. DISTRIBUTED SYSTEMS: ARCHITECTURE AND APPLICATIONS

2.1 OVERVIEW

Distributed systems [Coulouris01] [Tari01] use multiple workstations communicating to each other via a common network. The underlying networks have been developed to enable many distributed systems to exchange and share resources and services. In the literature, different definitions of distributed systems have been given:

- *A distributed system is a collection of independent computers that appears to its users as a single coherent system* [Tanenbaum95].
- *A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages* [Coulouris01].
- *A distributed system is an information-processing system that contains a number of independent computers that cooperate with one another over a communications network in order to achieve a specific objective* [Puder06].

A *distributed application* executes over a distributed system. In a distributed system, many *autonomous elements* are distributed over different hosts. These components/elements may be used exclusively by a single host and do not need to be homogeneous. Theoretically, the primitives offered by the network allow the elements to interact with each other and to request and give access to their services.

Distributed objects are units that are designed to work together. These units can be in multiple computers connected via a network or in several processes inside the same computer. An object-based distributed system denotes that the object-based model is well-suited for the distributed system. *A distributed object architecture DOA* consists of a collection of interacting objects. Each object consists of a set of data and a set of methods.

An *object* has a set of attributes that represent the state of the object. Attributes that are not accessible from other objects are considered as private or hidden attributes. Private attributes are used to achieve data abstraction. Data stored in hidden attributes can

only be accessed and modified by operations. Changing the hidden attributes will not affect other objects. This is particularly important if objects are designed and maintained in a distributed setting. Objects may export a set of operations that make known the state of attributes or that allow modifying their values. Other objects may request execution of an exported operation. Each attribute has a name. A name is used to identify an attribute within the context of an object. Attributes also have a type. The type determines the domain of the attribute.

Service-oriented architecture SOA [Baker05] is a way of developing distributed systems where the components/elements of these systems are *stand-alone services*, where these services can execute on distributed computers. Service-oriented systems support the reusability of existing implementations and the modification of their run-time behaviour based on the execution environment; in service-oriented engineering, systems can be constructed by composing independent services that encapsulate reusable functions.

The service-oriented architecture has also changed the image of the *web* from the presentation of information to computational infrastructure to satisfy the clients' needs. Web services cover all the aspects of service-oriented architecture. They are platform and implementation-language independent and commonly adopted on *XML-based standard*.

2.2 COMMUNICATIONS IN DISTRIBUTED SYSTEMS

Since there is no shared memory in the distributed systems, nearly all the communications between processes are based on message passing. When process *A* wants to communicate with process *B* it first builds a message in its own address space, then it executes a system call that causes the system to send it over the network to *B.*

2.2.1 CLIENT/SERVER ARCHITECTURE

In *client/server architecture* [Tanenbaum95], an application is modeled as a set of services offered by *servers, a group of cooperating processes,* to the *clients,* figure 1.1. The client/server architecture is often based on a simple connectionless *request/reply protocol*. The client sends a request message to the server asking for some service. The

server does the work and returns the data requested or an error code. The *two-tier client-server architecture* is the simplest client/server architecture. They are in two forms: *Thin/Slim Client Model* and *Fat/Thick Client model*.

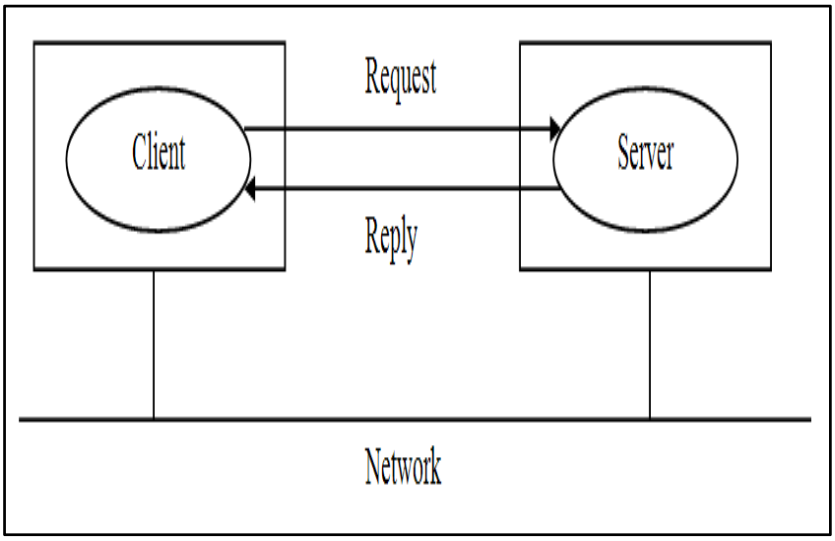


Figure 1.1, Client/Server Architecture

The discovery solution in the client/server architecture is simple. The server has a port and network address known by the client. The clients connect to the server on its listen port. Once the connection is established, the client and server can communicate with each other. The network protocols, for example IP, do the work of sending the packet from the client to the server and back. The system calls can be invoked through the library procedures: *send* and *receive*.

Client/server architecture has some advantages:

- The client/server architecture allows the division of applications into a client part and a server part.
- In the client/server architecture, the resources are used effectively when a huge number of clients are accessing a high-performance server.
- Another advantage of the client/server architecture is the possibility for concurrency.

However, the client/server architecture has a disadvantage that it does not exploit the computing power of the client efficiently as it does with the server.

2.2.2 PEER-TO-PEER ARCHITECTURE

Peer-to-peer P2P [Subramanian05] [Verma04] architecture is decentralized systems where computations can be performed by any host on the network. The term *peer-to-peer* refers to the concept that there are no distinctions between clients and servers in a network of peers using appropriate information and communication systems. In P2P applications, systems are designed to benefit from the computational power available through a huge network of computers. In difference with the client/server architecture, P2P networks gives scalability, lower cost, self-organized and decentralized coordination of limited resources. Middleware may be implemented with a peer-to-peer or a client/server approach.

The P2P architecture has the following characteristics:

1- Decentralization: No central coordinating authority is necessary to coordinate the hosts in network, the use of resources or the communication between the peers in the network (communications between peers are direct).
2- Distributed resources and services sharing: Each host can have/offer both client and server functionality; providing and consuming services or resources.
3- Autonomy: Each host can alone decide when and to what degree its resources are available to other hosts.

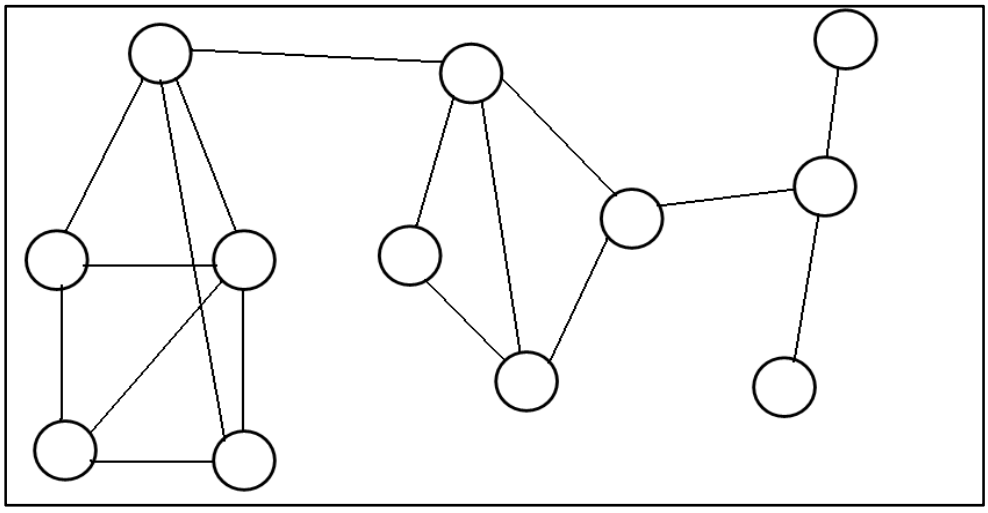


Figure 1.2, Peer-To-Peer Architecture

In P2P architecture, every host could be aware of every other host, could make connection to it and could exchange data with it. However, this is impossible, so hosts are organized into "localities" with some hosts acting as *bridges* to other host localities, figure 1.2. One solution is the use of *semi-decentralized architecture* where a server is used to help establishing connection between peers in the network or to coordinate the results of a computation. For example, the network hosts can communicate with the server to observe the other available hosts. Once they are discovered, direct connection can be established and the connection to the server is unnecessary.

2.3 IMPLEMENTING DISTRIBUTED SERVICES AND APPLICATIONS

To implement a service or application, different methods can be used

2.3.1 REMOTE PROCEDURE CALL RPC

Remote Procedure Call RPC [Srinivasan95a] was invented in the early 1980s by Sun Microsystems as part of their *Open Network Computing ONC* platform. It was included in Sun OS. Sun submitted RPCs as a standard to the *X/Open consortium* and it was adopted as part of the *Distributed Computing Environment DCE*. Remote procedure call systems are the origin of *object-oriented middleware*.

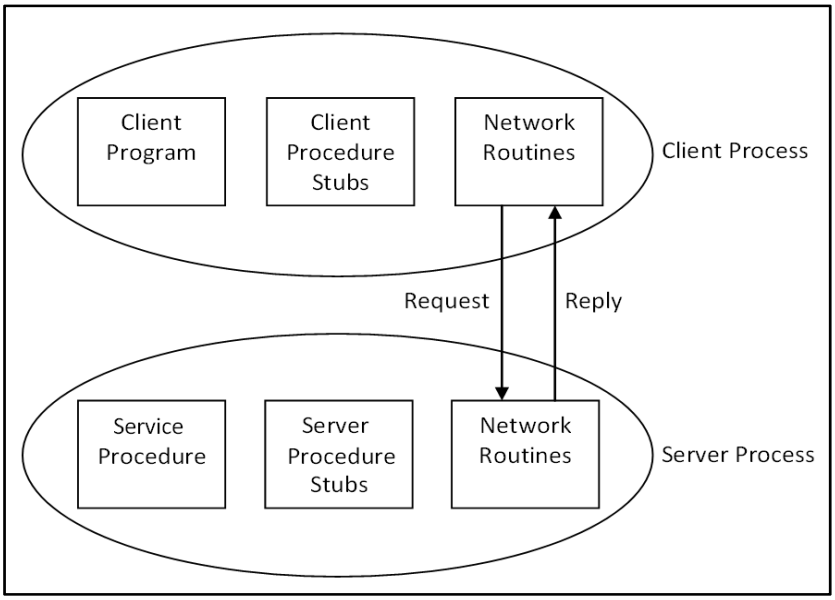


Figure 1.3, RPC Based System

Figure 1.3 shows the role of client and server in RPC. The client and server stub procedures are generated by an interface compiler from the interface definition of the service. The idea in RPC is to hide the message passing, and make the communications look like an ordinary procedure call. RPCs are operations that can be invoked remotely across different hardware and operating system platforms. In some systems, distributed object requests are implemented by RPCs. The server components that execute RPCs are called *RPC programs*. Servers may be clients of other servers to allow chain of RPCs.

2.3.2 REMOTE METHOD INVOCATION RMI

Java [Sun09a] is an object-oriented programming language developed by Sun Microsystems. *Remote Method Invocation RMI* [Sun09b], or *Java Application Programming Interface "Java API"*, was introduced in the version 1.1 of the *Java Development Kit JDK*. Java RMI was considered essentially as an *object-oriented Java RPC*. It extends the Java object model to provide support for distributed objects in the Java language.

Java RMI allows the invocation of methods of Java objects located in a Java *Virtual Machine "Java VM"* (a self-contained Java operating environment that simulates a separate computer) by a remote Java VM by using the same syntax as for the local invocation. The semantics of parameters in Java RMI are not the same as the distributed object model integrated into Java because the *invoker* (an object making a remote invocation) and the *target* (the implementer of a remote object) are remote from one another.

```
import java.rmi.*;
import java.util.Vector;
public interface Profile extends Remote {
        int getVersion() throws RemoteException;
        GraphicalObject getAllState() throws RemoteException;
}
```

Figure 1.4, Interface in Java RMI

Remote interfaces in Java RMI are defined by extending an interface provided in the java.rmi package called *Remote*. Figure 1.4 shows an example of a remote interface. The *GraphicalObject* is a class that holds the state of a graphical object. Sun is providing higher-level services on top of RMI. An example of these services are the *Enterprise JavaBeans* [Matena03] which aim to provide component-based development support for server components and adding higher-level services, such as persistence, security and transactions.

2.3.3 DISTRIBUTED COMPONENT OBJECT MODEL DCOM

Component Object Model COM [Rofail99] [Eddon99] is the Microsoft standard for creating software components. Microsoft has presented the first version of COM in 1993. COM supports the reusability since it allows building applications and systems from binary components supplied by different software suppliers. COM is a specification for the construction of *binary-compatible software components*; it adopts the structure of C++ *virtual function tables "vtables"*. This means that COM is not a library of code, a programming language, or a compiler.

COM is designed to allow the interaction of heterogeneous objects in terms of programming languages. These needs have been solved by the *Object Linking and Embedding OLE* technology based on *dynamic libraries DLL*. In the second version of OLE, a generic object model has been introduced for applications that run on the same workstation. In the structure of the second version of OLE model, nothing prevents that the applications can be distributed. Thus, the implementation of the new middleware named Microsoft *Distributed COM "DCOM"* [Microsoft96] is appeared. DCOM provides a distributed framework based on object-oriented model. *Dynamic Data Exchange DDE* was the first Microsoft object technology in the middle of 1980s, then it has changed to OLE and finally to DCOM.

The DCOM/COM technology support three methods of servers' implementation: *DLL, local shared objects* by locally running applications and remote objects. The DCOM/COM model defines a pattern of access based on interfaces, common to all three

types of servers. In this context, COM is the implementation that supports local communications, and DCOM offers support for access to remote servers.

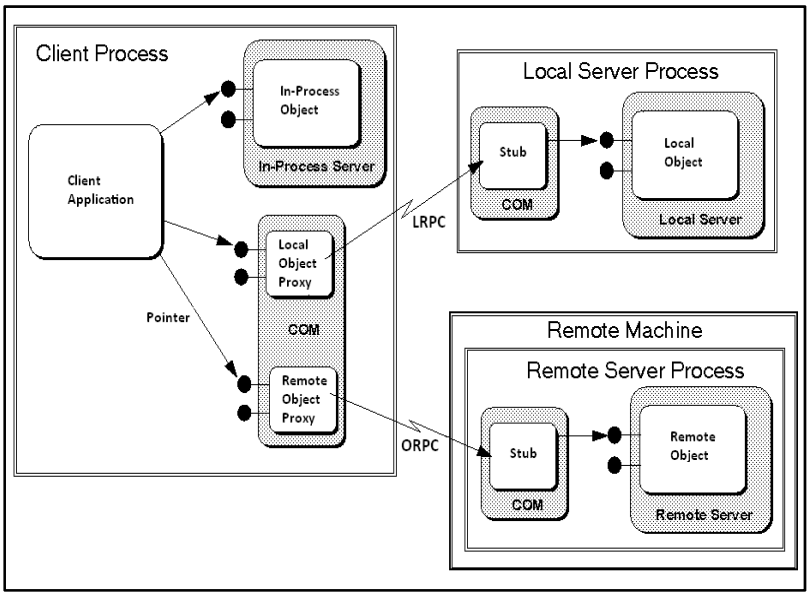


Figure 1.5, Client/Server in Microsoft® DCOM/COM

Client/Server dialogue in DCOM/COM technology can be in three ways, figure 1.5:

1- If the client and server belong to the same addressing space, COM loads the server code and gives the client a pointer offering direct access to the server.
2- If the server does not belong to same client addressing space, but locally on the workstation, the dialogue is done through local method calls LRPC (Lightweight Remote Procedure Call).
3- Finally, if the server belongs to a remote address space, DCOM uses object-oriented RPC (ORPC).

2.3.4 COMMON OBJECT REQUEST BROKER ARCHITECTURE CORBA

Common Object Request Broker Architecture CORBA is specified by the *Object Management Group OMG* [OMG09a]. CORBA is based on the concept of the *Object Request Broker ORB.* The OMG is a consortium primarily composed of the research and software industry. It was created in 1989 to support the adoption of standards for the development of distributed object applications.

The first specification published by the OMG was the *Object Management Architecture OMA* [OMA09]. The OMA defines the object model used in all the OMG technologies that offers the objects interoperability through heterogeneous environments. CORBA is the second stage of the OMG specifications. In version 1.1, 1991, CORBA has defined the *interface definition language IDL*, (figure 1.6). In version 2 of CORBA, 1996, the OMG made the interoperability its priority by defining the Internet Inter-ORB Protocol IIOP. In addition to the interoperability, many services were added.

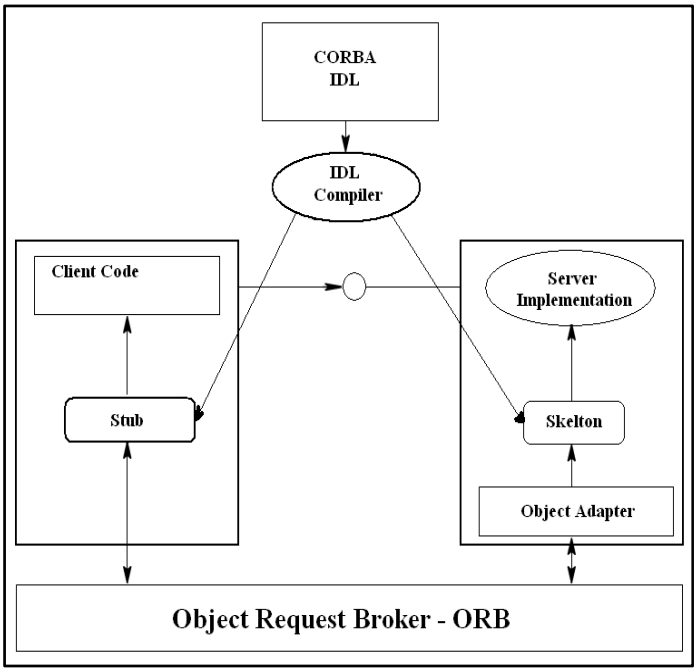


Figure 1.6, IDL CORBA

CORBA [Tari01] architecture is independent from the programming language used, the machine type and the operating system. CORBA is composed mainly of the ORB which is the heart of the CORBA architecture and the objects supported by the ORB. ORB assures the transport of requests on the network. It is responsible for intercepting methods invocations, locating the objects, carrying the invocations parameters through the network and transmitting any return values of the methods. CORBA specification has rise in several variants, such as a specification for the Wireless CORBA and a specification for the real-time called RT-CORBA. The *CORBA Component Model CCM* is the main novelty of the version 3 of CORBA.

2.3.5 WEB SERVICES

The development of *World Wide Web* [W3C09] [Lerner02] has allowed client computers to access to remote servers outside their own organization. *Web Services* [Erl05] are basically any software infrastructure that provides a service accessed remotely via the web. They are considered as a collection of operations that can be used by a client over the Internet. Each web service is identified by a *Uniform Resource Identifier URI* [W3C01]. Web services are *client/server model*. However, they are capable to play different roles according to the surrounding context, so they are *not absolutely a client or a server*. Services can be distributed on the Internet. To communicate, they exchange messages, expressed in the *Extensible Markup Language XML and* distributed by using Internet transport protocol such as *TCP* and *HTTP*. A service defines its needs from another service by setting out its needs in a message and sending it to that service

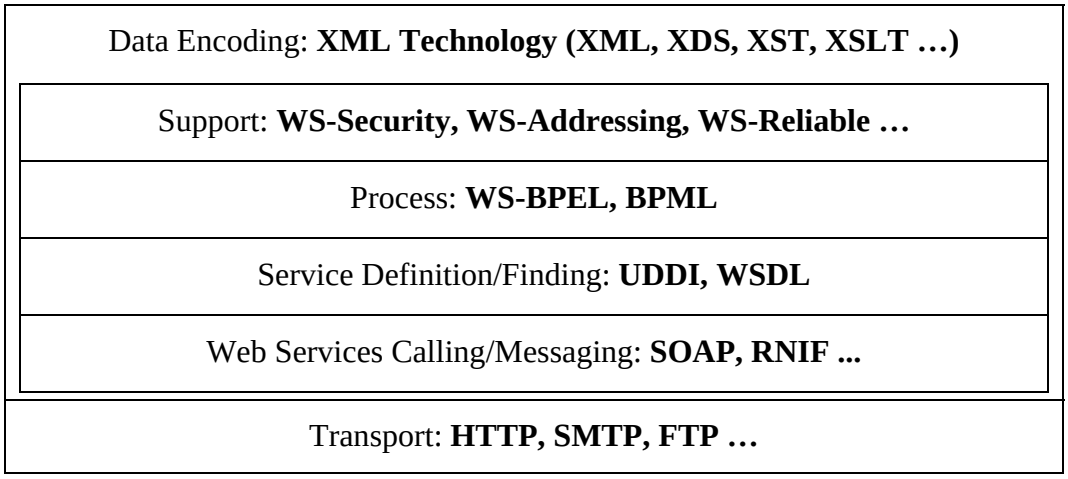


Figure 1.7, Web Service Standards

Although the definition of web services is large and flexible, in practice the major part of Web services is based on *HTTP*, *SOAP*, *WSDL* and *UDDI* [Newcomer03] [Cer02]. Web service protocols cover all the aspects of SOA from the service information exchange (SOAP) to programming language standards (WS-BPEL: a standard for a workflow language that is used to define process programs involving several different services). These standards are based on XML (figure 1.7), which is a human-machine notation that allows the definition of structured data, where text is tagged with a meaning identifier.

3. MODELING MANUFACTURING SYSTEMS

3.1 MODELING FMS WITH PETRI NETS

Modeling the manufacturing systems is not new. Different modeling formalisms are used to model FMS. Petri nets and Automata theory are of the most used techniques to describe the DES. [Peterson81] has introduced Petri nets to model and analyze the manufacturing systems. Other works like [Ramadge87] has also proposed to model and analyze controllable events DES. Later, [Lin90] has introduced a decentralized control model. [Zamaï98] has presented a hierarchical and modular architecture for real-time control and monitoring of FMS. [Cho99] has proposed a centralized and decentralized control model. However, the most new modeling approaches are diverting towards the distributed modeling for the manufacturing systems [Petin05] [DaSilveira02b].

In the previous section we have introduced the different methods that are used to implement the distributed system design. In this section we will introduce the new techniques to model the manufacturing systems with Petri nets and other formalisms. We will meanly focus on the distributed approach model proposed by LAGIS/OSSc team.

3.1.1 PETRI NETS OVERVIEW

Petri nets have been proposed by C. A. Petri in 1962 in his PhD thesis "*Communications with Automata*" [Petri66]. Petri nets [Merlin76] [Murata89] are a mathematical and graphical tool used for modeling, formal analysis, and design of different systems like computer networks, process control plants, communication protocols, production systems, asynchronous, distributed, parallel, and stochastic systems; mainly *discrete event systems*.

As a graphical tool, Petri nets provide a powerful communication medium between the user and the designer. Instead of using ambiguous textual description, mathematical notation difficult to understand or complex requirements, Petri nets can be represented graphically. The graphical representation makes also Petri nets intuitively very appealing. They are really easy to understand and grasp. This is due to the fact that

Petri net diagrams resemble many of the informal drawings which designers make while they construct and analyze a system.

A Petri net graph [JWang07] contains two types of nodes: *Places* "*p*" and *Transitions* "*t*". Graphically, places are represented by circles, while transitions are represented by bars or rectangles, figure 1.8. Places and transitions are directly connected by *arcs* from places to transitions and from transitions to places with. A place *P0* is considered as an input place of a transition *t* if there is an arc from *P0* to *t*. A place *P1* is considered an output place of a transition *t* if there is an arc from *t* to *P1*. By default the weight of an arc is 1. However, an arc may be annotated with a positive number *k* called weight (or multiplicity). This value can be seen as k-parallel arcs. Places can contain tokens represented by dots. These tokens are the marking of places. The initial marking of places is represented in the initial marking vector m_0. The graphical presentation of Petri nets shows the static properties of the systems, but they also have a dynamic properties resulting from the marking of a Petri net.

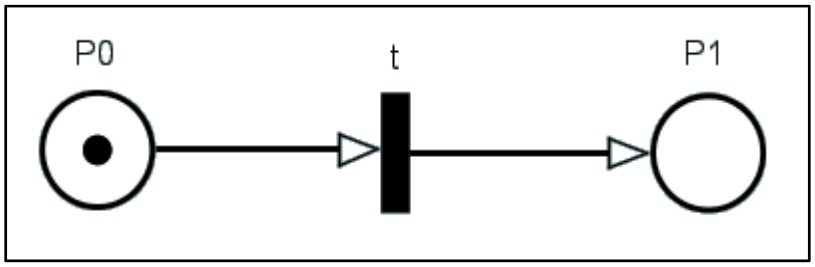


Figure 1.8, A simple Petri Net

As a mathematical tool, a Petri net model can be described by a set of linear algebraic equations, linear matrix algebra, or other mathematical models reflecting the behaviour of the system. This allows performing a formal analysis of the model and a formal check of the properties related to the behaviour of the system: deadlock, concurrent operations, repetitive activities…

3.1.2 WHY PETRI NETS?

A *Discrete Event System DES* [Cassandras08] [Ramadge89] is a *discrete-state, event-driven, dynamic system*. The state evolution of DES depends completely on the

occurrence of asynchronous discrete events over time. Figure 1.9 shows the states jumps in a DES from one discrete value to another whenever an event occurs during the time. Nearly all the DESs are *complex* and require a high degree of correctness. Information systems, networking protocols, banking systems, and manufacturing and production systems are falling into this classification.

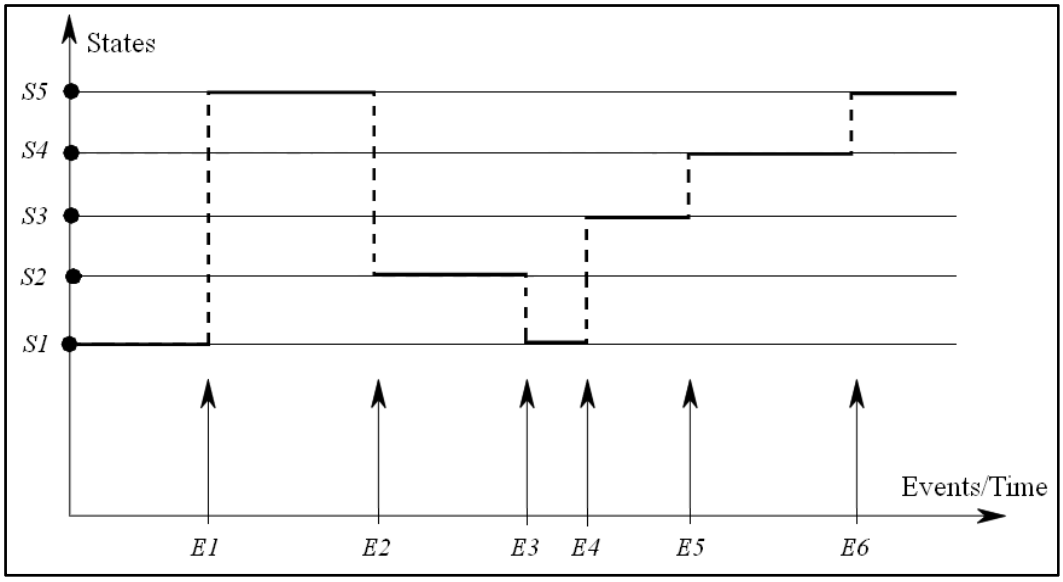


Figure 1.9, Discrete Event System

One way of dealing with these problems is to model discrete event systems with Petri nets since:

1- Due to their flexibility and the formalism power, Petri nets can express concurrency, asynchronous and parallel actions, nondeterministic choice, synchronization, distribution, causality and most system properties. All these features raise the modeling power to various types of system behaviour at different abstraction levels and provide an excellent formal framework for modeling a variety of systems.

2- The simple graphical presentation of Petri nets model: the ease to visualize the state-flow of a system and to see dependencies between the parts of a system. This simplifies modeling and understanding systems because of its declarative, logic-based modeling principles.

3- The mathematical basis: Most of the computer systems developments are determined by formal mathematical methods. The main element is linear algebra; therefore many properties like net analysis and verification are possible by theoretical means.

4- Petri nets are supported by a variety of extensions and tools. These tools support the simulation of the model. The resulting provides an abstract view of systems behaviour.

5- The Petri net model permits the simultaneous occurrence of multiple events, without increasing the model complexity.

6- Many other methods and formalisms can often be transformed into Petri nets supporting them with a formal semantics.

3.2 THE ARIZONA STATE UNIVERSITY/INTEL APPROACH

The work of at the Arizona state university/Intel [Sarjoughian05] proposes a simulation modeling combined with decision control for semiconductor supply-chain manufacturing. The importance of this proposition is the benefits for the analysis, design, and operation of supply-chain network systems. The work proposes a discrete-event system specification with four types of modules: *inventory*, *factory*, *shipping link*, and *customer modules* with a common interface specification for control/decision commands, figure 1.10.

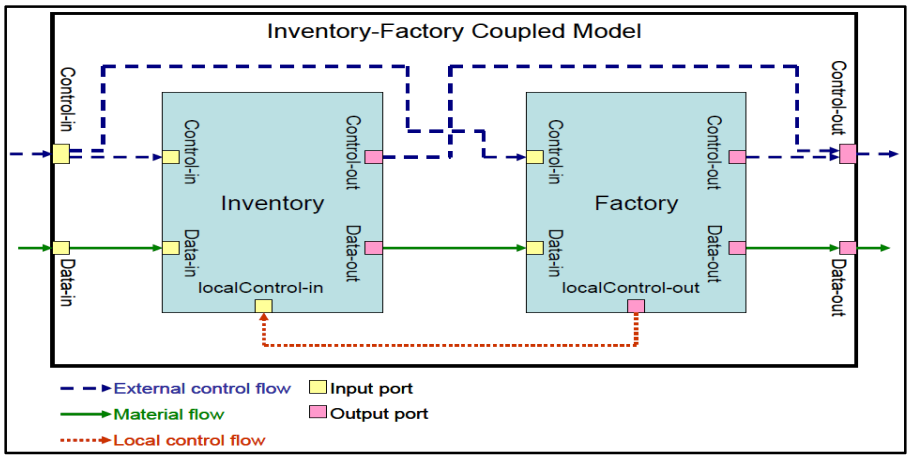


Figure 1.10, Structural Composition of Inventory and Factory Models

The control model is generally used for the control of highly stochastic processes where selection of control actions is desired. In this model, the current and historical measurements of a process are used to predict its behaviour for future time instances. Figure 1.11 shows a model predictive control. The simulated system sends its current outputs to a system prediction model. The system prediction model then computes future outputs (i.e., controlled outputs) for some number of time steps.

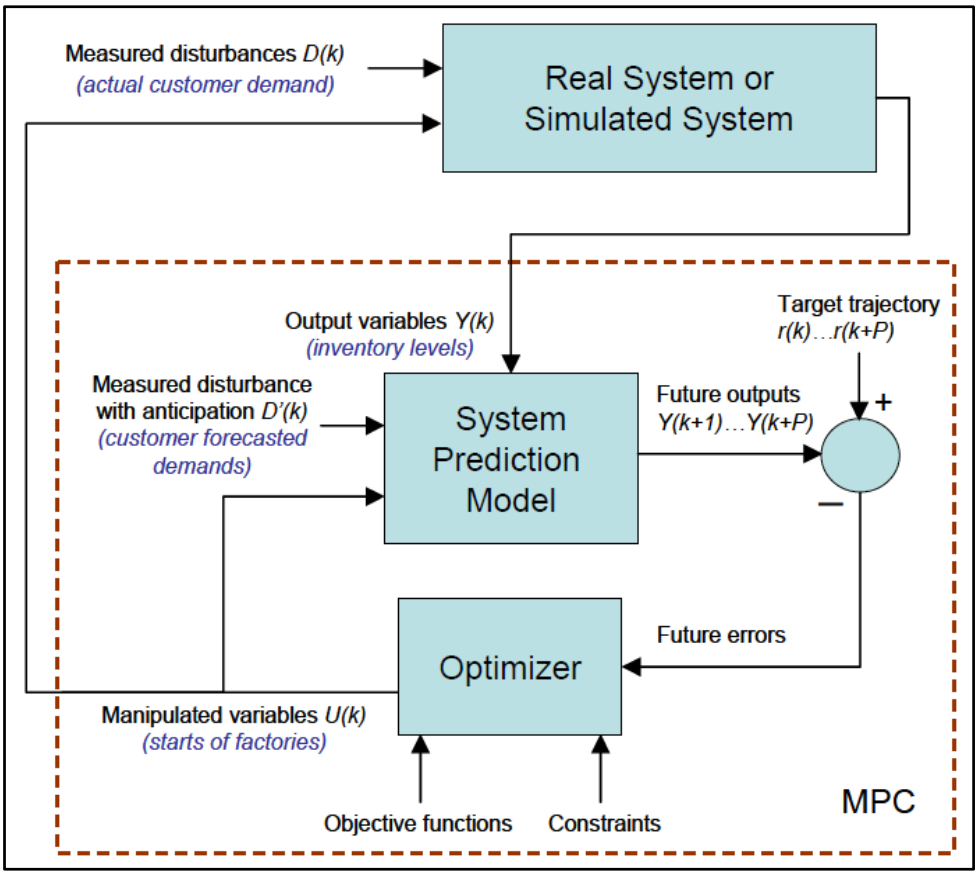


Figure 1.11, The Model Predictive Control

To compose models regard the less distinct syntax and semantics for each module, figure 1.12, a *knowledge interchange broker* is used. This structural specification provides well-defined structural information translation from the manufacturing process network model to the model predictive control and vice versa.

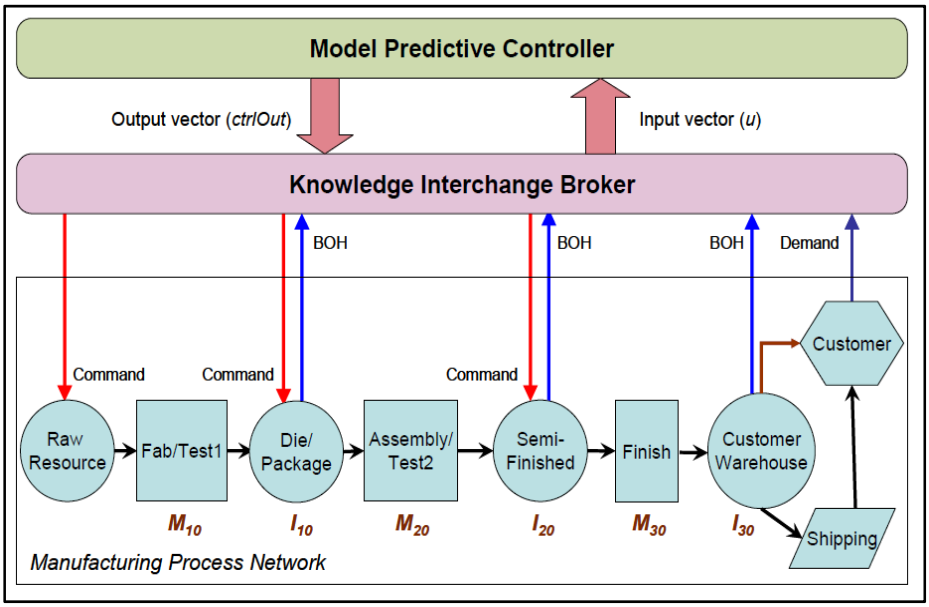


Figure 1.12, Composition of Manufacturing Process Network and MPC via KIB

This approach supports systematic specification of interactions between process dynamics and control decisions. The model composition is used to embedding model interactions inside the process and control models as is required when using interoperability in combination with model exchange.

3.3 THE NHIT (TAIWAN) APPROACH

The National Huwei Institute of Technology [Tsai05] has proposed web-based model (figure 1.13) for distributed manufacturing control systems with problems such as process routing, allocating resources and scheduling work-pieces. The manufacturing parts and resources were presented by *agent-based approaches*. The supervision of the system is done by a web-based cell controller. This work presents the new evaluation of modeling with web-based technology. This new technology can support collaboration between geographically distributed work centres and makes the implementation easier.

Agents are composed of sub modules responsible for negotiation between agents, executing the different operations of the agents and recognising and analysing errors. Agents represent the machines and resources in the system. Agents communicate together through a local network or Internet. C*oordination and negotiation protocols* CNP were

used for the decision-making of resource allocation and message exchanging, based on TCP/IP protocol. Figure 1.14 shows the description of the contract net protocol using unified model language agent.

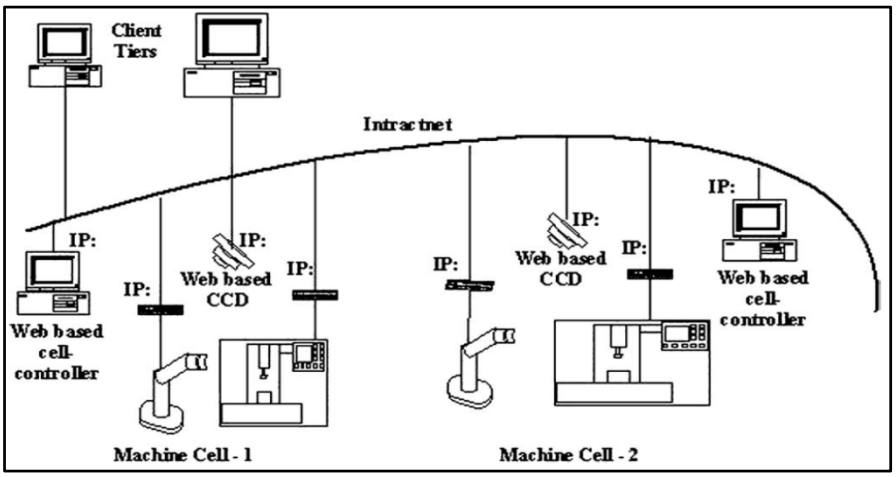


Figure 1.13, The architecture of a web-based distributed manufacturing control system

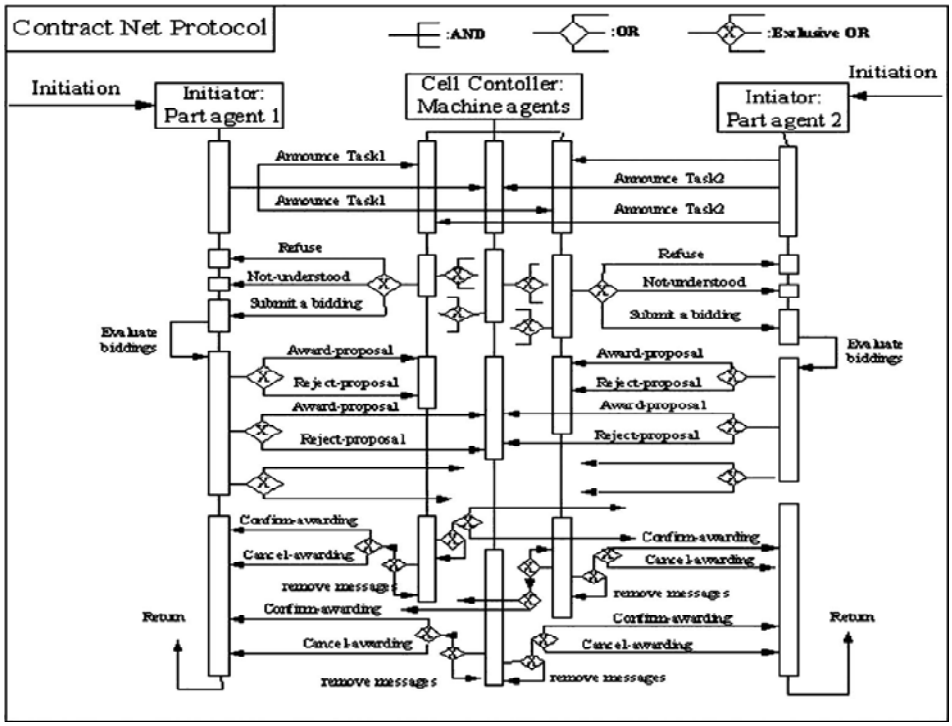


Figure 1.14, the CNP described by a unified model language agent

This work shows a new architecture for a web-based distributed manufacturing control system for the design of co-operative mechanisms for better system performance.

3.4 THE CRAN APPROACH

The recent work of CRAN [Gouyon04] has proposed a product-driven control, hierarchical architecture. This work is based on the "*agile manufacturing*". The modeling technique is based on two models of the systems: a control model for the operative part elements (OPE) and a behavioral specifications model of the control part (OPE). The control model is decomposed in two points of view: production resources and products, figure 1.15.

The production control of the product is composed of (i) a routing control sub-process to the different resources and (ii) an operation sub-process of the performing resource coordinated by the product. While the resource control is composed of: (i) a control sub-process that receives and deals with the messages send by the product and sends to product its actual status after transformation and (ii) an operating sub-process that transforms the sent messages of the control sub-process to physical action over the product. The products, according to their needs, will ask for resources operations.

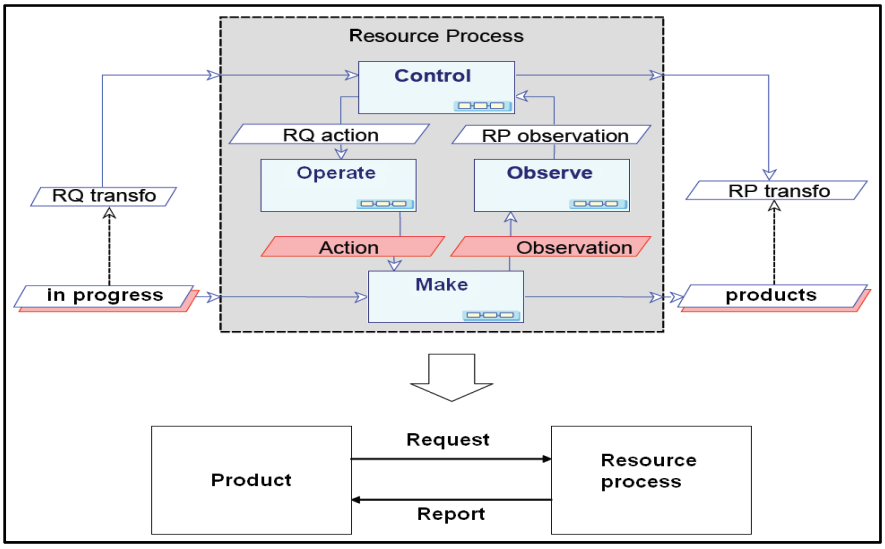


Figure 1.15, Product-Resource Model [Gouyon04]

The modeling is realized with *finite-state automata* to allow modeling a process resource controlled by the product. It models the behaviour of the resource seen by the product through exchanged messages between them (request/report).

This work puts the product at the center of the process automation. It ensures the interoperability between the control of the resources of the manufacturing system and the product control through its routing on the system.

3.5 THE LAG/G-SCOP APPROACH

The work of LAG/G-SCOP [Henry05] [Mendez02] [Zamaï06] proposes a control, supervision, and monitoring module. [Henry05] has proposed a coordination model for the functional chain. The model is composed of two levels: coordination level to manage and coordinate the local control/communication and functional chain level that groups all the elements of the operative part, figure 1.16.

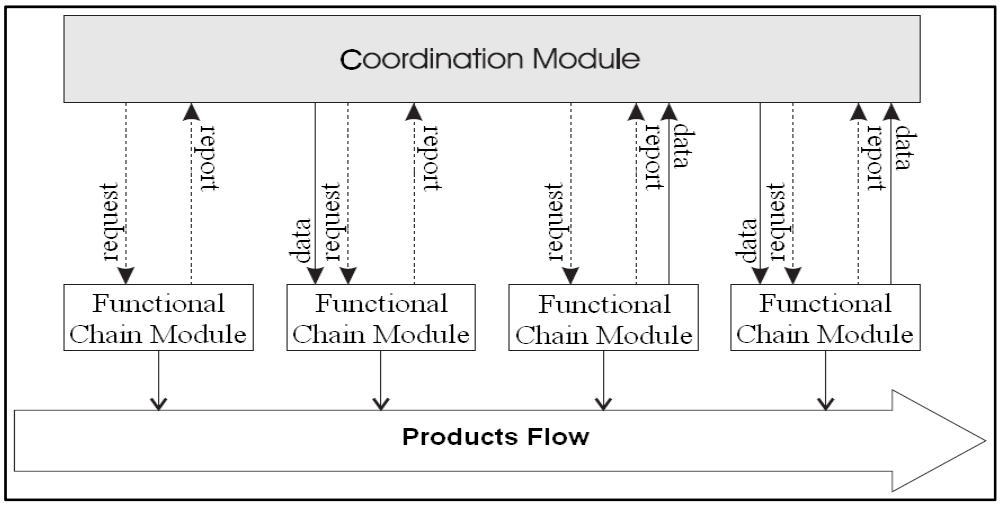


Figure 1.16, The LAG/G-SCOP Approach [Henry05]

A decision module is applied on the approach CERBERE of [Mendez02]. This module is charged to take all the required decisions to generate or select a control module, a resumption module or an urgency module (figure 1.17), depending on criteria (quality and production) and constraints (security in the operative part). To execute the requested services from the different functional chains, control rules are used. The control model has a set of the orders (request a service, report the end of a service, information sent from the environment) that it executes to impose some evolutions over the operative part and the products. These evolutions, that answer a request, must satisfy a set of security constraints.

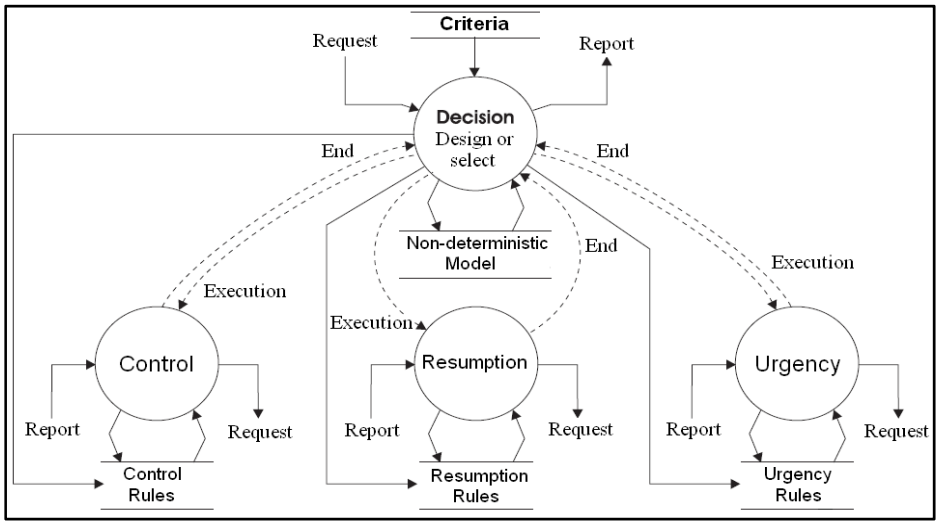


Figure 1.17, Decision analysis [Henry05]

A research technique is also used in the control model that aims to find the shortest path to make a product (initial state to final state) [Zamaï06]. This technique is achieved by transforming the proposed model into a states representation containing the set of executable operations and the legal parallelism between these operations and the one in progress. This representation is modeled in automata and Petri nets.

This works have two objectives, the first one is to propose a modeling formalism for the control part and the second is the elaboration of a technique for the control synthesis.

3.6 THE LAAS APPROAH

The objective of the work developed by LAAS is to propose a *generic, "heterarchical" and distributed architecture model* for the manufacturing systems [DaSilveira02a] [DaSilveira02b]. The proposed "*heterarchical*" architecture is based on *no client/server architecture* for the communication between entities, no external higher levels of control to coordinate processes and the addition or modification of existent *entities* without significant structural changes. The work proposes an a*cquisition/routing block* that deal with control, supervision and monitoring of the system, figure 1.18. The work proposes also a systematic procedure for distributing a centralized model of supervision and control [DaSilveira02b].

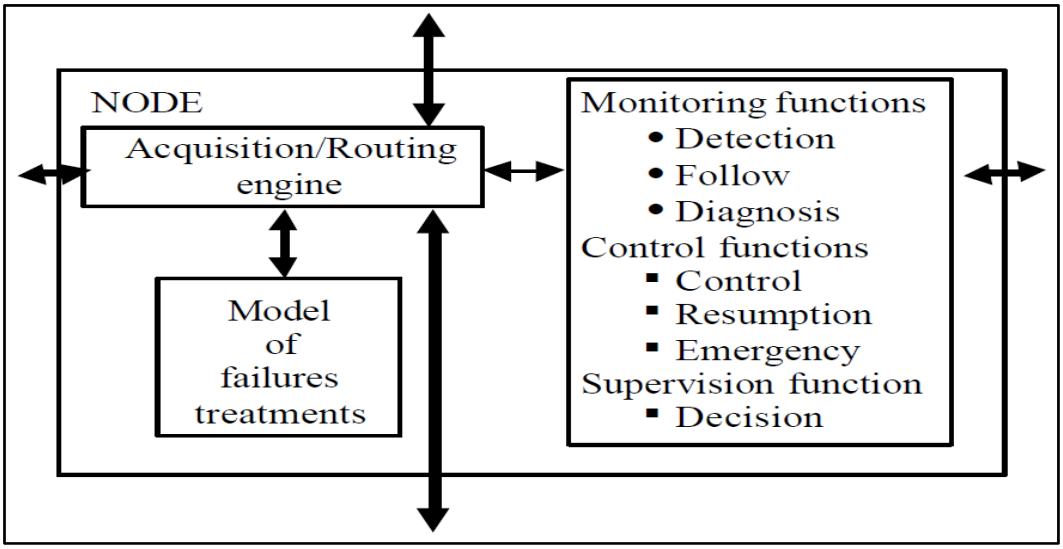


Figure 1.18, A generic module for control, supervision and monitoring [DaSilveira02b]

To distribute a centralized model, the model is divided into "*sub-models*". Each sub-model represents a resource or a set of resources and the relations with other resources. Two different models are used. The *control model* represents all constraints associated to the transformation of raw parts to finished products. While the *process model* represents the physical and functional characteristics of the process. An *entity* or a *module* has a process sub-model and a control sub-model.

To achieve this distribution, the centralized Petri net model of two processes is transformed and split up to two Petri nets sub-models. To maintain the data coherence after distribution, a *communication protocol* is used. The proposed technique for the communication protocol is to centralize the decision part in one module called *centralized decider module* to optimize time.

The work presented concerns the quantification of the redundancy inserted by the distribution methodology by proposing a systematic procedure, from a centralized process model (specified by Petri nets), to obtain a distributed model with partial redundancy. However, the intra-module and inter-modules communications are not detailed but assumed to be *client/server protocol*. Also, the work did not give a solution for the complexity associated to the process distribution as well as the coherence between entity models.

3.7 THE LAB-STICC APPROACH

The research works developed at Lab-STICC concerns the design of reconfigurable DES systems such as manufacturing systems or electronic systems. These works are structured around three main ideas [Berruet07]: reconfiguration of complex systems, a top-down methodology associated with "pivot" description languages for the co-design of systems, and a bottom-up methodology based on a component approach to allow rapid prototyping and the reuse of the code.

1- Reconfiguration of complex systems:

The reconfiguration consists on organizing the system to react in two cases: in case of a new demand of the system and in case of a reaction to a failure. To implement the reconfiguration process, they propose two key ideas. First they propose different point of views to describe a system. A system can be described according with a physical point of view or a logical point of view. For example, the logical sequence is a logical view of the architecture of a system. A second way to describe a system is the distinction between the architecture of a system and the configuration. The architecture defines the potentialities of a system. Its configuration defines a specific way to exploit the system.

The second key idea to implement this concept is to propose the introduction of a configuration task in the structure of the supervision function of the control system (figure 1.19). This introduction seems to be suitable with the context of the exploitation of the system. In this case, the decision task can select a new configuration. The role of the configuration task is to define the mode of resources that participate to the production and to define the operation that can be held by the system in this configuration.

A drawback of this proposition is to not consider the role of the maintenance function to define a configuration. For us at LAGIS, we consider that a configuration results of a negotiation between the maintenance function and the planning function because the engagement of a resource in a production depends also on the maintenance planning of this resource.

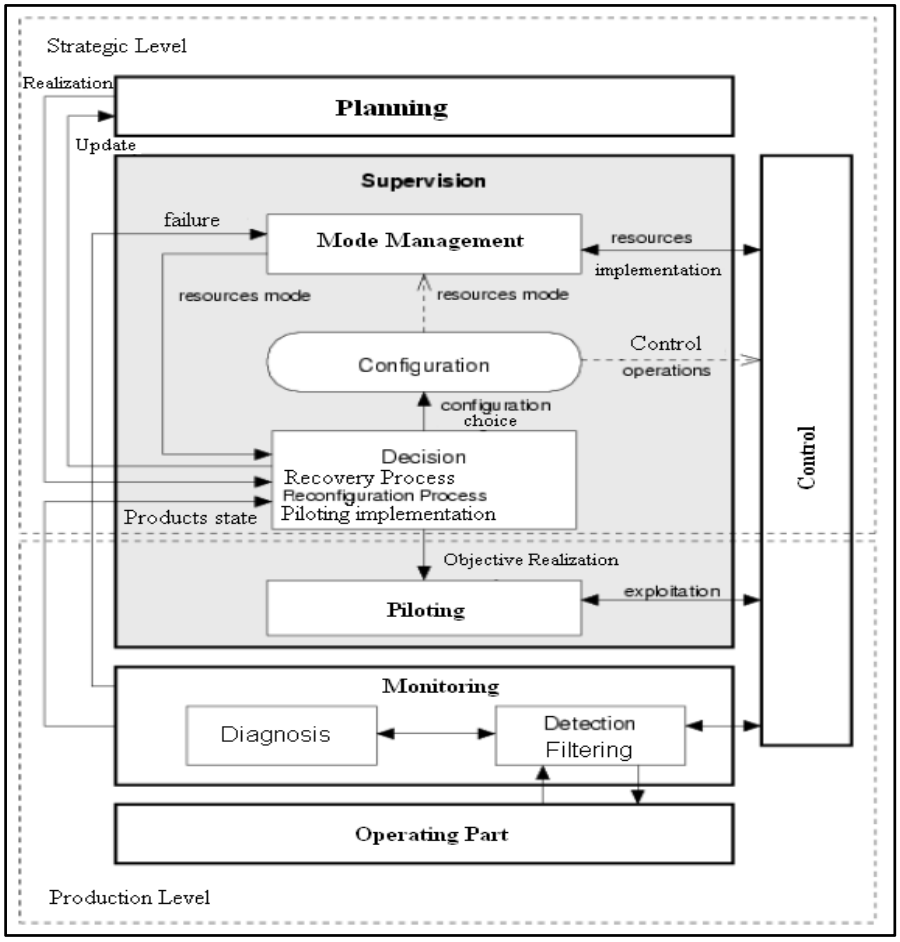


Figure 1.19, Proposed Control Architecture [Berruet07]

2- A top-down methodology for the design of complex systems:

One of the conclusions of the Lab-STICC is the complexity of these systems leads to decompose their control function in different tasks that are studied and implemented separately. The consequence is that they used different models of the system to be controlled without any guarantee of coherence and in consequence, without guarantee of interoperability of these tasks. To deal with this problem, the Lab-STICC proposes to use a description language to build a reference and principal model of the system. Then, all the other models required by the design methodology or the different tasks of the control can be derived from this basic models using techniques such as extraction to focus on specific aspects of the system or enrichment to take into account additional viewpoints (Figure 1.20).

In this context, they use techniques of model translations to automate the translation from one model to the other. In this context and also taking into account a principle like co-design that is well-known in electronics area, they propose a top down approach that allow developing incrementally the operative part and the control function of manufacturing systems. In this methodology, they propose different techniques do make static and dynamic analyses of a system. For example, using model translation technique, they can build a model that enable to evaluate the criticality of the functions of a plant. Another translations, allow to derived other models suitable with dynamical analyses by the means of simulation (joint simulation [Lallican07] and reflexive simulation [Berruet07])

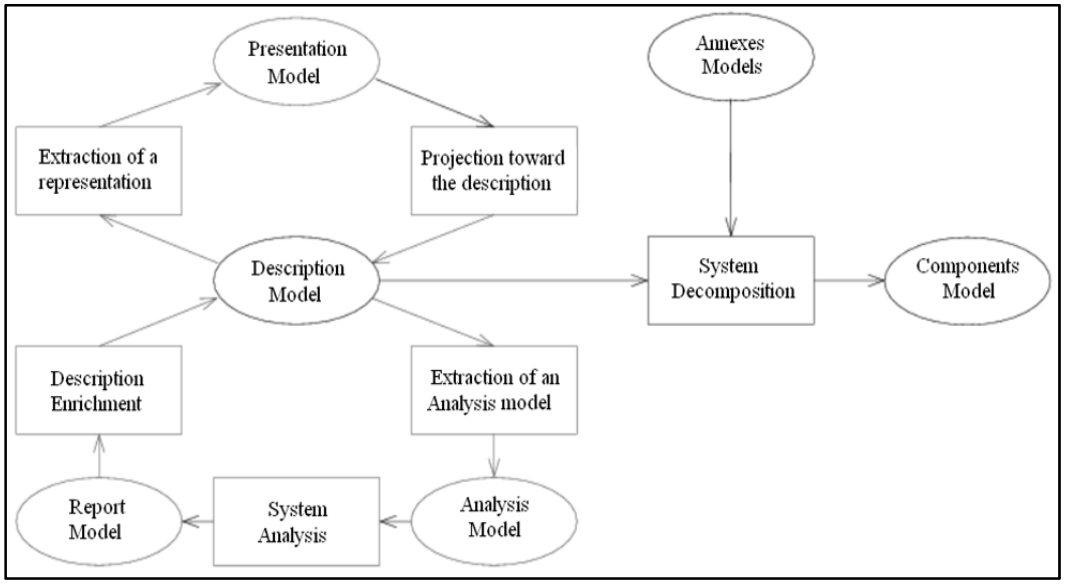


Figure 1.20, Synthesis of the process of model the management proposed by the Lab-STICC [Delamotte05]

3- A bottom-up methodology based on a component approach:

In an effort to reuse and accelerating the design stage of a production system, till several years the Lab-STICC is developing a methodology for generating the control command of the transport system of reconfigurable systems. This methodology is based on the concept of component. In this approach, a component is characterized by a set of operations and a set of views. Each point of view of a component is associated with a

model. The main views are the operative part, the graphical view, the constraint view, the control part view, monitoring/supervision view.

It may be noted that most of these views correspond to tasks of the control command. Indeed, the idea is for each view, to get a global model from the models associated with its different components. The construction of the system model is made by a bottom up approach from basic components defined in a component library. The composition of components is done according to different levels defining other types of components: *support component, effective contextual component, enriched basic component,* and *system component*. A component system includes all components of the system. The nature of each component depends on the type of its operations (basic, contextual, contextual effective) [Lallican07] and its position in the hierarchy of components.

A major feature of this approach is the constraint view. It enables the designer to express constraints that must be checked during the integration of the component in a given system. The current constraints are taken into account functional and operational features, and also safety. One can think that in future they will integrate all requirements including also aspects of reliability and performance. These constraints allow linking the functional capacities of component with its state. So, they are taken into account for the generation of the model corresponding to any of previous views.

This approach has been developed and supported by software for the generation of control function of DES transport system. Compared with the top-down approach, it assumes that the designer already has the plan instrumentation diagram of the plant. Consequently, it is limited to the generation of control models for system transitique.

This approach seems to have also inspired other works such as the automation of the life environment of persons with reduced mobility [Belabbas07]. In these works, we find the concept of the component from the perspective of a black box with input and output interface that allow a rapid building of a control by aggregation of components (Figure 1.21). We will see in chapter III that the approach we propose is close to the main

general principle of this work. However, the Lab-STICC approach does not take into account underlying communication protocols.

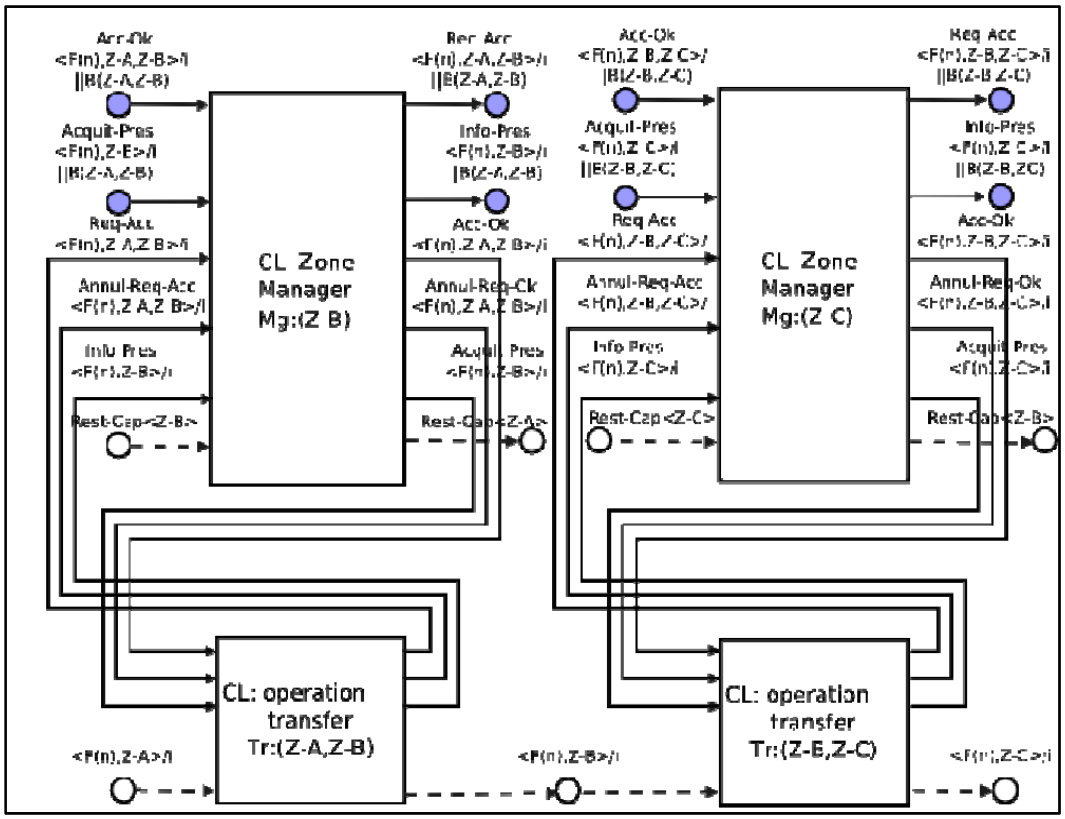


Figure 1.21, PN intellectual properties proposed by the Lab-STICC [Belabas07]

3.8 THE LAGIS/OSSC APPROACH

The *LAGIS* laboratory develops a consistent, progressive and complete design approach of FMS. This approach is implemented by modeling controllers' components in Petri nets formalisms and in a client/server distributed architecture [Huvenoit93] [Toguyeni06] [Bourdeaud_huy06]. The idea here is to implement each operating sequence as a PN model where each place represents the state of a product with regard to its sequence of machining operations and also its location in the plant.

The approach distinguishes two categories of controllers [Toguyeni06]: operating sequence that controls the different operations applied to a product and Graph of Coordination of Complex Resource GCCR that controls the operations applied to a

complex resource (a resource with several machining areas connected by transport resources). The aim of this decoupling is to reduce the complexity when designing the control of complex systems, figure 1.22:

1- *Process sequences:* They describe the different operations to apply to raw parts to obtain a finished product. The *Extended Operative Sequence* describes the different ways to obtain a *finished product* from *raw parts* using the available machines of a plant.

2- *Resource Sharing*: The resource sharing implies that resources are allocated to the requestors. Resources can be simple (*mono service*), or complex (*multiple services*).

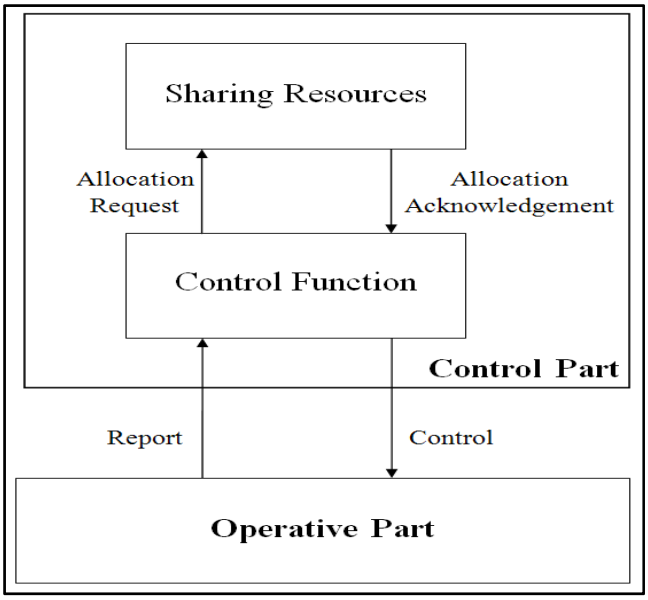


Figure 1.22, LAGIS Control System Architecture

To illustrate the approach, we consider the example of figure 1.23. This workshop is made of three machining machines M1, M2 and M3. The arrows in this figure represent the reachability capacity of each robot R1 to R4. R4 performs transfer operations from *FIFO IN* (a buffer that permits the entry of parts in the plant) to Z1, and from Z1 to *FIFO OUT*. Z1 to Z4 are to transfer a part from a resource to another one. IS1 to IS6 are intermediary stock within the conveyor. For more information, please refer to [Toguyeni06].

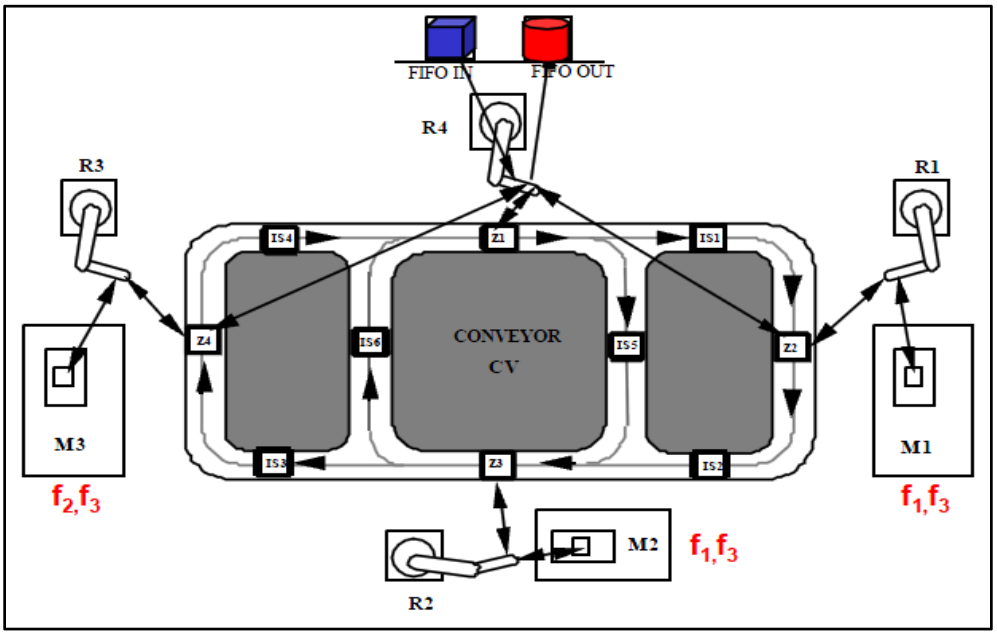


Figure 1.23, An Illustrative example of Manufacturing Plant with Flexibilities [Berruet98]

To model the system, colored Petri nets are used:

1- *Extended Operating sequences:* A resource operation is requested through a pair of Request/Acknowledge places, figure 1.24. The doublet **<op, id>** represents the type of operation to apply to a product and the identification of each product. The request place enables an asynchronous coordination with the CPN controller of the resource.

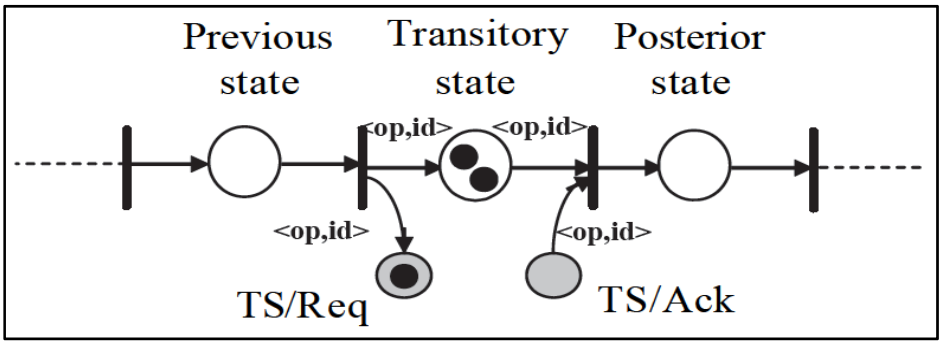


Figure 1.24, Coordination between an operating sequence and a GCCR.

2- *Resource allocation:* In flexible manufacturing plant, it is necessary to solve allocation problems. So, an allocator is needed corresponding to each resource.

The allocator can be modeled by CPN (or by other formalism), figure 1.25. The client/server technique permits the communicating between operating sequences with these allocators. The approach is also characterized by the use of *pre-allocation notion* which is useful to increase the performance of the FMS by enabling objects' transfer in masked time.

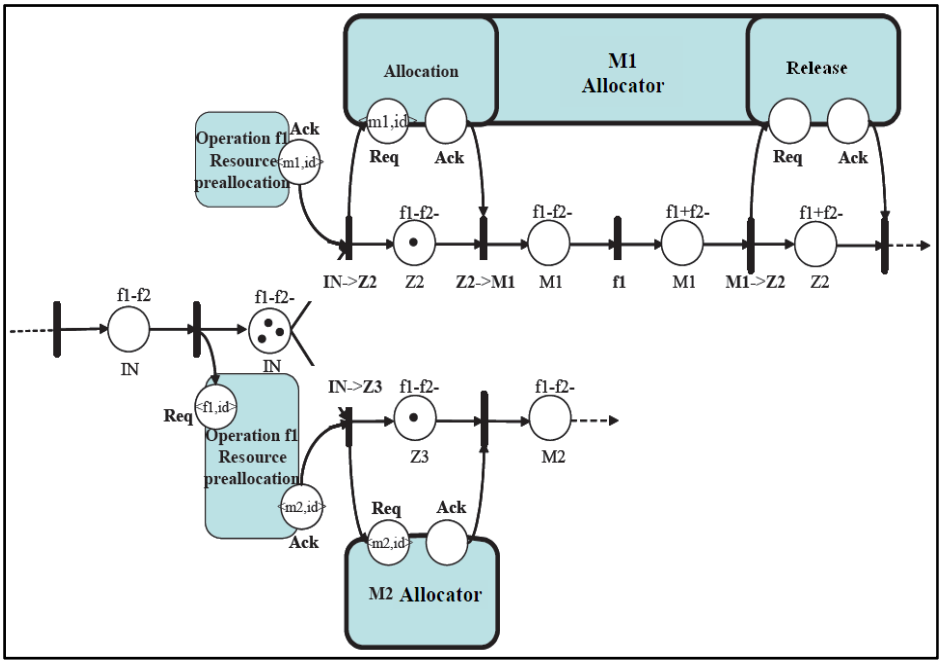


Figure 1.25, Resource Allocation based on Client/Server Technique

3- *Graph of coordination of Complex Resources:* The transport resources represent the direct access for a transfer between two physical areas: requestor and destination. The requestor sends a request to start the transfer. When the transfer finishes, the requestor area receives an end acknowledgment. However, a transfer cannot be done without the acceptance of the destination area (a free place at that area).

Figure 1.26 shows an example of extended operating sequence. In the figure a product is performed on machine M2. The pre-allocation/allocation of resource is requested at each state. The "–" sign means the product before machining, and the "+" means the product is finished.

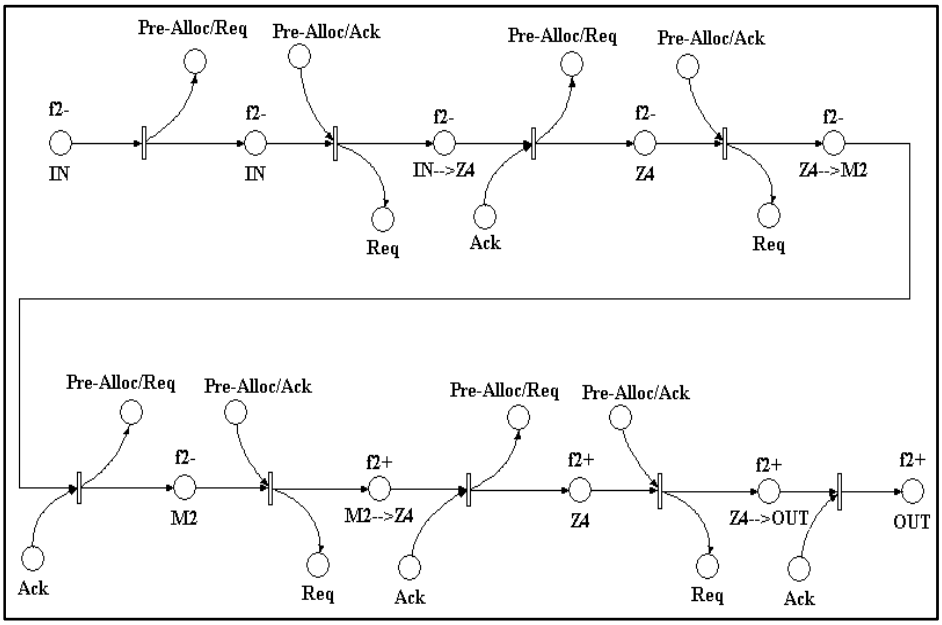


Figure 1.26, Extended Operating Sequence to perform service f2 on machine M2

Figure 1.27 shows the transfer process between two physical areas P1 and P2 by a robot R. At the beginning the product arrives to the source area, P1. The source area sends a request to the transfer component (robot) to transfer the product to the destination area, P2. However, a free place in the destination area must be available to perform this transfer. If this is the case, the robot takes the product from the source area releasing up a free place in P1. The robot then puts the product on the destination area ending the transfer process.

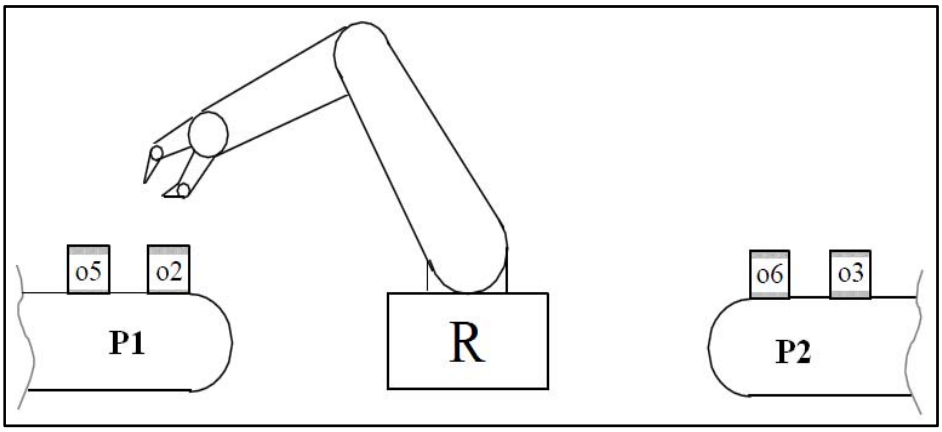


Figure 1.27, Transfer between two Areas [Bourey88]

Figure 1.28 shows the developed Petri net. Each area has four interfaces modeled represented as places:

1- Place "*CONS"* models the number of free places in the destination area.
2- Place "*PROD*" models the parts that are waiting in the source area.
3- Place "*REQ*" models a request of evacuation of the current part.
4- Finally, place "*ACK*" models a response of the process to confirm that the part is out of the area.

The figures also shows the messages exchanged between the two physical areas and the robot, represented by P1→ P2 sub net.

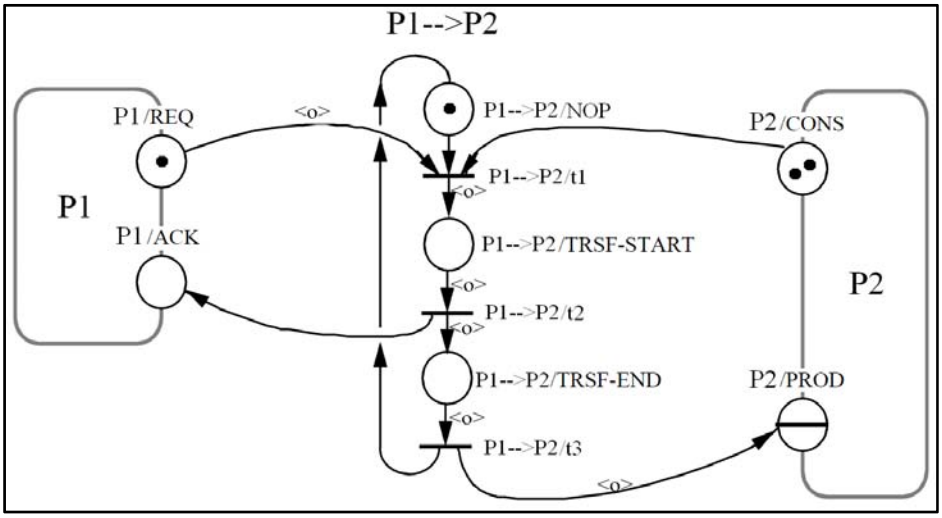


Figure 1.28, Developed Petri Net of Figure 2.25

4. CONCLUSION

In this chapter we have introduced different proposed approaches to model manufacturing system. The proposed models show the systems at the design stage where none of these models focused on the implementation stage. In addition, these models do not take into account the underlying network during the design stage. However, different to the universities laboratories' approaches (as the work in this thesis), the industrial companies are more and more interested in presenting the implementation of these models with web services, CORBA, RMI or DCOM where the code can be distributed over several computers [SOCRADES09] [Tsai05] [Schoop00].

In this chapter, we have introduced the different techniques that can be used to build the design stage model in a distributed form. However, in the rest of this thesis for the design stage, we will focus on the classical client/server approach for the modeling of manufacturing systems. The implementation stage approaches presented in the first section of this chapter will not be treated in the rest of this work. In the next chapters we will focus on the modeling of communication protocols and the implementation of the LAGIS/OSSc approach in a client/server distributed model, chapter 4. We will also evaluate the impact of the underlying network on the system, chapter 3 and 4.

In the next chapter, we will mainly focus on the modeling of the communication protocols underlying the manufacturing systems. Many formalisms are proposed such as *UML*, *Timed Automata*, and *Petri Nets*. Petri nets formalism is a powerful methodology for modeling manufacturing systems since it mainly allows the modeling of concurrent, distributed and parallel systems. This formalism has an outstanding mathematical basis as well as its graphical interface that allows easily the designer to study the behaviour of the model.

Chapter 2

COMMUNICATION SYSTEMS AND MODELING TECHNIQUES

1. INTRODUCTION

The modern advances in hardware technologies has played a big role in the rapid development of communication networks and distributed control systems. Distributed systems use networks for communication. Communication networks are generally built from various transmission media including wire cables, fiber and wireless channels, hardware devices including routers, switches, bridges, hubs, repeaters, and network interfaces and software components including protocol stacks, communication handlers and drivers.

The collection of hardware and software components provides communication facilities for distributed systems. This collection forms the communication subsystem. The computer and other devices are referred to as hosts. A node is referred to any computer or network device attached to a network. The cooperation between communicating devices is governed by a set of rules called a *protocol*. Protocols form a major aspect for distributed systems design. A familiar example of distributed systems is Internet and its services.

To evaluate protocols, modeling and simulation approaches and tools can be used and executed on computer. Different formalisms are used to model distributed services or communication systems (*UML, Timed Automata, SDL, Petri Nets* ...). However, Petri nets are one of the most appealing and used methods for modeling distributed systems. Petri nets were used initially to study the interconnection properties of concurrent and parallel activities. Thus, it is not surprising that we use them to model both protocols and distributed services.

The underlying network of a manufacturing system is the *industrial local network*. So, we will mainly focus on this type of networks and mainly the LAN MAC sublayer protocols. In this chapter we will introduce the architecture of a communications system, communication networks and protocols models. In the second part, we focus on the different methods and techniques used to model protocols and services. We mainly focus on the use of Petri nets and their advantages over the other methods.

2. COMMUNICATION SYSTEMS ARCHITECTURE

2.1 COMMUNICATION SYSTEMS OVERVIEW

Communication systems [Gebali08] [Proakis02] are designed to send messages or information from a source that generates the messages to one or more destinations. In general, a communication system can be represented by the functional block diagram shown in figure 2.1. The original telecommunication system was developed for voice communications. Today communication networks include all types of voice, video and data communication over copper wire, optical fibers or wireless medium.

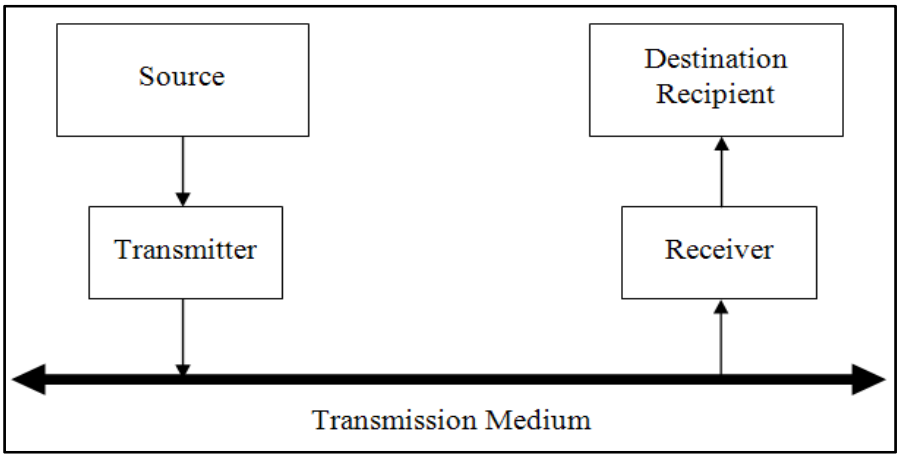


Figure 2.1, Functional Diagram of Communication System

With Internet, increasingly numbers of computer networks are now connected via the Internet. The concept of telecommunication system has increased the complexity significantly. These systems can be divided into different types based on their requirements:

- **Point-to-point Communication**: In this type, communication takes place between two end points.
- **Point-to-multipoint Communication**: In this type of communication, there is one sender and multiple recipients.
- **Broadcasting**: In a broadcasting system, there is a central location from which information is sent to many recipients, as in the case of audio or video

broadcasting. In a broadcasting system, the listeners are passive, and there is no reverse communication path.

- **Simplex Communication**: In simplex communication, communication is possible only in one direction. There is one sender and one receiver; the sender and receiver cannot change their roles.
- **Half-duplex Communication**: Half-duplex communication is possible in both directions between two computers or persons, but one at a time. These types of systems require limited channel bandwidth, so they are low cost systems.
- **Full-duplex Communication**: In a full-duplex communication system, the two parties can communicate simultaneously, as in a telephone system. The ability of the communication system to transport data in both directions defines the system as full-duplex.

2.2 NETWORK LAYERING ARCHITECTURE

Networks [Peterson03] [Mir07] [Stallings07] are organized into a hierarchy of layers where each layer has a well defined function and operates under specific protocols. The number of layers can vary from one network reference model to another but the goal of a layered structure remains common to all models, (figure 2.2). *OSI model* [Zimmermann80] is structured in a series of 7 layers, while the *TCP/IP model* includes only four layers. Each layer consists of hardware or software elements and provides a service to the layer immediately above it.

OSI is a general model which is therefore applied to many kinds of networks. Each layer of a host will "talk" to the layer of the same level of the recipient host, (figure 2.3). The set of rules that makes two layers of the same level can communicate is called *protocol*.

However, the dialogue between two layers of level *n* is not direct from one layer to another layer. Instead, each layer of network communicates by local procedure calls with the layers above and below it. The sending layer (or layer *n*) transfers the information to the layer immediately below it (layer *n-1*). Layer *n-1* transfers the

information in its turn to the under layer until reaching the physical medium. The message is then transmitted in bits. It then goes upward the same layers of the recipient until it reaches layer *n*.

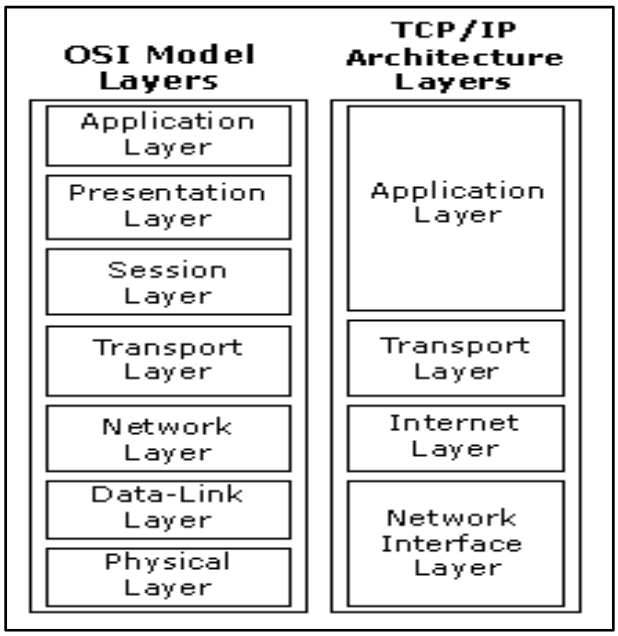


Figure 2.2, OSI and TCP/IP Reference Models

At the transmission, each layer adds a header in the message and sometimes a tailor to check transmissions' errors. This feature will provide necessary information for the same layer of the recipient. This information may relate to the size of the message, its time-to-live and the source and destination addresses. At the reception, each layer reads the header reserved for it, processes the information, disguards this header and then transmits the message to the upper layer. All these operations are of course transparent to the user. Moreover, if the transport layer has limitations on a messages size with respect to the layer immediately above it (for example the network layer over the transport layer), this message will be fragmented, and then sent as several independent packets. Layering provides some important features:

- Layering provides a more modular design. New service can be added easily only by modifying the functionality at one layer, and at all the other layers just reusing the functions provided by them.

- Networks are decomposed into more manageable layers, instead of implementing it in one layer that does everything.

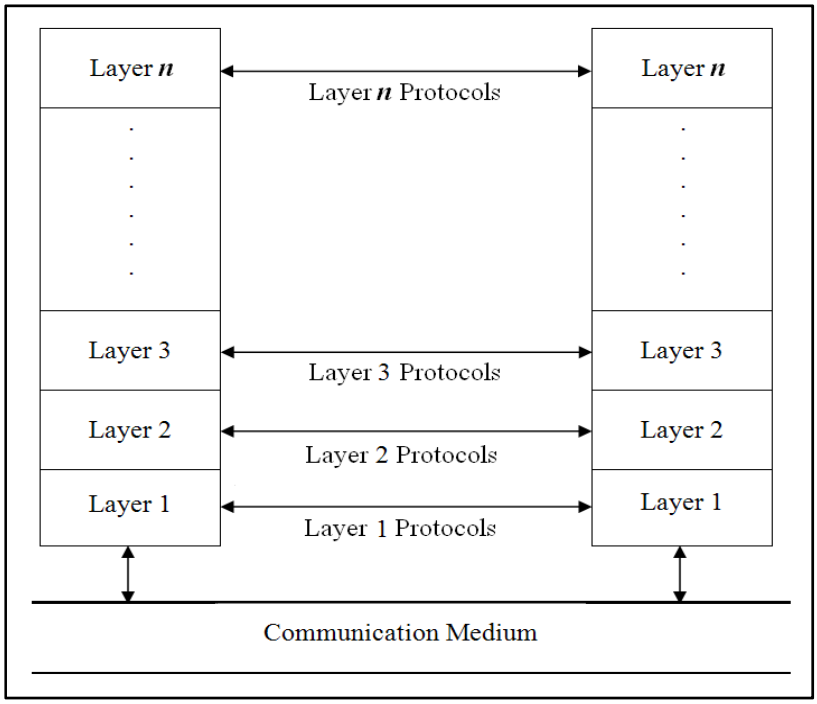


Figure 2.3, Network Layering

2.3 PACKETS ENCAPSULATION MECHANISM

Each layer uses *Protocol Data Units (PDUs)* to communicate and exchange information. The data transmitted from one host to another host should go down all the layers. Each layer adds a header (PDU attaches to the data) and the data are enclosed with protocol information. The data changes name at each level. From TCP/IP architecture viewpoint, the different names are

- *Data message* at the application layer.
- The message is then encapsulated in a *segment* in the transport layer.
- Once the segment encapsulated in the Internet protocol layer, it is named *datagram* or *packet*.
- It changes to *frame* at the network access layer.
- Finally, the physical layer encodes these digits into a digital signal.

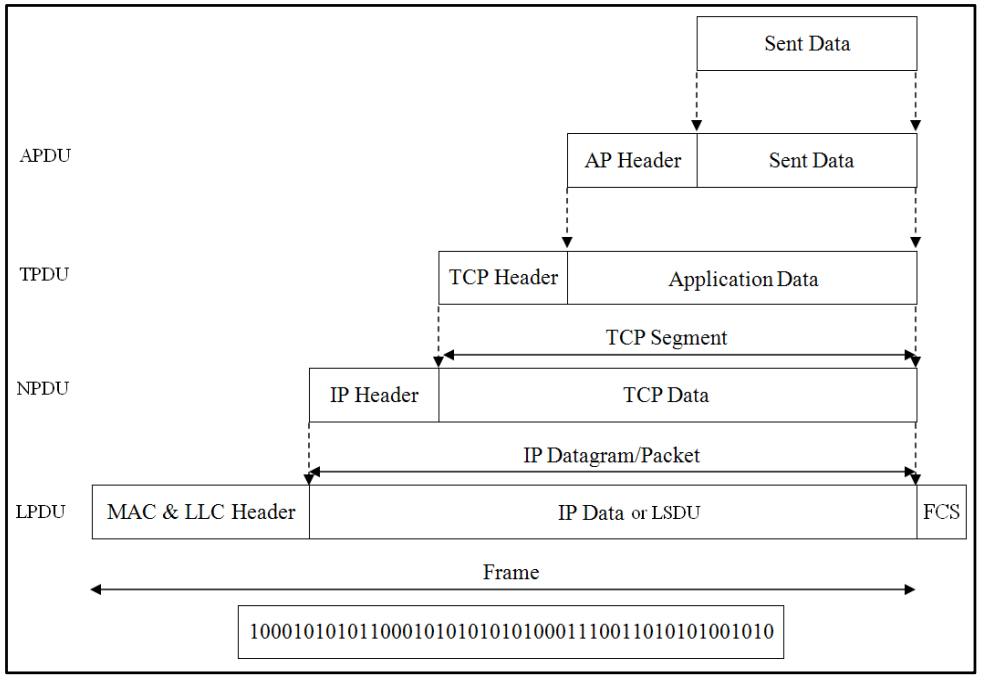


Figure 2.4, Data Encapsulation in TCP/IP architecture

When the PDU in layer *n* is passed down to layer *n-1*, it becomes data in layer *n-1*. The layer *n* PDU is now referred to layer *n-1* as its *service data unit SDU*. Layer *n-1* transports this SDU by placing it into its own PDU envelop. This process is called *encapsulation* [Dulaney09], (figure 2.4).

2.4 PROTOCOLS AND INTERFACES

Computer networks use well defined protocols to communicate. A *protocol* [Lammle08] is defined as a set of rules and formats controlling communications between processes at the same layer and agreed by all of the communication participants. Protocols' rules specify the sender, the receiver and the message sent by the sender. Each participant plays a certain role. An application on a host wishing to transmit a message to another host may issue a call to a transport protocol. It passes it a message in the specified format. Each protocol is designed to achieve a certain goal. The goal can be expressed by one or more properties the execution of the protocol should satisfy. Properties are generally dependent on the environment in which the protocol is deployed. The definition of a protocol has two important parts to it [Jia05]:

1- A specification of the format of the data in the message,
2- A specification of the sequence of messages that must be exchanged.

As an example, the TCP header has a fixed length of 20 bytes, (figure 2.5). It may be followed by options. Each header field has its own information. For example, the fields source and destination port number identify the ends of the connection. The sequence number field specifies the sequence number of the sent data (within a TCP stream, each byte of data is numbered). The length of the options field is variable.

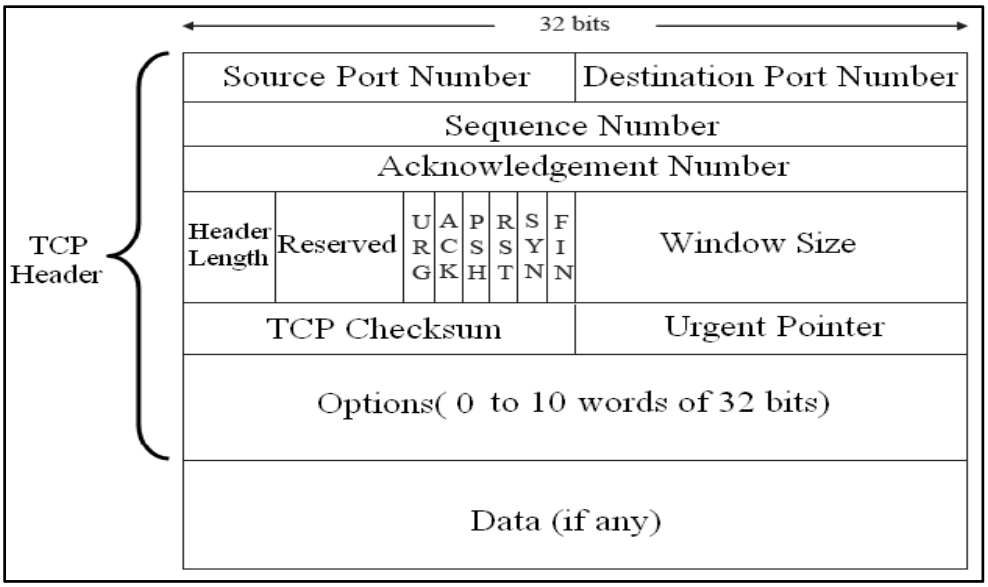


Figure 2.5, TCP Header

In TCP protocol, the flags field has in important role in the connection and transmission processes. A TCP connection is opened with the *3-way handshake* that creates and negotiates the data connection, figure 2.6. At first, a synchronization packet (SYN flag is set to 1) is sent from the transmitter side to open a new connection, with a sequence number equals to *m* (initial sequence number of the transmitter). The recipient side receives this packet and sends again a synchronization acknowledgment where the SYN and ACK flags are set to 1. When the transmitter receives the recipient acknowledgement packet, it sends an acknowledgement packet to receiver but this time with ACK is set to 1 and SYN is set to zero indicating that the packet is monger a synchronization packet with *m+1* sequence number. Once these messages are exchanged,

the transmitter can send its data to the recipient workstation. However, to end a connection, the transmitter sets the FIN flag to 1 indicating the end of transmission.

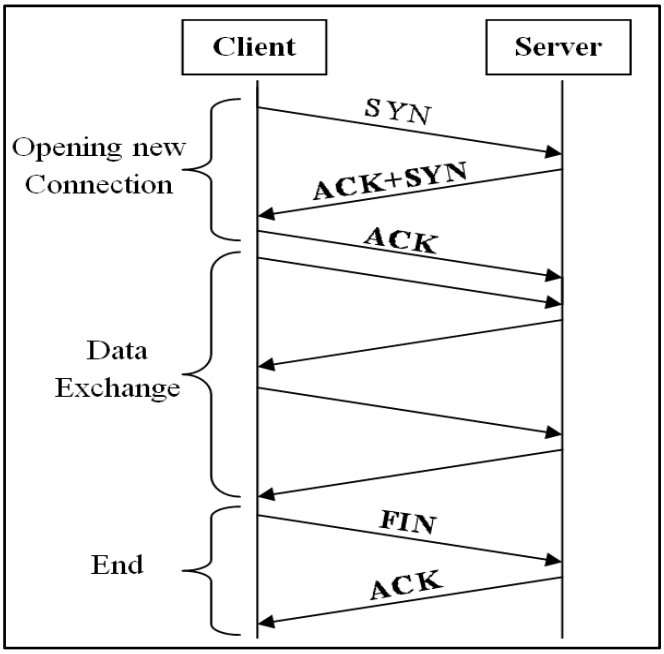


Figure 2.6, TCP connection

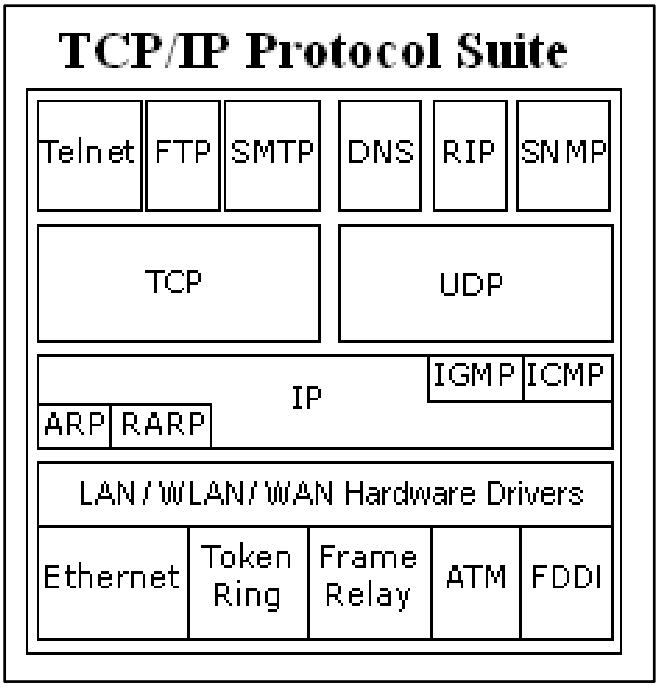


Figure 2.7, TCP/IP Protocol Suite

Protocols are basically a way to ensure that hosts are able to communicate with each other successfully. Protocols working together to guarantee effective communication are grouped into what is known as a *protocol suite* or *protocol stack*. A *protocol suite*, also called a *protocol family*, is the collection of protocols from many layers that forms the base of a useful network. The *TCP/IP protocol suite* [Kozierok05] or *DARPA Internet Protocol suite* [RFC1180], figure 2.7, is an example.

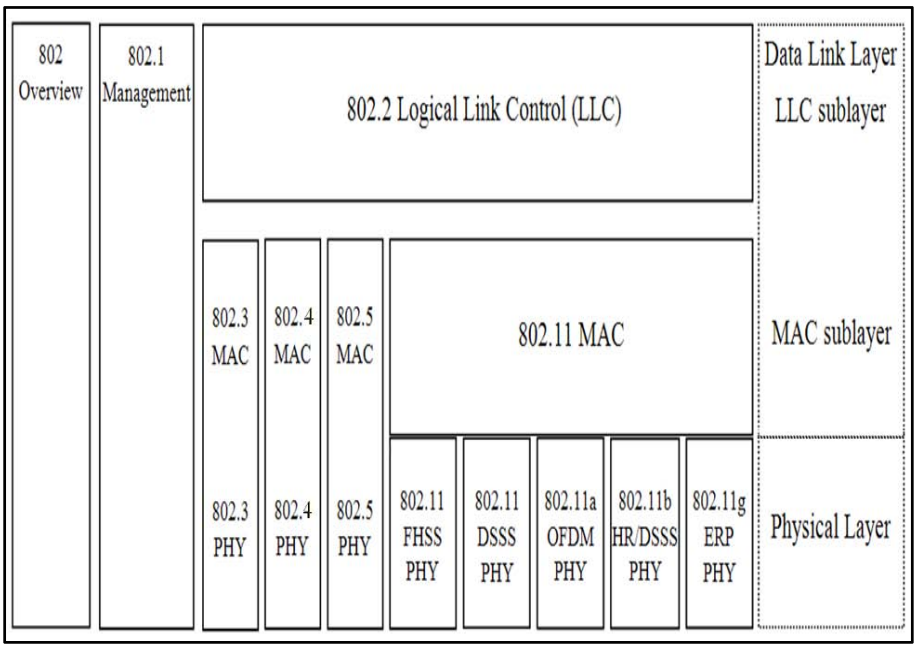


Figure 2.8, IEEE norms for LAN networks

The *Media Access Control MAC* [IEEE802] is a LAN sub-layer of the Data Link layer specified in the OSI model (layer 2), figure 2.8. It offers hardware addressing and channel access control mechanisms that facilitate for different networks hosts to communicate on a shared medium. Different MAC protocols are used for different shared networks. Common MAC layer standards are the *Carrier Sense Multiple Access/Collision Detection CSMA/CD architecture* [IEEE02] used in Ethernet (Token Ring uses token passing method, FDDI uses a dual-attached, counter-rotating token ring topology). *Medium Access Controller* implements the MAC protocols.

The common MAC Layer standard for wireless networks is *IEEE 802.11 standard* [IEEE07] which offers many functions that support the operation of 802.11-based

WLANs. The *Carrier Sense Multiple Access/Collision Avoidance CSMA/CA* [Brenner97] is MAC layer standard used by the 802.11 family. The MAC layer maintains and manages communications between wireless workstations by coordinating access to the shared channel and using protocols that improve communications over the wireless medium.

Since network architecture is based on layering, adjacent layers in the layer stack communicate vertically between them. This mechanism is called an *interface*. An *interface between layers* is the process by which data is passed from layer n+1 to layer n or conversely [Kozierok05]. In other words, a higher layer is allowed to use the services of the lower layers without necessitating knowledge of the implementation of these layers. This interface is the only way for layer n+1 to access the lower layer *n* and is called the *layer n protocol service access point n-SAP*.

Moreover, one of the results of encapsulation is the need for an interface. This interface separates the outer view from the inner one. It encapsulates activities and hides a lot of the underlying implementation details. The outer view provides information about the functionality of representation. The inner view reveals the implementation details. The interface decreases the dependencies between representations. Implementations can be changed only if the outwardly visible functionality is preserved.

The term interface is also used in other computer and networking domains, since its meaning refers to connecting several things together. For example, user interface handles the interactions between the program and the user. The component interfaces [Puder06] declare the services that a component offers and the parameters to be specified. They are used as an access point to the component functionality by other components. The mechanical and electrical interfaces such as RS232, RS449 and X.21 at the physical layer are also other examples.

Other example is *TCP/IP sockets*, [Peterson03]. Sockets are the points where a local application process attaches to the network. The interface describes operations for creating a socket, attaching it to the network, sending/receiving messages through the socket, and closing it.

2.5 THE OSI REFERENCE MODEL

The Open Systems Interconnection OSI [Zimmermann80] [ITUT94] architecture has been developed by the International Organization for Standardization (ISO) [ISO09] in 1977, to describe the operation and design of layered protocol architectures. This forms a valuable reference model and defines much of the language used in data communications. The OSI Reference Model is a hierarchical model, consisting of seven layers divided into two layer groups:

- **Upper Layers:** *Session, Presentation and Application Layers*.
  This layers' group defines how the end workstations applications can interact with the users, and communicate with each other (how applications running over the network can be implemented).
- **Lower Layers:** *Physical, Data Link, Network and Transport Layers*.
  This layers' group defines how data is transmitted from end to end (formatting and transmitting data over the network).

In the OSI model, each layer has some characteristics that define it, and also a variety of protocols associated with it.

- ***The Physical Layer PHY:*** This layer is special since it represents the hardware and circuit that drive the network. It transmits data in sequences of bits by *analogue* signaling using amplitude or frequency modulation of electrical signals on cable circuits, *light signals* on fiber optic circuits, or other *electromagnetic signals* on radio and microwave circuits. In addition, it may detect errors by monitoring the quality of the received electrical or optical signals. RS485 and 10Base-T are examples of two different physical links.
- ***The Data Link Layer DLL:*** This layer is responsible for:
  a. Addressing since it provides access to media using the hardware address.
  b. Detecting and dealing with errors but it does not make any correction.
     The proposed protocol by ISO is HDLC (High Level Link Control).

- ***Network Layer:*** This layer defines how interconnected networks function. It is responsible also for providing logical addressing and packets routing (which routers to use to determine a path). The network layer performs the interconnection implementation of several nodes in a network. An example protocol of this level is X25 level 3.
- ***Transport Layer:*** This layer is responsible for establishing connection between two hosts. Protocols in this layer are *connection-oriented protocols for reliable delivery* the TP4 protocol is comparable to Transmission Control Protocol TCP. In the connection-oriented mode, the transport layer is responsible for acknowledgments and retransmissions.
- ***Session Layer:*** This layer maintains the separation of data for different applications. Its protocols concern the establishment of sessions between two or more users or distributed components.
- ***Presentation Layer:*** This layer deals with special processes that must be done to data during the whole connection such as the encryption of data and data compression. It performs the representations of higher level objects.
- ***Application Layer:*** This layer represents the user interface. At this layer users can use programs. As examples, two well-known protocols of application layer are FTAM for file transfer protocol or X400 for electronic mail. In this layer protocols are made to satisfy the communication needs of a specific application such as the availability of resources for the intended communication.

2.6 NETWORKS SIZES AND TYPES

Computer networks [Mir07] [Stallings07] have grown rapidly. Networking is used in every aspect of life. In the 1970, the Internet was a research project. Today, the Internet has grown enormously and many users have high speed Internet access through cable modems, ADSL, or wireless technologies. In order to communicate between hosts in a network, a transmitter device must interface with the transmission system. However, there must be some forms of synchronization between transmitter and receiver to conform to the requirements of the transmission system.

2.6.1 WIRED LOCAL AREA NETWORKS LAN

Local Area Network LAN [Tanenbaum03] is a network of small and medium size that covers a building or a company (within a limited geographical area up to a few kilometers), developed in the early 1980s. LANs are high-speed, low-error networks, except when message traffic is very high. Token ring, Ethernet, and FDDI are between the most popular LAN technologies. Ethernet is the dominant technology for wired LANs. It offered originally a bandwidth of 10 Mbps and after extended to 100 Mbps and 10 Gbps. *Ethernet IEEE 802.3, Token Bus MAP IEEE 802.4* and *Token Ring IEEE 802.5* are examples of wired LANs. LANs have three characteristics:

1- **Transmission Technology:** twisted copper wire, coaxial cable, or fiber optics.
2- **Size**: LANs are limited in size. However, this eases the management of networks.
3- **Topology**: LANs use different topologies. *Bus topology*, *Ring topology* and *Star topology*, figure 2.9, are the most known and used topologies. Transmitter and receiver communicate over a shared medium. A transmission from any host is broadcast to and received by all other hosts.

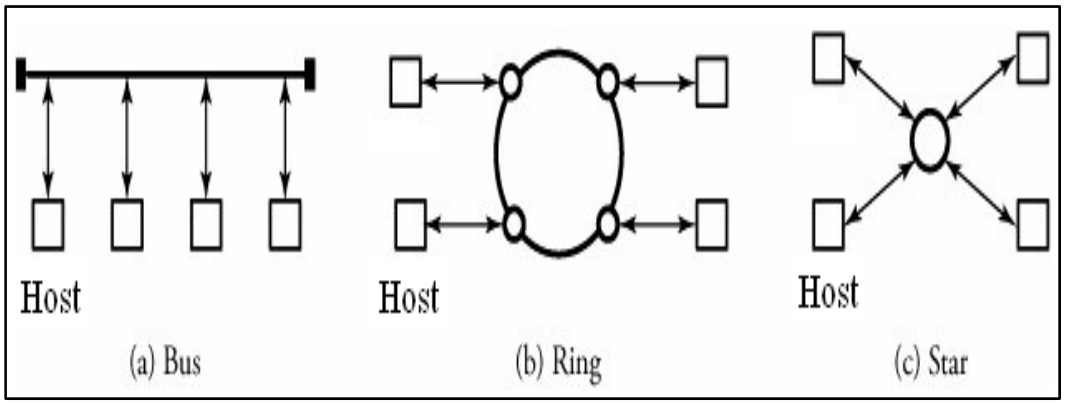


Figure 2.9, LAN Topologies

LANs performance is suitable for the implementation of distributed systems and applications. More recently, examples of switched LANs have occurred. Switched-mode high-speed Ethernet has been developed to overcome the bandwidth and latency guarantees in original Ethernet. *ATM LANs* use simply an ATM network in a local area, in addition to *fiber channel*.

2.6.2 WIRELESS NETWORKS WLAN

Wireless Local Area Networks WLANs [Sankar05] are local area networks that connect computers and devices to each other and also to the wired network. Connection in WLAN is without wires, using radio frequencies or light where signals propagate through space. This gives freedom in movement and the facility to extend applications to different parts of a building, city, or anywhere in the space. However, the limits of most wireless technologies usually make WLANs able to connect devices that are very close to each other (within a few hundred of meters at most).

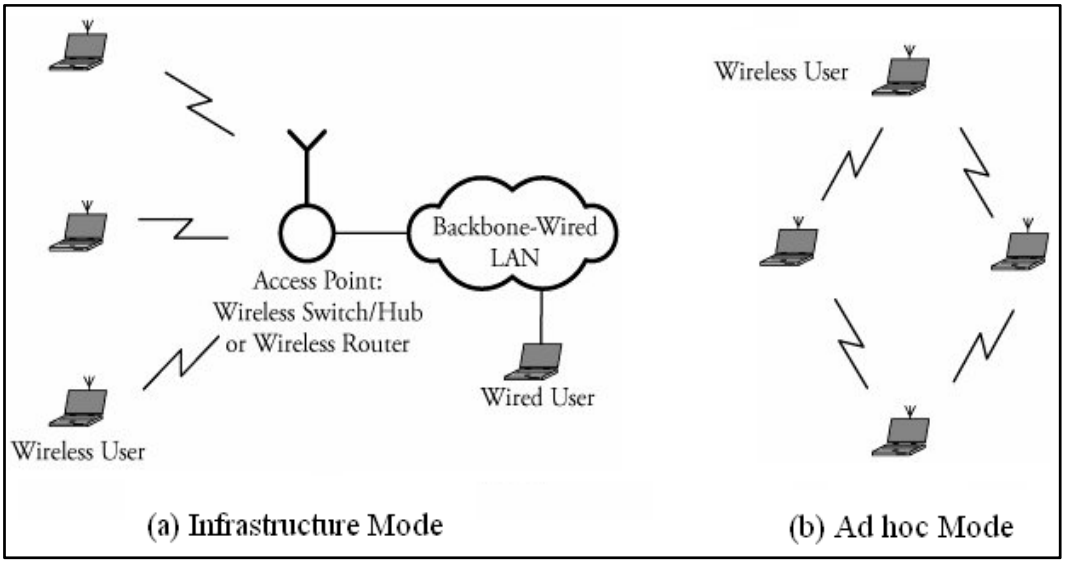


Figure 2.10, Wireless Network Modes

The basic topologies in wireless LANs is shown in figure 2.10. In infrastructure mode, access points, wireless switch/hub, are used as an interface between the wired and the wireless LANs. In this mode, all wireless users transmit to an access point to communicate with other users on the wireless or wired LAN. In Ad hoc mode, each user in the wireless network communicates directly with others users, without a backbone network. WLAN performance enables higher-end applications to run easily. With data rates of up to 54 Mbps, a WLAN can satisfy any office or home network application. *IEEE 802.11* is a WLAN standard which most systems implement and which is becoming very widespread.

2.6.3 WIDE AREA NETWORKS WAN

Wide Area Networks WANs [Tanenbaum03] are used to connect devices or networks over a large geographical area. WANs carry messages at lower speeds between nodes. The communication medium is a set of interconnected switching nodes/circuits linking a set of switching elements which form what is called a *subnet*. Switching elements or *routers*, connecting two or more transmission lines, are specialized computers that manage communication and route messages or packets to their destination. Routers are not concerned with the content of the data. Transmission lines are the channels that carry information (bits) from one workstation to another. Transmission lines can be made of copper wire, optical fiber, or radio links. In general, the WAN contains numerous transmission lines, each one connecting a pair of *equipments*. However, it may happen that the network shared resources are saturated which is called *data congestion*. To overcome this problem, several algorithms for congestion avoidance exist and are implemented at the transmitters and receivers level.

The largest existing network is the Internet. Internet covers the entire planet. It owns its name from the fact that it was the result of connecting multiple heterogeneous networks. It is a network of networks. There is a hierarchy in the Internet infrastructure. At the highest level, the backbones which are composed of fast routers and ccommunication lines at high data rates. At the medium level, the regional and national networks. Finally, at the lowest level, the LAN. WANs may be implemented using one of these technologies: *circuit switching, packet switching, frame relay* and *ATM networks (Asynchronous Transfer Mode)* which have assumed major roles.

3. COMMUNICATION PROTOCOLS MODELING METHODS

In the previous we have showed the different communication architectures models. As we have mentioned before, in this thesis we are interested in modeling *both* communication protocols and distributed services. Modeling communication protocols and distributed systems is not new. Several formalisms, simulators and modeling tools exist. However, one of the main challenges in modeling nowadays is establishing a precise relationship within a wide range of available modeling formalisms and comparing their verification capabilities and descriptive power. Thus, we propose to *unify* the modeling of both, protocols and distributed systems, in one formalism. In this way, we eliminate the need to any transformation from one formalism to another and so facilitate the modeling process. Since time is an important factor in communication protocols, any formalism capable to model them in any level must include a concept of time.

A *formal model* [Geoffrion89] [Broy07] is a mathematical and abstract representation. It is always close to the real system since it reflects a certain view of it. The Formal modeling consists of introducing system requirements (cost, security, manufacturing facilities, maintenance, evaluation, reliability and availability [Barger03]) into a small fragment. This introduction must be inside the chosen mathematical framework for the modeling process. The main purpose of a formal modeling is to clarify largely inexplicit information. During the construction of a formal model, many ambiguities must be removed. This consists in general of taking a decision or making a choice. A good model is initially a model that one can understand easily and can be explained simply. The procedures of verification must be simple and convincing.

The basic steps used for building a model are the same in all the modeling methods. The details vary a little from one method to another. However, the understanding of the common steps, combined with the typical statements needed for the analysis, provides a framework in which the results from almost any method can be interpreted and understood. Three basic steps are used iteratively until an appropriate model for the system is developed:

1- **Model Selection:** In this step, schemes of the system, system knowledge and assumptions about the system are used to determine the form of the model to be fit to that system.
2- **Model Fitting:** An appropriate model-fitting method is used to estimate the unknown parameters in the model. Then, the model is carefully evaluated to see if the underlying assumptions of the analysis appear possible.
3- **Model Validation:** In this step, if the model validation identifies problems, the modeling process is repeated using information from the model validation step to fit an improved model.

Two additional steps can be added to the basic sequence between model selection and model-fitting: the *experimental design* and the *data collection.*

3.1 UNIFIED MODELING LANGUAGE UML

Unified Modeling Language UML [UML09] is a graphical notation proposed by the *Object Management Group OMG* [OMG09], designed to represent, specify, build and document software systems. Its two main objectives are:

- The object oriented modeling,
- The use of abstract language comprehensible by man and machine interpretable.

UML allows building several models of a system, each emphasizing different aspects: functional, static and dynamic organization. UML [Rumbaugh99] is considered as the successor of three modeling languages: *Booch*, *Object-Oriented Software Engineering OOSE* and *Object Modeling Technique OMT*.

As a descriptive mechanism, UML makes distinction between the model and the diagram concepts. A model contains all the system elements and the diagram displays some types of a model elements. UML does not impose any design methodology, i.e. UML does not impose a particular way for the use of diagrams it offers, except the use of syntax rules defined in its specification.

UML1.x, 2003, is based on a single meta-model. While *UML2.0* [Haugen04], 2005, has introduced new structures such as *component*, *provided and required*

interfaces, port, connector, protocols description machine and composite structure. It provides a component model that allows the definition of components and systems architecture to be developed. UML2.0 described 13 types of diagrams which allow capturing different aspects and views of the system, figure 2.11.

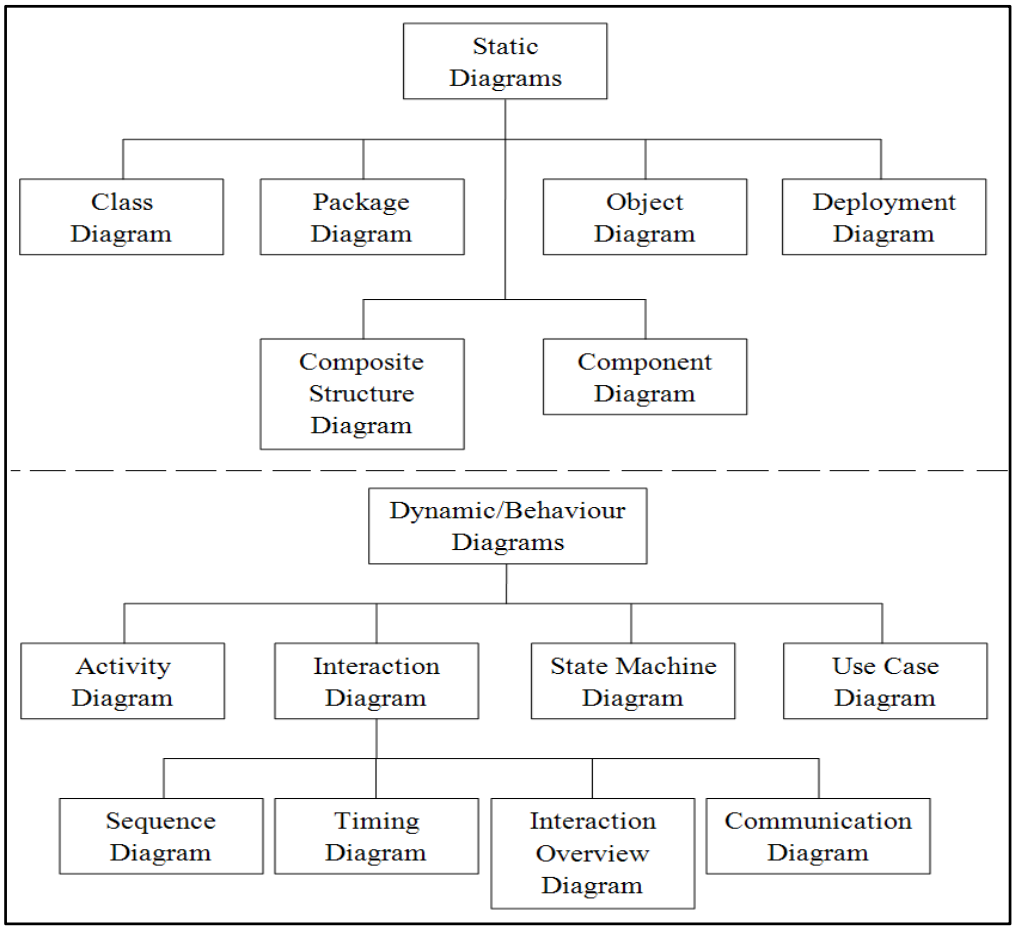
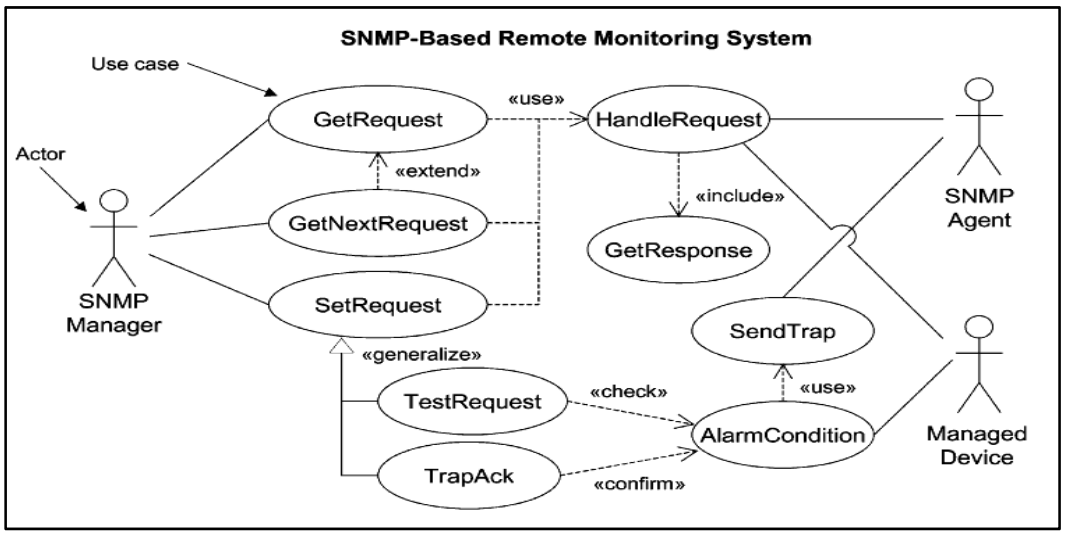


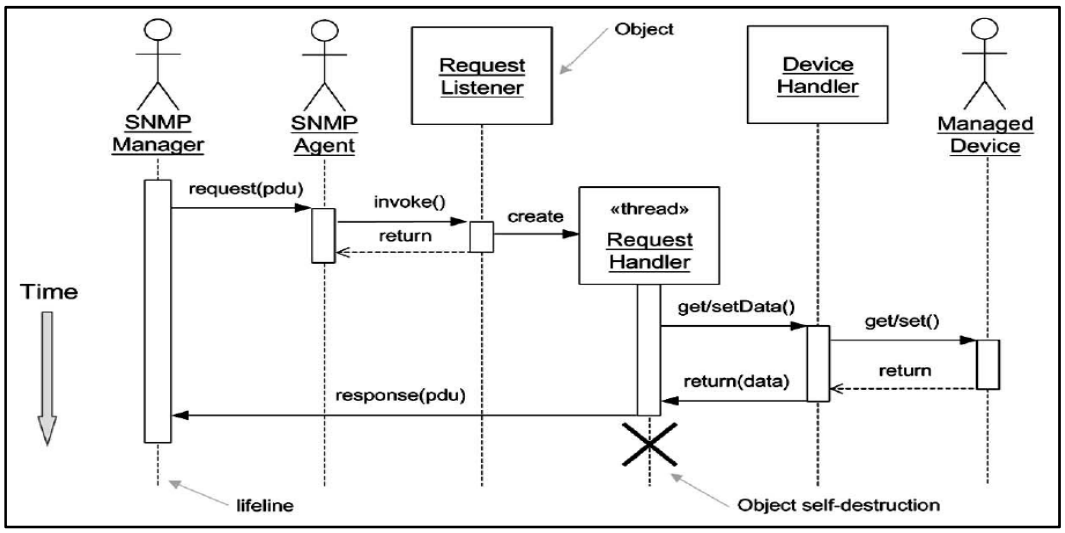
Figure 2.11, UML 2.0 Diagrams

One of the benefits obtained in using UML as an *architecture description language ADL*, especially UML2.0, is allowing users who have little knowledge in the field of formal specifications to realize and manipulate architectural description. UML is not directly associated to a method, it is only a specification language. The process that implements the toolbox UML is intentionally not specified. UML is one of the most popular methodologies. The representation of process as a flow of activities, the graphical syntax, and the simple notations are the key factors of UML, figure 2.12(a) and (b).

In the figure 2.12(a), the three actors represent users and other external systems that interact with the described system, while the 9 use cases represent the scenarios of the system (request, ack, trap and condition). In figure 2.12(b) shows the sequence diagram of the request scenario, which includes the 5 types of requests (*GetRequest*, *GetNextRequest*, *SetRequest*, *TestRequest*, and *TrapAck*) described in the use-case diagram in Fig. 2.12(a) [Lee04].



(a) Use-case Diagram



(b) Sequence Diagram

Figure 2.12, SNMP Protocol in UML [Lee04]

However, UML has also weakness points:

1- UML is based on informal semantics, language elements are not defined on the basis of mathematics. So, it is hard to verify the consistency semantics of a UML diagram mathematically, which is not the case of Petri nets [Bernardi02].

2- UML provides 13 types of diagrams. However, if one uses two different types of diagrams to describe the same system in two different views, it is difficult to formally verify the consistency of the global model (for example UML dynamic diagrams [Engels02]). Hence, at the implementation level, it is essential to overcome this weakness.

3- UML is not well suited for analyzing and modeling (temporal) behaviour in concurrent and distributed systems as Petri nets (highly used aspect) [Kristensen04] [Wu07]. In most of the recent research, UML is widely enhanced and supported by Petri nets to overcome different weakness aspects: the definition of an analysis model for temporal properties [Mallet06], time inscriptions in sequence diagrams [Eichner05], verifying *dynamic behavioural modeling* for real applications [Lee04].

4- "*A UML diagram, such as a class diagram, is typically not refined enough to provide all the relevant aspects of a specification"* [UML09]. Thus, the *formal* Object Constraint Language OCL is used to describe expressions (operations/actions) on UML models.

3.2 SPECIFICATION AND DESCRIPTION LANGUAGE SDL

Specification and Description Language SDL [SDL09] [IEC09] is a formal notation evolved and standardized between 1976 and 1992 by International Telecommunication Union ITU-T. Originally focused on telecommunicating systems [Latkoski07] [Kim07] [Melia06], SDL is a high-level general-purpose description language with more application field such as dynamic distributed systems [Díaz08], event-driven systems and real-time systems [Drosos01]. SDL provides *textual (Phrase Representation SDL/PR)* and *graphical (Graphic Representation SDL/GR)* formats.

Based on *Finite State Machines FSM* [Lai02], SDL consists of a set of concurrent processes, extended with variables and data space, communicating by exchanging control signals or structured messages instead of shared memory. The system architecture is drafted by deploying building blocks and channels connecting them, figure 2.13. Many versions of SDL were released. *SDL-2000* is the latest released version completely based on object-orientation. This version is accompanied by an SDL-UML-Profile. *JADE* (Java) [JADE99] and *Cinderella SDL* [Cinderella07] are existing tools supporting SDL.

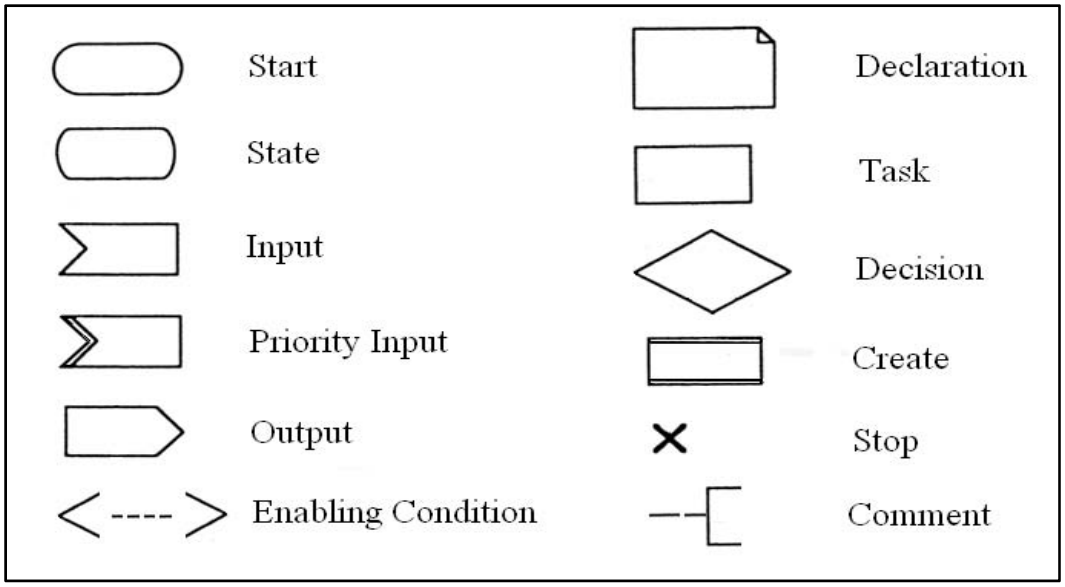


Figure 2.13, Important SDL Symbols

Figure 2.14 shows the process in the Go-Back-N protocol which is nearly optimal for channels characterized by small propagation delays, and thus is widely used for control in many classical computer-communication networks.

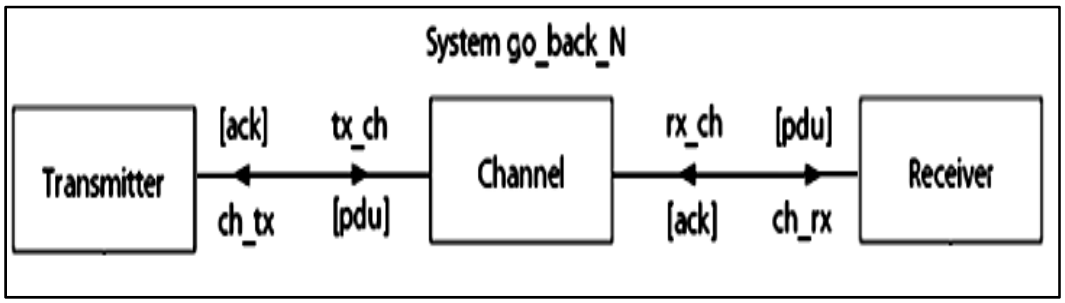


Figure 2.14, Go-Back-N Protocol in SDL

In the Go-Back-N protocol, the sender continuously transmits packets until the number of the sent packets "*win*" equals to the transmission window size *W*. Each packet contains a unique identifier, "*ns*". It then starts a timer *T*. The receiver returns a positive acknowledgment with the identifier of the last correctly received packet "*nr*". However, if the receiver does not send an acknowledgment or the acknowledgment is lost, before the timer end, it repeats the transmission of the packets (ns=nr).

In figure 2.15, branch in the middle, the transmitter verifies if the number of sent packets is less than the transmission window (condition: win<W). If not, it sends a packet (output: pdu(ns)). It increments the value of sent packets "win" and the sequence number "ns" by one (N is the total number of packets to be sent). If the value of "win" equals to W, it starts the timer T (branch to the right). Otherwise, it continues transmitting.

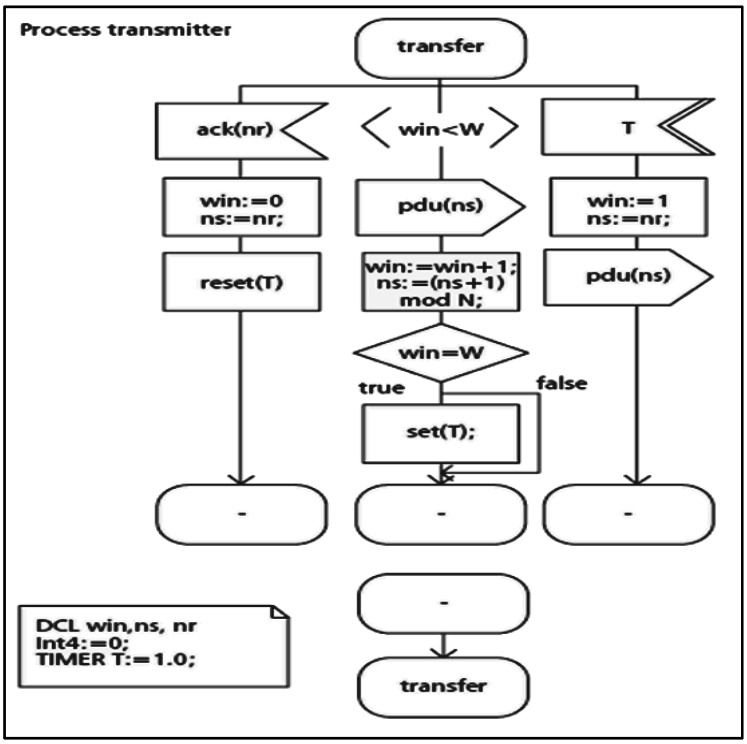


Figure 2.15, Transmitter side in Go-back-N in SDL

Once *win* equals W, the timer T (branch to the right) is set to a certain value (in the example one unit of time) with priority input. During this time, the transmitter waits for an acknowledgement from the receiver (branch to the left). If the timer time is out

before the reception of an acknowledgement, “ns” is reset to nr (last correctly packet) and w to 1, restarting the transmission process (output: pdu(ns)). However, if an acknowledgment is received (branch to the left), *win* is reset to 0, *ns* is set to *nr* and the timer T is reset to 0.

Figure 2.16 (a) shows the channel side which represents the medium to exchange the messages and acknowledgments between transmitter and receiver. Figure 2.16 (b) shows the receiver side. The receiver starts receiving data. If the sequence number of the received packet equals to the expected one (decision: ns=nr?), then it returns an acknowledgment of the last correctly received packet (nr=nr+1). Otherwise, it returns a negative acknowledgement with the current value of nr, and discards the erroneous packet.

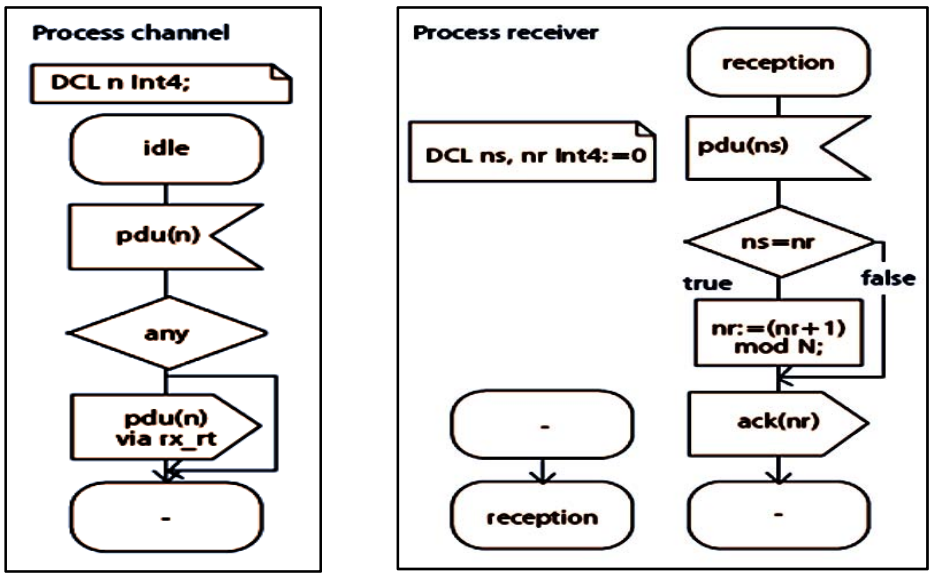


Figure 2.16, (a) Channel side (b) Receiver Side

SDL is a user-friendly language and a model-oriented specification. It has a high modeling capacity. However, SDL has some disadvantages [Werner06]:

1- Most of the existing tools are commercial.
2- In SDL, there is no verification calculus (no proofs).
3- Temporal interval expression is not allowed in SDL, (necessary at the communication level, mainly for the clocks asynchrony).

4- Recent studies in the modeling and robustness analysis of multi-hop Internet signalling protocols have shown that SDL is not well suited to create certain network topologies [Werner05] [Melia06].
5- Although the graphical form of SDL is good for showing flow of control, it is completely inadequate for showing the flow of data.

To overcome some problems in SDL, Petri nets are used as the formal model underlying an SDL model [Godary04] [Capellmann99], in addition to the translation of SDL specifications into different types of Petri nets [Nepomniaschy08] [Kryvyy08] [Aalto03].

3.3 TIMED AUTOMATA

Timed (finite) Automata TA [Alur92] [Bengtsson04] is a formalism for modeling and verification of real time systems. The *timed automata* consist of a set of states, transitions between them and a set of *stopwatches* or *clocks* to measure the time. An *automaton* is a simple state machine or labeled transition system extended with real-valued variables. A timed automaton accepts a timed word (infinite sequence in which a real-valued time of occurrence is associated each variable). The variables model the logical clocks in the system. The transitions of the automaton put certain constraints on the clock values: a transition may be taken only if the current values of the clocks satisfy the associated constraints, or *the condition on the clock called guard.*

Definition: A timed automaton is a tuple $\boldsymbol{A = (\Sigma, S, X, T, I, F, V)}$, where:

- $\boldsymbol{\Sigma}$ is a finite alphabet of actions,
- **S** is a finite set of states (locations, or nodes),
- **X** is a finite set of clocks,
- $\mathbf{T \subseteq S \times [C(X) \times \Sigma \times 2^X] \times S}$ is a finite set of transitions (or edges), written as $s \xrightarrow{g,a,C} s'$ or $s \xrightarrow{g,a,C:=0} s'$, and **C(X)** is a set of clock constraints over **X**,
- $\mathbf{I \subseteq S}$ is the subset of initial states,
- $\mathbf{F \subseteq S}$ is the subset of final states.
- $\mathbf{V: S \rightarrow C(X)}$ assigns invariants to states or location.

A *path* in **A** is a finite sequence of consecutive transitions:

$$P = s_0 \xrightarrow{g_1,a_1,C_1} s_1 \ \dots \ s_{p-1} \xrightarrow{g_p,a_p,C_p} s_p$$

where $\{q_{i-1}, g_i, a_i, C_i, q_i\}$ *timed words* $\in T$ for each $1 \leq i \leq p$.

The path is said to be *accepting* if it starts in an initial state, $q_0 \in \mathbf{I}$, and ends in a final state, $q_p \in \mathbf{F}$.

When its guard *g* is true, the transition can take place and assigning new values to clocks while emitting visible action. In some situations, states have *invariants* which must remain true as the automaton is in the state. Invariants are used most often to force the execution of a transition. The time scale is supposed to be continuous: the values of the clocks are real numbers. In addition, transitions are instantaneous.

In figure 2.17, starting from state *p* where the value of the clock *x* is zero (p, [x = 0]), delay may last for any nonnegative real number *t* to reach the valuation (p, [x = t]). This is called a time *elapsing step*. Being in state *p* and $t \leq 2$, a *discrete step* can be performed by taking the transition *a*, since the guard $x \leq 2$ is satisfied, reaching to the state *q* (q, [x = t]). In the state *q* delay may be for at most *5-t* time units because of the invariant $x \leq 5$ associated with the state *q*. As soon as the value of the clock *x* is at least 3, the discrete transition *b* can be taken and return to the initial state *p* (p, [x = 0]) because the value of the clock *x* is reset to 0 by taking this transition.

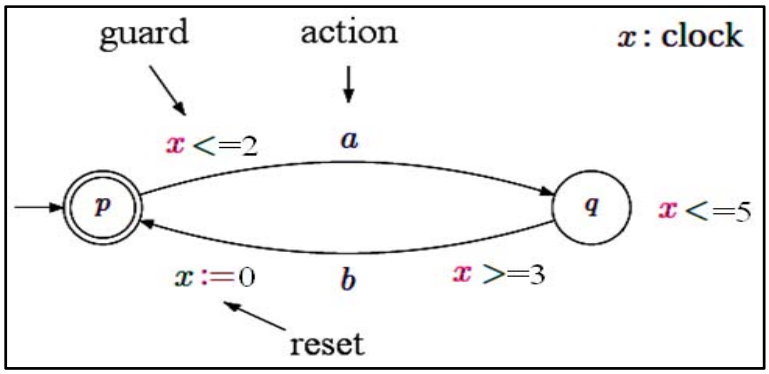


Figure 2.17, Example of Timed Automata

Timed Automata are used to model and analyze timeliness properties of embedded system architectures [Hendriks06], real-time systems [Larsen05] and communication protocols [Corin07] [Diaz06] as real-time components. *UPPAAL* [UPPAAL09] and

KRONOS [KRONOS02] [Yovine97] are two of the most well-known tools in TA. KRONOS is a tool that aims to assist the design of real-time systems and to provide a verification engine integrated into design environments. KRONOS is developed in France at *Verimag laboratory*, a joint laboratory of *UJF*, *Ensimag* and *CNRS*. It is free distributed tool used to model timed automata components and is expressed in the real-time temporal logic *TCLT*, *timed computation tree* logic. However, KRONOS is textual and does not support a graphical design interface. In addition, the work on this tool was stopped in 2002 and no new development is done since that date.

UPPAAL is developed by the Uppsala University and Aalborg University. UPPAAL is an open source toolbox for modeling, simulation, and verification of Timed Automata. It is a graphical, platform-independent interface, written in Java and extends FSMs by means of timers. Its model checker is used to verify some of the system properties. In timed automata, there are two classes of algorithms based on the notion of zones and implemented using data structures like the *Difference Bounded Matrices DBM*: the class of *forward analysis algorithms* and the class of *backward analysis algorithms*. UPPAAL implements only the forward analysis algorithms that permit the feature of bounded integer variables, for which backward analysis is inappropriate. Conversely, KRONOS implements the two kinds of algorithms.

Figure 2.18 shows the Go-back-N protocol in UPPAAL. The upper part of the figure shows the transmission side. S1 is the initial state. The transmitter sends its data over the channel *tx_ch* to state S2 (the sign "!" is used to represent the emission of data and the sign "?" represents the reception of data). It receives data channel *ch_tx* from state S3, (figure 2.14). A retransmission, timeout or acknowledgments are all checked by the guards associated to the transitions.

The lower part shows the channel and reception sides. The channel just forwards data and acknowledgments between the transmitter and the receiver. However, at the receiver side, committed states are used for S2 and S3 (the letter C inside the circle). These locations freeze time which means that delays are not allowed and the committed state must be left to the next state (a committed location if there are several ones).

Outgoing transitions of a committed state have priority and must be carried out immediately. The receiver receives data on the *ch_rx* channel and returns its acknowledgments on channel *rx_ch.* As SDL, UPPAAL is user-friendly platform. Its mean usage is for validation of hard temporal constraints.

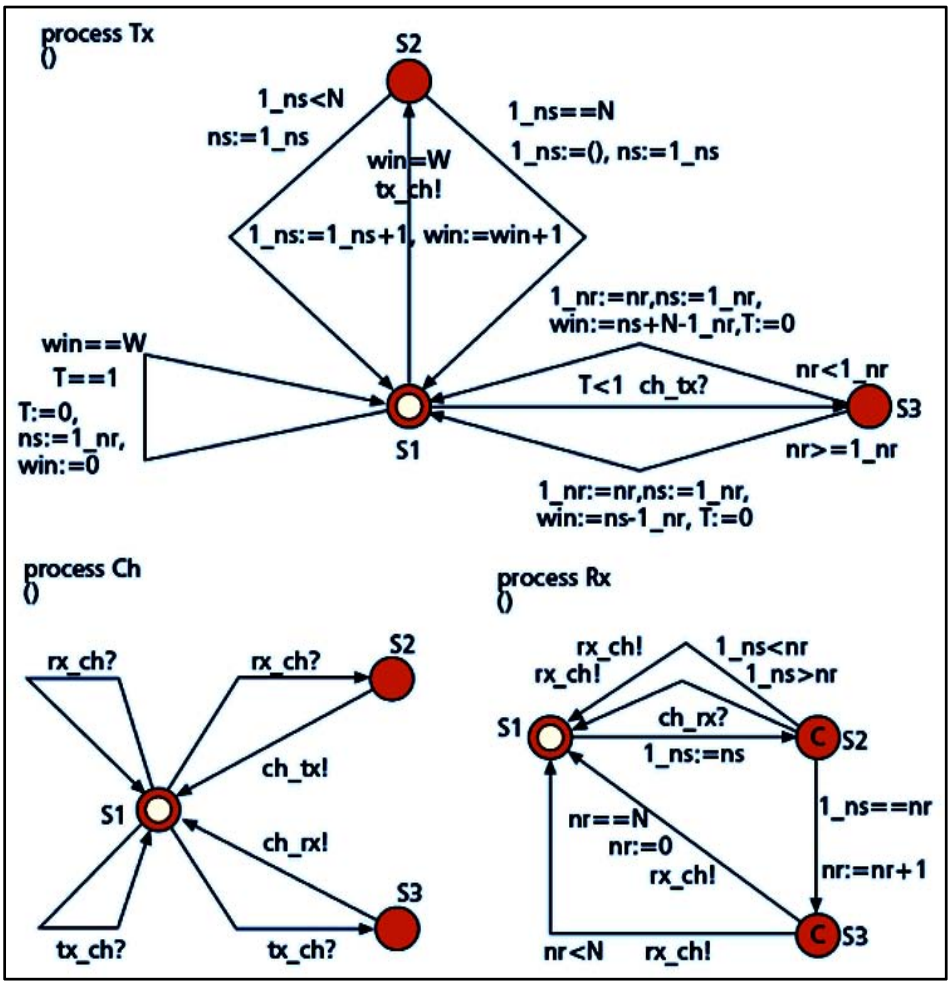


Figure 2.18, Go-Back-N Protocol in Timed Automata

However, timed automata have some weakness points:

1- "Timed automata" is not general purpose formalism (limits its application field to the hard timing constraints applications).
2- Some timed automata tools like KRONOS lack the support for high level composable graphical patterns to support systematic design of complex systems [Dong04].

3- Performance analyses are of quantitative measures (limited to the upper bounds) which make it unsuitable for systems that depend upon real-time considerations.
4- UPPAAL and KRONOS implement the forward analysis algorithms, however according to [Bouyer03] the two implementations are not correct.

3.4 SIMPLE PROMELA INTERPRETER SPIN

SPIN [Bolognesi87] is a tool for specification, simulation, and validation of communication protocols. It uses a C-like specification notation *ProMeLa* or *Process Meta Language*; it is a textual notation for *Extending Communicating Finite State Machine ECFSM*. SPIN supports efficient model-checking, validation of consistency requirements, invariant assertions, and temporal properties expressed in an ad hoc Linear Temporal Logic.

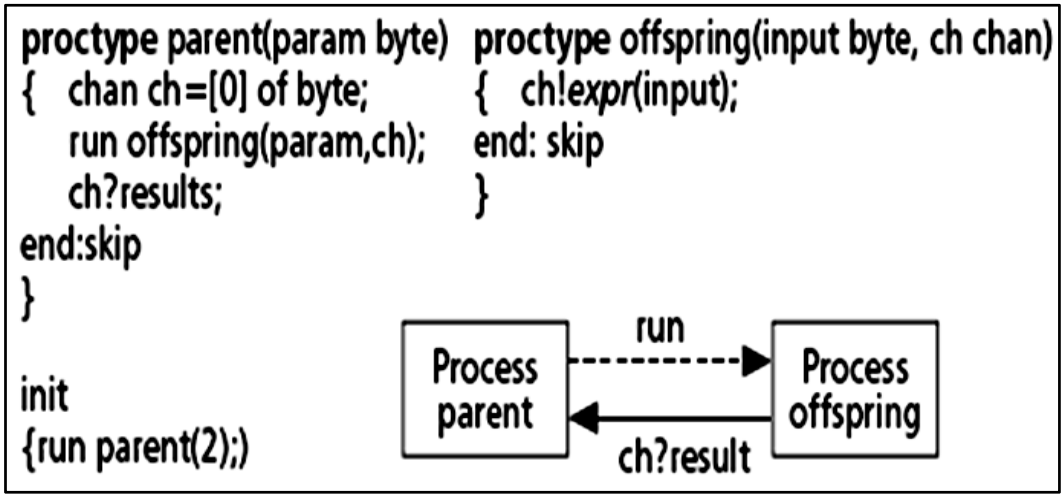


Figure 2.19, Processes Declaration in ProMeLa SPIN

Figure 2.19 shows the declaration processes and the messages exchanged between both of them (input and result values). In SPIN, system components are processes whose internal behavior is described as a set of possible transitions and processes can communicate on channels. SPIN can perform simulation and logical validation on a ProMeLa file. *Xspin* or *X-interface SPIN* is the graphical version of SPIN, generating graphical FSMs. SPIN main application domain is for validation of logical properties, with a good friendliness usage and modeling capacities.

However, SPIN has some disadvantages:

1- No hierarchical structuring facilities,
2- Worst-case complexity is exponential in number of processes,
3- Performance analyses are not supported.

3.5 PETRI NETS WITH TIME

Historically, the application of Petri nets to communication protocols [Merlin76] [Diaz87] [Billington88] dates back to the earliest attempts to use formal techniques to solve problems specific to communication protocols. One of the extensions of Petri nets used to model communication protocols and real-time systems is the *Time Petri nets TPN* [Diaz91] [Berthomieu07] [Bucci05]. TPN is a powerful formalism used to model systems where time is the main constraint. They allow the description of the *watch–dog mechanism* (restricting and monitoring the time) which is not allowed in *timed Petri nets* [Ramchandani74]. Since transitions are labeled with time intervals in TPN, this makes it possible to model unpredictable timing where the exact durations of events are not known. They can also represent the timed Petri nets, just by making the two limits of the interval equal.

Definition: A Time Petri Net is a six–tuple $\boldsymbol{N= (P, T, Pre, Post, m_0, \tau_s)}$, where:

- **P** and **T** are the sets of nodes respectively called places and transitions ($|\mathbf{P}|=m$, $|\mathbf{T}|=n$) and $P \cap T=\Phi$;
- **Pre: P×T→N and Post: T×P→N** are the weighted flow relation representing the arcs;
- $\mathbf{m_0}$**: P→N** is a mapping associating to each place $p \in \mathbf{P}$, an integer $m_0(p)$ called the initial marking of the place *p*.
- $\boldsymbol{\tau_s: T \rightarrow R^+ \times (R^+ \cup \{+\infty\})}$ is a function called *Static Interval function*, and, $\forall t \in T$, $\boldsymbol{\tau_s}(t_i) = [\alpha_t, \beta_t]$, where α_t and β_t are rationals such that:

$$0 \leq \alpha_t < \infty \text{ and } 0 \leq \beta_t \leq \infty$$

$$\alpha_t \leq \beta_t, \text{ if } \beta_t \neq \infty \text{ and } \alpha_t < \beta_t, \text{ if } \beta_t = \infty$$

Time represented into intervals (the lift bound is called *static earliest firing time* or *EFT* and the right bound is called *static latest firing time* or *LFT*) makes it easy to model events with unknown occurring time. These limits are related to the date when $\mathbf{t}_i$ was enabled for the last time. Let θ be the date when $\mathbf{t}_i$ becomes enabled; then $\mathbf{t}_i$ cannot be fired before $\theta + EFT$ and must fire no later than $\theta + LFT$ (if max is finite), except if the fire of another transition $\mathbf{t}_j$ un-enables $\mathbf{t_i}$ before it is fired. Transition firings have no durations. In addition, a transition **t** of a time Petri net is enabled at marking **M** if and only if $\mathbf{Pre(p, t) \leq M}$. If a transition $\mathbf{t_i}$ is fireable at a state defined as $\boldsymbol{S= \{M, V\}}$, where **M** is a marking and $\boldsymbol{V}$ is a dynamic firing interval set, where:

$$\boldsymbol{V \in \tau_s: T \rightarrow R^+ \times (R^+ \cup \{+\infty\})}$$

Then the new marking M' is reachable from M: $\boldsymbol{M'(p) = M(p) + C(P, t_i)}$, as usually in Petri nets.

A transition $\mathbf{t_k}$ is said to be *newly enabled* by the firing of a transition $\mathbf{t_i}$ if M – Pre(p, t) + Post(p, t) enables $\mathbf{t_k}$ and M – Pre(p, t) did not enable $\mathbf{t_k}$. If $\mathbf{t_i}$ remains enabled after its firing then $\mathbf{t_i}$ is newly enabled. The number of entries in vector **V** is given by the number of the transitions enabled by marking M.

The transition firing in a Time Petri Net has two firing semantics. The first semantics is called the *strong timing semantics STS*, which imposes that any enabled transition ***t*** must be fired at its latest firing time at most unless it is disabled by the firing of a conflicting transition at a time no greater than the latest firing time of transition *t* LFT_t. In other words, when the LFT_{ti} is reached, the transition must be fired and time cannot disable firing of the transition t_i. This requires some form of global coordination of the net.

On the contrary, when using the *weak timing semantics WTS*, the firing time of a transition is not constrained by firing conditions over other transitions. The decision about a transition firing is local and it can be fired only in its time interval. If it does not fire within its upper bound, then it cannot fire anymore. To be able to fire, it should be *reset* by being disabled once. WTS is nearer to the original untimed Petri nets. However, STS

is the most widely adopted one and Petri nets with STS are more powerful than Petri nets with WTS.

Figure 2.20 shows an example of a time Petri net. A transmitter workstation attempts to send data to a receiving workstation. It waits a backoff of 5 units of time (transition tx_ch). The vector V= {tx_ch[5,5]} since one transition is enabled. The firing of "tx_ch" puts a transition in place "Send Packet". A packet can be lost due to external noise or interference. This is represented by the transition noise. Three transitions are enabled, so, V= {Noise[0,9], ch_rx[0,1], Retransmission Timer[10,10]}. However, transitions "ch_rx" and Noise can fire before transition "Retransmission Timer" since their EFT and LFT are less than those of transition "Retransmission Timer" (0, 0<10; 1, 9<10). So, one of them will fire first. The firing of one of them disables the other one.

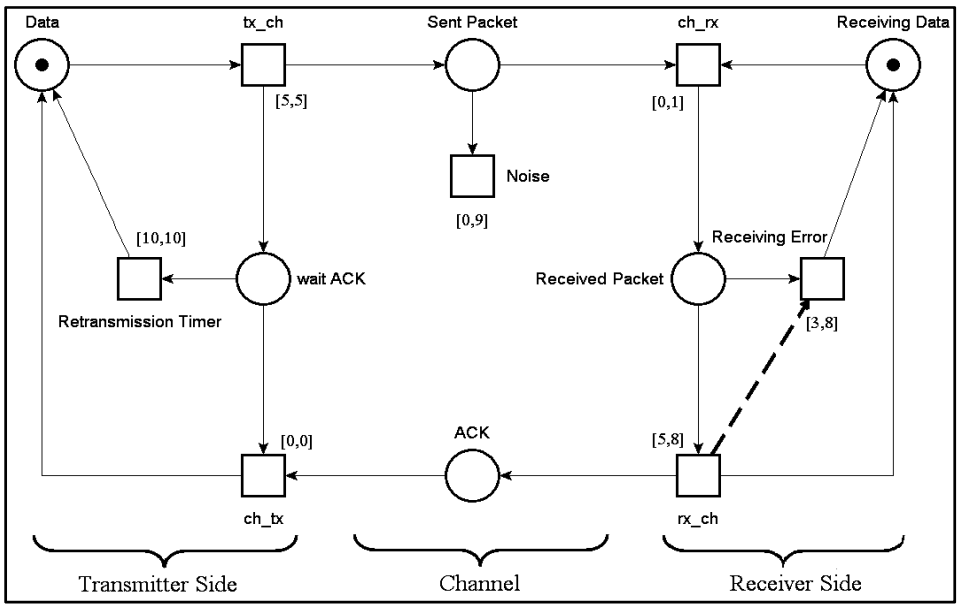


Figure 2.20, Time Petri Net Example

- If transition Noise is fired, then V= {Retransmission Timer[10,10]}. This means that the packet is lost and it will be retransmitted after 10 units of time. Hence, the local timer on this transition continues incrementing which means that if transition "Noise" is fired after 4.15 units of time, transition "Retransmission Timer" will fire after 5.85 units of time (4.15+5.85=10).

- If transition “ch_rx” is fired, a token is put in place “Received Packet” and V= {Receiving Error[3,8], rx_ch[5,8], Retransmission Timer[10-θ,10-θ]}, where θ is the time before firing transition “ch_rx” and $\theta \in [0,1]$.

Again, transitions “rx_ch” and “Receiving Error” will fire before transition “Retransmission Timer”. However, transition “Receiving Error” can fire before transition “rx_ch”, since its EFT (3) is less than “rx_ch” EFT (5). In Tina [Tina09], TPN was improved with priority selection. Transition “rx_ch” has priority over transition “Receiving Error” if it is not fired before 5 units of time, which means that it can fire in an interval of [3,5[units of time. This is useful to represent error checking, assuming that an error can be checked just after the reception of a packet.

Now, if transition “Receiving Error” is fired, the packet must be retransmitted (as in point 1 above). On the other hand, firing “rx_ch” puts a token in place “ACK” and V={ch_tx[0,0], Retransmission Timer[10-θ-λ,10-θ-λ]}, where λ is the random time before firing transition “rx_ch” and $\lambda \in [5,8]$. The maximum value of θ+λ is 9 (8+1), this means that to fire transition “Retransmission Timer”, the local clock needs one more unit of time. However, transition “ch_tx” is immediate, so it must fire first according to strong timing semantics STS.

To conclude, the use of Petri nets for modeling communication protocols has several advantages:

1- Petri nets are simple formal definition, supporting powerful analysis capabilities, with powerful semantics that support execution and simulation,
2- Their visually graphical interface makes Petri nets easy to realize and manipulate for a wide range of users,
3- Petri nets with time allow to describe protocols in a very adequate way, particularly by supporting directly the fundamental notions of concurrency and synchronization [JWang07] which are inherent to communication protocols,
4- Time Petri nets are able to analyze temporal and stochastic properties [Masri08] [Moraes06],

5- Their analysis techniques prove many important structural properties [Billington04],
6- There exist many verification techniques by simulation and free computer-aided tools based on Petri nets with time [Renew08] [CPN07],
7- Successful use in many different application areas: complex, manufacturing and distributed systems [Moreno08] [Glabbeek08], real-time systems [Chen08] [Liu08], safety critical systems [Xu07], multimedia systems [ZWang07] …

However, there are some weakness points:

- The inflexibility to cope with system changes.
- Petri nets lack computations expressiveness with tokens considered as black dots. This limits its modeling power since no value is transferred by communications. For example, in the figure 2.20 only one packet can be sent (one token in place "Data") since it is not possible to send an acknowledgment without identifying the token that models the sent packet.
- The locality of the Petri net semantics, firing transition only influences its local neighbours, reactions to external events and exceptions are difficult to model.
- One of the basic problems in Petri Nets is that modeling a system needs many places and transitions. In other words, as the system size and complexity grow, the state-space of the Petri net grows exponentially, which could become too difficult to manage both graphically and analytically.

Many extensions of Petri nets were suggested to overcome these problems. High-level Petri nets with object concepts offers practical support for Petri nets through the provision of flexible and powerful structuring primitives. Combining Petri nets with *object-oriented* concepts is an excellent solution to profit from the strengths of both approaches and eliminates nearly all these problems.

4. CONCLUSION

A distributed system is a collection of computers connected to a network and data applications distributed on these computers. In the first part of this chapter, we have focused on the networking concepts and techniques that are the basis for distributed systems. Layered protocols are the base for communications in distributed systems. Two main reference models for computer networks are the OSI model and the TCP/IP model.

In the next part we have focused on the methods used in modeling communication systems and distributed services. UML, Timed Automata and Petri nets are of the most used formalisms in this domain. Formal methods are a big supporting means for increasing correctness and reliability in system design and implementation. However, none of these formalisms were used to model both the communication protocols and the distributed services.

Nevertheless, Petri nets provide a clear formalism for concurrency and distribution. Combining both Petri nets and Object-orientation concepts together allows benefiting from the strengths of both approaches. High-Level Petri nets HLPN, described in the following chapter, are considered as a special kind of Petri nets which allow the representation and manipulation of an object, a *token*. This powerful formalism allows modeling complex systems like the distributed systems and communication protocols.

Since the manufacturing systems use the industrial local networks, the upcoming chapters will mainly focus on modeling the *LAN MAC sublayer protocols*, more particularly *Ethernet* and *Wireless LAN*. To build the model's components, we will first analyse these protocols. Later, we will evaluate the complete model to verify some of its properties.

Chapter 3

MODELING COMMUNICATION PROTOCOLS WITH COMPONENT-BASED APPROACH

1. INTRODUCTION

Component-based engineering [Brown96] [Brereton00] has a huge importance for rigorous system design methodologies. It is based on the statement which is common to all engineering disciplines: *complex systems can be obtained by assembling components* [Gössler03], ideally *commercial-off-the-shelf (COTS)* [Carney00] [Weyuker98].

Reusability and extensibility [Meyer97] are key factors that contribute to this success and importance. Component-based development aims at decreasing development time and costs by creating applications from *reusable, easily connectible* and *exchangeable building blocks*. Components are usually characterized by abstractions that ignore implementation details (the internal functionality) but describe the way these components can be connected together to complete the model through their connecting interfaces. In the context of utilisation, the main factor in building such components is the ability to make them reusable in different future models by a well-define internal functionality and connecting interfaces.

From my opinion, one of the most efficient high-level formalisms used for modeling both network protocols and distributed services are the *High-Level Petri Nets HLPN* [Lakos95] [Kohler05]. Petri nets have become very popular for representing distributed computer systems because of their capability of clearly describing concurrency, conflicts, and synchronization of processes, and because they present a simple and elegant formalism for their description. *They offer an excellent framework that can be used for the specification and performance evaluations of distributed systems and protocols* [Masri08a]. Thus, Petri nets can be used to unify the modeling of service and protocols in a single formalism.

HLPN fulfil all the requirements of communication systems distributed services. The graphical representation of the net gives the user an easier understanding of the modeled process. In addition, they support simulation which is one of the main methods used to verify properties of communication protocols. Simulation can be interactive, i.e. with user intervention, or automatic. Formal modeling usually gives a set of expressions

for its components describing the behaviour of the investigated system. It also provides precise descriptions of the component function and structure.

In the first chapter, we have presented in Section 3.6 the modeling of the control functions of a manufacturing system. In this part we have emphasized the use of client/server approach to allow the distribution of resulting code on a distributed architecture of the control. In the following sections, we aim to model network protocols to be able to analyze their impact on the productivity of the manufacturing system with a selected architecture of the control. Since our control function is modeled with a High-Level Petri Nets, we select the same formalism to be able to obtain a global model that integrates both services (here the modules of the control function) and the model of the network protocols.

In this chapter, we will introduce the different component-based modeling properties, particularly with the formal methods techniques in which we are interested. In the second part, we will describe the used modeling formalism *High-Level Petri Nets*. In the third part, we will specify the methodology used for building the patterns and components for communication networks. Since we are interested more particularly in industrial networks, our modeling of network protocols will focus on the medium access control. In consequence, we will illustrate our methodology on two illustrative examples: Ethernet and DCF 802.11.

2. COMPONENT-BASED MODELING PROPERTIES

Component-based methodology is a promising technique and implies method developed from the *object-oriented design*. It uses *hierarchical* and *modular* concepts to design and analyze systems. To reduce the cost of system developments, it makes use of *independent, interactive, and reusable components*. In component-based engineering research literature, several approaches [Gössler07] [Bastide04] [Heck03] [Cheesman01] [Penix98] have focused on the aspects of the *development* of components. However, *reusing available, ready-to-use components* decreases *time-to-market* for new systems and applications [Seyler02]. This may be done by selecting the appropriate components from the available components based on the needs and then assembling them to build a new component system-model.

Different methods of component specification software exist; from the *Interface Description Language IDL* (*Object Management Groups' CORBA* [OMG09a], *java-based components such as JavaBeans* [Matena03] and *Microsoft's .Net* [Thai02]), passing by the *design-by-contract based* (*UML* [UML09] [DSouza99]), to *formal methods* (*Petri Nets* [Chachkov01]). In spite of their widely difference in the details, they have a common concept: *a component is a black box that is accessed through exposed interfaces*. In this section we will precise the main features that a component-based method must verify.

2.1 GENERICITY

Genericity is used in component-based engineering to raise time-to-market, productivity, and quality in systems development. The term generic component refers to a component that implements a process or part of a process-set and that is adaptable to accommodate different needs. Genericity of a component is based on its independence compared to its type. This is an important concept for high-level methods because it can increase the level of abstraction of these methods.

A generic component can be seen as a parameterable element. Parameters should be specified and a specific version of the component (an instance) will be created and

used. Another advantage is that a generic component can be represented as a *generic factory* that will create as many components as necessary for the application. Thus, the main objective of genericity is to integrate the component-based approaches with the technical approaches.

2.2 MODULARITY

Modular models are easier to design compared to similar complex models. "*Modularity is having a complex system composed from smaller subsystems that can be managed independently yet function together as a whole*" [Langlois02]. In another definition, "*modularity is used to describe the use of common units to create product variants*" [Huang98]. The objective of modularity is the ability to identify homogeneous, compatible, and independent entities to satisfy the needs of a system or an application. In many domains, modularity is essential to manage the design and the production of complex technology.

Modular design aims at organizing complex systems as a set of components or modules. These components can be developed independently and then joined together. They can be separable or inseparable units [Foster95]. The decomposition of a system model into smaller modules has the following advantages:

1- A modular model can be very near to the real system, since it reflects the hierarchical structure inherent to the system.
2- It is possible to concentrate on each component as a small problem.
3- Components which are too complex can lose some of their details and their interactions can be confused. A component can be divided into smaller components until each module is of manageable size.
4- Modular model allows testing each component separately.
5- Implementation changes and corrections on simple components can be done easily.
6- Documentation in modular structure becomes also easier.

2.3 REUSABILITY

The implication of *reusability is that the available components must give enough information to ease the assembly of components into a new system* [Geisterfer06]. The information must include dependency and configuration information. To make well decisions about selecting and reusing components, the following information is required:

1- *Operational specification*: the semantic interaction of the component,
2- *Operation context*: where and how the component will be used,
3- *Non-functional properties*: describe the properties such as performance, security and reliability,
4- *Required interfaces and resources*: the functionality and resources needed by the specified component to execute its provided functionality.

Since all real systems are made of components, *component-based* systems are comprised of multiple components [Brown96] that:

- are ready "*off-the-shelf*," either from a *commercial source (COTS)* or reused from another system;
- have significant combined functionality and complexity,
- are self-contained and can be executed independently,
- will be used "*as is*" without modification,
- must be combined with other components to get the desired functionality.

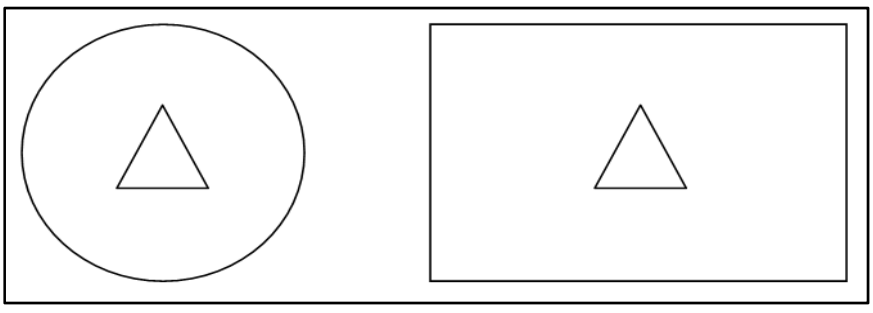


Figure 3.1, Basic Module Reusability

All these benefits and more lead us to use the component-based approach to model the distribution of manufacturing systems and the underlying protocols. The reuse of

components is very important in the modeling level since most of the system parts and machines are the same. In addition, protocols share many properties. With the reuse of already modeled components, the time and modeling-cost are reduced. As we can see in figure 3.1, models are sharing some properties (the triangle). Once this part is modeled, it can be reused in any model that has a need to such component.

2.4 COMPONENTS ABSTRACTION

The modeled components are seen as *black box* where the internal functionality is hidden, while the interfaces represent the service that can be offered by this component. Every component or module is characterized by its internal hidden *behaviour*. Its interfaces are chosen to reveal as little as possible about its inner implementation.

Components abstraction is useful for reducing the designing complexity by decomposing a problem into connected components. Abstraction (or specification) describes the functional behavior of the components [Sametinger97], i.e. components are considered to be specific to an application. Abstraction focuses on the important characteristics of component upon the designer viewpoint. This definition supports the abstraction of data, hiding internal function, reusability and self-contained component behaviour descriptions [Edwards97].

Thus, during the design of components we must focus on well-defining the service offered by the component by its interfaces and the parameters that can be adapted to the application requirements, rather than spending the time on describing its internal behaviour. This can be achieved by giving appropriate names to the interfaces and parameters and documenting these interfaces and parameters.

3. CHOOSING THE METHOD: HIGH-LEVEL PETRI NETS

As we saw in the last sections, component-based approach can deeply support the reusability of ready-to-use components and the distribution in modeling manufacturing systems. The use of component-based technology in real-time systems progress has been the main object of recent research such as in the *UML* and *high-level Petri Nets*.

Moreover, systems are required to be more flexible and dynamic and to be designed in less time. This point together with the fact that the design of complex system often represents a considerable investment forces for an increased reuse of flexible and configurable design elements. This is one of the central motivations for the use of component-based techniques. Components contain attributes representing the component state, and operations that access and manipulate the state. The inner behaviour of the component is kept hidden from the outside world and only the interfaces are visible.

In this work we aim to model and evaluate the distributed services and protocols with the same modeling formalism. Time-dependence and data identification characterize these systems. One of the choices was UML because it is well-defined, powerful formalism and widely used in this domain [Bigand04]. However, UML lacks one important feature to achieve the desired needs which is the lack of a formal semantics and hence it is not possible to directly verify *timing requirements* which are *necessary in communication systems*.

Another choice was to use Petri nets, (since pervious work of OSSc team concerning the manufacturing systems' services is already made in ordinary and colored Petri nets). Petri nets are also a powerful formalism widely valid for the modeling of concurrent and distributed systems. Since many extensions and tools are available, mainly for time, identification of tokens and stochastic issues which are very important in the communication protocols and services. So, the integration of the previous work with our work for modeling the whole systems elements (services and protocols) will be easier (no need to make a transformation).

3.1 SELECTION CRITERIA

Petri nets are mathematical formalism intended to be used for modeling, simulation and analysis of different kinds of systems. In computer science Petri nets are used for modeling a great number of either hardware and software systems, or various applications in computer networks. A special advantage of Petri nets is their graphical notation, which reduces the time to learn Petri nets and simplifies their use. However, this formalism has different extensions and tools. As we saw in chapter 2, communication protocols have some characteristics and requirements. Thus, the tool selection criteria were depending on these requirements:

1- *Time*: Communication protocols are real-time demanding applications. Transmitting and receiving data, accessing to the channel, backoff and other needs depend on time. Time Petri nets allow this feature.
2- *Headers and Data fields*: Data packets have many fields which may be modeled as tuples. This feature is supported in high-level Petri nets.
3- *Probabilistic and Stochastic Properties*: Messages exchanged over the network may be lost or perturbed. Bit rate error is a function of noise which means that it is not fixed. The representation of such feature can be made by stochastic functions. Stochastic Petri nets have it in its definition.
4- *Sent and Received Packets*: Messages exchange over the network needs the identification of packets. Colored Petri nets are made for this need.

3.2 PROPERTIES OF OUR HIGH-LEVEL PETRI NETS

3.2.1 DEFINITION

In this subsection we will give a brief definition on the desired high-level Petri nets. This definition is not far from the definition of colored Petri nets [Jensen91]. However, we add to this definition time notation which is not identified.

Definition: A High-Level Petri Net is a tuple $N = (P, T, A, m_0, \Sigma, \Lambda, G, E, D)$ where:

- **Σ** is a *finite set of non-empty color sets*.
- **Λ** is a *color function*, $\Lambda: P \rightarrow \Sigma$
- **G** is a *guard function*, G: T → Boolean expression, where:
  $\forall \mathbf{t} \in T: [Type(G(t)) = B_{expr} \wedge Type(Var(G(t))) \subseteq \Sigma]$,
  where *Type* is the color type of the guard function, B_{expr} is a Boolean function and Var is the variables of the guard function.
- **E** is an *arc expression function*, E: A→E(a), where:
  $\forall a \in A: [Type(E(a)) = \Lambda(p(a)) \wedge Type(Var(E(a))) \subseteq \Sigma]$, p(a) is the place of arc *a*.
- **D** is a *delay function*, D: E → TS, where TS is a delay/time stamp associated to the arc inscription with the annotation symbol "@".

The arc expression function can contain any sign and/or mathematical or logical functions of the programming language used with Petri nets, such as Java. The delay function can be associated to both output arcs (from places to transitions) and input arcs (from transitions to places). The implementation of this definition will be given by different examples in the following subsections.

3.2.2 INSCRIPTIONS, GUARDS AND TUPLES

Arcs are the connectors between places and transitions. Arcs can have arc inscriptions. When a transition fires, its arc expressions are evaluated and tokens are moved according to the result. Arc inscriptions can be simple, tuples or even mathematical operators. They can be also variables or constants. However, inscriptions do not have the same meaning on both input arc and output arc.

Figure 3.2 shows different arc inscriptions. In figure 3.2 (a), the arc inscription contains mathematical operation. The resulting of firing T1 is a token with value 8. While in (b) T2 can fire only if place P4 contains a token with value 5. Also tokens can be numbers or strings as in place P5. The resulting of firing T2 is a token with value "hello" in place P6. However in (c), T3 can fire with any value in place P7, but the resulting of this firing is a token with the value 5 put in place P9. Other Java signs can be also used like the "!" sign which means the not-equality, while "|" is an "*or*" sign.

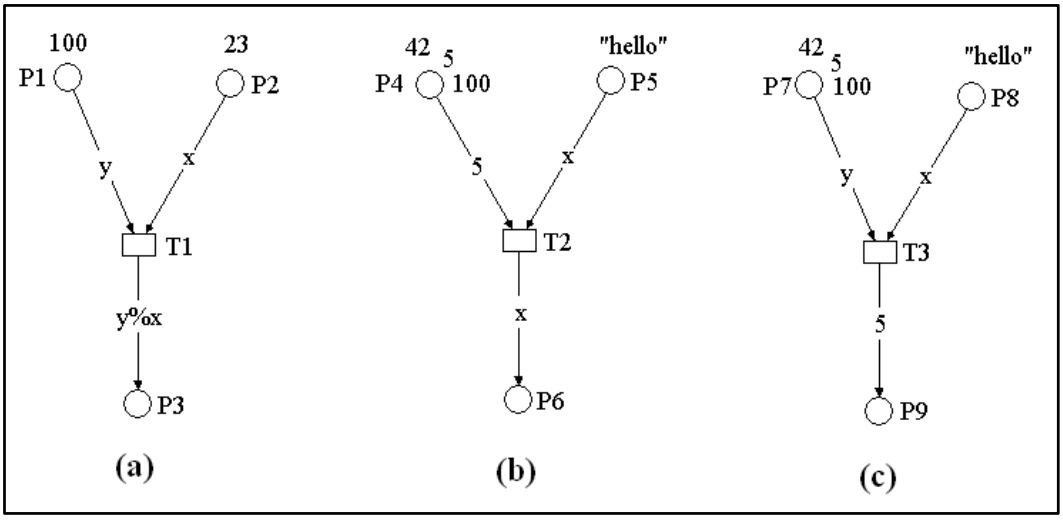


Figure 3.2, Arc Inscriptions

Not like the arc inscriptions, guard inscriptions are expressions that are prefixed with the reserved word guard associated to the transitions. A transition may only fire if all of its guard inscriptions evaluate to true. Guards are the conditions that must be satisfied to fire transitions. They can be used as if statements.

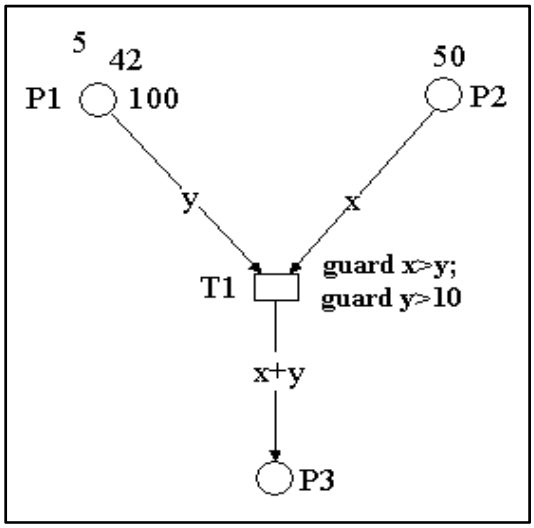


Figure 3.3, Guard inscription

Figure 3.3 shows an example of the guard inscription. To fire T1 both conditions must be true: *y* greater than 10 and *x* greater than *y*. The tokens with value 42 and 100 in place P1 satisfy the second condition. However, the value of token *x* is 50. So, only the

token with value 42 can be used to satisfy the first condition. The resulting of firing T1 is a token with value 50 + 42 = 92 put in place P3. Guards are also useful to identify the tokens.

A tuple is denoted by a comma-separated list of expressions that is enclosed in square brackets. [1,“abc”,1.4] denotes a 3-tuple which has as its components the integer 1, the string “*abc*”, and the double precision float 1.4. Tuples are useful for storing a whole group of related values inside a single token and hence in a single place. A tuple, [[1,2],[3,4,5]], is a 2-tuple that has a 2-tuple as its first component and a 3-tuple as its second component. This might be useful if the components are hierarchically structured.

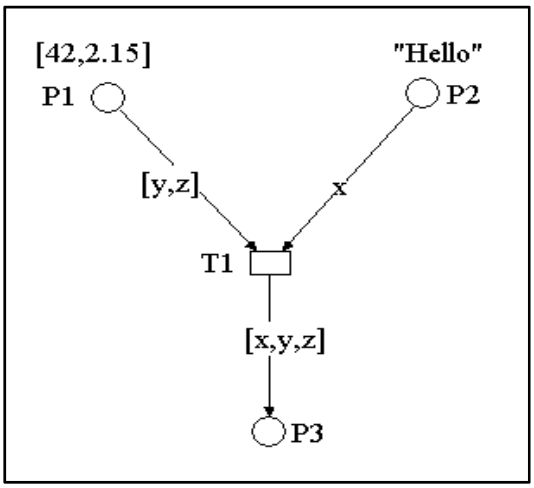


Figure 3.4, Tuples

Arc inscription can modify tokens and the structure of a tuple. Figure 3.4 shows an example of tuples. Tuples can be used to represent *Protocol Data Unit PDU* in communication protocols.

3.2.3 STOCHASTIC AND PROBABILITY FUNCTION

A stochastic process or random process is a collection of random variables. In Stochastic Petri nets, the function **Γ** is a set of firing rates that maps the set transitions T into a probability density function *f*. The entry $\delta_i \in \Gamma$ is an exponential distributed random variable, whose *f* is a negative exponential, associated with transition $\mathbf{t}_i$.
F is a function that represents a probability distribution in terms of integrals such as:

$$p(a \leq x \leq b) = \int_a^b f(x)\, dx = 1, \text{ for any two numbers } a \text{ and } b \qquad (3.1)$$

The probabilistic measure **P** is a function transforming the random variables to the interval [0, 1] such that:

- P(x) is non-negative for all real *x*.
- The sum of P(x) over all the possible values that *x* can have is 1:

$$\sum_i P_i = 1 \qquad (3.2)$$

Where ***i*** represents all the possible values of *x* and P_i is the probability at x_i, consequence P(x) ∈ [0, 1].

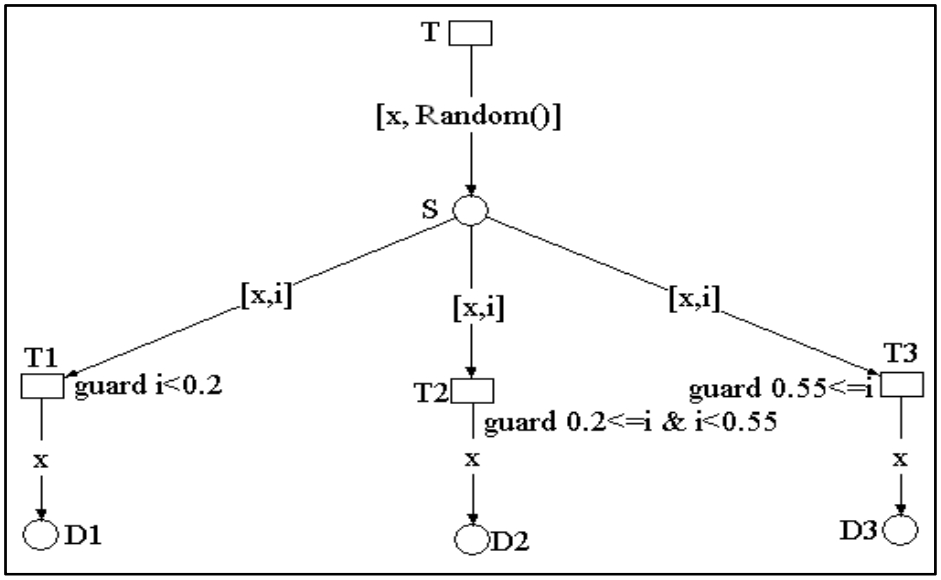


Figure 3.5, Probabilistic Process with the *Random()* Function

Figure 3.5 shows a possible probabilistic process with the *Random()* function. The function represents the generating of a random variable that can be easily implanted in *Java* to create any type of random variable (class ***RandomVariable()*** in the package ***java.lang.object*** or any Java random function).

In the figure, the firing of transition T generates a 2-tuple token [x, i] in place S. In this token, *x* models the type of the object and *i* is, for example, the type of the measure of a characteristic of this object. Let us assume that *i* is a random variable in the interval [0, 1]. Because of the guards on the transitions, the token in place S can only enable one of

the three transitions T1, T2 and T3. The value of *i* equals to randomly generated value of this function. The firing of the enabled transition depends on the value of *i*:

- If the value of *i* is less than 0.2, T1 can be fired and hence a token of value *x* is put in place D1.
- If the value of *i* is greater than or equals to 0.2 and less than 0.55, then T2 is the enabled and the fired transition.
- However, if the value of *i* is greater than or equal to 0.55 then T3 can be fired and hence the token *x* is put in place D3.

3.2.4 TOKEN IDENTIFICATION

Workstations exchanging messages put the source and destination addresses in the header of the message. The workstation which has the destination address can pick up the message. Token identification is very important to model this process. High-level Petri nets allow the identification of tokens.

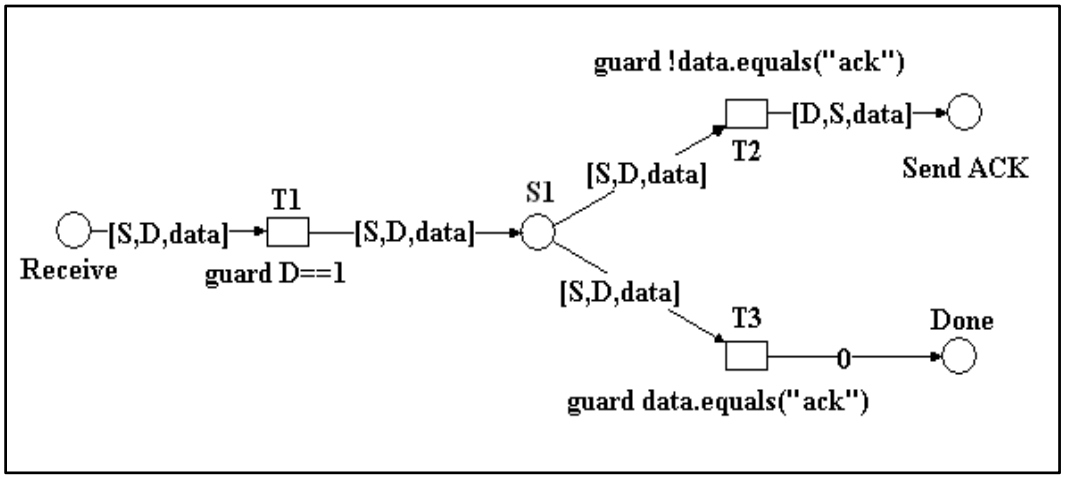


Figure 3.6, Token Identification

The guard inscription on the transitions can be used to identity a token depending on its fields (in the input places). Consider the example in figure 3.6, a workstation, sensing the channel for reception, can only pick up the packet if its destination address is "1" (assumed to be its address). Other verifications can be done such as the identification of the contents of the packet if it is an acknowledgement packet or data packet.

3.2.5 TIMING

Time notation is added to PN formalism to model time dependencies. A time stamp is attached to each token. Delays are associated to arc inscriptions in order to control the time stamps of token and the firing times of transitions. To add a delay to an arc, the symbol "@" and an expression representing the number of time units are added to the arc inscription. For example, the inscription x@5 indicates that the token must stay or will be available after 5 units of time.

Delays at the input arcs (from places to transitions) mean that a token must remain available for given time before firing the transition (timed transition). However, delays at the output arcs specify that a token is only available after some time (immediate transitions). Delays can be created by a random number generator or depend on the result of an action. Delays may depend also on the token values to delay the input token itself, which means that [x, t]@t is legal.

Timing adds another firing rule. Immediate transitions have more priority over timed transitions. To construct the vector of enabled transitions *V(t)* in the net, *local remaining time of the tokens LRT in the input places* with respect to the *arrival time* of token in the place is used. The time inscription at the output arcs of a place (input arc for a transition) only indicates the time a token must stay in that place before firing the transition. The time for each place is computed locally for each arc-transition delay, but to compute the *effective remaining time* α_t for each enabled transition, the *maximum local remaining time* for each input place of that transition is used:

$$\alpha_t = \max\{LRT(p_i),\ \forall\, p_i\ \in {}^\circ t\} \tag{3.3}$$

Where $^\circ t$ is the set of input places of transition *t*, with LRT = 0 for the input arcs with no time inscription.

Once V(t) is constructed, the transition with the *minimum remaining time* is first fired:

$$F_{Fired}(t) = \min\{\alpha_{t_i}, \text{ such that } t_i \in V(t)\} \tag{3.4}$$

Where $t_i \in V(t)$ is an enabled transition in the vector V(t).

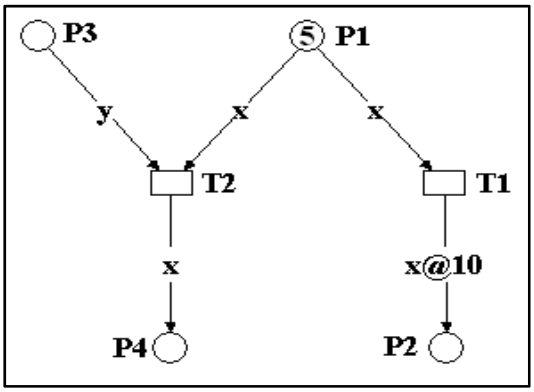


Figure 3.7, Time Inscription on the Output Arcs

In figure 3.7, transitions T1 and T2 are immediate. The inscription on the output arc between T1 and P2 indicates that the token is put (available) in place P2 after 10 units of time, but it is immediately removed from place P1. So, the arrival of a token to place P3 during the 10 units of time would not have any effect on the net since the token in place P1 has been already removed by the fire of T1. This case is similar to the firing rules found in Timed Petri nets.

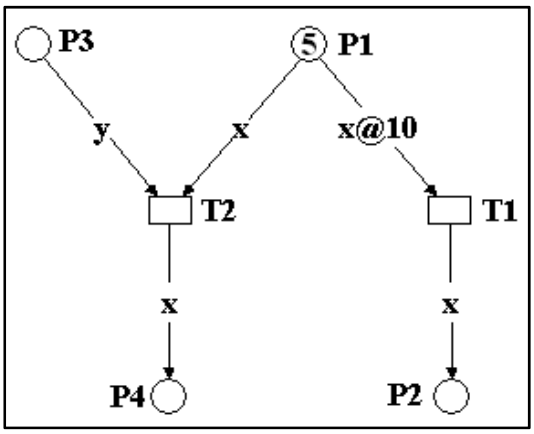


Figure 3.8, Time Inscription on the Input Arcs

In figure 3.8, transition T1 is enabled but cannot fire before 10 units of time, (tokens in place P1 must stay available for 10 units of time before firing T1). After firing T1, a token with value 5 is put in place P2. However, T2 is an immediate transition since time delays are not added to any of its input arcs. So, if a token is put in place P3 during the 10 units of time, it is fired immediately and transition T1 is no longer enabled. In this

special case, the firing of transition T1 is as the firing rule of a T-time Petri net with interval [10, 10].

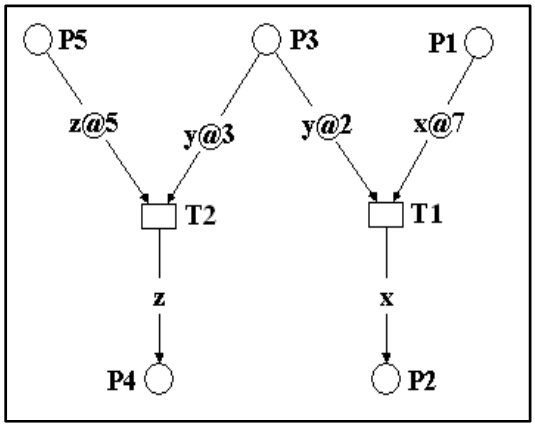


Figure 3.9, Computing the Effective Firing Time

Figure 3.9 shows the general case to find the fired transition. In the figure, the firing of T1 or T2 depends on the token arrival time in each input place (for T1: places P1 and P3; for T2: places P3 and P5). If we assume that one token is put in each place at the same time, both T1 and T2 are enabled. To compute the effective firing time, we get:

$\alpha_{T1} = \max \{2, 7\} = 7$, $\alpha_{T2} = \max \{3, 5\} = 5$

$F_{Fired}(t) = \min \{\alpha_{T1} = 7, \alpha_{T2} = 5\} = \alpha_{T2}$

So, T2 is the fired transition.

However, if we assume that a token is put in place P1 3 units of time before the arrival of the other tokens, we get:

$\alpha_{T1} = \max \{2, 4\} = 4$, $\alpha_{T2} = \max \{3, 5\} = 5$

$F_{Fired}(t) = \min \{\alpha_{T1} = 4, \alpha_{T2} = 5\} = \alpha_{T1}$

Here, we used the local remaining time for place P1 (7 – 3 = 4 units of time). Thus, the fired transition is T1 since the token in place P1 has already resided part of its staying time (time inscription on the arc).

4. BUILDING COMPONENTS TO MODEL LAN MAC PROTOCOLS

As we have announced in the previous sections, the manufacturing systems uses the local industrial network to exchange messages between the workstations. So, before starting the construction of modeling components, we will analyze the data link layer protocols that we are interested in this thesis. These analyses will help us identify the basic behaviours common to different protocols. Each basic behaviour will be modeled in order to create a basic component. In our method, basic components are the initial brick of the library that will serve to model all the complete behaviour of the different protocols concerned in this study.

4.1 ANALYZING THE DATA LINK LAYER PROTOCOLS

The *Data Link Layer DLL* is the second layer in the OSI model (figure 2.2). In communication networks errors can occur. This means that some mechanisms can be used to detect and correct them. The data link layer makes errors processing by adding a checksum field at the end of each frame (FCS field, figure 2.4). This filed will help the receiver in comparing the recomputed checksum of the received packet with that in the FCS field to accept or drop the frame.

The data link layer is often split in two sublayers: the *logical link control LLC* and the *media access Control MAC* (figure 3.10). This division is based on the architecture used in the *IEEE 802 Project*, which aims to facilitate the interoperability of different LAN technologies. LLC sublayer provides the functions needed to establish and control the logical links between local devices on a network. It offers services to the network layer and conceals the other details of the data link layer. This allows different technologies to work similarly with the higher layers. Most LAN technologies use the IEEE 802.2 LLC protocol.

The MAC sublayer provides hardware addressing and channel access control mechanisms that enable hosts to communicate. Different MAC protocols are used. The

data link layer defines most LAN and wireless LAN technologies: *IEEE 802.2 LLC, IEEE 802.3 Ethernet, IEEE 802.5 Token Ring, FDDI and CDDI, IEEE 802.11 WLAN.*

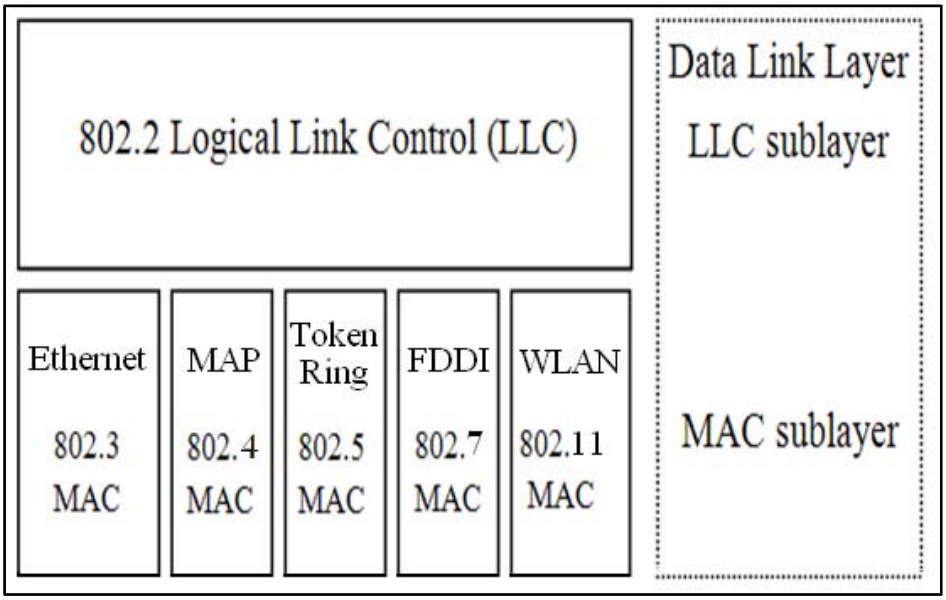


Figure 3.10, IEEE MAC Sublayer

4.1.1 CHANNEL CHECK

A workstation attempting to send data must at first check if the channel is free or not. FDDI and Token Ring technologies use *token-passing protocol*. Data is sent from one workstation to another around a physical ring. A workstation can only send its data if it possesses the token (a small frame).

Ethernet uses the *CSMA/CD Protocol.* The workstation must check if the channel is free or not. If the channel is busy, the workstation defers for 9.6 μs before rechecking the channel. If the channel is free, the workstation defers for a period of 9.6 μs before it starts its transmission. The IEEE 802.11 DCF uses the CSMA/CA protocol. To use the network, a workstation must before check if the channel is free for more than a period of time called *Distributed Inter-Frame Space DIFS*, figure 3.11. If so, the workstation starts a random *backoff* before starting its transmission.

If the channel status is changed in both Ethernet and IEEE 802.11 deferring and backoff times, the workstation must restart the process of sensing the channel.

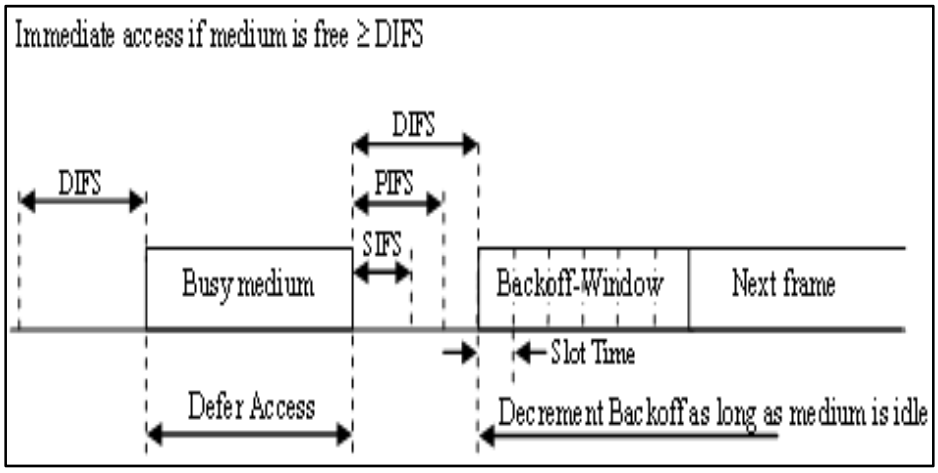


Figure 3.11, Channel Access in IEEE 802.11 DCF

4.1.2 SENDING AND RECEIVING: DATA, ACKNOWLEDGMENTS AND JAM

Workstations send and receive packets. These packets can be data packets, acknowledgment packets or JAM frame (a 32-bit frame, put in place of the correct MAC CRC). In Ethernet networks, workstations receive either data packet or a JAM after a collision. The destination workstation does not need to send an acknowledgment to the transmitter at the MAC layer. However, in wireless LANs, the destination workstation must send an acknowledgment to the transmitter after a successful reception of a packet, (figure 3.13). Otherwise, the transmitter will consider that its packet is lost or a collision is occurred, so it retransmits this packet causing an extra load on network worthlessly.

In FDDI and Token Ring, the receiver must send an acknowledgment to the transmitter because it cannot free up the network by itself. When the transmitter receives the acknowledgment, it releases the free token back on the network.

On the other hand, to send data, workstations need only to put the destination address in the packet. Since the medium is shared in most LAN technologies, all the workstations will see the packet. However, only the workstation that has the destination address reads the packet and the others will either forward it, or drop it.

4.1.3 RANDOM AND BINARY EXPONENTIAL BACKOFFS

As we mentioned above, in communication networks errors can occur. This is due to many factors like the surrounding environment, noise and interference, or because of collisions. FDDI and Token Ring use the token-passing protocol. This guarantees the non-occurrence of collisions since only one workstation can send its data over the network at any particular time.

However, Ethernet and IEEE 802.11 networks use the channel check and the inter-frame space to decide the medium access. Thus, collisions may occur due to that more than one workstation transmits on the shared medium at the same time. In Ethernet, the maximum time needed to send the first bit from one end to the other end of a 10BaseT medium is 25.6 μs. During this time, another workstation(s) may attempt to send its data, as they think that the channel is free.

As a result, a JAM signal is propagated over the shared medium informing the occurrence of a collision. Each workstation concerned by a collision starts a binary expositional backoff procedure, called *BEB*, to decide when it can do a new attempt to access the medium. The BEB algorithm computes randomly a waiting delay that increases with the number of the attempts *Tn* of the workstation.

At the beginning *Tn* equals zero, (figure 3.12). Each time a collision occurs, the workstation increments *Tn* counter until it reaches 15. Before trying to transmit its data again, the workstation starts a BEB by taking a random value between 0 and 2^X and multiplies it by 51.2 μs, where:

$$X = \begin{cases} \text{Tn}, & \text{if } \ 0 < Tn \leq 10 \\ 10, & \text{if } 10 < Tn \leq 15 \end{cases} \tag{3.5}$$

This helps in decreasing the possibility for a collision occurrence. In case of no collision, the workstation continues transmitting and when it is done it leaves the channel. However, If *Tn* reaches 15, (the load on the channel is very high), then the workstation aborts its transmission and tries it again later.

In wireless LANs, after a collision, no JAM signal is sent. However, if the workstation does not receive an acknowledgment after a period of time equals to Short IFS (*SIFS*, figure 3.11), it considers that a collision has occurred and starts a backoff procedure. For each retransmission attempt, the backoff grows exponentially as:

$$ST_{backoff} = R(0,CW) * Slot\text{-}time \quad (3.6)$$

Where:

- *ST* is the backoff time.
- *CW* is the *Contention Window*.
- *R* is a random function.

In general, the initial (starting) value of CW (CW_{min}) is 16. After each unsuccessful transmission attempt, CW is doubled until a predefined maximum CW_{max} is reached (often 1024).

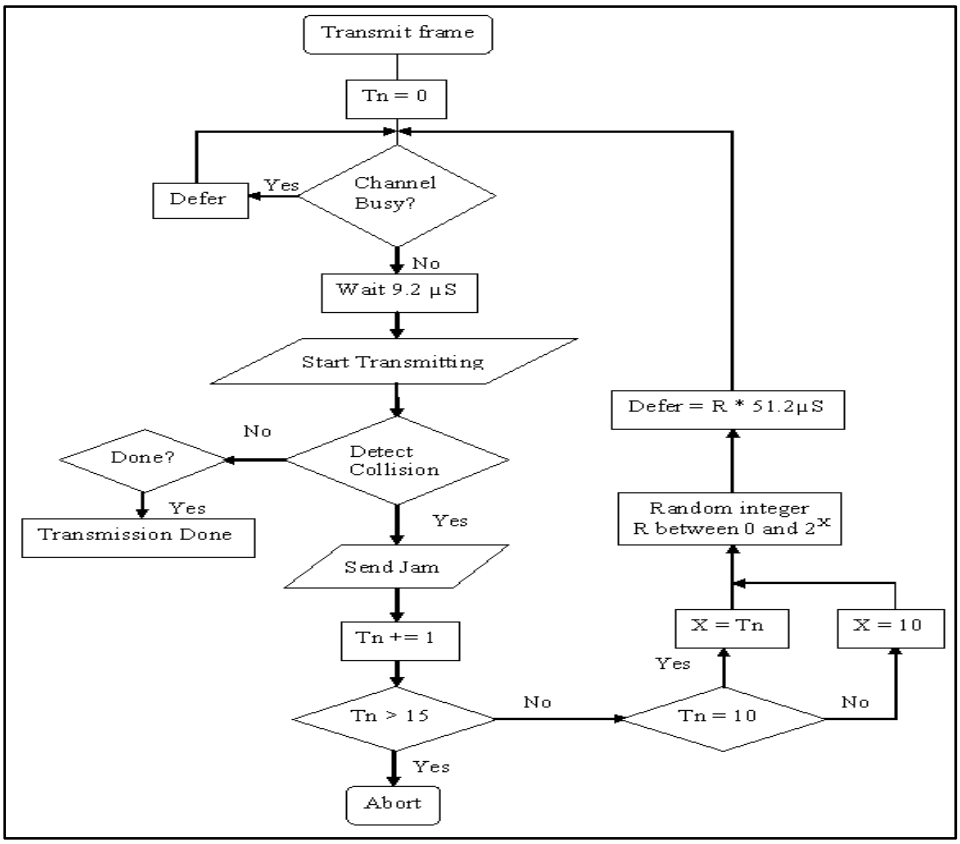


Figure 3.12, Transmission in Ethernet

There are two major differences between Ethernet and IEEE 802.11 backoff processes:

1- The wireless LAN starts a backoff procedure even at the first attempt to send its data (figure 3.13), while Ethernet does not. This is one of the mechanisms used to implement the Collision Avoidance feature of CSMA/CA.

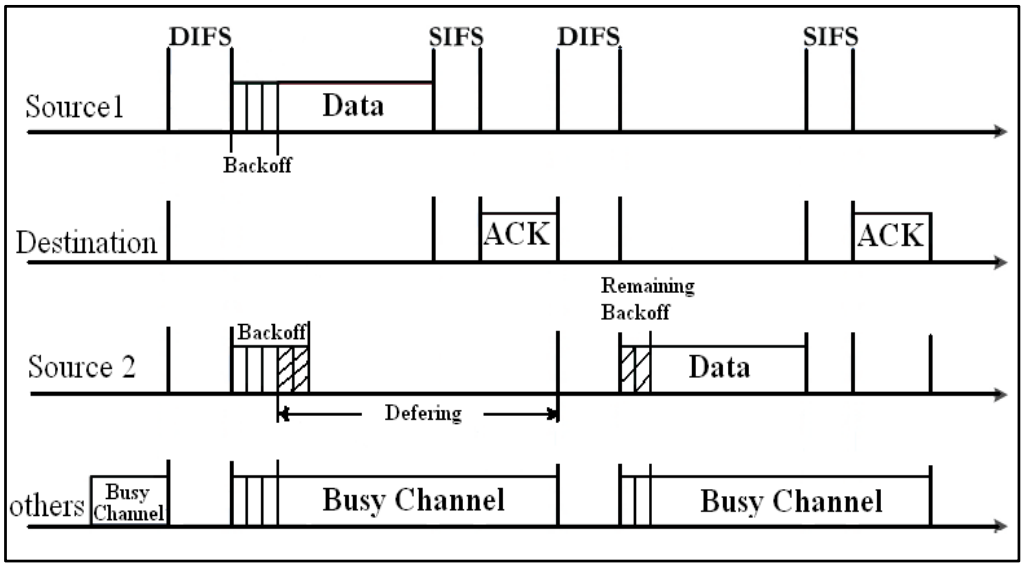


Figure 3.13, Backoff mechanism in IEEE 802.11 DCF without RTS/CTS

2- Ethernet starts its BEB algorithm after a collision (without conceding the status of the channel) and then restarts checking the channel to send its data.

While in IEEE 802.11, the workstation checks first the channel status as in subsection 4.1.1, and then it decrements its backoff by:

$$R = \begin{cases} R\text{-}1, & \text{if the channel is free during one Slot time} \\ R, & \text{if the channel becomes busy} \end{cases}$$

The design of CSMA protocol offers fair access in a shared medium. This means that all the workstations have a chance to use the network and workstations cannot capture the channel for ever.

The remaining value of R is reused after the channel status becomes free for more than a DIFS period. The workstation starts sending its data when R equals zero, figure 3.14.

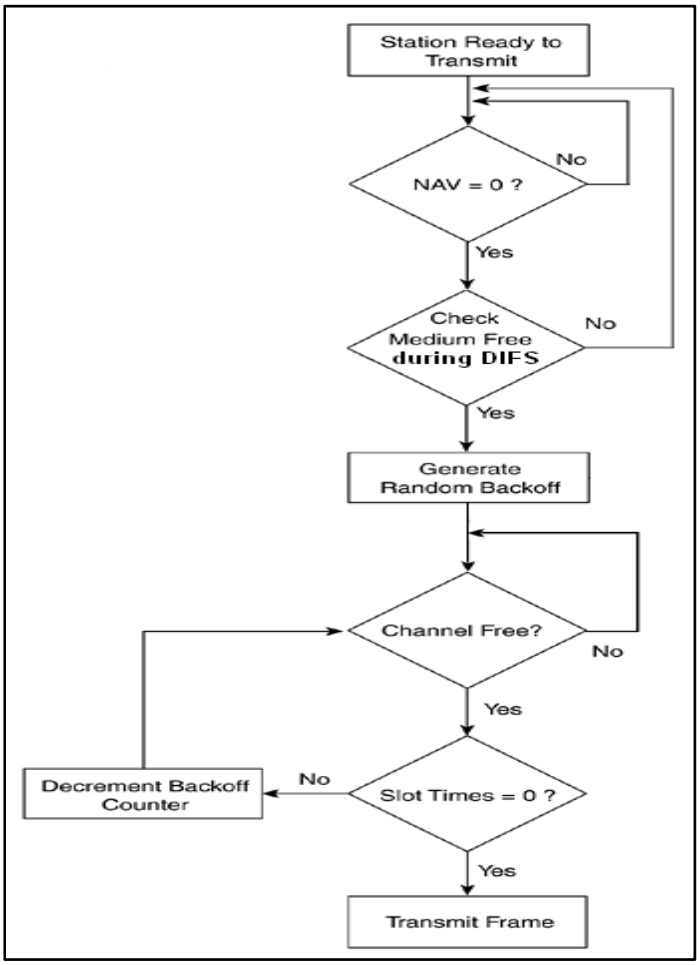


Figure 3.14, Medium Access Process for 802.11 Protocol

4.1.4 THE CONNECTING MEDIUM

In most LAN technologies, the medium is shared. Workstations connected to the network share the medium to send their data. The medium can have two characteristics:

1- Workstation sense the medium to check if it is busy or not. Channel status change means that the medium is in use by a workstation sending its data. In FDDI and Token Ring, if the token is possessed by one workstation the medium is busy for the other workstations until is releases the token. The *Carrier Sense* portion of the CSMA protocol means that before transmitting, each workstation must wait until there is no signal on the channel. In wireless LAN channel, the channel becomes busy if the *carrier sensing mechanism* indicates that the channel is busy.

2- Collisions occur at the medium. Workstations transmit their data over the medium. In CSMA, if two or more workstations use the medium at the same time to send their data, a collision will occur due to many signals propagating over the medium. As a result, no one of these signals will be usable at its destination. In Ethernet, the channel becomes free after a collision after a maximum time of 51.2 µs (two times the maximum time needed to send a bit from one end to other end on the channel) due to the use of collision detection technique. While in wireless LANs, the medium becomes free when the workstation with the longest packets finishes from transmitting.

4.2 BUILDING PATTERNS COMPONENTS

In our approach, we want to model reusable components. In this section, we will build the components that will be used to model the communication protocols. We specify interfaces to enable the assembling of these components to build other *composite-components*.

4.2.1 COMPONENTS INTERFACES

The component interfaces declare the services that a component offers. They are used as an access point to the component functionality by other components. Since we use Petri nets to model the different component behaviours, we used *places* to be the input interfaces of components and the output interfaces are *transitions*. This choice is coherent with the traditional way to model asynchronous communication between processes modeled by Petri Nets. A producer component fires an output transition and puts tokens in the input places of consumer modules. The connection between transitions and input places between two blocks can be 1-to-many or many-to-1.

As an example, figure 3.15 (a) shows a many-to-1 connection that is used to connect workstations output transitions to a medium input place since workstations put their data on the medium only. While figure 3.15 (b) shows a 1-to-many connection. This connection can be used to connect the medium output transitions to workstations input places, since all the workstations can see the signals propagating over the medium. A 1-

to-1 connection is possible when one workstation is connected to the medium. The light gray gates represent the connection part of the workstation and the light yellow gates represent the connectors on the medium module.

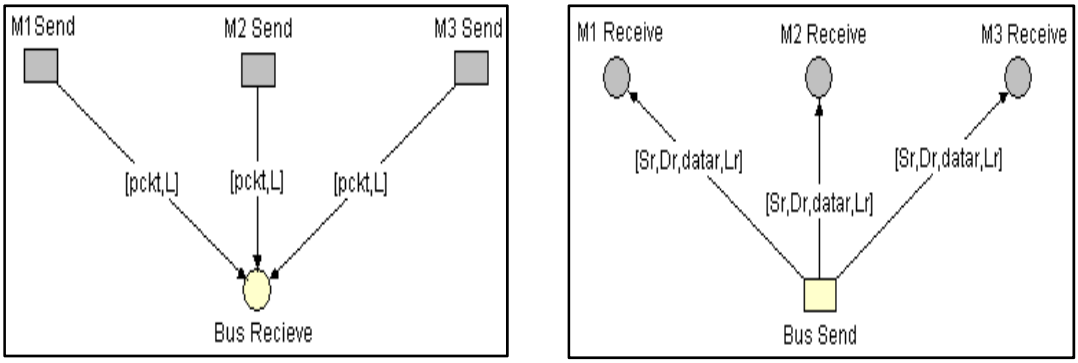


Figure 3.15 (a) Many-to-1 connection, (b) 1-to-many connection

In our approach, the interfaces' transitions put only tokens while places consume tokens. So external arcs cannot consume tokens from places, or be condition to fire transitions. However, some exceptions, where high dependency between components exists (like the dependency between the channel check and the backoff components) or when tokens will not be put in any place, may exist.

This approach is very useful to deal with the complexity due to the size of a system. Indeed, if one has already a model of some workstations connected to the medium and wants to increase the size of its model, the insertion of new workstations can be done easily by adding an arc connecting an output transition to an input place.

4.2.2 CHANNEL CHECK COMPONENT

On a network, the workstation is the most active part; it detects signals and checks channel changes. When a workstation wants to transmit, it checks at first if the channel is idle, then it defers for a period of time before it sends its data.

Figure 3.16 shows a channel check component. Elements in *light gray* represent the places and transitions used to build the component. Elements in *dark gray* represent the interfaces of the component. Initially, the channel is idle for all the workstation. This is represented by a token in place "Idle". A workstation that wants to send data (a token in

place “Data send”) must first check the channel. In wireless LANs, the channel must be free for a period more than DIFS, while in Ethernet, it is 9.6 µs. This is represented by the *@t’* at the arc between place “Idle” and transition “TF” (*t’* equals 9.6 µs in Ethernet and 50 µs in 802.11b). The workstation must wait before it starts transmitting, represented by a token put in place “sdata”. In Ethernet the wait “@t” equals to 9.6 µs, while in 802.11 it is equal to random value between CW_{min} and CW_{max} slots time. Place “Backoff/Deferring Time” and transition “FC” is used to decrement the backoff in wireless LAN, while for Ethernet, it can be left as it is in the figure (no dependence to that transition in the model).

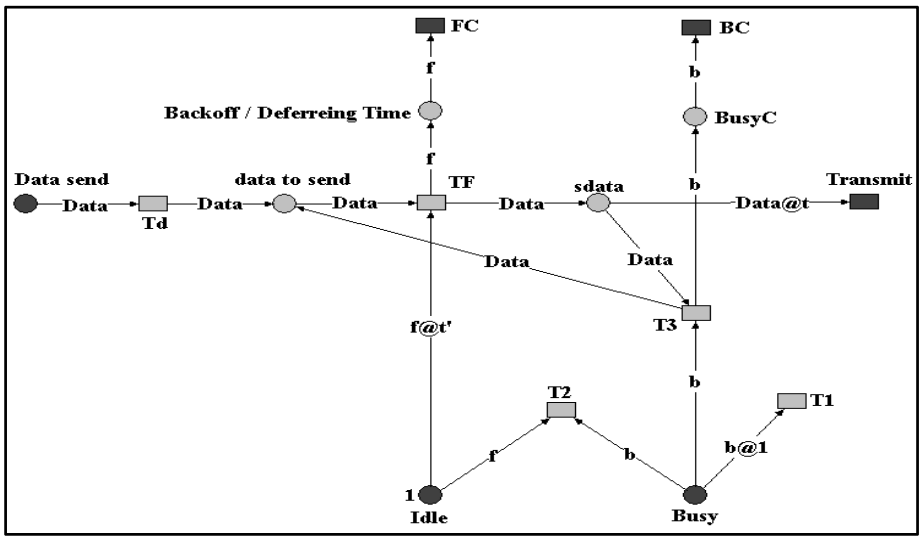


Figure 3.16, Channel Check Component

Consequently, if the channel status is changed (a token is put in place “Busy”), the workstation can be in one of the following cases:

- It is the transmitter (there is no more tokens in place “sdata”), then nothing is changed and the token in place “Busy” is consumed by transition T1;
- It attempt to send or it has no data to send, then T2 is fired;
- It is in the backoff/deferring phase, then T3 is fired (the workstation rechecks the channel again) and a token is put in place “BusyC” to stop decrementing the backoff. Hence, in wireless LAN, the workstation stops decrementing the backoff, but it keeps its remaining value.

In the three cases the channel status is changed from idle to busy.

Initially, this component has one token with value 1 (representing the free channel) in place Idle. The use of this component is possible in any protocol that demands the sensing the channel before transmitting data. It represents also the status of the channel free or idle. Let us notice here that, for genericity, we use two parameters t' and t to define the delay on the arc Idle-FT and arc Backoff/Deferring Time-Transmit

4.2.3 RECEIVING AND SENDING ACK COMPONENT

Workstations receive two types of packets: data packet and ACK/JAM frames. In Ethernet network, no acknowledgment is sent after the reception of packet. Therefore, the received packer can be either a data packet or a Jam frame. While in wireless LAN, FDDI and Token Ring, the received packet is either a data packet or an acknowledgment frame.

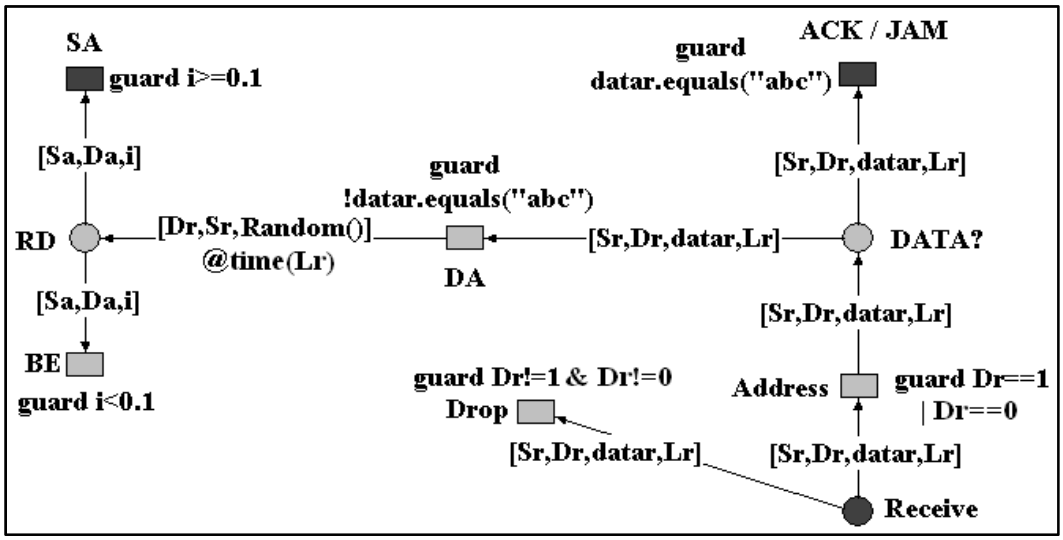


Figure 3.17 Receiving and Sending ACK Component

Figure 3.17 shows the receiving and sending acknowledgment component. One assumes that a token is put in place "Receive". The fields of the token represents: the source address "Sr", the destination address "Dr", the received data "rdara" and the last field represents the lengths of the packet. The workstation checks at first the destination address "Dr" of the packet. The guard condition on transition "Address" checks if the received packet belongs to this workstation, a token is put in place "Data?". Otherwise, the token in place "Receive" is eliminated by transition "Drop". Hence, "Dr==1" is

considered as the own address of the workstation, while “Dr==0” is used to represent the multicast or JAM frame reception.

Next, the guard condition of transition “ACK/JAM” is used to checks if the received frame is an ACK frame or a JAM frame (for Ethernet only). The “abc” in the guard can be modified according to the needs of the designer and the type of network. However, if the received packet is a data packet, transition “DA” is enabled. This transition is fired after a time equals to the time needed to receive the packet modeled by the “@time(Lr)” at the outgoing arc. This “@time(Lr)” is a function that returns the time corresponding to the length “Lr” of the packet.

Let us notice here, the functions dynamicity can be used to model mobility of a wireless networks nodes. This can be done since the bit rate is a function of the signal strength and that the signal strength is a function of distance. This means if the source knows the location of the destination, then the distance can be computed, and hence the time needed to send a packet is determined.

The last step is to represent the bit rate or receiving errors. The random function R*andom()* is used to generate a random variable *i*. Assuming that the bit rate error is less than or equal to 10% of the transmitted/received packets. So, if the value of *i* is less than 0.1, then the packet is discarded (the token in place RD is consumed by transition “BE”). Else, the packet is received correctly and then an acknowledgment is sent, by firing transition “SA”. This interface can be left unconnected in Ethernet. Again, the *Random()* function can be implemented by using any of the Java random functions. Also, as we can see in figure 3.17, the modification of tuples can be done easily, just by modifying the arc inscriptions according to our needs.

As we saw above, this component has an important functionality since it is used to identify the received data (own or not), the type of the received data (JAM, ACK, data frame) and the process of sending an acknowledgment after a successful reception. Thus the use of this component is possible for the protocols demanding the identification of data and the send/receive process.

4.2.4 BACKOFF / BEB COMPONENT

The third component is the backoff / BEB component shown in figure 3.18. As we can see in the figure, retransmitting the packet depends on the value of *n*, (transitions T6 and T7). If the packet is correctly sent/received (a token is put in place "Done"), then *n* is reset to *z* (0 for Ethernet and 1 for wireless), for the next attempt to transmit, place N.

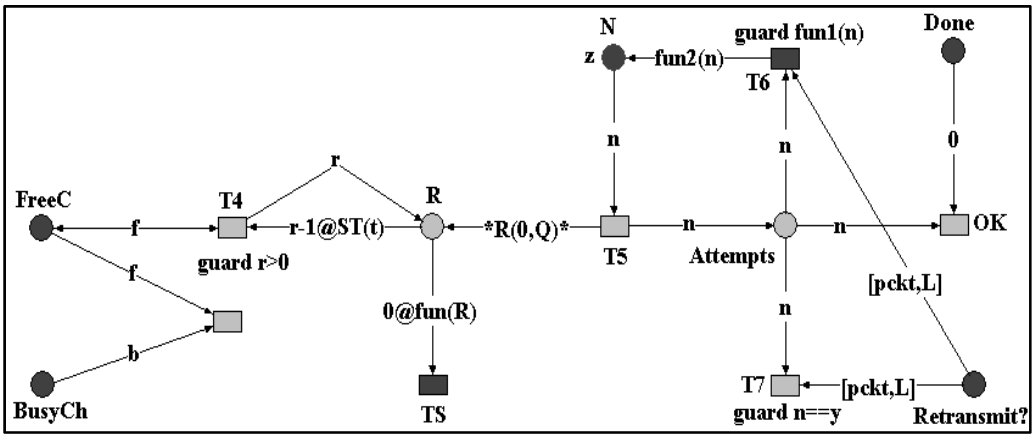


Figure 3.18, Backoff / BEB Component

Variable Value	Ethernet	IEEE 802.11b
fun1(n)	n<15	n<33
fun2(n)	n=n+1	n=n*2
y	16	64
z	0	1
R(0, Q)	random(0, 2^X), X depends on n	random(0, CW)
Fun(R)	R*51.2μs	0
ST(t)	0	Time slot (20μs)

Table 3.1, Differences between Ethernet and IEEE 802.11b networks

However, the component inscriptions depend on the type of the network. As an example, table 3.1 shows the differences between Ethernet and IEEE 802.11b networks. In addition to table 3.1, in Ethernet, places "FreeC" and "BusyCh" are not used (they can be left as it is), since the backoff decrement in Ethernet does not depend on the status of

the channel. While in 802.11b, this interface is very important in decrementing the backoff each time the channel is free for a slot time or the backoff is conserved if the channel status is changed to busy. The firing of transition TS represents the (re)transmission allowance of a packet (backoff equals to 0).

The backoff component is useful for the protocols that may need a specific timing procedure since it can be related to another components (which the case of wireless: by checking channel always) or just for standalone use.

4.2.5 MEDIUM COMPONENT

Figure 3.19 shows the medium sharing component. A token is put in place "Receive" representing packet transmission over the medium. Firing M1 increment the variable *x* by 1 (initially x equals to zero, place "P5"). Once x equals to 1, the channel status is changed to busy or a transition "Busy" is fired (hence P7 contains only one token with value 0, so transition "Busy" is fired one time only).

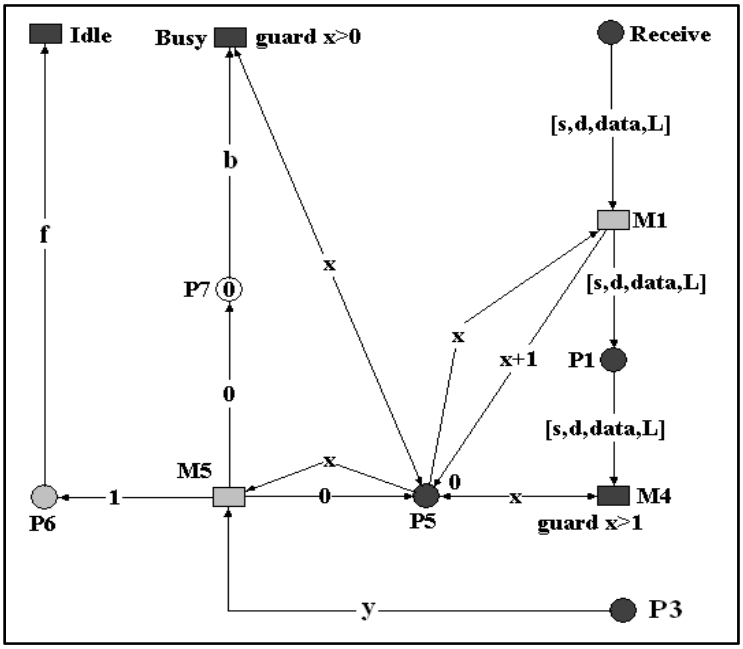


Figure 3.19, Medium Sharing Component

The variable *x* is also used to count the number of workstations transmitting their data over the medium at the same time. If two or more workstations are sending at the

same time, this means that a collision is occurred, then transition “M4” is fired to remove the token modeling the frame to transmit. However, the firing of transition “M4” depends on the type of the network. Place “P1” represents the “data transfer” to the other workstations. However, if a token is put in place “P3”, then the channel can become free again and transition “M5” is fired; a token with value 0 (at the arc inscription between M5 and P7) is put in place P7 and a token with value 1, (the arc inscription between M5 and P6) is put in place P6 (representing the free channel) enabling and firing the transition Idle.

4.3 PROPERTIES ANALYSES OF THE BUILT COMPONENTS

The basic properties of Petri nets include reachability, boundness, and liveness. Several methods are used to analyse Petri nets: *reachability analysis*, *invariant analysis* and *reduction* [Lewis98]. However, applying these methods on high level Petri nets is delicate and not easy. In literature, different methods are proposed to verify some properties. [Boukadi07] and [Hinz05] have proposed to use the *state space method* that consists of designing a graph with a node for each reachable marking and an arc for each occurring binding element. [Evangelista05] and [Liang06] have proposed reduction algorithms for the number of states in the model to verify some properties. However, most of these methods are complex and can be only used to verify some properties, but not all [Lakos02]. Another proposition is to subdivide the complete model [Khomenko03] into smaller parts. However, in [Petrucci05] study, the analyzed part does not guarantee the same results for the whole model.

In this work, properties of the built components are not done. This is due to the inability of the used tool to perform these analyses. Supporting the tool with these capabilities is not an easy task. Another possibility is to transfer the model to basic Petri nets or any other modeling formalism. Again, transformation cannot be made easily. For the future work, this part must be covered by developing new modeling tool capable to perform such properties or by combining two (or more) modeling tools together to achieve this goal.

5. APPLICATION PROTOCOLS

In this section, we will illustrate our modeling approach through two examples: IEEE 802.3 Ethernet MAC protocol and IEEE 802.11 MAC protocol because both protocols are based on CSMA. One of the objectives is to illustrate the advantage of having generic components and the hierarchical composition that allows building composite-components.

5.1 MODELING IEEE 802.3 ETHERNET PROTOCOL

5.1.1 ETHERNET OVERVIEW

Ethernet [Spurgeon00] is the most widely used LAN technology in the world. Ethernet was designed at its beginning at the *Xerox Palo Alto Research Center PARC*, in 1973. The used protocol differs from the classical protocols like token control, where a station cannot send before it receives an authorization signal, the token. With Ethernet, before transmitting, a workstation must check the channel to ensure that there is no communication in progress, which is known as the *CSMA/CD Protocol.*

Supplement	Topology	Medium	Max Segments	Nodes/Segment
802.3 –1985	10Base5	Thick coax	500	100
802.3a –1985	10Base2	Thin RG-58 coax	185 m	30
802.3i –1990	10Base-T	CAT 3/5 two-pair UTP	100 m	1024
802.3j –1993	10Base-F	Two-strand multimode fiber	2000 m	1024
802.3u –1995	100Base-T	CAT 5 two-pair UTP	100 m	1024
802.3z –1998	1000Base-X	Gigabit Ethernet variant mediums	25–10000 m	1024

Table 3.2, Ethernet Supplements

The original 10 Mbps Ethernet standard was first published in 1980 by the *Digital Equipment Corporation Intel-Xerox DECIntel-Xerox Vendor Consortium*. This standard is called *DIX Ethernet* based on thick coaxial cable. The first Ethernet card appeared was

in 1982 called Ethernet II. In 1983, the *Institute of Electrical and Electronics Engineers IEEE* develops the Ethernet standard with *802.3 Ethernet* identifiers [IEEE09], table 3.2. The IEEE standards have been adopted by the International Standards Organisation, and is standardised in a series of standards known as ISO 8802-3.

The IEEE standard was first published in 1985 with the title *IEEE 802.3 Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications.* At the same year it publishes the standard 802.3a or the *Thin Ethernet*. In 1987, fiber optics was used with the 802.3d standard. Despite the availability offered by other high-bandwidth networks like ATM and FDDI, Ethernet is very interesting because its low cost, maturity and stability.

5.1.2 CSMA/CD MECHANISM

In Ethernet network, to control which host is allowed to transmit at any given time, a protocol is necessary. The simplest protocol is known as *ALOHA* [Tanenbaum03]. ALOHA allows any host to transmit at any time, but conditions the hosts to add a *checksum/CRC* at the end of its transmitted frame to let the receiver to identify whether the packet was correctly received. So, ALOHA offers a best effort service, it therefore relies on *ARQ protocols* to retransmit any corrupted data.

Ethernet uses *CSMA*, a refinement of ALOHA, to improve the performance when the medium is highly used. The *Carrier Sense* portion of the CSMA protocol refers to that before transmitting each host must wait until there is no signal on the channel. If another host is transmitting, there will be a signal on the channel. With *Multiple Access* all the Ethernet hosts have the same priority to use the network, and can attempt to access the channel at any time. The next portion of the access protocol is called *Collision Detect*. Since each host has equal opportunity to access the channel, it is possible that multiple hosts start transmitting their packets simultaneously. When this happens, the hosts sense the *collision signal, JAM*, which informs the hosts to stop transmitting. Each host will then choose a BEB time and retransmit their packets. Collisions are normal events on Ethernet and they are an indication that the CSMA/CD protocol functions as designed.

5.1.3 MODELING AN ETHERNET WORKSTATION

An Ethernet workstation is the most active part in the network. In our approach, we want to model reusable components. In this section we will show the *reusability* of the modeled components above to model Ethernet workstation. These components are reused with the *needed modifications* to answer the specification of an Ethernet workstation. Figure 3.20 shows the detailed and complete module for the Ethernet workstation. As we can see in the figure, the three components: Backoff component, Channel Check component and Receive/Send component are *reused* to build the workstation. To complete the model and to bind the used components together, some additional places and transitions (in white) are used.

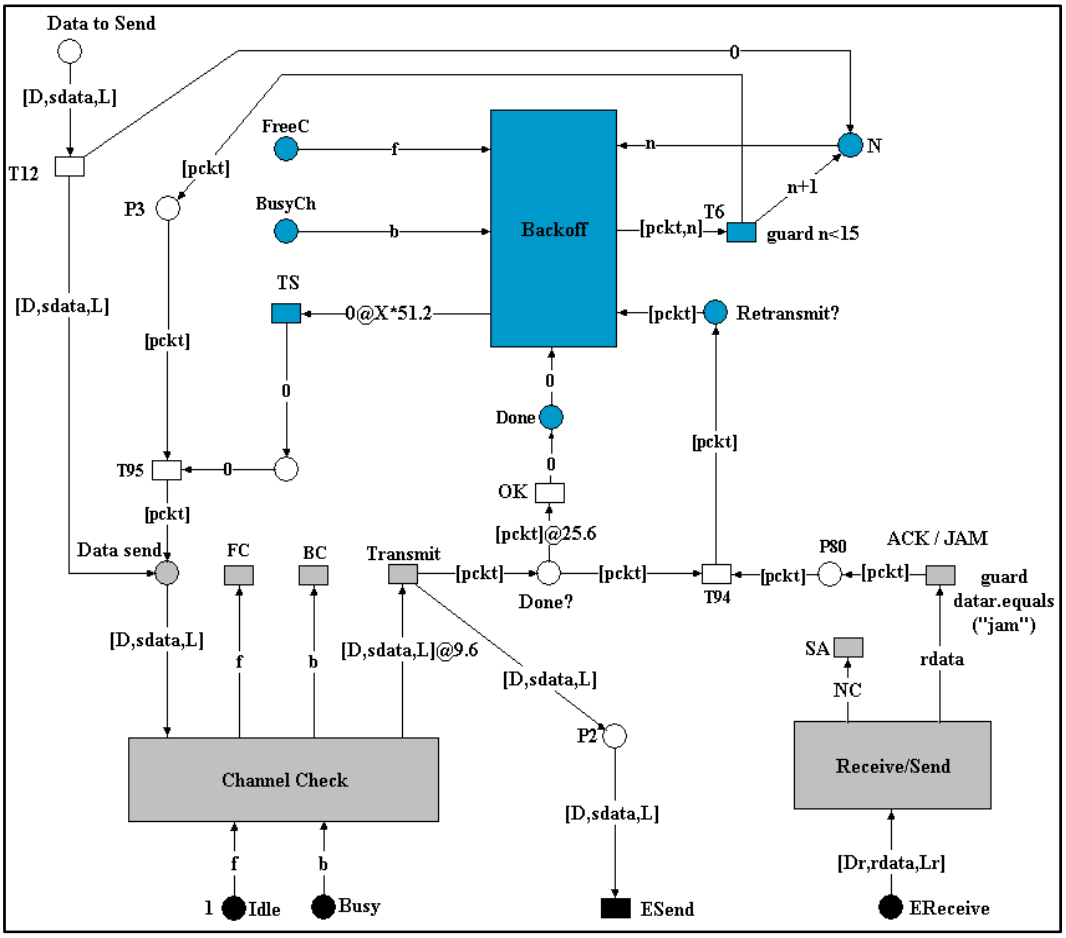


Figure 3.20, Hierarchical Design of an Ethernet Workstation Component based on Generic Basic Components

In the figure we can see that five interfaces were not connected:

- The “FreeC” and “BusyCh” interfaces of the Backoff component, and the FC and BC interfaces of the Channel Check component, since Ethernet workstations decrement their backoff without the need to check whether the channel is idle.
- The SA interface of the Receive/Send component, because in this part we do not model the service offered by an Ethernet workstation. However, in the next chapter we will use this interface to classify/identify the received packets.

An important notice we can see also in the figure is that the whole component can be reused as one component for the Ethernet workstations to build a complete Ethernet network. In other words, this new component is seen as a *composite-component* with the black places and transitions as the interfaces of this new component.

5.1.4 MODELING ETHERNET MEDIUM

The second part in the Ethernet network is the medium where the workstations exchange the messages. Figure 3.21 details and completes the module for the Ethernet medium. The Medium Sharing component is reused to model Ethernet medium component. Several transitions and places (in white) are necessary to complete the model.

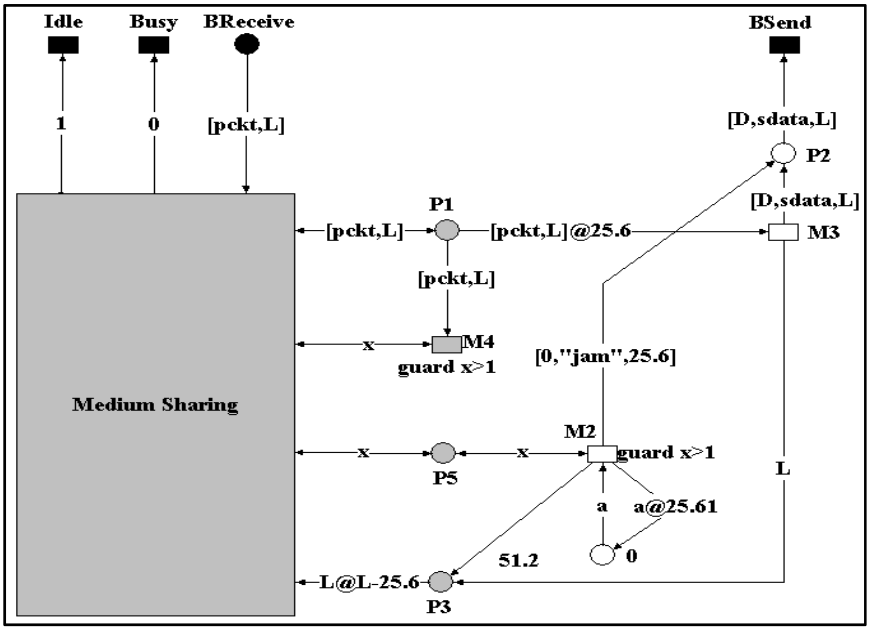


Figure 3.21, Hierarchical Design of an Ethernet Medium

Transition M2 is used to model the collision part. If two or more workstations send their data during the period of 25.6µs over the channel (x is greater than 1) then a collision occurs. In this case, the conditions on transition M4 and transition M2 are satisfied enabling these two transitions. The firing of M2 will generate the JAM signal, or a token put in place P2. While M4 is used to eliminate all the signals on the medium, tokens in place P1.

As the workstation component, the medium module can be reused as one component of the Ethernet medium to build a complete Ethernet network (Ethernet network with bridges for example). Thus, the black transitions and places are the new interfaces of the new complete component.

5.2 MODELING IEEE 802.11B WLAN PROTOCOL

The second application protocol is the DCF IEEE 802.11b WLAN protocol.

5.2.1 IEEE 802.11 PROTOCOL OVERVIEW

Wireless technology has become popular to access to internet and communication networks. The IEEE 802.11 [IEEE07] offers the possibility to assign part of the radio channel bandwidth to be used in wireless networks. The IEEE 802.11 protocol is a member of the IEEE 802 family, which is a series of specifications for local area network technologies, (figure 3.10). IEEE 802.11 is a wireless MAC protocol for *Wireless Local Area Network WLAN*, initially presented in 1997. The IEEE 802.11 standard defines *Medium Access Protocol and Physical (PHY) frameworks* (layer 2 in the OSI model) to provide wireless connectivity in WLAN. This independence between the MAC and PHY has enabled the addition of the higher data rate 802.11b, 802.11a, and 802.11g PHYs. The physical layer for each 802.11 type is the main differentiator between them. However, the MAC layer for each of the 802.11 PHYs is the same.

Many other 802.11 variants have appeared also. In 2004, the ***802.11e*** is an attempt enhancement of the 802.11 MAC to increase the quality of service. The ***802.11i*** and ***802.11x*** were to enhance the security and authentication mechanisms of the 802.11 standard. Many other variants like *802.11c, 802.11d, 802.11h.*

IEEE standard	Net Bit Rate	Frequency band	Notes
802.11	1 Mbps – 2 Mbps	2.4 GHz	1997.
802.11a	Up to 54 Mbps	5 GHz	1999. Products not released until 2000.
802.11b	5.5 Mbps – 11 Mbps	2.4 GHz	1999. The most common 802.11.
802.11g	Up to 54 Mbps	2.4 GHz	2003. Applies the coding techniques of 802.11a for higher speed in the 2.4 GHz band, while retaining backwards compatibility with existing 802.11b networks.
802.11n	Around 500 Mbps	2.4/5 GHz	2009-2010. Build on the previous 802.11 standards. max speed of 600Mbps with the use of 2 spatial streams at a channel width of 40 MHz

Table 3.3, IEEE 802.11 PHYs variants

5.2.2 IEEE 802.11 OPERATION MODES

Two operating modes can be used for setting up an IEEE 802.11network: the *infrastructure mode* and the *ad hoc mode*. In the ad hoc mode, hosts communicate directly with each other without intermediates. The mobile workstations located within the reach of each other create an *IBSS* or *Independent Basic Service Set*. In the infrastructure mode, all communications between mobile workstations or between the workstations and the outside network pass through an access point AP who takes the role of the relay. The coverage of an AP constitutes of a *BSS* or *Basic Service Set.*

Access points can be linked together through a *distribution system DS*. The standard does not give specifications on the nature of this interconnection but it is usually a wired network (such as Ethernet). The whole interconnected wireless LAN including all the hosts, access points and the distribution system, is seen to the upper layers of the OSI model as a single 802 network, and is known as the *Extended Service Set ESS*, figure 3.22.

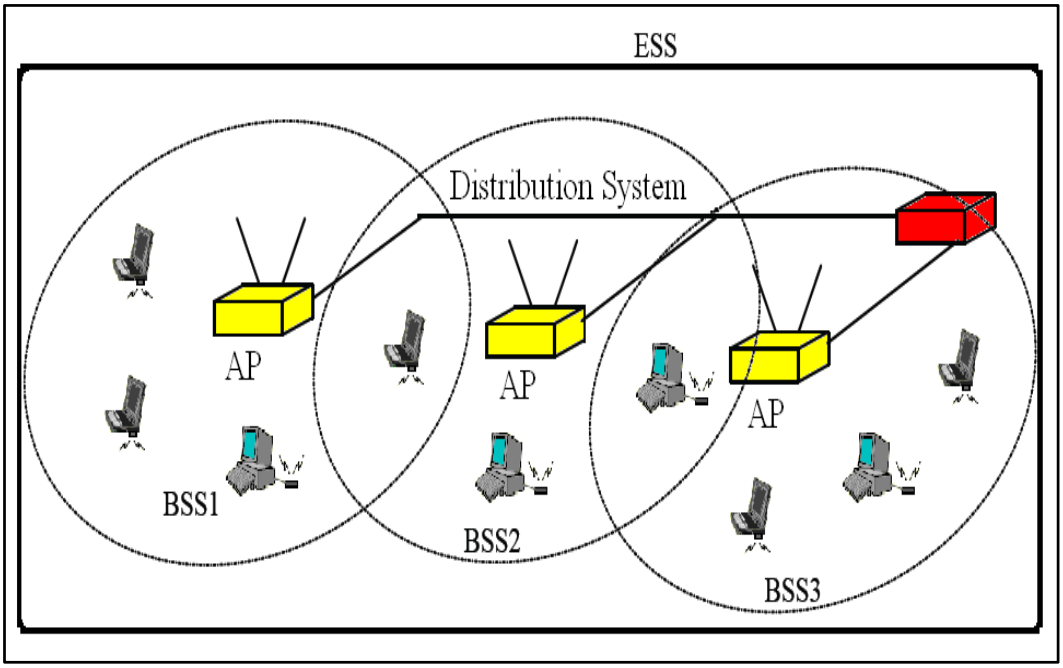


Figure 3.22, Typical Architecture of the IEEE 802.11 LAN

The IEEE 802.11 MAC layer defines two coordination functions to access the wireless medium: A *distributed coordination function DCF* and a *centralized coordination function PCF (Point Coordination Function).*

5.2.2.1 POINT COORDINATION FUNCTION PCF

PCF allows an 802.11 network to provide an enforced fair access to the medium. It is an optional part of the 802.11 specification. The access to the medium under the PCF looks like token-based medium access control schemes, with the access point holding the token. Direct communications between wireless workstations are not possible; they must all pass through the access point. As a result, half of the bandwidth is wasted. This method has been launched by the standard to meet the needs of users with real-time traffic. It is based on defining a *contention free period CFP* to be held alternately with the *contention period CP* managed by the DCF method. Alternating periods at regular intervals is known as the *contention-free repetition interval*.

The contention period, figure 3.23, must be long enough for the transfer of at least one maximum-size packet and its associated acknowledgment.

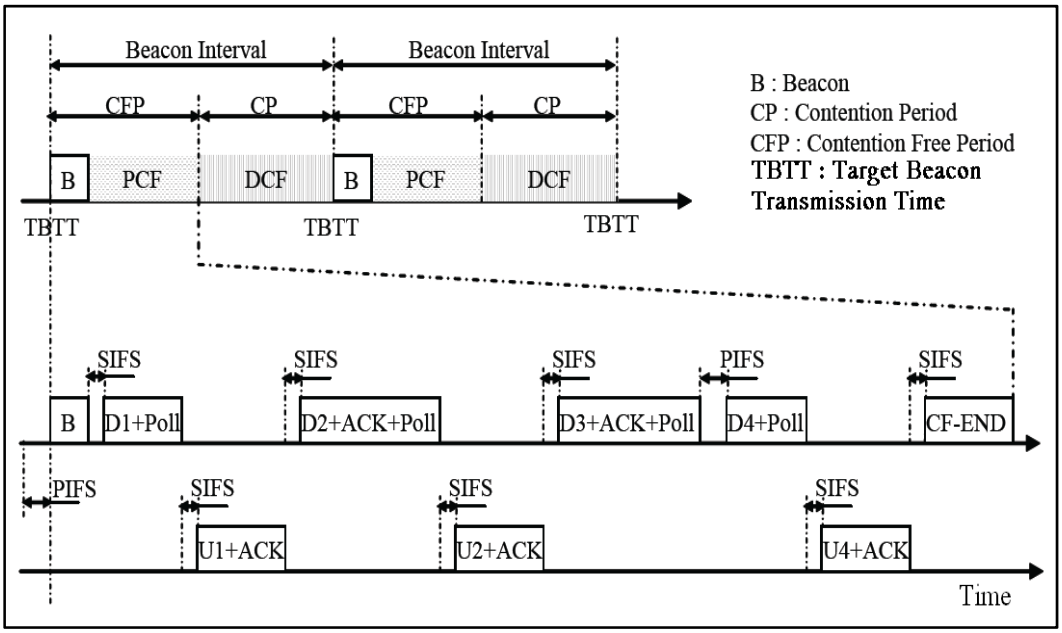


Figure 3.23, Point Coordination Function PCF

5.2.2.2 DISTRIBUTED COORDINATION FUNCTION DCF

DCF [Bianchi00] is the fundamental access method used to support *asynchronous data transfer on a best effort basis*. As identified in the specification, all the workstations must support the DCF. The DCF is based on the CSMA/CA protocol. The reason is that even though the wireless LAN is a broadcast medium, the traditional CSMA/CD will not function properly because the workstation is unable to listen to the channel for collisions while transmitting, due to the big difference between transmitted and received power levels. To overcome this problem CSMA/CA protocol uses a positive acknowledgement mechanism.

In IEEE 802.11, carrier sensing is performed at both the physical layer and the MAC sub-layer. On the physical layer, the carrier sensing is referred to as the *physical carrier sensing* which detects an activity in the channel via relative signal strength from other workstations. On the MAC sub-layer, carrier sensing is known as the *virtual carrier sensing* which is used by the source to inform all the workstations in the BSS for how long the channel will be used for successful transmission of a *MAC protocol data unit*. The source set the duration field in the MAC header of data packets (or in the *Request to*

Send RTS and in the *Clear to Send CTS* control frames, figure 3.24). The cannel is marked busy if either the physical or the virtual carrier sensing mechanism indicates that the channel is busy.

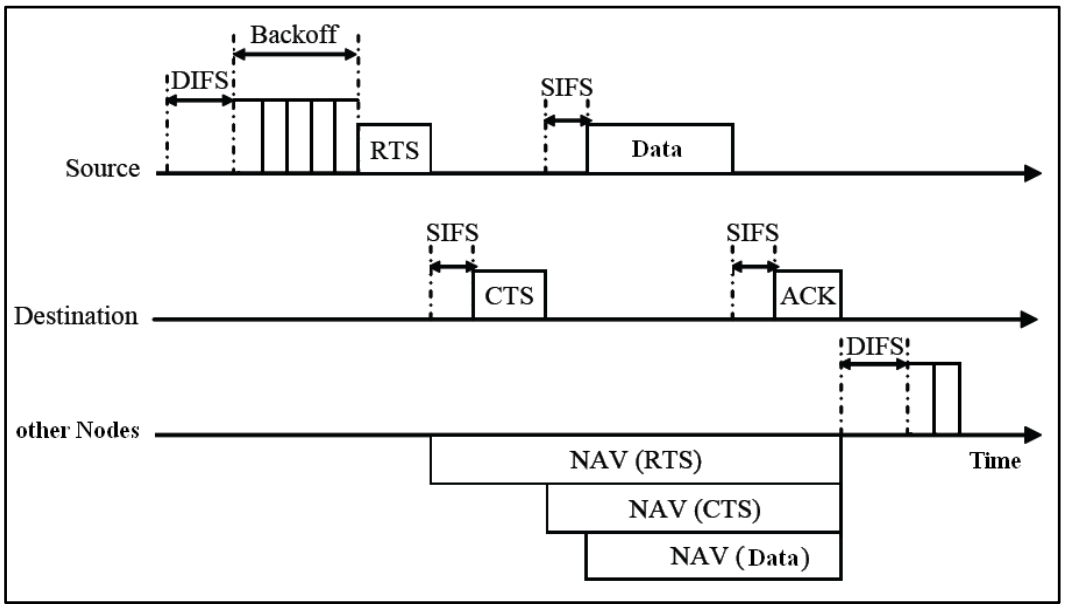


Figure 3.24, DCF Transmission with RTS/CTS

Priority access to the wireless medium is controlled through the use of *Inter-Frame Space IFS* time intervals between the transmissions of packets. The IFS intervals are mandatory periods of idle time on the transmission medium. Three IFS intervals are specified in the standard: *Short–IFS (SIFS), Point Coordination Function–IFS (PIFS)* and *Distributed Coordination Function–IFS (DIFS).* The SIFS interval is the smallest followed by the PIFS followed by DIFS.

5.2.3 MODELING A DCF IEEE 802.11B WORKSTATION

In the last sections, we saw how the built components can be reused to model the IEEE 802.3 workstation module. Figure 3.25 shows the detailed and complete module for the DCF IEEE 802.11b workstation model by the reuse of *ready-to-use components* in the previous sections.

The workstation sets the value of N to 1 (place “N”), sense the channel (transition “TF”), sends its data (place and transition “Send”) and waits for an acknowledgment

(place “Wait”). If no acknowledgment is received during the SIFS period or 10µs, Transition T11 will fire putting a token in place “Retransmit?” to check if the packet can be retransmitted (transition T6) or not (transition T7). As we can see in this figure, all the components are reused to composite the workstation module. All the interfaces were also used in this module.

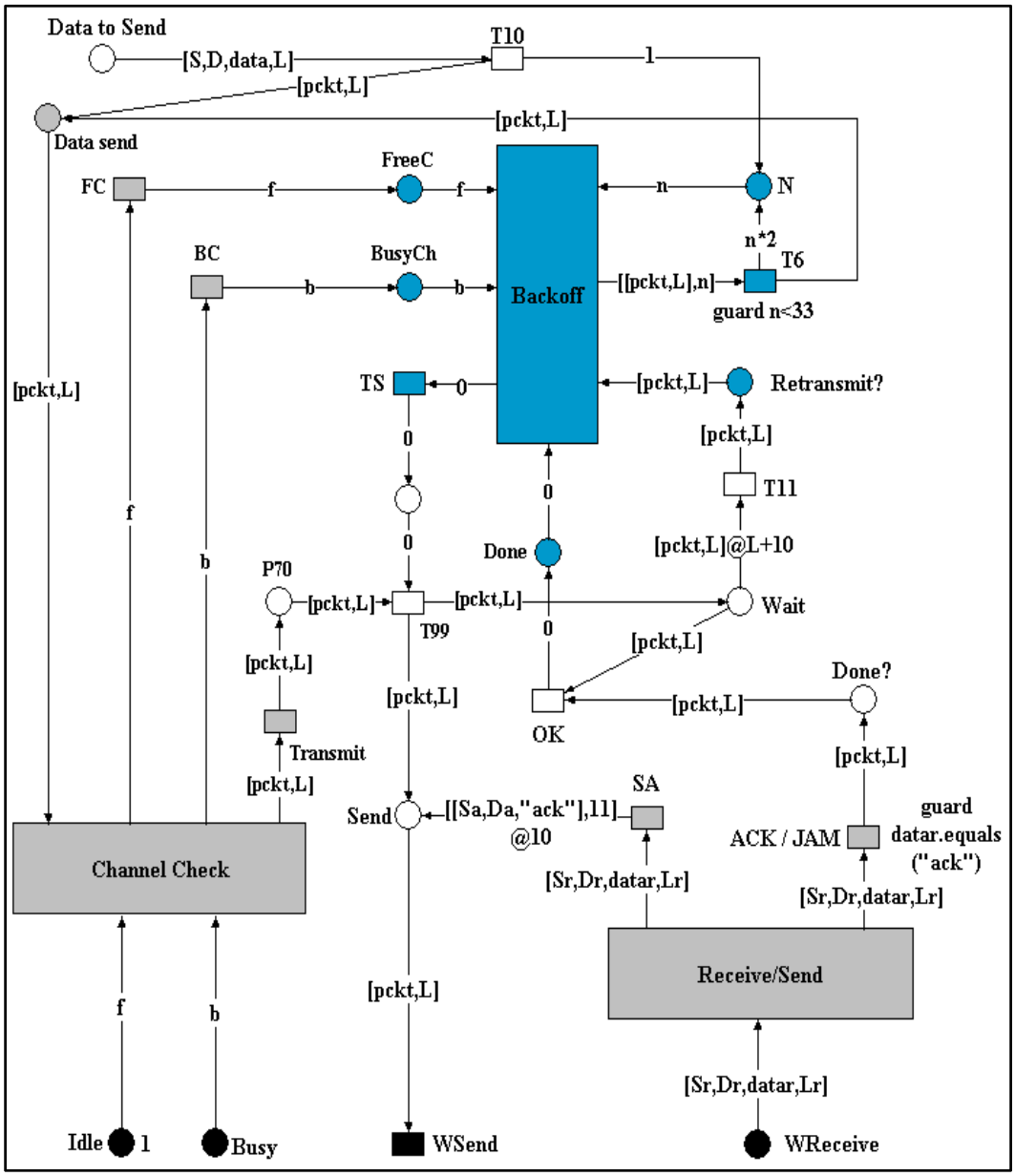


Figure 3.25, Hierarchical Design of a DCF IEEE 802.11b Workstation Component based on Generic Basic Components

5.2.4 MODELING THE WIRELESS MEDIUM

Figure 3.26 shows a detailed and complete component of the wireless medium. In this model we used more functions to complete the modeling. The *Math.max(L,R)* function is used to compare the different packets lengths. Place P4 has a token with assumed value of 10 bytes as the smallest packet size. When many workstations send their data over the medium, a collision occurs. However, workstations continue transmitting even after the collision.

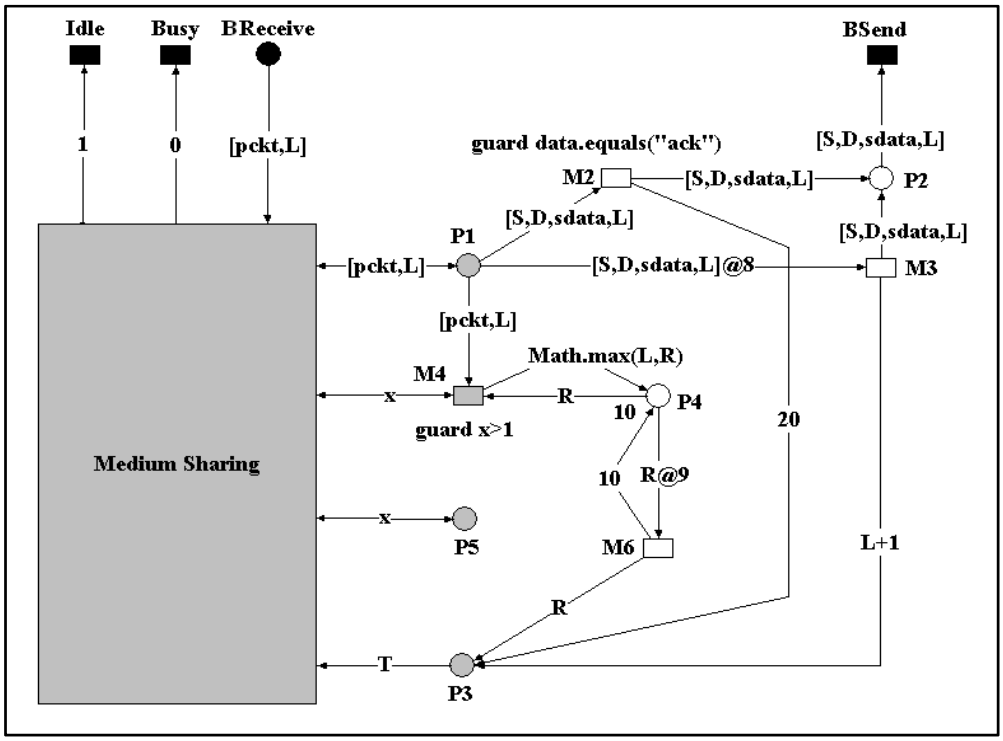


Figure 3.26, Hierarchical Design of DCF IEEE 802.11b Medium

Here, the *Math.max(L,R)* function compares the length of the sent packets L with the assumed value R. If the L is greater than R, it has the value of L, otherwise R keeps its value. We assumed also that all the workstations in the medium will know that the channel is busy during 9µs so no workstation transmits after this time. As a result R will have the value of the packet with the longest size L. The medium becomes idle after the end of this transmission. The firing of transition M6 will put a token in place P3 with the value of *R* which will be used to free up the channel (to fire transition Idle).

6. CONCLUSION

Component-based methodology is a great technique and designing method derived from the object-oriented design. A component-based approach defines that a component designer does not need implementing the technical concerns several times. The designer only identifies the services required by a component, and makes sure that these services are available by the developed components. System models can be assembled from working together components, accessing each other through distinct component interfaces that hide the component implantation details.

Since the functionality and the way to access the components are well-defined, pre-existing, ready-to-use components can be reused in several models. Our approach is based on the component-based technique and designing method to model communication protocols and distributed systems. HLPN permit the representation and manipulation of an object. Petri nets are a powerful formalism for modeling concurrency and distribution.

Constructing a library of ready-to-use components can help in modeling new systems easily. To build such library, analyses, constraints and user requirements are the key factors to complete it. To examine and motivate the reusability of generic components which reduces the time and the cost needed to build new models, we have chosen wireless protocol IEEE 802.11b and Ethernet IEEE 802.3 as two application examples. As a result, our component-based design approach ease the development of generic components, supports their reusability, increases adaptability of service composites, and enables continuous extensibility of components functionality. In the next chapter we will verify the accuracy of our approach and evaluate the complete model with a study case of the manufacturing systems.

Chapter 4

PERFORMANCE EVALUATION OF DISTRIBUTED SYSTEMS

1. INTRODUCTION

Performance evaluation [Haverkort98] [Fortier03] is often important in the design, development, and configuration of complex systems especially for computer and distributed systems [Masri09a]. A system may work properly, but it must also work efficiently. Performance evaluation deals with existing or planned systems, and aims to compare different configurations, or to find a favourable configuration of a system. Three techniques are used for evaluating the performance of a system: *measurement, analytical models and simulation models*.

Measurement can offer the most exact and accurate results. The system is observed directly. However, measurement is often the most costly of the techniques since it is only possible if the system already exists. In some cases, measurements may not be accurate since it depends on the state of the system. For example, if network measurements are done during peak period, the observed results would be not the same if the measurements were done during a low use period of the network.

Analytical models may provide exact answers. Analytical modeling uses simple mathematical expressions to obtain the performance results for a system. However, these results may not be accurate, because of their dependencies on the made assumptions in order to create the model. The behaviour of computer systems including processing periods, communication delays, and interactions over the communication channels is difficult to model. Analytical models are excellent to model small to medium systems that fulfil some requirements but it is not the case for industrial-sized, networked and distributed systems.

However, simulation models can be used as an alternative choice to analytical techniques. Larger and more complex models can be built and analysed. Simulation models allow creating very detailed, potentially accurate models. However, developing the simulation model may consume a good amount of time, but once the model is built it takes a little time to get results.

In this chapter we will introduce the different models and methods used to evaluate the performance in communication and distributed systems. We will particularly focus on the simulation model approach (where we are interested in) and the advantages and disadvantages of this method over the other methods. The second part will be dedicated to the simulation results of the communication protocols models presented in chapter 3. A comparison with NS-2 simulator and other studies will be given to prove the accuracy and quality of our models. The last part will combine the modeling of communication protocols and the manufacturing system presented in chapter 1. The impact of using different communication protocols over this system will be verified.

2. PERFORMANCE EVALUATION TECHNIQUES

This section will focus on the analytical and simulation modeling techniques.

2.1 ANALYTICAL MODELS

Analytical modeling is one of the fast and cheapest techniques for evaluating the system performance. It is based on the modeling the systems under a form of parameters, variables and a set of mathematical formula that control their relations. With analytical models, one can model and evaluate the behavior of the system in the flexible manner. However, these techniques need many simplifications and hypotheses to obtain a coherent system. These simplifications and hypotheses are one of the major limits and challenges for the accuracy of the obtained results with the analytical techniques. Analytical techniques take different forms, from simple bounds analysis to the evaluation of complex Markov Chains.

2.1.1 STOCHASTIC MODELS

Stochastic models [Mieghem06] [Ethier86] are mathematical description represented by random variables with uncertain results and where one can only compute the probabilities of possible outcomes. In a stochastic model, the total behaviour of the system is expressed as a *stochastic process in time*. A stochastic process denoted as $\{X(t), t \in T\}$,is a collection of random variables $X(t)$ that change their value throughout the time t

which runs over an index set *T*. The set of all the possible values of *X(t)* is called *state space*.

A Poisson process and *Markov process* are two stochastic processes. *A Markov process* is also a stochastic process with some additional properties: the future behaviour of the process depends only on the present value, but not on the states assumed in the past. A Markov process is called a *Markov chain* if its state space is discrete.

Stochastic processes are distinguished by:

- Their state space,
- The index set T,
- The dependence relations between the random variables X(t).

Stochastic models are interesting when the process has a strong element of random motion. It gives also more information about statistical uncertainties involved in the process. The performance measures are expressed as functions of stochastic process. These functions can be more or less easy to determine depending on the type of the stochastic process and of the desired measures.

2.1.2 QUEUEING MODELS

The *queueing theory* [Cassandras08] [Mieghem06] is one of the most used methods for the performance evaluation in computer systems. It represents and analyzes resource-shared systems, such as production systems and communication systems. A *queueing model* is presented as a set of *servers* interacting together (which represents the systems resources) and a set of clients (represents the users sharing these resources). Graphically, this model is represented as a direct oriented graph with nodes representing the servers and the links between them represent the requests behaviour of the clients from these servers. In distributed systems, many clients attempt to access the shared resources.

Because the request rates vary in time, waiting situations occur when more than one user wants to access a single resource. The idea of queueing systems is to model shared resources as service providing entities preceded by waiting queues, figure 4.1.

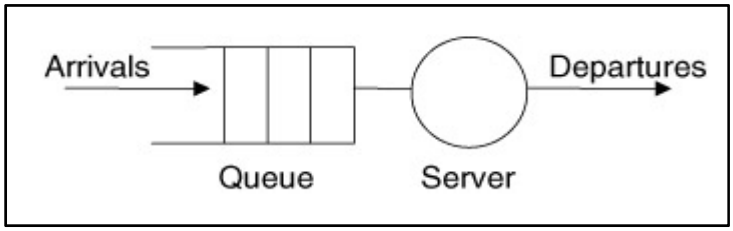


Figure 4.1, Single server model

Queueing models have some characteristics:

- The arrival process of clients.
- The clients' population.
- The amount of waiting room.
- The amount of service a client requests.
- The service capacity.
- The service discipline, such as:
  - *FCFS or FIFO*: First Come, First Served,
  - *LCFS(-PR) or LIFO*: Last Come, First Served, with or without Pre-emption,
  - *RR*: Round-Robin.

The queueing theory makes use of a particular type of notation *A/B/m/K* so as to describe a system, where:

- *A* is the arrival time distribution
- *B* is the service time distribution
- *m* is the number of present servers, $m = 1, 2, \ldots$
- *K* is the storage capacity of the queue, $K = 1, 2 \ldots$ If the *K* position is not present, it means that $K = \infty$.

Moreover, the distributions of *A* and *B* has some common notation:

- M is Markovian (exponential) service time or arrival rate,
- G is general service time or arrival rate,
- *D is the Deterministic* case where the service and arrival times are fixed.

Thus, queueing systems exist in several models:

- The M/M/1 system is a basic and simplest queue model. It consists of a Poisson arrivals process with exponential distribution of the service such as $A(t) = 1 - e^{-\lambda t}$ *and* $B(t) = 1 - e^{-\mu t}$, for some positive parameters λ (arrival rate) and μ (service rate), one server, infinite capacity and population, FCFS (FIFO) queue ordering discipline.
- The M/M/1/K system: the same as M/M/1 system but with a finite queue size.
- The M/M/C system: the same as M/M/1 but with multiple servers, C.
- The M/G/1 system: the service time does not have the Markov property.
- The G/M/1 system: the service time is random but the arrival process is non-Markovian.

Layered queueing networks LQN [Woodside95] is one of its extensions. It allows to model the client/server distributed architecture with concurrent interactions. In this extension, servers can become a client of other servers but it continues serving its own client.

Performance can be evaluated in such models once the queueing model has been characterised completely. Several performance measures can be done: steady-state probability of having n clients in system, service rate of one server, total time spent in the waiting line by client n, the number of clients in queue at time t, long-run average time spent in system per client, and so on.

Analytical models try to abstract details of a system and can be used at any stage of the systems design. They can directly present statistics for the modeled system. In addition, they represent a flexible modeling methodology with low time consumed in constructing the model. However, accuracy, trustability and believability of the obtained results of such models are not high since simplifications and hypotheses are needed to solve the systems equations which affect the results. In addition, the larger the system is, the more complex the model becomes.

2.2 SIMULATION MODELS

Real systems are very complex to accept analytical solutions. However, mathematical models may be still applicable but their problem is the need to have the tools able to solve

the equations of such models. *Simulation* [Fortier03] allows modeling and observing the system behaviour. It facilitates the understanding of the real system. Simulation allows the evaluation of a system model in an executable form, and the data of such process are used to *calculate* different measures of interest. Simulation is generally used because the complexity of most systems requires the use of simple mathematical ways for practical performance studies. This makes simulation as a tool for evaluation.

The use of simulation for performance verification of distributed systems is highly used in the research because of its large modeling capacity. Analytical solutions for discrete event systems are mainly hard to acquire. This makes simulation a very attractive for their study. As an advantage over the analytical models, every system can be modeled, as many dynamic and complex interactions can be taken into consideration for the simulation. Moreover, in most of the cases the functioning and performance of a system can be verified with the same simulation environment.

Another advantage of simulation methods is that they are generally reusable over different abstraction levels. In other words, the level of abstraction for the simulation can be customized to the necessary degree of accuracy. However, the major disadvantage of this technique is that it needs (in some cases) important and consistent calculation resources and time. Many communication networks, manufacturing, transportation, economics and space systems can be easily and satisfactorily analyzed with simulation models: increased requirement for better quality and utilization of resources, shorter time, and reduced costs, the impacts of failures, etc.

In the case of distributed event systems, simulation is also used in several applications such as the design of manufacturing systems and communication networks: evaluating different protocols and network resources, or designing road networks to deal with traffic loads. Reminding that, these systems are all highly complex systems. So, building and experimenting the real system is in fact unrealistic. Concurrently, building such systems based on rough approximations is simply too "*hazardous*". However, such systems are subject to possible use of approximations. In this case, simulation becomes a way to examine their accuracy before making any real confidence on them.

Simulation models are normally described as being either continuous or discrete (figure 4.2), where these terms involve the behaviour of system. In the continuous simulation models, the events change as time goes forward. However, in discrete-event simulation models, the time in the time-based models is assumed to go forward in fixed steps, but the number of events pet time step varies. While in the event-based models, a time step is limited to one event (each time an event occurs) [Haverkort98].

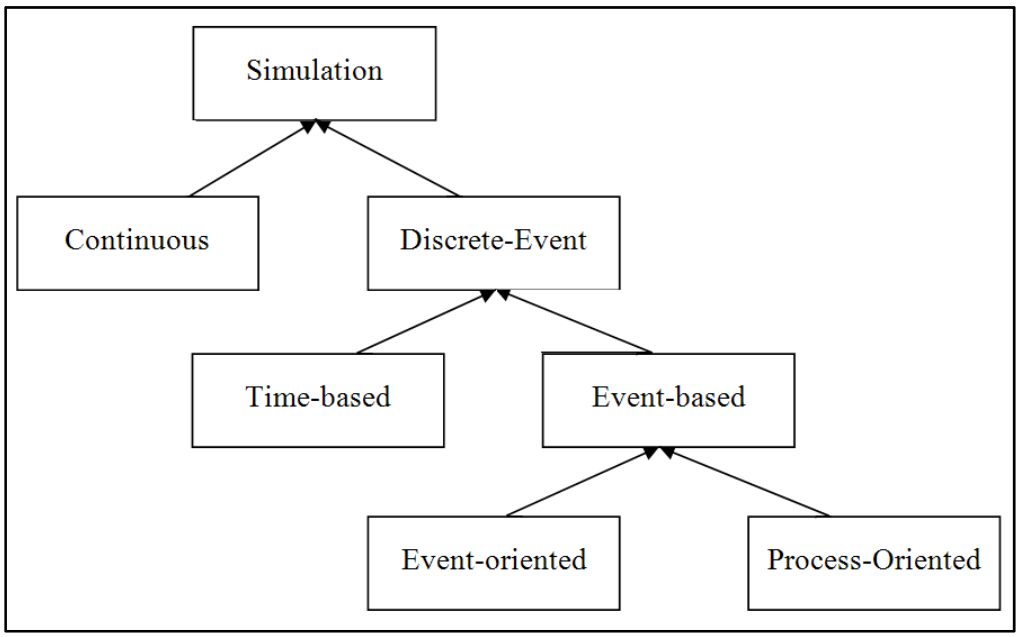


Figure 4.2, Simulations Classifying

2.2.1 CONTINUOUS EVENT SIMULATION

Continuous event simulation [Alony07] [Haverkort98] copes with the modeling of physical events / states such as processes, behaviours, or conditions, which can be expressed by a set of continuous variables changing with time. In a continuous-event simulation, simulated time advances regularly by some fixed increment and at each simulated time the simulation checks if a change happens at that time. Generally, these physical events can be described by systems of differential equations with boundary conditions. In continuous systems, time is a continuous parameter, but the system can be observed at fixed time instances only, producing a discrete time parameter. Continuous-event simulation is widely used in chemical, gas and fluid systems studies.

2.2.2 DISCRETE EVENT SIMULATION

More suitable for our aims are the discrete-event simulations [Cassandras08] [Schriber05]. Discrete-event simulation is a powerful computing technique for understanding the behaviour of systems. In discrete-event simulations, the state changes occur at discrete steps in time. Discrete event simulation mainly used in more real environments such as communication networks and protocols [Lim08] [Weber07] [Kumar07] [Carothers06], manufacturing systems [Hong08] [Kumar09], material handling [Giordano06] [Gan06], etc. General purpose programming languages like C/C++ and Java and several simulators are based on the discrete event simulation such as NS-2 [NS208], OPNET [OPNET09], OMNet++ [OMNet09], IBM Cell SDK [IBM09], P2PSim [P2PSim05] and many other tools.

The *state or state variable changes* are also called *events*. In such simulation, events take place one-by-one discretely in time. Distributed systems consist of objects and components that exchange variables and information data between them. These variables and data are subject to actions and conditions coming from the communication channels and/or the components interfaces. Such conditions or events may be in many forms such as the arrival and departure times, waiting time, stop points, service variables and handling and so on. The resulting of these events over the variables and data attributes provides information on how to deal with them.

In communication networks, the arrival of packets only makes a workstation starts a receiving process event. If that packet does not belong to it, the workstation drops the packet and stops that service. Thus, the sensing of the channel (event) and the internal attributes (another event) were responsible in starting and ending a service. In addition these starting and ending of events may affect other events to occur and to be simulated.

However, the modelling of such events needs a good definition, development and scheduling of these events. The needs and constraints, all the activities that could be performed, and the interaction between all the system-model elements and components must be taken into account and clearly be present in the model. Otherwise the resulting and measures of the simulation would not reflect the desired quantitative values attended

from realistic simulation. Our approach is made for the discrete event systems, so, a discrete event simulation could be made based on the event and attribute changes. This leads to give some characteristics for the discrete event simulation [Fishman01]:

1- It enables the organization of theoretical observations about a system,
2- It improves the understanding of a system,
3- The speed to achieve the desired analysis,
4- It is less costly than real systems measurements.

However, the main disadvantage of discrete event simulation is the occurrence of two or more events at the same time. In this case the system must choose one of these events. Here, the results may not be the same if the simulation is repeated many times, since the simulator may choose another event than the one chosen in a previous simulation.

The second disadvantage is that simulation must be repeated many times to get accurate (average) results. The detection of exceptional events during simulation is difficult since the simulator itself generates the events (depending on our modeling technique). However, this will be possible if the simulation is repeated several times. Again this will cost longer time to verify the system-model.

2.3 COMPARISON BETWEEN THE DIFFERENT METHODS

Table 4.1 shows a qualitative comparison between the different methods used to evaluate the systems performance. This comparison is based on the following criteria [Chhabra07] [Jain91]:

1- *Stage*: Which performance evaluation technique should be used at any point in life cycle,
2- *Time required*: The time consumed/required by a particular technique,
3- *Tools*: Which analytic tools, simulators, measurement packages are used,
4- *Accuracy*: It represents the degree to which the obtained results match the reality (evaluates the validity and reliability of the results obtained).
5- *Scalability*: It represents the complexity degree to scale a particular technique

6- *Trade-off evaluation*: It represents the ability of a technique to study the different system configurations.

7- *Cost*: This cost must not be considerable in term of time and money needed to perform the study.

8- *Flexibility*: The under test system-model should be easy to modify and extend. The used evaluation technique should provide the possibility to integrate these considerations easily in the developed model.

Criterion	**Analytical**	**Simulation**	**Measurement**
Stage	Any	Any	Post prototype
Time Required	Small	Medium	Varies
Tools	Analysts	Computer Languages	Instrumentation
Accuracy	Low	Moderate	Varies
Trade-off evaluation	Easy	Moderate	Difficult
Cost	Small	Medium	High
Scalability	Low	Medium	High
Flexibility	High	High	Low

Table 4.1, Comparison of the different Performance Evaluation Techniques

Simulation seems to be the mostly used technique used to evaluate the performance of the computer systems. It represents a useful means to predict the performances of a system and compare them under many conditions and configurations. One major advantage of this technique is that even if the system is already implemented, it offers flexibility difficult to reach with measurement techniques.

Our modeling formalism, Petri nets, combines both the analytical and simulation models which let the possibility to model system mathematically. However, communication networks and distributed systems are so complex that building and solving the equations' system are too difficult and needs tools capable to perform this process.

3. PERFORMANCE EVALUATION OF NETWORK PROTOCOLS

As we saw in the previous section, simulation is more appropriate to evaluate the performance in computer systems since it can give more accurate results and easier than the analytical methods. *To evaluate the quality and accuracy of our* model, we will show the simulation results of our model for the DCF IEEE 802.11b protocol part without RTS/CTS. We will show a comparison between the time needed to send packets over Ethernet and the DCF protocols. Next, to we will compare the results for IEEE 802.11b with the data given by NS-2 simulations performed in the same conditions, since it is one of the well-known and highly used simulators in the domain of communication networks. In addition, we compare them with the other studies about the IEEE 802.11b network [Anastasi05] and [Heusse03].

3.1 CHOOSING THE TOOL

Many tools and extensions of Petri nets exist such as *PNtalk, PROD, QPME, CoopnBuilder, ALPHA/Sim, Artifex* and other tools [PNW09]. However, the development of most of these tools has been stopped since long time, or they do not support our needs or these are commercial. Two main, free of charge tools were possible to cover the previous features "*CPN Tools*" [CPN07] and "*Renew 2.1.1*" [Renew08] [Kummer04].

"*CPN Tools" is a discrete-event modelling language combining Petri nets with the functional programming language Standard ML* [Jensen07] developed and maintained by the *CPN Group* at the *University of Aarhus*. CPN Tools is a tool for editing, simulating and analysing colored Petri nets with GUI graphical interface. However, during simulation, "CPN Tools" has shown an important problem that does not apply to our timing needs. Figure 4.3 shows this feature in "CPN Tools". In this example, the sender waits for an acknowledgement (a token in place "ACK?") during 10 units of time, a token put in place "Receiving ACK". However, if it does not receive an acknowledgment during this time it starts a collision procedure, T1 is fired.

However, during simulation this need never happens. The tool showed that since there is initially no token in place “Receive ACK”, then, even after the putting a token in that place, T0 would never be fired. Moreover, even if initially a token is put in place “Receive ACK” and in place “ACK?”, T1 stills a possible fired transition even T1 has timing of 10 units of time and T0 is immediate. This problem is not appropriate for our modeling methodology. When the sender receives an acknowledgement, the collision procedure is more possible. However, “CPN Tools” did not give us the possibility to model this feature.

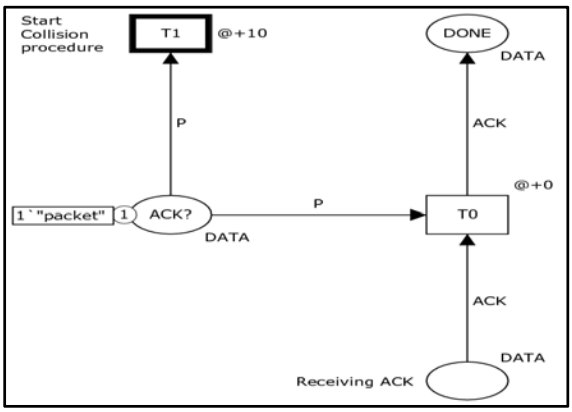


Figure 4.3, Timing in “CPN Tools”

Another possible tool was “*Renew 2.1.1*”. “Renew” is a Java-based high-level Petri net *discrete-event simulator* that provides a flexible modeling approach based on *reference nets* [Moldt03], developed and maintained at the *Theoretical Foundations Group* of the *University of Hamburg*. *Renew* combines the Petri nets formalism with the Java Object-Oriented programming language. This combination has permitted modeling all the selection criteria defined previously and more, since it allows the use of nearly all the functions offered by Java. However, in this work we did not make use of the *reference nets feature* of this tool, but we were limited to the same classical features found in “CPN Tools” for modeling and simulation the system depending on our needs.

Renew is available free of charge including the Java source code, allowing the insertion of new plug-ins. Its editor has: easy to use interface, minimal input for the user, direct relation to the functionality and provision of a high-level formalism, which

facilitates its use. Moreover, it is always up-to-date; the last version 2.1.1 is dated from July 2008. Simulations can be made with different *plugged-into-application compilers*; in our application we use the Timed Java Compiler.

As a simulation tool, "*Renew 2.1.1*" allows the *continuous* and the *step-by-step* simulations. The later permits to check the system changes one-by-one and getting better understanding of the system-model such as deadlocks, unexpected event, warnings and so on. This allows to better reconstruct or modify the system-model easily and to find the errors one-by-one for better performance measures and systems design and implementation. Once the model goes straight forward, the continuous simulation can be performed. This allows getting the desire performance measurements more rapidly. Nevertheless, the tool "Renew 2.1.1" does not have a package (or a plug-in) able to perform these analytical verifications.

3.2 SIMULATION AND RESULTS

Our simulations are based on full-mesh dense networks with different numbers of workstations:

1- The simulations were performed for different number of workstations sharing the medium.
2- For each case, the simulations were repeated 100 times to get average measures.
3- Each simulation assumes that all nodes transmit at 11Mbps
4- All the nodes attempt to send data as soon as possible.
5- Each node has 1000 packets (to get the average possible measures) with average packet length of 1150 bytes (packet length varied from 800 byte to 1500byte)
6- All simulations are accomplished on Intel® Core™ 2 Duo Processor T2300, 2G of RAM.

3.2.1 AVERAGE BANDWIDTH PER NODE

The first result is the average bandwidth per workstation. Figure 4.4 shows the throughput of 802.11b nodes sharing a bandwidth of 11Mbps. As we can see in the figure, the bandwidth per node decreases logically with the increase of nodes number. In the

figure, when the number of nodes is small each workstation has more bandwidth from the shared effective bandwidth. However, when the number of the nodes on the network increases, the bandwidth is decreasing exponentially. This is due to the increased number of collisions on the network, and so more bandwidth will be lost.

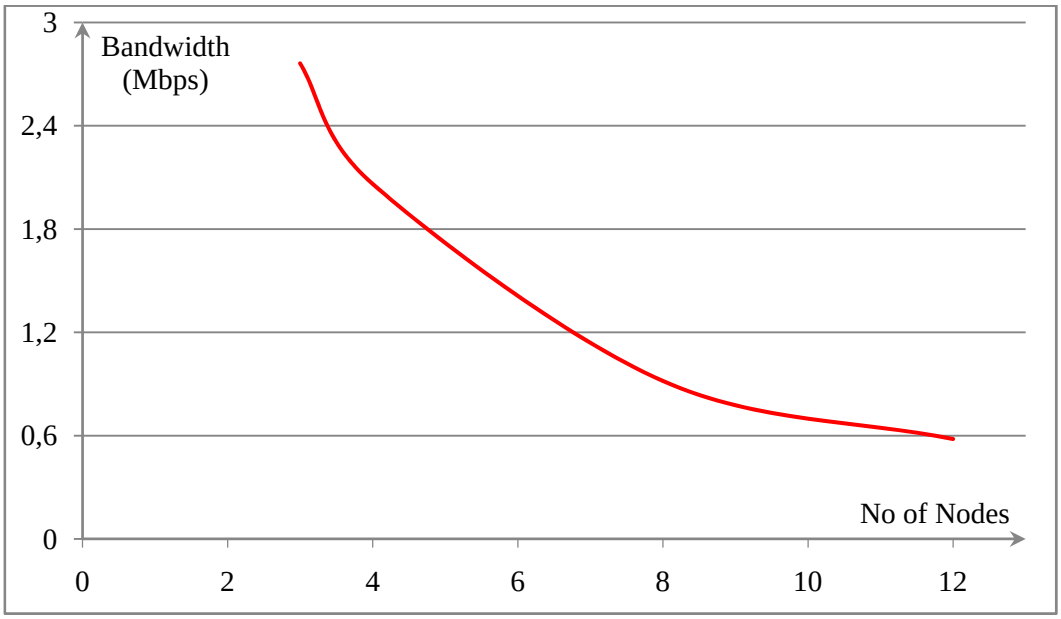


Figure 4.4, Bandwidth Variation with Number of Nodes

The other factor is that CSMA gives fair timing to the machines to access the channel. Thus, workstations must wait longer time to have access to the channel. Another factor is after a collision, the workstations must double their contention window which means longer backoff time. So, more time is spent to decrement the backoff or less total bandwidth.

3.2.2 COLLISIONS RATE PERCENTAGE

The next step is to compute the collision rate percentage or errors versus the network utilization. Figure 4.5 shows how the collision rate increases when the number of workstations increases. As we can see in the figure, when the three workstations are sharing the medium, the collision rate is nearly 8%. However, when there are 12 workstations sharing the medium, the collision rate reaches 23.2%. These results confirm the results obtained in the previous section and our explanation.

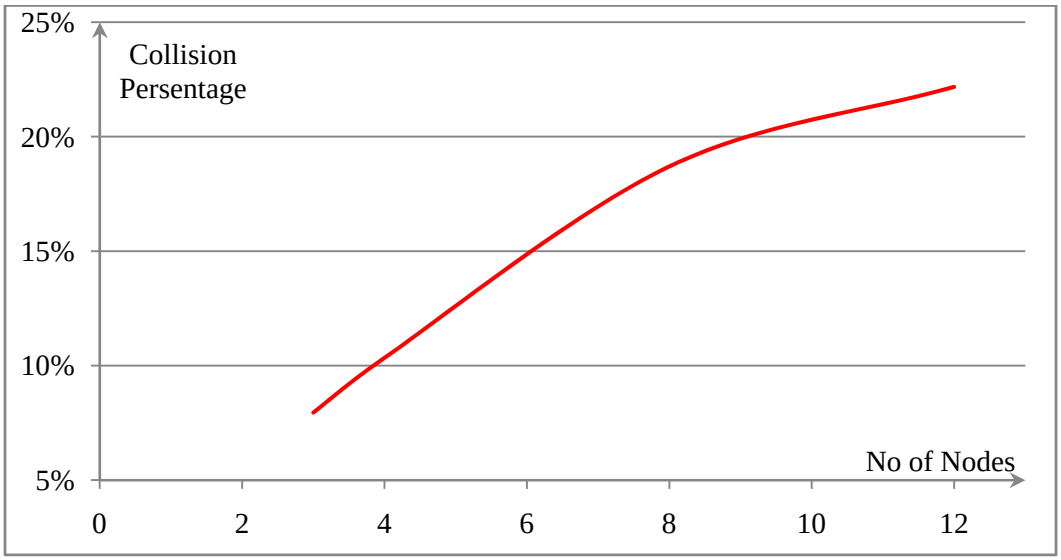


Figure 4.5, Collisions Rate Percentage

As we can see, the collision rate is increasing linearly until certain point (8 workstations). The reason is when more workstations attempt to send, more packets are on the shared channel and hence the probability that a collision occurs increases. However, when the number increases more, the collision rate increase becomes slower. The explanation for this *slowly state* is the *backoff procedure*. With more workstations, the number of collisions increases, and the value of CW also increases (backoff time). On the other hand, this increment of backoff time decreases the probability of a collision, since workstations in collision must wait for longer time before attempting to send again. So, the collision rate increment becomes slower.

3.2.3 TRANSMISSION TIME PER PACKET

The next test is to measure the overall time needed to send a packet over Ethernet or DCF protocols (from sender side to receiver side). Figure 4.6 shows the time required to transmit one packet versus the number of nodes on the network. The transmission time increases linearly due to the increased number of sent packets on the network and collision rate.

However, sending a packet over Ethernet requires less time than sending it over DCF. The figure shows that with three nodes on the network, DCF seems to be the same as

Ethernet. However, with the increase of nodes the difference becomes obvious. This is due to:

1- A workstation attempting to use the channel in wireless networks needs to ensure that the channel is idle during a DIFS period or 50µs, while in Ethernet it only needs 9.6 µs.
2- From the first attempt to transmit, wireless nodes starts a backoff procedure (B_{avg} = 8 * 20 µs) decremented only if the channel is idle, while in Ethernet, workstations defers only for 9.6 µs.
3- After a collision, in wireless networks, the channel status becomes idle only when all the workstations finish their transmissions (no collision detection process), while in Ethernet the channel becomes idle after 51.2 µs (channel acquisition slot time).
4- The backoff procedure used after each collision (Chapter 3, Section 4.1.3 and figures 3.13 and 3.15) in wireless networks doubles the contention window value which is already 8 times greater than the one used in Ethernet. This makes the backoff in wireless greater than Ethernet BEB even with slot time (20µs) less than the 51.2 µs used in Ethernet.

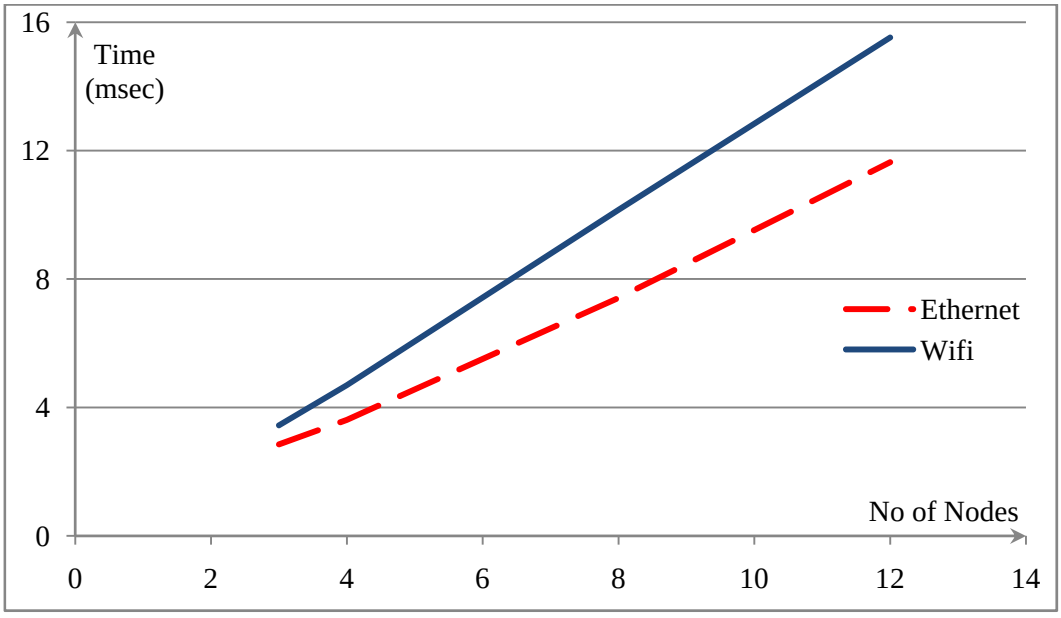


Figure 4.6, Transmission Time per Packet

Table 4.2 shows the collision rate, the bandwidth per node, the time needed to send a packet and the total effective bandwidth. The total effective bandwidth decreases with the increased number of workstations on the medium and the increment of collision rate.

No of Nodes	Collision Rate	BW/Node	Time/Packet	Total Effective BW
3	7,95%	2.76 Mbps	3.541 ms	8.29 Mbps
4	10,34%	2.06 Mbps	4.694 ms	8.25 Mbps
8	18,70%	0.92 Mbps	10.15 ms	7.34 Mbps
12	23,18%	0.58 Mbps	15.52 ms	6.97 Mbps

Table 4.2, Different simulation results of 802.11b DCF Protocol

3.3 COMPARISON WITH NS-2 SIMULATOR AND OTHER STUDIES

To evaluate the quality and accuracy of our model, we have used the network simulator NS-2 as a helping tool since it is widely used to model communication protocols. The NS-2 simulator is a *discrete-event network simulator* that allows simulating many scenarios defined by the user. It is commonly used in the research due of its *extensibility*, since it is an open source model. NS2 is widely used in the simulation of routing and multicast protocols and ad-hoc network. The network can be represented / modeled by *traffic sources, protocols, routers* and *links that connect them.*

The source code of NS-2 is written in C++ language for the internal functioning of the network components (core engine) and O-TCL scripts, which is the object-oriented version of TCL language, for the configuration and simulation scripts. The modeling of a network consists of different elements:

- *Nodes*: They correspond to workstations (traffic generation) or routers.
- *Communication Links*: They model the physical connection between two nodes.
- *Agents of communication*: They represent the transport layer protocols (TCP and UDP). To establish communication between two nodes, we must attach an agent on each node, and connect them so that the communication can take place.

- *Applications*: They are responsible for generating traffic (file transfer, random traffic, etc.) and they use the communication agents.

Each element in NS-2 model can be adapted to the users' needs: the packet size, bit rate, maximum number of packet, etc.

To carry out our comparison, the NS-2 model simulations were performed in the same conditions (a dense network and the workstations are in the same domain) and on the same computers. Figure 4.7 shows a sample declaration of a node with packet size of 1150 bytes and 1000 packets for each node

```
set udp(1) [new Agent/UDP]
$udp(1) set prio_ 1
set null01 [new Agent/LossMonitor]
$ns_ attach-agent $node_(0) $udp(1)
$ns_ connect $udp(1) $null01
set cbr(0) [new Application/Traffic/CBR]
$cbr(0) set packetSize_ 1150 ;
$cbr(0) set rate_ 1Mb
$cbr(0) set maxpkts_ 1000
$cbr(0) attach-agent $udp($i)
```

Figure 4.7, Sample code of NS-2 Model

Figure 4.8 shows the results obtained from NS-2 and those from our model, (figure 4.3). As we can see the results of both simulations Renew and NS-2, are nearly identical which confirms the correctness of our model. Moreover, if we compare our obtained results with those in [Anastasi05] and [Heusse03], we can get also the same results from both the simulation technique and the equation we obtained from the results.

The other comparison is the *effective simulation time*. As we can see in figure 4.9, the simulation time increases in a linear way when the number of nodes increases (confirmed by the results in Figure 4.6). The figure shows that NS2 needs less time to perform the same simulation. However, NS2 does not support the step-by-step simulation to verify the system event by event. The second important issue is that it is not possible to

model distributed services with NS2 (no supporting package). However, with “Renew” as Petri nets editor and simulator, it is possible to combine both services and protocols in one global model.

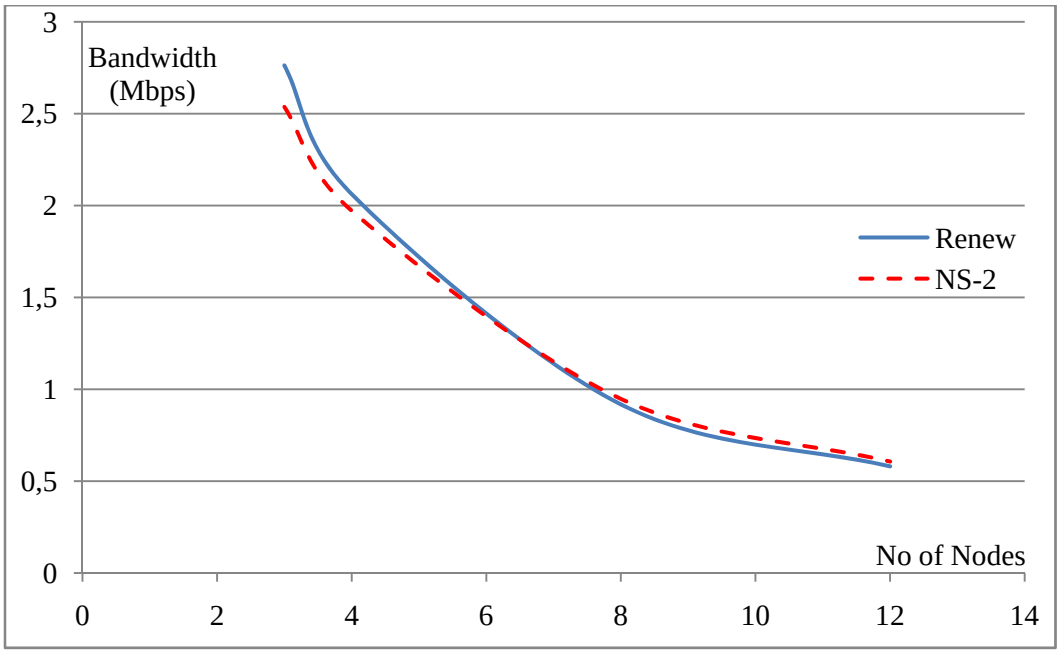


Figure 4.8, Comparison between Our model and NS-2

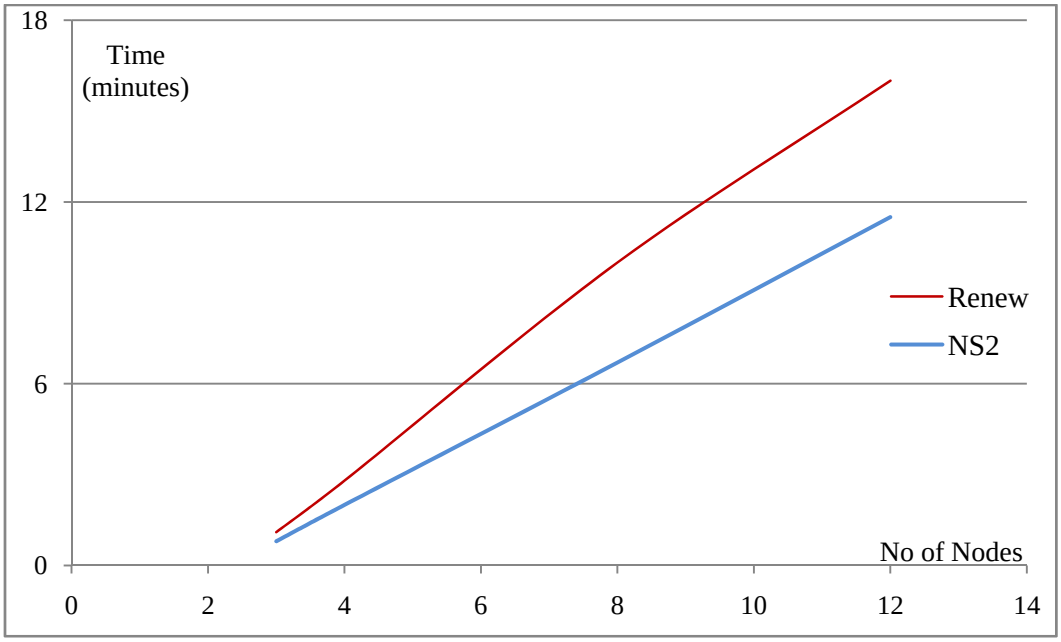


Figure 4.9, Effective Simulation Time versus number of nodes

4. A CASE STUDY: EVALUATING PERFORMANCE OF A DISTRIBUTED MANUFACTURING SYSTEM

In the last sections, we have shown the modeling part of the communication protocols. In this section we will show the modeling part that concerns the services. The illustrative example used in Chapter 1, figure 4.10, will be reused to model the services offered by a production system. The used modeling technique will be the same as the communication protocols, i.e. component-based methodology, where each part of the system is modeled in hierarchical composition: "*service-workstation*", i.e. each service is modeled over a workstation.

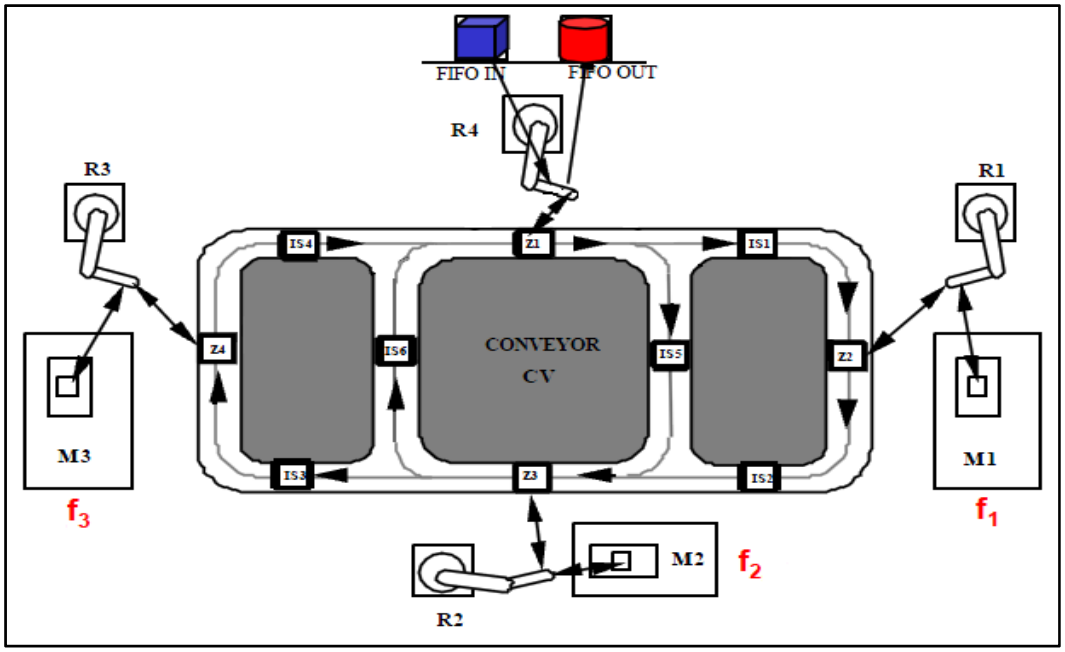


Figure 4.10, Manufacturing Plant with Flexibilities

4.1 ANALYZING THE SYSTEM

In *chapter 1*, we have shown the technique used to model the system, figures 1.10-13, which is based on transforming each place and transition to a complete Petri net figure 1.11. In this chapter, we will not make use of the intermediate model step and we will distribute the services over different workstations (no centralized control).

4.1.1 SYSTEM COMPONENTS

Figure 4.11 shows the complete system components used to transfer one product from *IN/OUT* area to any machine *M1*, *M2* or *M3*. *Z1* to *Z4* represent the input and output areas for each machine and the IN/OUT area.

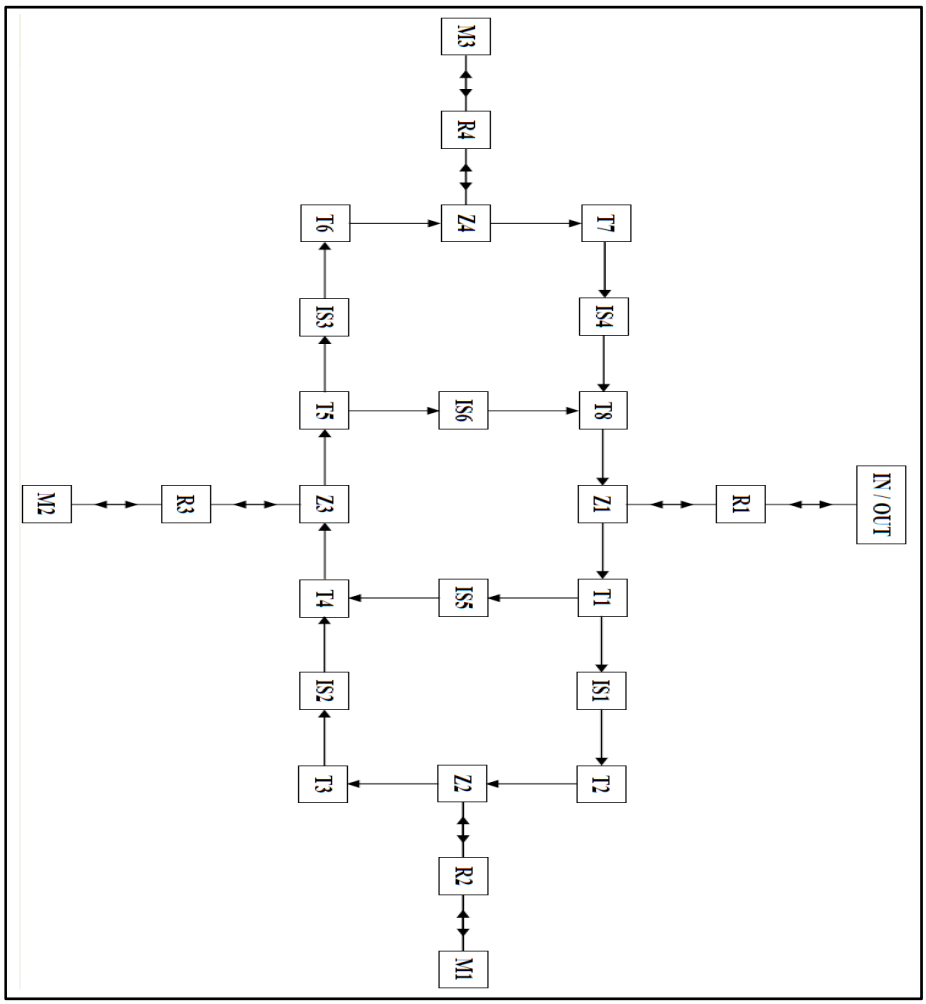


Figure 4.11, Complete Area and Transfer Components

The capacity of each is limited to one product. *IS1* to *IS6* areas represent the stock area before and after machining a product. The capacity of each stock area is greater than one. R1 to R4 represent the robots used to make a transfer from a Z area to a machine or IN/OUT area and vice versa. Finally, *T1* to *T8* represent the transfer elements from and to a Z and an IS stock areas.

The component represents the service offered by a resource, a robot or a transfer component. Machines, Z's and stock areas are considered as shared resources. The use of each of them needs a pre-allocation (chapter 1, section 3) made by a transfer component T_i or R_i. Each component-service is modeled as a Petri net over the workstation component model (Chapter 3) (workstation per service). In order to allocate a resource or machine, the request of a service is made by messages exchange over the network via the underlying workstations.

4.1.2 EXCHANGED MESSAGES BETWEEN COMPONENTS

In order to transfer a product from one area to another one, areas must allocate the required area/resource. Pre-allocation is passed through a transfer component. Transfer components check the possibility to allocate the destination area (depending on the capacity of each area). An acknowledgement is from the destination area when a place is free. During this time the source area and the transfer component are in waiting period (machines and Z areas do not perform any action during this time, while stock areas can receive products from other components).

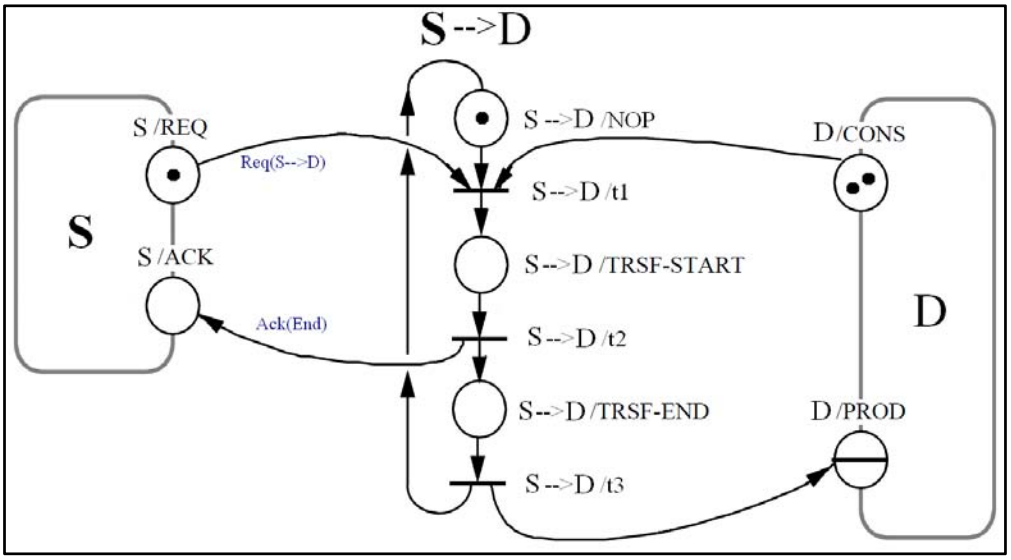


Figure 4.12, Product Transfer in Petri Nets

Figure 4.12 shows the centralized model of a product transfer from S to D areas. In the figure, to transfer a product, the product must be available in area S (a token put in

place S/REQ), the transfer component must be also available (a token in place S→D/NOP) and a free place in area D (a token in place D/CONS). These three tokens enable the transition S→D/t1 and a token is then put in place S→D/TRSF-START starting the transfer process. The transfer component takes the product from area S. the firing of transition S→D/t2 and the put of a token in place S/ACK inform that a place is released up in area S. The transfer process continues by putting a token in place S→D/TRSF-END. When the product arrives to area D (transition S→D/t3), the transfer component becomes free again (a token is put in place S→D/NOP) and an area is used in area D (a token is put in place D/PROD).

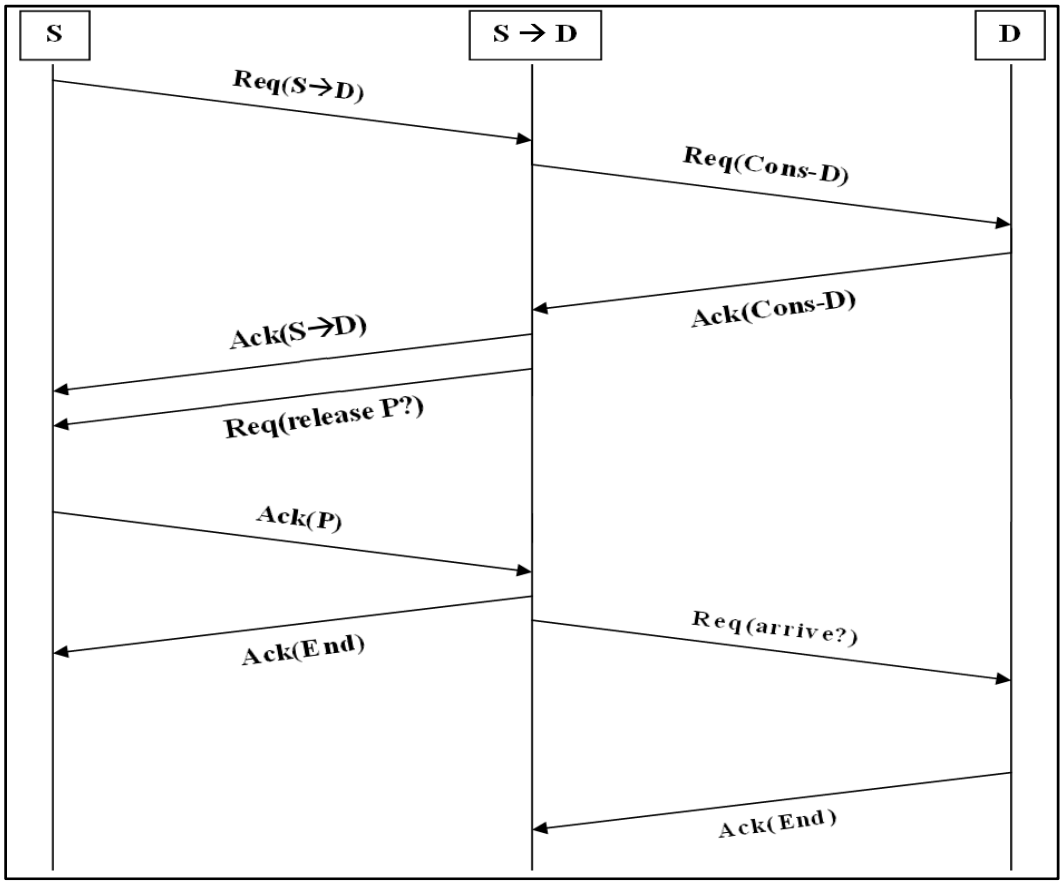


Figure 4.13, Exchanged Messages over the Network for the Transfer

Figure 4.13 shows the complete messages exchanged in case of implementation of the 3 processes (S, S→D, and D) in 3 different computers. Each process plays a different role with regard to the client/server mechanism. S is always a client and D is always a

server. The role of S→D varies depending on the message. At first, the source area *S* (workstation) sends a request message to the transfer workstation, *S→D* (T_i or R_i), containing the destination workstation *D*. T_i (or R_i) sends a request to D, requesting a free place (*Cons-D*). If there is a free place, *D* will send a positive acknowledgment to T_i (or R_i), otherwise S and T_i (or R_i) will stay in a waiting period.

Once T_i (or R_i) receives the acknowledgment, it sends two messages to S containing a positive acknowledgment and a request to release the product. When the product is released S sends an acknowledgment to T_i (or R_i) to start the transfer. When T_i (or R_i) takes the product, it sends an end message to S to free one its places (Cons-S). Finally, it sends a message to D asking the arrival of the product to its side. Once the product arrives to D, it sends an acknowledgement to T_i (or R_i) informing the end of the transfer.

4.2 MODELING THE COMPONENTS

The system contains two types of components: transfer or area components. A component is modeled by a workstation module and a service module. In this section we will show the Petri net module used to model the service of each component. In order to differentiate between the different workstations on the network, we used an addressing system for each workstation depending on its type. In this section we will model the physical transfer of a product represented be the dark gray interfaces of the service components. The transferred token between the different services represents the product. To make the transfer the message exchange procedure must be made before any transfer.

4.2.1 AREA COMPONENTS

When a Z, an IS or a machine receives a product (represented by a token), it starts a transfer procedure to transmit the product to its final destination. This token contains two fields: the final destination *FD* and the type of the product *ID* (will be used later to arrange the products after their arrival to the OUT area).

An area sends and receives messages depending on whether it is the source or the destination. Figure 4.14 shows the part concerning a source area (example of IS1: address

31). The gray places belong to the workstation itself, while the white ones model the service. When the area receives a product, a token put in place “From another Area”, it sends a transfer request to T2 (address 42) with the target area Z2 (address 22). The type of the product and its ID is contained in the token fields (in the example FD=12 and ID=1 representing a service on machine M1). The product is kept in place “Wait” and it is realised only when the area receives an end acknowledgement (the guard on transition “t11”). A token is put in place Cons of that area.

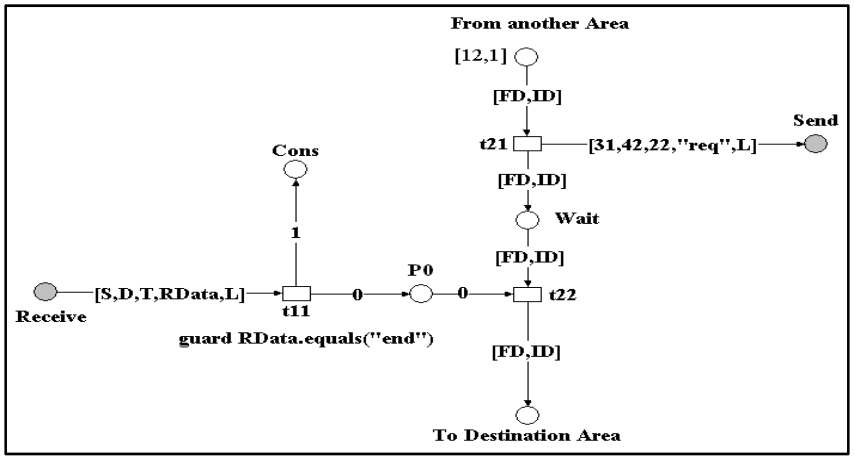


Figure 4.14, Part: Source Area

If the area is a machine, a place and a transition are added between place “From another Area” and transition “t21” to represent the machining process, figure 4.15. *MTime(ID)* represents the time needed to machine a product before the token becomes available.

Figure 4.15, Machining Time

Figure 4.16 shows the other messages sent and received by an area in order to transfer a product. If the area is a source area, it receives two messages: the end message (transition t11, figure 4.14) and the request message to release the product (transition 51). It then sends an acknowledgment to the T_i (or R_i) (in our example, to T2, and the value “99” means not used field).

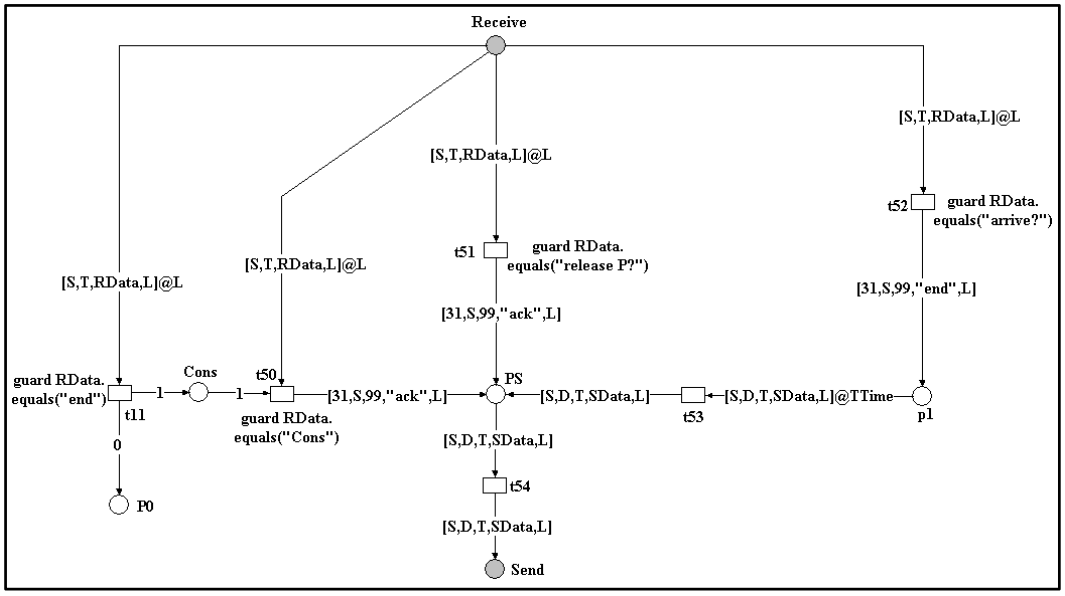


Figure 4.16, Messages Exchange for a Transfer from an Area

However, if the area is the destination, it receives two messages: a message to allocate a free place (transition t50) and a request message for a successful product reception (transition t52). The capacity of an area (the number of tokens in place Cons) decides whether this request is possible by send a message to the source, or the source must wait until a place is released up. The inscription *TTime* at the arc connecting place p1 and transition t53 models the time need to receive a product from one area to another.

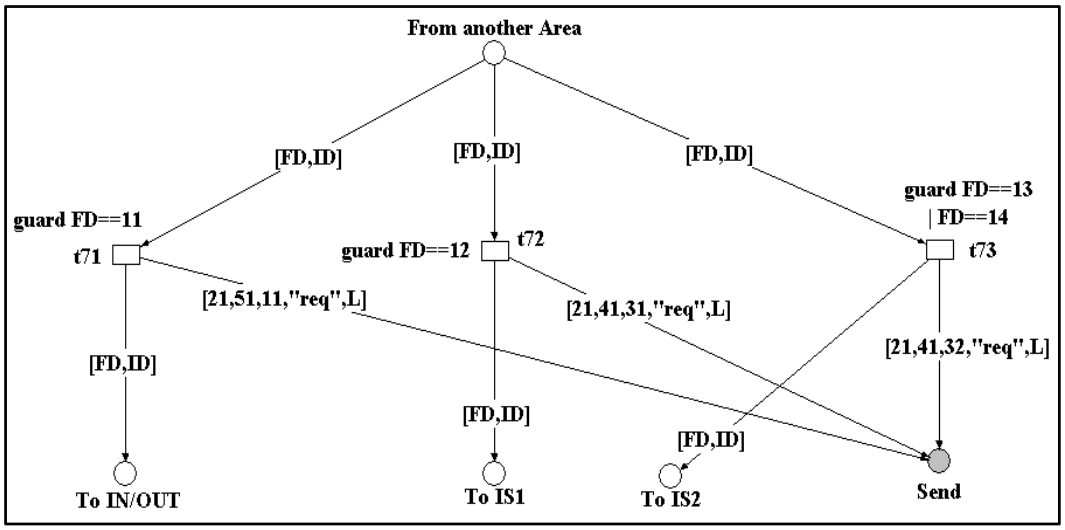


Figure 4.17, Products Routing

Another characteristic added to such components is routing the product to the correct direction. All the Z areas have this feature. Figure 4.17 shows the routing of a product in Z1. The products found in that place need either to be machined, depending on their final destination (to machine M1: transition t72, or to machines M2 and M3: transition t73) or to be forward them to OUT area (transition t71). To forward a product to area OUT, it must be before machined. This condition is satisfied by a machine by changing the field FD to 11. Hence, to rearrange the products (in OUT area) the field ID remains the same.

4.2.2 TRANSFER COMPONENT

A transfer component is used to perform a transfer between two areas. Figure 4.18 shows the complete messages sent and received by a transfer component. The component receives a request packet from an area (transition t60). In order to validate this request a token must be present in place “Cons”, representing the capacity of this component. It sends a message to the destination area requesting a free place. The component stays in a waiting period (place p60) until it receives acknowledgment packet from the destination area (transition t62).

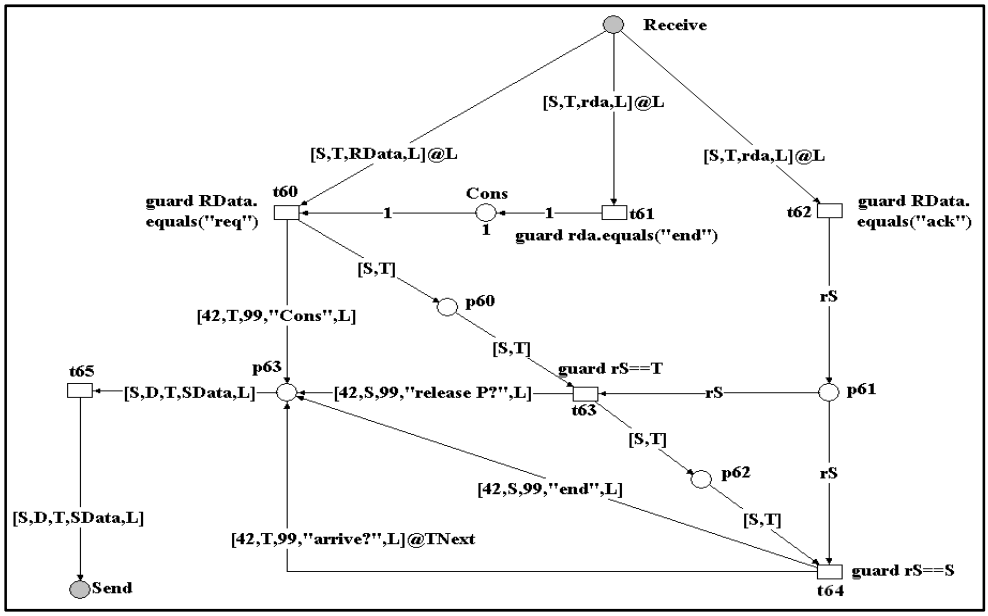


Figure 4.18, Transfer Component – Service Part

Once the acknowledgment arrives, the transfer component sends request ("release P?") to the source area. To insure an excellent functionality of this module, a guard is associated to transition t63 to assure that the sender of the acknowledgment is the destination area. Again, the component stays another time in waiting period (place p62) until it receives the second acknowledgment.

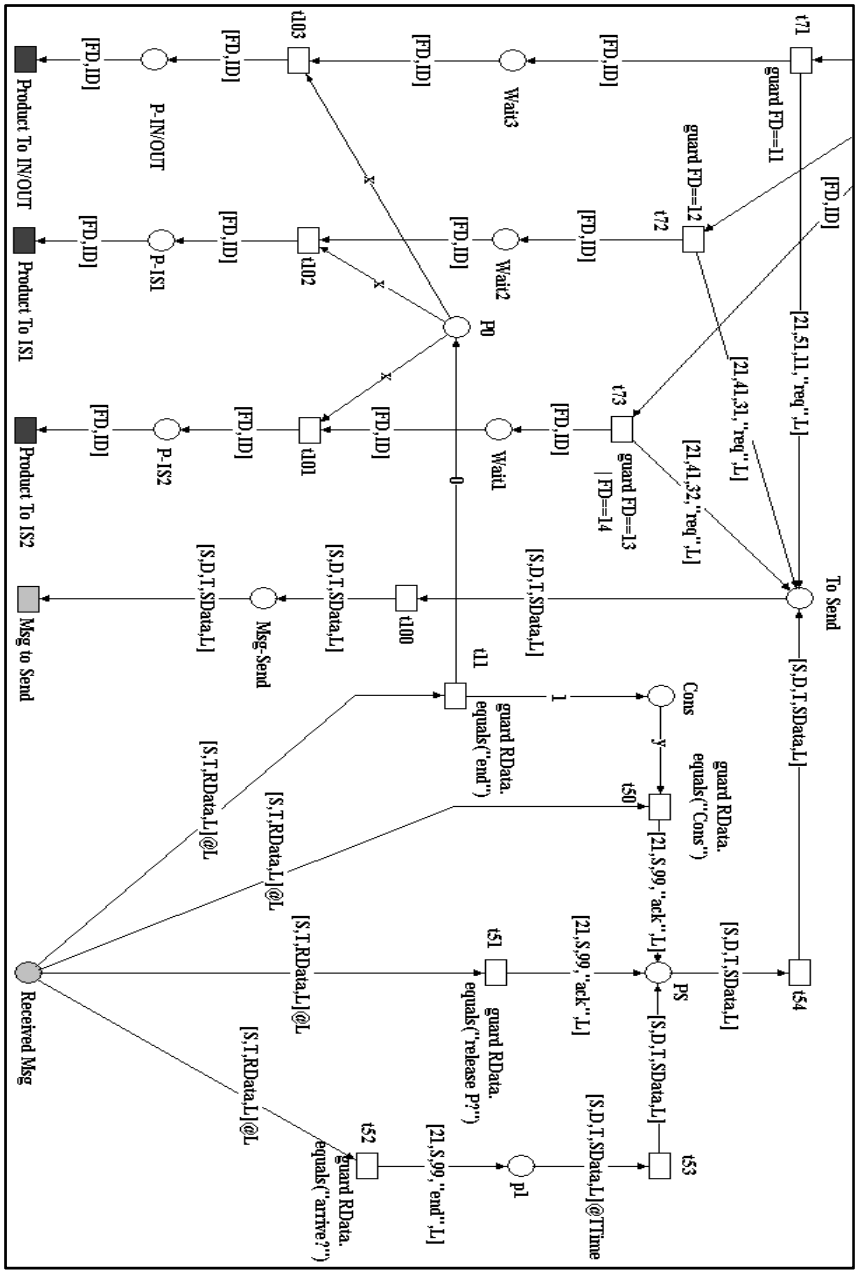


Figure 4.19, Hierarchical Service-Workstation Petri Net

When the acknowledgment arrives, transition t64 is enabled (condition: the sender must be the source), two messages are sent: to the sender releasing one place (the product is taken by the transfer component), and to the destination area requiring if the product has arrived. This second message is sent when the first message is sent (the *TNext* inscription on the arc between t64 and p63).

Figure 4.19 shows a complete Petri Net model for Z1 area. While figure 4.20 show the service component with the different interfaces. In the figures, the when a product arrives to the area, a procedure of exchanged messages starts depending on the destination area until that product is transferred to its final destination. Hence, the workstation component is connection of the service and network (light gray interfaces), while the output interfaces of the service component (dark gray) is connected directly to their destination components.

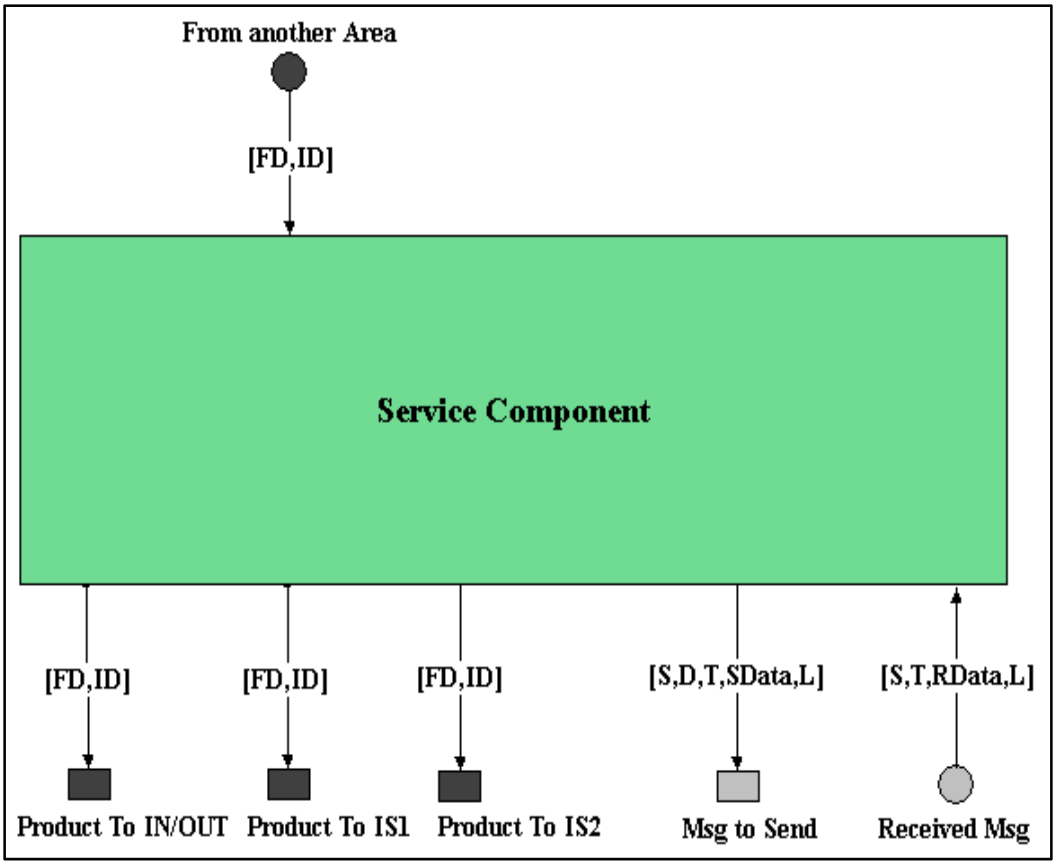


Figure 4.20, Service-Workstation Composite-Component

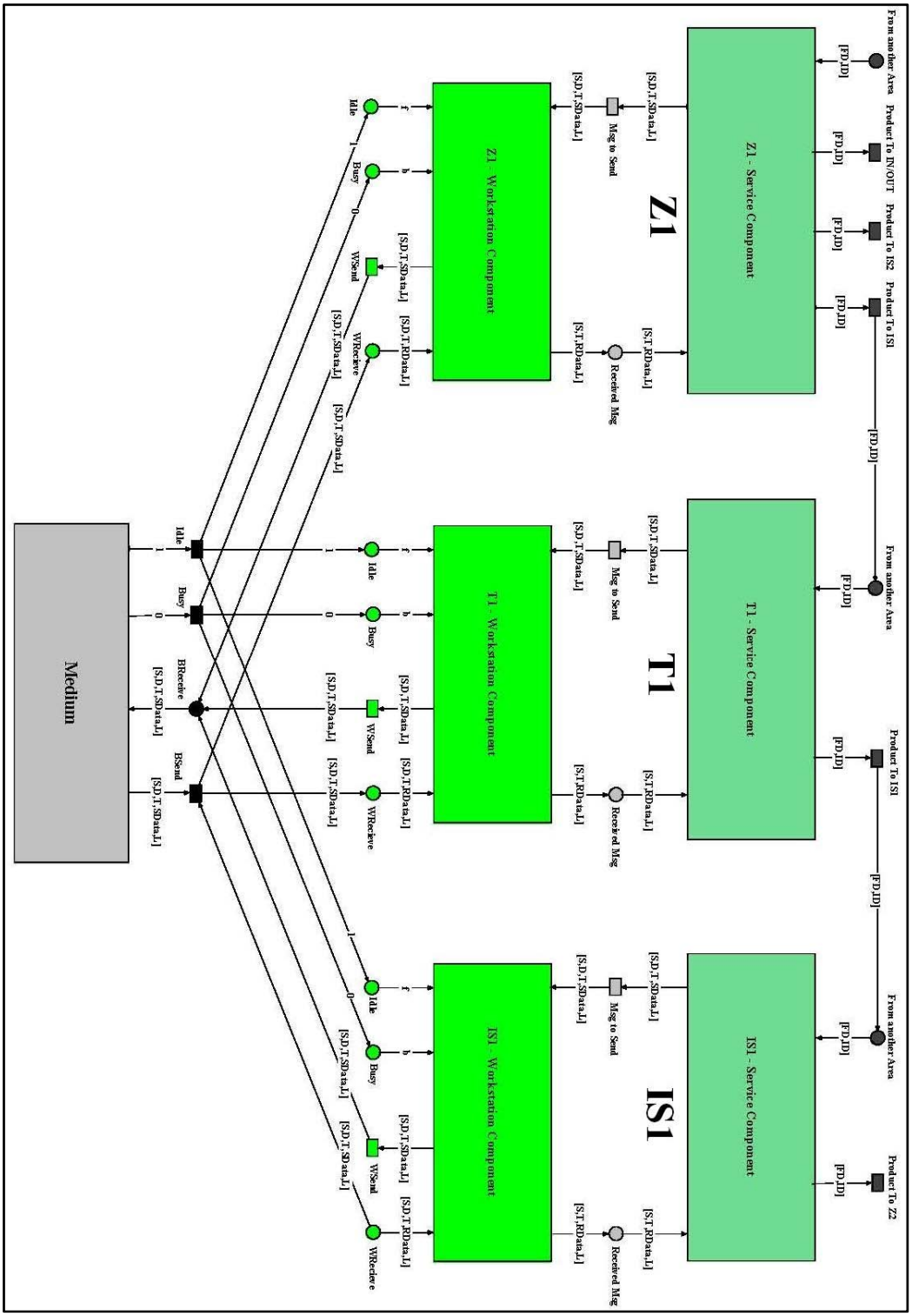


Figure 4.21, Sub-model for a transfer from Z1 to IS1

Figure 4.21, shows a sub-model for a transfer of a product from area Z1 to stock IS1 through the transfer element T1. In the figure, the modeled components are reused to build the whole system. This reuse is applicable for the other elements in the system. Some additional places and transitions are used to complete the system-model and to insure its functionality, table 4.3.

Type	No of Transitions	No of Places
Area Component	30	25
Transfer Component	24	21
Complete Model	748	652

Table 4.3, Petri Nets Complexity

4.3 SIMULATION AND RESULTS

As in section 3, the system is evaluated by simulation. The simulation was performed on the same PCs used in section 3. The system is assumed to perform 100 different products. The simulation aims to see the impact of using different type of products and different protocols over the system. The transfer time is supposed to be 50 msec and the machining time to be 100 msec. These values have been chosen in milliseconds to really verify the impact of the underlying network on the system. Otherwise, if we use the real values in minutes, the impact of the underlying network would not be obvious with the example we have used. The number of simultaneous products per type is varied from 2 to 5 products.

Each machine performs one function. Once the product is machined, the field *FD* is changed to "11". However, to accomplish a service different paths are used for each machine:

To perform f2: IN/OUT → Z1 → IS5 → Z3 → M2 → Z3 → IS6 → Z1 → IN/OUT

To perform f1: IN/OUT → Z1 → IS1 → Z2 → M1 → Z2 → IS2 → Z3 → IS6 → Z1 → IN/OUT

To perform f3: IN/OUT → Z1 → IS5 → Z3 → IS3 → Z4 → M3 → Z4 → IS4 → Z1 → IN/OUT

The type of services on the system affects the number of exchanged messages and transactions on the network. For example, to perform the service f2, the number of transactions is 72 exchanged messages per product. However, to complete service f1 or f2, the number of exchanged messages is 90 messages per product. This is in the case of one product only on the system. However, when there are several products on the system, this number increases due to collisions. So, this number may reach 90~100 messages per product for service f2, and 110~120 messages per product for service f1 or f3.

4.3.1 ONE PRODUCT

The first simulation is to get an idea about the time needed to machine one product over the system. Table 4.4 shows the impact of changing the communication protocol in the system over the time needed to finish one product. An important difference appears between Ethernet at 10Mbps and 100Mbps. However, the 1Gbps does not create a big difference, since the machining and transfer times are the dominant in this case.

Service	802.11b	E-10Mbps	E-100Mbps	E-1Gbps
f2	564.5 ms	567.6 ms	506.7 ms	500.7 ms
f1 or f3	680.2 ms	684.5 ms	608.5 ms	600.9 ms

Table 4.4, Time to Machine a Product

The other interesting result is the time difference when the required service is f2, or f1 or f3. Since the path to finish the product is longer, the time needed to make the product is clearly longer. In this part, 11M 802.11b *seems to be* better than 10Mbps Ethernet.

4.3.2 DIFFERENT PRODUCTS, SAME PROTOCOL

The next results concern the impact of changing the type of products with keeping the same communication protocol. Figure 4.22 shows the time needed to make one product (100 products over 10Mbps Ethernet on the system). Here, the number of products to machines 1 and 3 are always equal.

The first remark that we can get is the important time difference when one product is made over the system, and when 6, 9 or 15 simultaneous products exist at the system.

The second remark is the effect of changing the number of products to machine 2 (the overall number of products is always 100). Noting that:

1- From 10 to ~30, the time needed was decreasing. This is due to that the congestion on Z3 decreased, since M1, M2 and M3 uses it to transfer the products from and to IN/OUT.

2- On the other hand, when the number of products addressed to machine M2 increased (more than 30 products), the time increased since the number of products fabricated is 1 after M1 and M3 finish their products (M1 and M3 each has less than 30 products to fabricate). As a result, the machining and transfer times become more important.

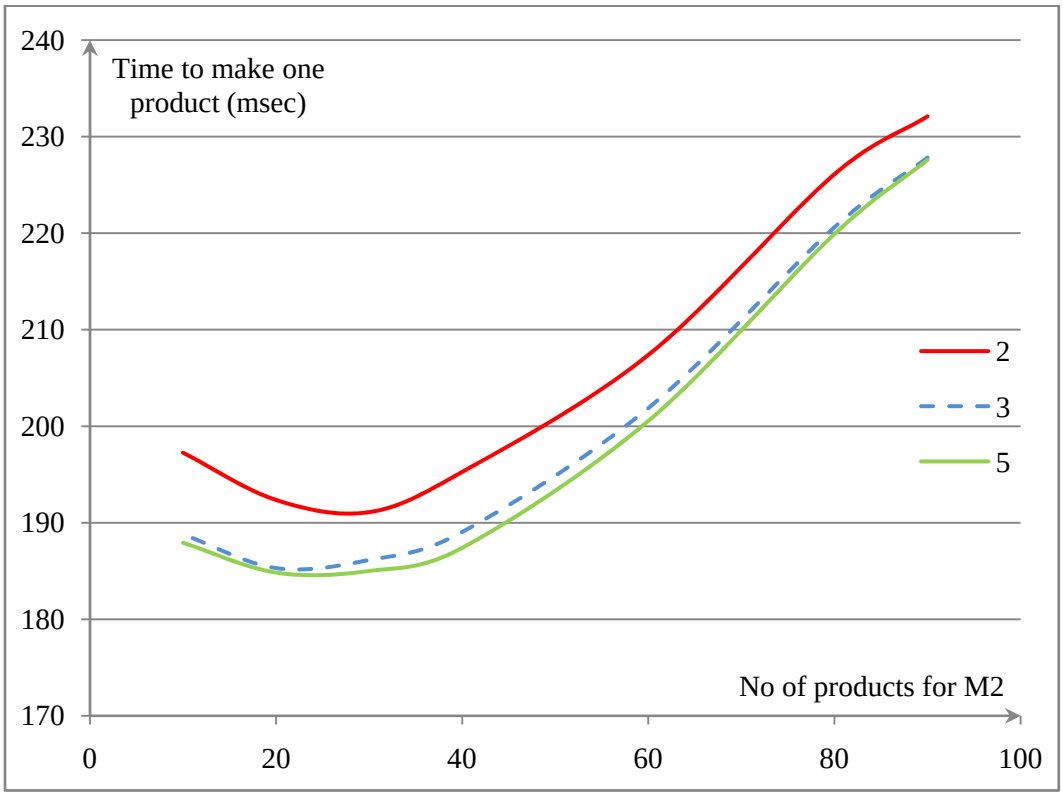


Figure 4.22, Time per Product

The figure shows also the impact of changing the number of simultaneous products attended to each machine. Increasing the number of products decreases the time needed to fabricate a product. This is explained since the more products are transferred at

the same time. However, entering 3 or 5 five products does not decrease the time. This is because the machines can fabricate one product only. So, entering more products will not affect the time.

4.3.3 Same Products; Different Protocols

The last results are the most important, since it shows the impact of changing the communication protocol over the system. Different remarks are found from the figure 4.23:

1- 802.11b protocol does not present a good choice. This result is conforming with the results of figure 4.6. This becomes clear when the number of simultaneous products increases (the number of exchanged messages increase also).

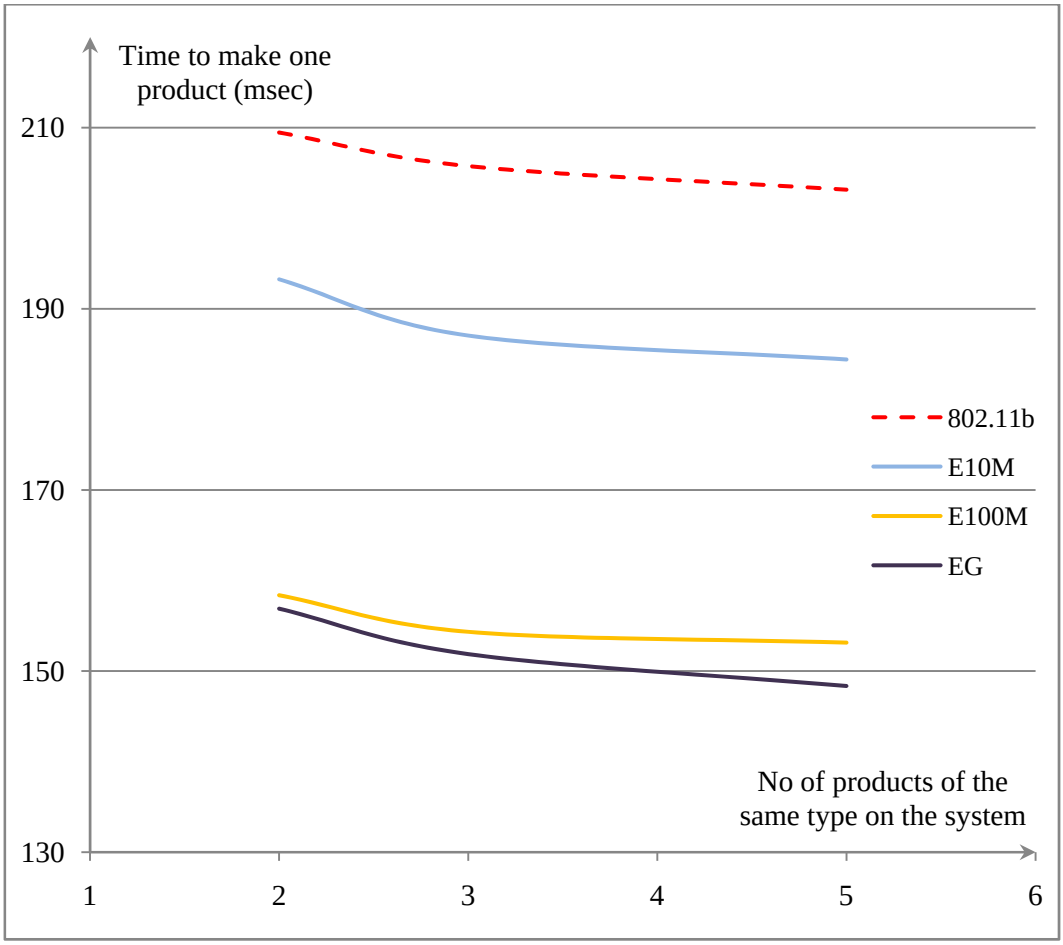


Figure 4.23, Impact of changing the communication protocol in the system

2- A big time difference is noticed when using 100Mbps Ethernet (compared to 10Mbps Ethernet and 802.11b). The number of messages is important. With 2 simultaneous products of each type, the number of exchanged messages reaches 500 to 600 exchanged messages. With 3 simultaneous products of each type, the number of exchanged messages reaches 900 to 1000 exchanged messages. While with 5 simultaneous products of each type, there are nearly 1400 to 1500 exchanged messages on the network.

   The type and speed of protocols is very important since to exchange this huge number of messages on the network, the bit rate is very important and decreases obviously the time needed to exchange these messages between the different resource/workstation on the system.

3- The use of 1Gbps Ethernet did not show a big difference with respect to 100Mbps Ethernet. However, this conclusion is not really correct. The impact of using Giga Ethernet can appear if the modeled system is larger (more machines, stock areas, resources, etc.).

   In that case, the number of exchanged messages over the network will be greater. Thus, the impact of using Giga Ethernet will become obvious since the time needed to send these messages will be shorter (for example, as the time difference between 10 and 100Mbps).

   However, in our model the number of modeled components is still medium (3 machines, 4 resource areas and 6 stock areas). So, the machining and transfer times are dominant here when using Giga Ethernet compared to 100Mbps Ethernet.

5. CONCLUSION

Performance evaluation is very important for the system development and configuration. Many measurements can be performed to analyze the performance of the system depending on the system itself. Simulation is the one of the most suitable technique for evaluating distributed systems because of its complexity.

In this chapter we have shown the results obtained from simulating the system presented in chapter 3. The tool "Renew 2.1.1" allows modeling and simulating the system. However, the choose for this tool is just to perform our needs, and can replaced by any tool that has more performance, especially for analyzing the properties of the basic and composite components.

The simulation results show the degree of accuracy, correctness and quality of our approach. The comparison with other results proves these features. We have also showed a complete service-workstation model for the manufacturing system presented in Chapter 1. The obtained results are very interesting and show how temporal features of the distributed applications are affected when the underlying network protocol and service is changed especially between 10M Ethernet and both 100M Ethernet and wireless protocol.

Thus, the obtained results are very encouraging and motivate to continue modeling new protocols (not only MAC layer but also from the upper layers: Network and Transport) and distributed systems such as Data Base Systems and transportation Systems. Our component-based modeling approach shows a high quality and easiness in modeling complex systems where another editors and simulators do not allow to model services and communication protocols in a single model and in a distributed form.

CONCLUSION AND PERSPECTIVES

1. CONCLUSION

Distributed systems are more and more present in our daily life. These systems are complex and can be distributed in one place or even everywhere in the world. The use of distributed allows sharing different and expensive resources by a several clients. Thus, the need to control the distributed systems is very important. New technologies play a big role to in the control process. These technologies underlie the distributed systems and are used to exchange messages between the different parts of the system.

Manufacturing systems are one kind of these systems. They are a subclass of the discrete event systems DES. The need to model these systems before their implementation is important. The design stage allows verifying some of their properties. A *well-designed* model that takes into accounts all the requirements and constraints of a system can save cost and time.

In this work, we have presented the problem of modeling manufacturing systems and the underlying communication protocols. However, modeling a huge and complex system implies to have also a big and complex model. So, we have proposed in this thesis a component-based modeling approach based on High-Level Petri Nets. This approach can meet the challenges of modeling the distributed systems and the communication networks. Genericity, modularity and reusability are the main and important characteristics of this approach since allows *reusing ready-to-use* components and easily fitting them to new system-models depending on the requirements of clients and applications. These advantages and more allow building complex system-models in an easier way.

In chapter 1, we have shown different models proposed to model the manufacturing systems with Petri nets. However, the proposed models do not take into

account the underlying network and their impact on the performance of the system. They are also focusing only on the design stage, but they do not give solutions of how to implement the models on a distributed service environment which is the main difference between the industrial companies and the universities' laboratories.

In chapter 2, we have introduced the different methods used to model communication protocols. UML, Timed Automata and Petri nets are suitable and the most used to model the communication protocols. But, Petri nets, as a powerful formalism show a high capacity and ability to model concurrent, and more particularity discrete event systems. Petri nets have also many extensions and tools. These two reasons in addition to that the service part is already modeled in Petri nets (the LAGIS/OSSc team approach) have persuaded us to greatly choose Petri nets as a unifying formalism for the modeling of the whole system (both services and communication protocols).

In chapter 3, we have shown our criteria to choose a Petri nets modeling tool. The high-level Petri nets contain nearly all the extensions of Petri nets and makes use of high-level programming language. This combination makes easier to use Petri nets extensions in one tool, reduces the complexity and size of the model and adds the modularity and reusability features to already powerful formalism.

We have also presented in chapter 3 our component-based modeling. The approach is based on building the system-model from smaller several components. This construction allows identifying each component separately. At the abstraction level, the overall component is seen as a black box hiding its internal implementation and behavior but offering only its interfaces and parameters that can be modified to be appropriate to the model needs. With this approach, addition, elimination, upgrade and modification of a component are possible to go with the systems requirements.

To build the different component, we have first analyzed the communication protocols of the MAC layer since the manufacturing systems use the local industrial networks to manage its exchanged messages. These protocols intersect in many points: channel check, backoff procedure, data send and receive processes. These similarities

have allowed building one component for each process, which are reused to model the different illustrative protocols of Ethernet and DCF 802.11 protocol.

The next step in this work was to evaluate and to verify the correctness and quality of our approach. In chapter 4, we have shown the different measures to evaluate the performance. Simulation is the most suitable in our case. The obtained results, compared to other studies and NS-2 simulator results, proved the correctness and precisions given by our approach.

The last step was to show a complete service-protocol model. For this, we have chosen the case study of the manufacturing system presented in chapter 1. However, in chapter 4, we have presented a new model of the system where the services are distributed over different PCs, eliminating the need to have an intermediate model. Each service is modeled in a separate Petri net. The overall service-protocol component is composited of one service component over one workstation component.

To complete the obtained results and verify the impact of the communication protocols over the productivity of the system, we have simulated the whole system. The simulation results show the direct impact and effects of the underlying network on the productivity of the system. They show the express relation between the network performance and the manufacturing system. Moreover, the obtained values are helpful and can be used in building and designing new systems, for example in our case the use 1Giga Ethernet does not provide to the system important benefits in case it uses 100M Ethernet and the use of 10M Ethernet is better than using wireless networks even with 11Mbps bandwidth.

In this thesis, the approach that we have proposed shows a well-defined modeling technique able to manage the complexity in modeling distributed systems and communication protocols by decomposing the model into small manageable and reusable components. Petri nets formalism has shown a high ability to model and simulate complex systems. The originality of our approach comes from unifying the modeling of both services and protocols in one formalism and completing the missed part of the

communication protocols of the previously presented models for the manufacturing system. This is proven by the obtained results of this approach and the benefits that a designer can have by saving a lot of lost cost and time where decomposing a complex model is a useful demonstration for difficult problems.

2. PERSPECTIVES

2.1 ENRICHING THE COMPONENTS LIBRARY

In this thesis we are interested in modeling manufacturing systems. This case study uses the local industrial network. For this reason, we have only presented LAN MAC sublayer protocols by representing two illustrative models for Ethernet and DCF 802.11b wireless protocols. This limited representation does not even cover all the protocols in that layer. In addition, what we have covered is only one type of the distributed systems.

However, as we have shown in the first chapter, distributed systems count as many as systems and distributed services and applications. The obtained results from our approach encourage generalizing this approach for new types of systems and applications like data base systems, ATM banking systems or transportation systems. This generalization requires to model new protocols since the modeled protocols does not allow doing that. The modeling of new protocols is not only limited to MAC sublayer protocols since such systems (data base or transportation) make use of the higher level layers protocols to perform their services. This implies and opens the door to model new components that can enrich the desired library and can help to cover more aspects in the distributed systems.

However, distributed systems are usually heterogeneous. To resolve this problem, a *middleware* layer between the services and protocols, figure 5.1, can be used. The use of this layer implies also the modeling of new components for that layer. Here, we can mention the need to cover the implementation stage. In the first chapter we have shown the different types of middleware used to implement a distributed service. In the work,

our modeling approach is based on a classical client/server. However, the aspect of implementing this structure is never discussed. Thus, this work must be completed by covering the implementation aspect which is very important.

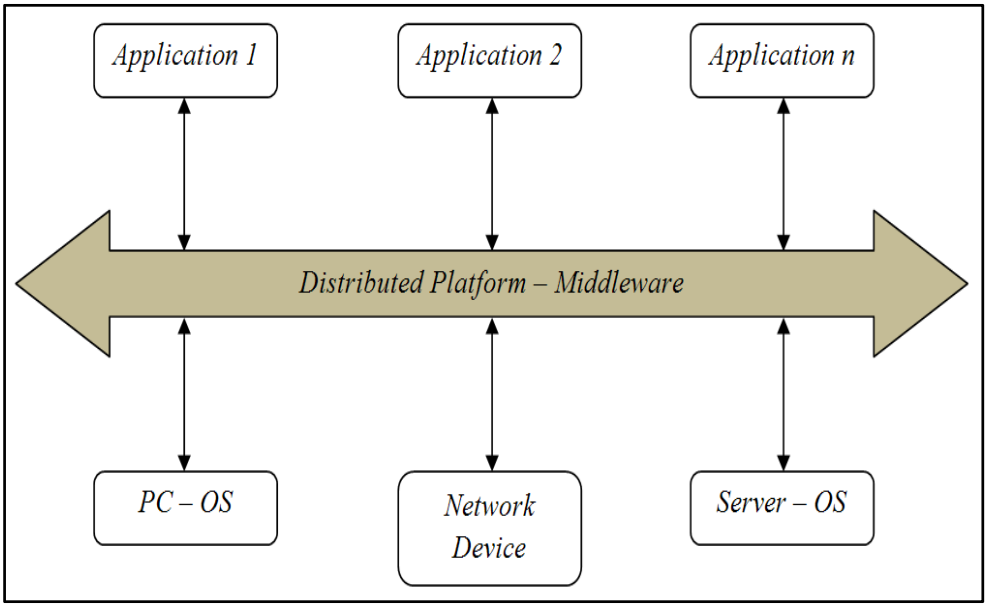


Figure 5.1, Middleware for Distributed Systems

With new protocols and middleware layer components, a real library of basic components can be created. Once these components are made, the modeling on new system-models becomes easier. Moreover, with middleware components, the results the can be obtained from simulating the whole system will be more accurate and reflects a better vision of the designed system.

2.2 QUANTITATIVE VERIFICATION

The thesis was based on using the simulation to verify some properties of the system. However, the work did not cover all the aspects of verifying some properties of the modeled components, especially for the unitary qualitative ones which is possible since we make use of Petri nets to model the system and since the components are small and can be verified easily. This aspect becomes more important once the components library is enriched with new components.

The verification process does not stop here since the complete system model will be build from these small components. Here, simulation can be useful for some properties but the need to guarantee the verified properties of the small components for the complete model or composite is also important.

2.3 TOWARDS A NEW SOFTWARE TOOL AND METHOD

From the previous needs appears the need to develop a new tool and method capable to perform the previous issues. The new tool must contain all the necessary functions and facilities to model distributed systems and communication protocols. Such tool facilitates unify the modeling. The used tools in this work do not cover the analytical verification of the system (no supporting packages). The new tool can be enforced with such packages.

In addition, this tool is not limited to model and simulate the system, but it can also be a helping tool for building the distributed systems by proposing the most appropriate components for the system or application to be modeled. For example, in this thesis we propose a workstation per service. This proposition may be costly and not the optimal solution. The proposed method can be useful to distribute the control/service part of the system on optimal workstation which can be more applicable and cheaper.

Résumé en Français :

Le développement des réseaux informatiques a permis l'émergence de nouvelles applications bénéficiant de la puissance et de la flexibilité offertes par la distribution de leurs fonctions sur de multiples ordinateurs. Ainsi, les systèmes dits « distribués » sont de plus en plus utilisés.

Dans cette thèse, nous nous intéressons plus particulièrement au contrôle par réseau des systèmes de production manufacturiers (SPM). Les systèmes de production manufacturiers sont une classe de systèmes à événements discrets dont les éléments interagissent ensemble pour construire des produits ou fournir des services.

Du point de vue industriel, ces systèmes sont composés d'un grand nombre de constituants physiques de nature multiples : machines d'usinage, systèmes de transport, de supervision, etc. La complexité de ces systèmes est encore renforcée par la présence de « flexibilités », qui permettent d'utiliser les mêmes machines pour produire plusieurs types de pièces, ce qui amène à multiplier les gammes opératoires à considérer. Le contrôle de ces systèmes est alors très important.

La mise en réseau est largement appliquée dans les applications industrielles. Les systèmes de production manufacturiers intègrent ainsi les réseaux de communications et les fonctions de contrôle à tous les niveaux. La connexion des éléments du système au travers d'un réseau réduit la complexité du système et le coût des ressources, et permet en outre de partager les données de manière efficace. Toutefois, la performance des réseaux de communication affecte les services offerts en termes de délais et de pertes de paquets.

La modélisation des systèmes de production manufacturiers est très importante pour vérifier certaines propriétés telles que l'absence de blocages, la vivacité, etc. ainsi que pour effectuer des analyses de performance. De nombreux travaux de recherche ont été proposés pour modéliser les systèmes flexibles de production manufacturiers :

1- L'approche du CRAN [Gouyon04] s'intéresse aux systèmes agiles. Elle traite l'interopérabilité entre contrôle des ressources et contrôle des produits.

2- L'approche du LAG/G-SCOP [Zamaï06] s'intéresse aux systèmes manufacturiers reconfigurables (SMR) plus particulièrement la génération en ligne de lois de commande.
3- L'approche du Lab-STICC [Berruet07] s'intéresse aux Systèmes reconfigurables. Elle propose la conception des systèmes de production manufacturiers par une approche composants.
4- L'approche du NHIT (Taiwan) [Tsai05] s'intéresse à l'exploitation des web-services pour le contrôle-commande.
5- L'approche du LAGIC/OSSc sur laquelle nous fondons nos travaux s'intéresse à la prise en compte de l'aspect distribué de la commande dès la phase de conception [Toguyeni06], figure 1.

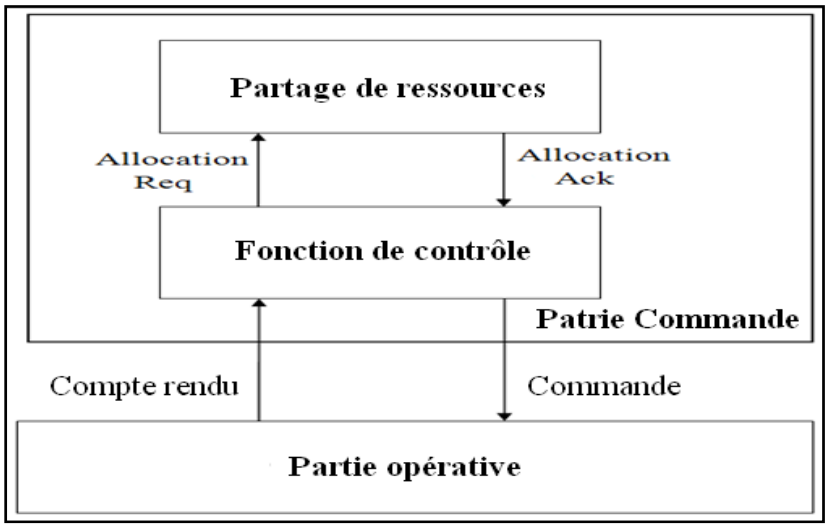


Figure 1, Approche du LAGIS/OSSc

Ces paradigmes classiques de modélisation sont généralement basés sur une vision centralisée. En effet, ce type de modélisation ne prend pas en compte le fait que le système sera distribué lors de l'implémentation et ne prend donc pas en compte l'architecture de communication. Aussi, les propriétés qui sont vérifiées lors la phase de conception ne sont pas nécessairement garanties après l'implémentation.

Dans ce contexte, nous proposons dans cette thèse la modélisation des systèmes flexibles de production manufacturiers et des protocoles réseau sous-jacents en un modèle

distribué sous la forme d'un client/serveur. Cette approche a été initialement proposée par l'équipe OSSc pour modéliser le système de production manufacturiers avec les réseaux de Petri. Dans cette thèse nous nous proposons de les modéliser avec les réseaux de Petri de haut niveau, qui est une méthode puissante capable de modéliser les deux points de vue. Cette possibilité vient de la capacité des réseaux de Petri à modéliser les systèmes concurrents et distribués.

Notre proposition : une approche par composants

Une manière de surmonter ces problèmes est de modéliser ces systèmes pour un fonctionnement en mode distribué. Un modèle distribué offre les moyens de décrire précisément l'ensemble des formes possibles d'incohérence à mesure qu'elles surviennent. Il tient compte de chaque partie dans le système, les ressources disponibles, la reconfiguration du système et le réseau sous-jacent. Une fois le modèle construit, son application et son implémentation sont plus faciles, car il a les mêmes caractéristiques que le système désiré. Néanmoins, ces systèmes sont complexes: la distribution, les dynamiques élevées, et ses composants d'une grande hétérogénéité. Par conséquent, il est nécessaire de modéliser ces systèmes d'une manière qui assure un haut degré de confiance et la rigueur des solutions.

Une façon de faire face à ce défi est l'utilisation de la méthodologie basée sur des composants qui est cohérente avec le principe des systèmes distribués dans lequel les composants sont réutilisables. L'approche par composants utilise les moyens génériques, modulaires et hiérarchiques de concevoir et d'analyser les systèmes. Elle prévoie que le modèle du système peut être assemblé à partir d'éléments travaillant ensemble, le concepteur n'a donc besoin que d'identifier les bons composants qui offrent de services appropriés en ce qui concerne les exigences des applications. Cette méthode permet la réutilisation et la généricité des composants qui réduit le coût du développement des systèmes.

L'approche par composants facilite et accélère la phase de modélisation. Cette dernière consiste à modéliser des composants élémentaires qui peuvent être réutilisés pour

construire des composants-composites. Ces composants sont génériques dans le sens où ils sont adaptatifs, paramétrables et réutilisables. Pour la réutilisation des composants élémentaires, nous proposons la construction d'une bibliothèque de composants de type commercial-off-the-shelf COTS [Carney00] [Weyuker98].

Les systèmes flexibles de production manufacturiers utilisent les réseaux locaux industriels pour échanger les messages entre machines. Ce type de réseau utilise les protocoles de la deuxième couche du modèle OSI. Ces protocoles sont des protocoles MAC. Pour illustrer notre technique de modélisation, nous proposons de modéliser les protocoles Ethernet IEEE 802.3 et IEEE 802.11b DCF.

La modélisation des components sera par les réseaux de Petri de haut niveau qui facilite la modélisation avec ses capacités de modéliser les comportements temporels, stochastiques et dynamiques. Ils donnent aussi la possibilité de identifier les jetons, ce qui très important pour la modélisation des protocoles informatique.

Les réseaux de Petri de haut niveau :

Plusieurs possibilités sont offertes pour le choix du formalisme de modélisation. Parmi ces choix nous avons comparé trois méthodes parmi les plus utilisées pour modéliser les services et protocoles de communication. Le tableau 1 montre les critères qu'on a considérés pour choisir le formalisme de modélisation. Dans cette thèse, nous nous intéressons aux techniques de modélisation formelle qui permettent des analyses poussées. Un autre besoin réside dans la capacité à modéliser des contraintes temporelles fortes. Le tableau montre bien que les réseaux de Petri sont les plus adapté à nos besoins.

	Automate Temporisé	**UML**	**RdP**
Méthode	Formelle	Semi-formelle	Formelle
Analyses formelles	Oui	Non	Oui
Modélisation temporelle	Oui	Récent	Oui
Applications	Applications temporelles	Usage général	Usage général

Tableau 1, Formalismes de modélisation

Les réseaux de Petri ont été introduits par [Petri66]. Ils sont présentés comme un graphe biparti où le marquage des places représente l'état du système et les transitions des événements et actions. Pour relier places et transitons, on utilise les arcs. Les réseaux de Petri disposent de nombreuses extensions et abréviations.

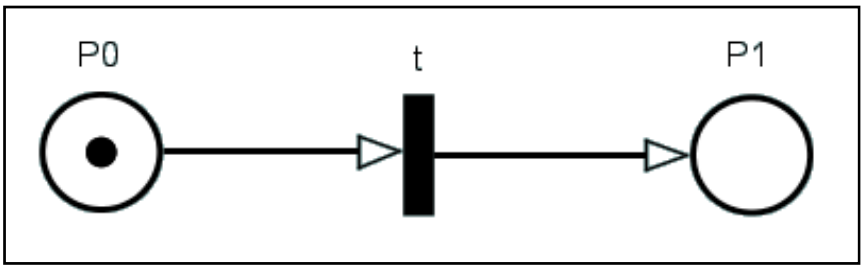


Figure 2, exemple d'un réseau de Petri

Les réseaux de Petri de haut niveau ont pour principales caractéristiques :

1- Des expressions sur les arcs et des gardes sur les transitions
2- La possibilité de distinguer les jetons en leur associant des identifiants
3- La capacité de modéliser des comportements temporels ou stochastiques

Le besoin d'identification de flux de données est très important dans les réseaux informatiques. Parmi l'envoi de multiples paquets entre source et destination, l'identification un flux précis entre plusieurs flux de données est nécessaire comme c'est le cas par exemple pour un arrêt d'urgence d'un robot dans les systèmes de production manufacturiers. La solution qu'on propose est l'utilisation des jetons caractérisés par des champs.

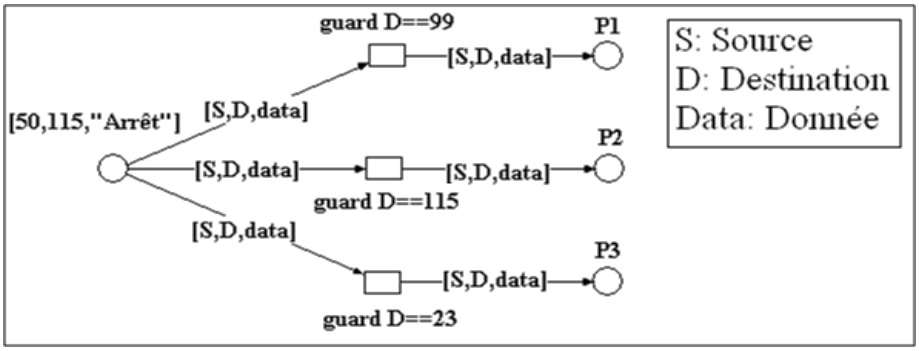


Figure 3, Expressions sur les arcs, gardes sur les transitions, identification des jetons

La figure 3 montre les expressions sur les arcs, les gardes sur les transitions, et l'identification des jetons. Dans la figure la seule transition franchissable est la transition avec le garde D==115, puisque le jeton a la valeur 115 dans son champ D.

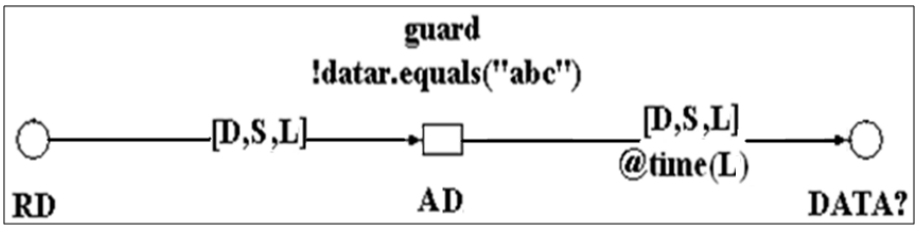


Figure 4, utilisation de fonctions dynamiques

La notion de temps est nécessaire puisque les protocoles réseaux ont des contraintes temporelles. Pour cela on propose d'ajouter la notion de temps (@) sur les arcs. Nous proposons aussi l'utilisation des fonctions dynamiques où les arguments sont les champs des jetons. Cette nécessité est due au besoin de modélisation des comportements dynamiques comme la mobilité des machines (réseaux wifi et robots mobiles), figure 4.

Démarche de modélisation :

Pour la modélisation, nous proposons de construire des composants qui sont caractérisés par des services offerts, une implémentation « cachée » où le code est accessible mais le concepteur n'a pas besoin de le modifier et aussi par des interfaces qui définissent les paramètres à fournir et à produire. L'implémentation effective du composant est fonction du point de vue et les exigences considérées.

Les interfaces choisies sont les places comme interfaces d'entrée et les transitions comme interfaces de sortie. Les places reçoivent autant de jetons que de composants producteurs et sont appropriées à la gestion de l'asynchronisme. Les transitions génèrent autant de jetons que de composants consommateurs. Ce choix permet de :

- prendre en compte l'asynchronisme des calculateurs,
- garantir le caractère générique des composants,
- faciliter les connexions.

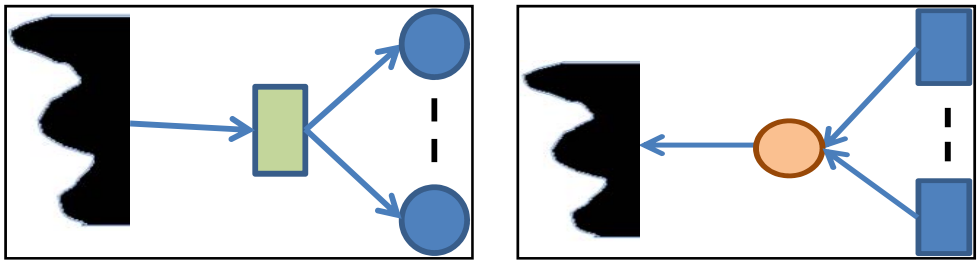


Figure 5, (a) interface de sortie (b) interface d'entrée

Construction des composants :

Pour construire les composants il est nécessaire d'analyser les protocoles. La démarche d'analyse est une démarche descendante pour identifier les comportements élémentaires communs. Chaque comportement commun identifié sera ensuite associé à un composant élémentaire. Une fois ces composants construits, on propose une d**émarche ascendante** à partir de ces composants élémentaires pour une construction hiérarchique de composant-composite.

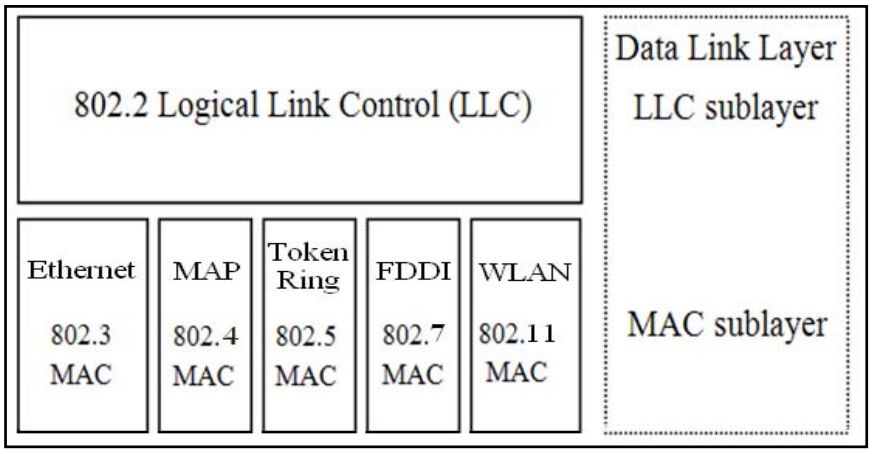


Figure 6, Protocoles MAC

Pour l'illustrer les deux démarches, nous utilisons les deux protocoles MAC Ethernet IEEE 802.3 et IEEE 802.11b DCF, figure 6. Ces deux protocoles sont des protocoles de la même couche (protocoles MAC Data Link) qui se fondent sur le même principe de détection de porteuse (protocoles CSMA). Ils disposent de comportements communs comme :

- Vérifier le statut du canal pour l'envoi
- Traiter des collisions
- Traiter des données

Trois composants élémentaire peuvent être trouvés sur la base des résultats de ces analyses: le composant channel/check, le composant Send/Receive modélisant l'envoi et la réception de données, d'accusé de réception et de JAM, et le composant Backoff/ BEB.

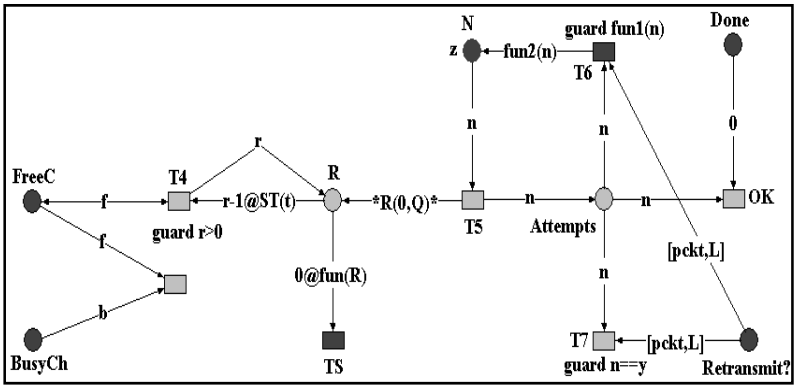


Figure 7, Composant Send/receive

Le composant Send/Receive est présenté en figure 7. Le composant représente la procédure de backoff et le traitement de collision. Avant un envoi, la machine met le nombre des tentatives (place N). Elle le garde dans la place « Attempts ». Solen ce nombre, elle choisi de manière aléatoire le backoff. Pour Ethernet il est entre 0 et 2^X, X dépend de nombre des tentatives. Par contre pour Wifi il est entre 0 et CW (contention window). Pour décrémenter le backoff, le canal doit être libre (un jeton dans la place « FreeC »). Si le statut de canal change la machine arrête de décrémenter le backoff, par contre elle le garde jusqu'à ce que le statut redevienne libre. Ici, si le backoff devient 0, la machine peut envoyer son message.

Après l'envoi, si une collision se produit (un jeton dans la place « Retransmit? ») la machine doit incrémente le nombre des tentatives. Si le nombre des tentatives de dépasse pas le nombre maximal, elle doit le retransmettre le paquet. Sinon elle arrête de le transmettre et elle le jette. Le tableau 2 montre les différents paramètres à modifier lors de l'utilisation de ce composant pour chaque protocole.

Variable	IEEE 802.11b	10M Ethernet	100M Ethernet
fun1(n)	n<33	n<15	n<15
fun2(n)	n=n*2	n=n+1	n=n+1
y	64	16	16
z	1	0	0
R(0, Q)	Aléatoire(0, CW)	Aléatoire(0, 2^X), X dépend de n	Aléatoire(0, 2^X), X dépend de n
Fun(R)	0	R*51.2µs	R*5.12µs
ST(t)	20µs	0	0

Tableau 2, paramètres de généricité

Modélisation ascendante :

Pour continuer la modélisation, un assemblage des composants élémentaires est utilisé pour la construction de composants-composites. Cet assemblage consiste à sélectionner les composants élémentaires approprié au composant-composite. Il faut parfois ajouter des éléments de RdP pour finir l'assemblage, comme le montrent les figures 8 et 9.

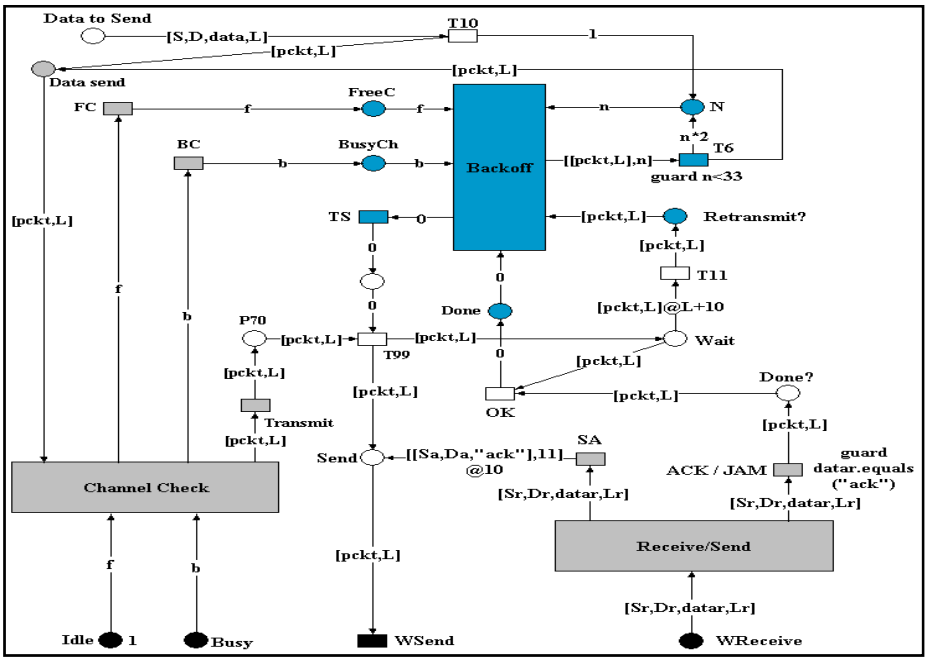


Figure 8, machine du protocole 802.11b DCF

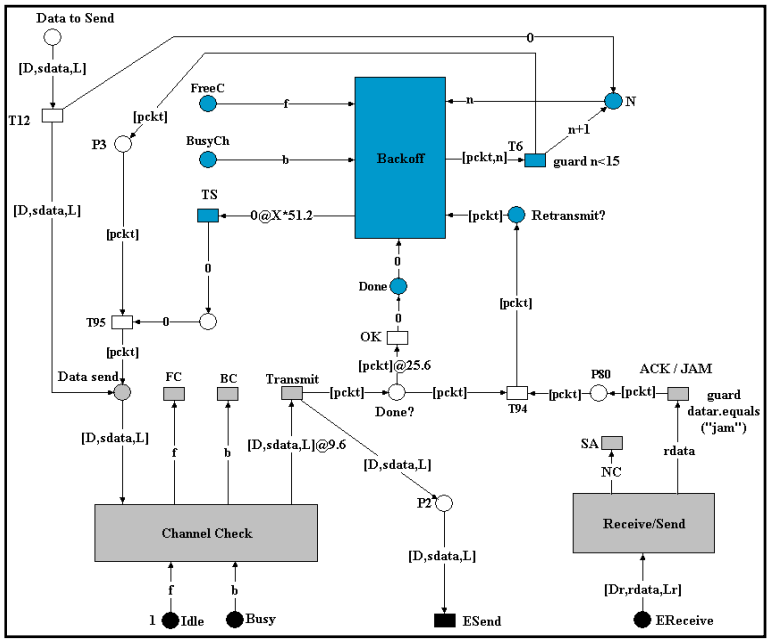


Figure 9, machine du protocole Ethernet

Validation expérimentale :

Pour voir la correction et la qualité de notre approche, la validation de la modélisation des protocoles est très importante. Cette validation est faite par l'utilisation de simulation (tableau 3) et une comparaison des résultats obtenus par notre modèle avec ceux obtenus par d'autres méthodes.

Critère	**Analytique**	**Simulation**	**Mesure**
Étape	Tout	Tout	Aval
Temps requis	Moyen	Moyen	Variable
Précision	Basse	Moyenne	Variable
Coût	Faible	Moyen	Élevé
« Scalabilité »	Faible	Moyen	Élevé
Flexibilité	Élevé	Élevé	Faible

Tableau 3, critères de choix d'une méthode pour la validation des modèles

L'outil le plus adapté à nos besoins est Renew : la modélisation de comportements temporels, stochastiques, dynamiques, et l'identification des jetons.

Résultats de Simulation :

La comparaison des résultats de simulation obtenus par Renew et par le simulateur NS-2 (figure 10 et 11) permettent de valider notre approche. La figure 12 montre également le temps nécessaire pour simuler le modèle sur Renew et NS-2.

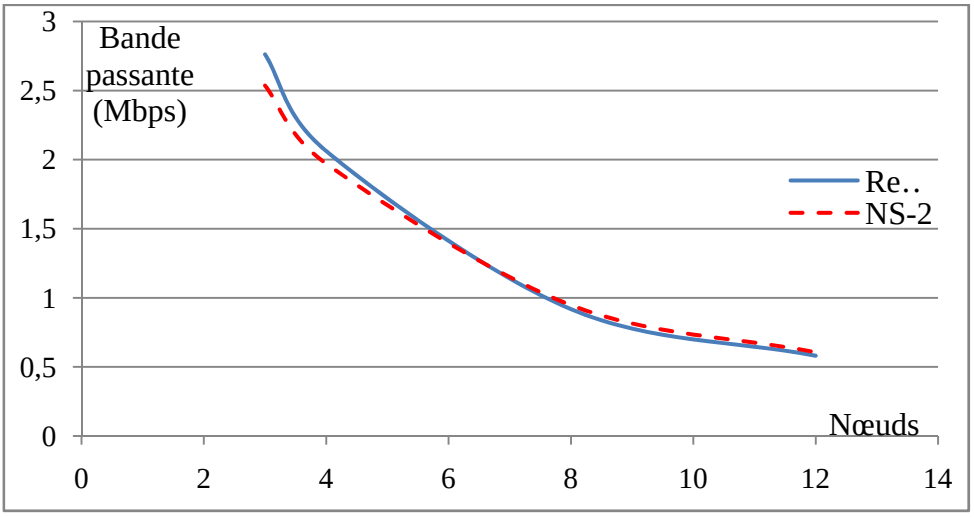


Figure 10, Partage de bande passante

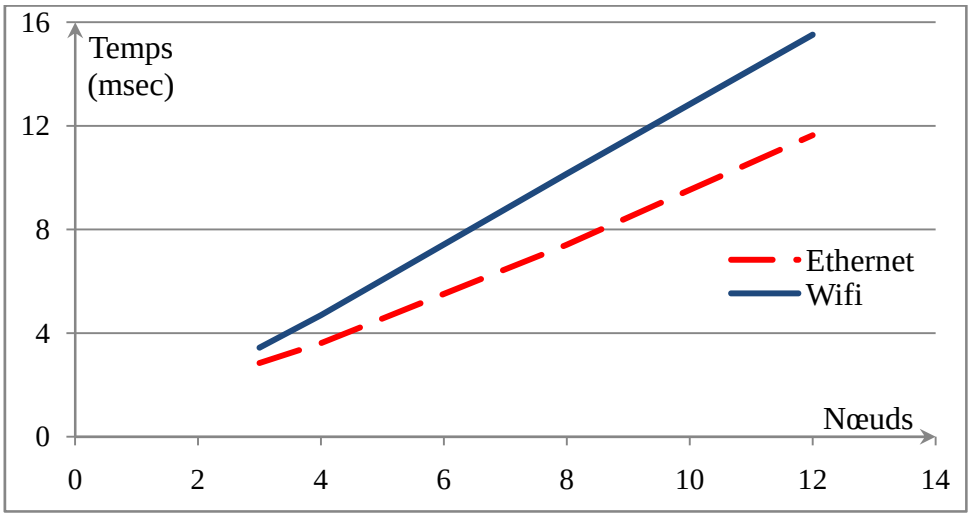


Figure 11, Wifi et Ethernet

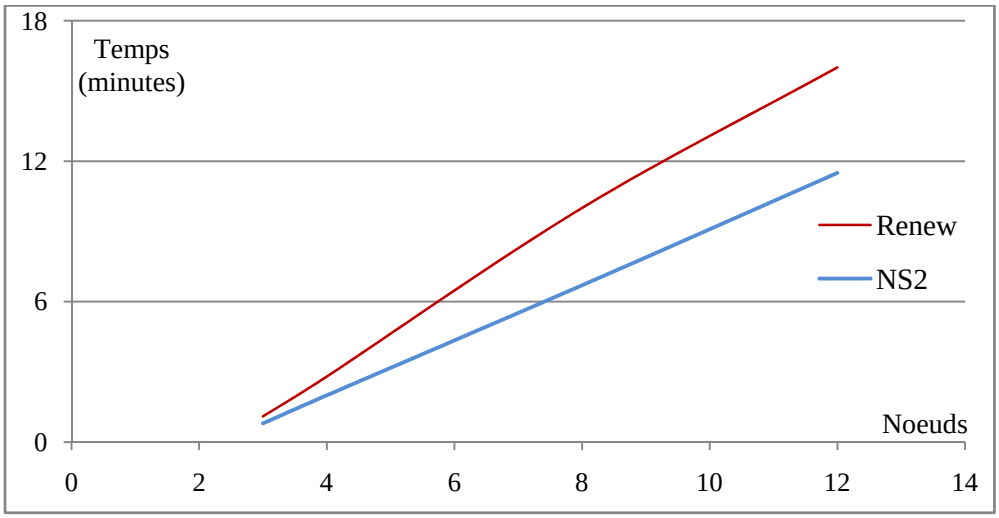


Figure 12, Temps de Simulation

Cas d'étude :

Un exemple illustratif est utilisé pour modéliser les services offerts par un système de production manufacturier, figure 13. La technique de modélisation utilisée a été la même que pour les protocoles de communication, c'est-à-dire à base d'approche par composants, où chaque partie du système est modélisé hiérarchiquement : un service par machine.

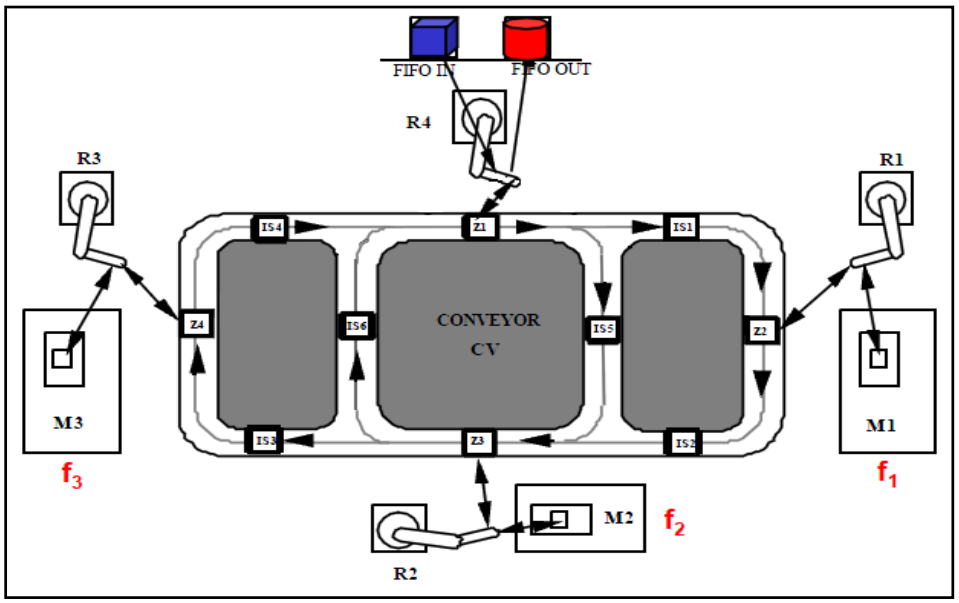


Figure 13, Exemple d'un système de production manufacturier

La figure 14 montre les messages échangés pour faire un transfert

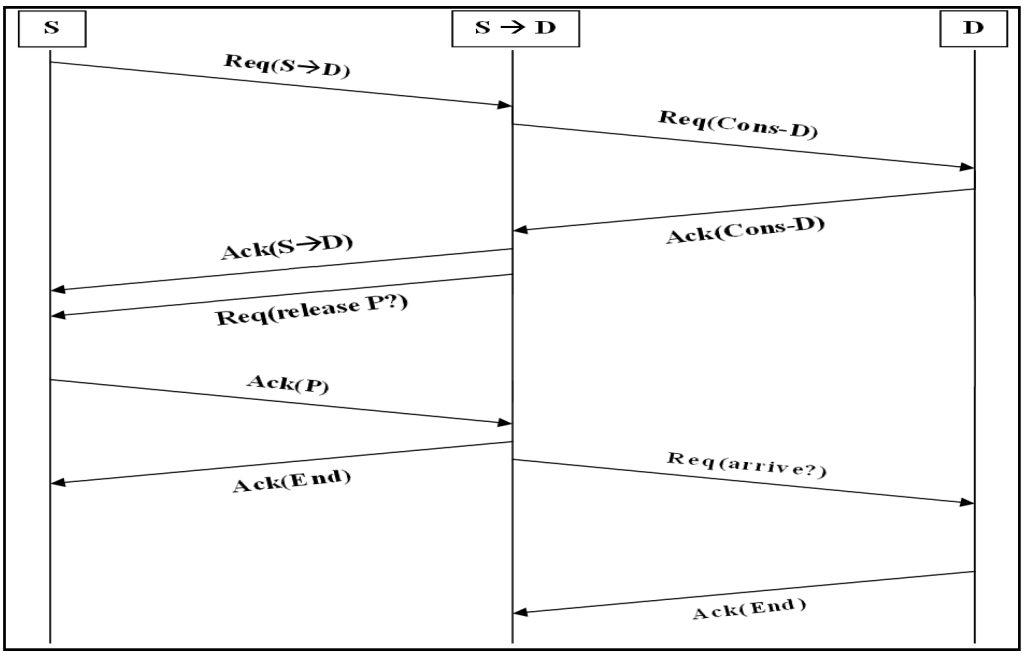


Figure 14, messages échangés pour faire un transfert

Conclusion :

L'approche que nous proposons pour la modélisation des systèmes de productions manufacturiers en distribué dès la conception pour la prise en compte de l'architecture de communication. La modélisation s'intéresse aux couches basses correspondant aux réseaux locaux industriels. Pour la modélisation, nous avons adopté une approche par composants. Ces composants sont réutilisables de type COTS qui baisse le coût de développement pour la modélisation d'autres système. Ils facilitent aussi la modification et la mise à jour de modèle sans touché l'ensemble de modèle globale. La modélisation des composants était en fonction du point de vue qui peut être changé selon la nécessité des concepteurs.

Les résultats que nous avons obtenus montrent la correction des modèles des protocoles et des composants élémentaires proposés. Ce qui amène à faire étendre notre approche aux différents systèmes distribués : bases de données, transport …

BIBLIOGRAPHY

[Aalto03] A. Aalto, N. Husberg, and K. Varpaaniemi, "*Automatic Formal Model Generation and Analysis of SDL*." Lecture Notes in Computer Science, Vol. 2708, pp. 285-299, 2003.

[Alony07] I. Alony and A. Munoz, "*The Bullwhip Effect in Complex Supply Chains*." International Symposium on Communications and Information Technologies, ISCIT '07, pp. 1355-1360, 2007.

[Alur92] R. Alur and D. Dill, "*A Theory of Timed Automata*." Lecture Notes in Computer Science, Vol. 600, pp. 45-73, 1992.

[Anastasi05] G. Anastasi, E. Borgia, M. Conti, and E. Gregori, "*IEEE 802.11b Ad Hoc Networks: Performance Measurements*." Cluster Computing, Vol. 8, No. 2-3, 2005.

[Baker05] S. Baker and S. Dobson, "*Comparing service-oriented and distributed object architectures.*" In Proceedings of the International Symposium on Distributed Objects and Applications, Lecture Notes in Computer Science, Vol. 3760, pp. 631–645, 2005.

[Barger03] P. Barger, J. Thiriet, and M. Robert, "*Dependability analysis of a distributed control or measurement architecture*", Proceedings of the 20th IEEE Instrumentation and Measurement Technology Conference, IMTC '03, Vol. 1, pp. 473- 477, 2003.

[Bastide04] R. Bastide and E. Barboni, "*Component-Based Behavioural Modelling with High-Level Petri Nets*." In MOCA'04 Aahrus, Denmark, DAIMI, pp. 37-46, 2004.

[Belabbas07] A. Belabbas, "*Conception de systèmes de contrôle/commande reconfigurables pour l'assistance technique aux personnes handicapées*." PhD Thesis, University of Bretagne-Sud, 2007.

[Bengtsson04] J. Bengtsson and W. Yi, "*Timed Automata: Semantics, Algorithms and Tools*." Lecture Notes in Computer Science, Vol. 3098, p. 87–124, 2004.

[Bernardi02] S. Bernardi, S. Donatelli, and J. Merseguer, "*From UML Sequence Diagrams and Statecharts to analysable Petri Net models*." Proceedings of the 3rd international workshop on Software and performance , pp. 35-45, 2002.

[Berruet98] P. Berruet, "*Contribution au recouvrement des Systèmes Flexibles de Production*." PdH thesis at Ecole Centrale de Lille, 1998.

[Berruet05] P. Berruet, J. Lallican, A. Rossi, and J-L. Philippe, "*A component based approach for the design of FMS control and supervision*." IEEE International Conference on Systems, Man and Cybernetics, Vol. 4, pp. 3005-3011, 2005.

[Berruet07] P. Berruet, "*Ingenerie de la Commande et Analyse des Systèmes Reconfigurables*." Habilitation to supervise the rechearch, University of Bretagne Sud, 2007.

[Berthomieu07] B. Berthomieu, F. Peres, and F. Vernadat, "*Model-checking Bounded Prioritized Time Petri Nets*." In Proceedings of ATVA 2007. Springer Verlag, LNCS 4762, 2007.

[Bianchi00] G. Bianchi, "*Performance Analysis of the IEEE 802.11 Distributed Coordination Function*." IEEE Journal on Selected Areas in Communications, Vol. 18, No. 3, Mar. 2000.

[Bigand04] M. Bigand, O. Korbaa and J-P. Bourey, "*Integration of FMS performance evaluation models using patterns for an information system design*", Computers & Industrial Engineering, Vol. 46, No. 4, pp. 625-637, 2004.

[Billington88] J. Billington, G. Wheeler, and M. Wilbur-Ham, "*PROTEAN: A High-Level Petri Net Tool for the Specification and Verification of Communication Protocols*." IEEE Transactions on Software Engineering , Vol. 14, No. 3, pp. 301-316, 1988.

[Bolognesi87] T. Bolognesi and E. Brinskma, "*Introduction to the ISO Specification Language LOTOS*." Computer Networks and ISDN Systems, Vol. 14, pp. 92-100, Apr. 1987.

[Billington04] J. Billington, G. Gallasch, and B. Han, "*A Coloured Petri Net Approach to Protocol Verification* ." Lectures on concurrency and Petri nets : advances in Petri nets, Vol. 3098, pp. 210-290, 2004.

[Boukadi07] K. Boukadi, Ch. Ghedira, Z. Maamar, and D. Benslimane, "*Specification and verification of views over composite web services using High Level Petri-Nets*." In proceedings of the 9th International Conference on Enterprise Information Systems ICEIS, 2007.

[Bourdeaud_huy06] T. Bourdeaud'huy and A. Toguyéni, "*Analysis of Reconfiguration Strategies Based On Petri Nets Models and Optimization Techniques*." 8th International Workshop on Discrete Event Systems, pp. 319-324, 2006.

[Bourey88] J-P. Bourey, "*Structuration de la partie procedurale du systeme de commande de cellules flexibles dans l'industrie manufacturibre*." PhD Thesis, University of Lille I, 1988.

[Bouyer03] P. Bouyer, "*Untameable Timed Automata! (Extended Abstract)*." Lecture Notes in Computer Science, Vol. 2607, pp. 620-631, 2003.

[Brenner97] P. Brenner, "*A Technical Tutorial on the IEEE 802.11 Protocol*." BreezeCOM, 1997.

[Brereton00] P. Brereton and D. Budgen, "*Component-Based Systems: A Classification of Issues*." IEEE Computer, Vol. 33, No. 11, pp. 54-62, 2000.

[Brown96] A. Brown and K. Wdlnau, "*Engineering of Component-Based Systems*." Second IEEE International Conference on Engineering of Complex Computer Systems, 1996.

[Bubnicki05] Z. Bubnicki, "*Modern Control Theory*". Springer-Verlag, 2005.

[Bucci05] G. Bucci, L. Sassoli, and E. Vicario, "*Correctness verification and performance analysis of real-time systems using stochastic preemptive time Petri nets*." IEEE Transactions on Software Engineering, Vol. 31, No. 11, 2005.

[Burns01] R. Burns, "*Advanced Control Engineering*". Butterworth-Heinemann, 2001.

[Capellmann99] C. Capellmann, H. Dibold, and U. Herzog, "*Using high-level petri nets in the field of intelligent networks*." Application of Petri Nets to Communication Networks, Advances in Petri Nets, Vol. 1605, pp. 1-36, 1999.

[Carney00] D. Carney and F. Long, "*What Do You Mean by COTS? Finally, a Useful Answer*." IEEE Software, 2000.

[Carothers06] C. Carothers, R. Lafortune, W. Smith, and M. Gilder, "*A Case Study in Modeling Large-Scale Peer-to-Peer File-Sharing Networks using Discrete Event Simulation* ." Proceedings of the International Mediterranean Modeling Multiconference, p. 617–624, 2006.

[Cassandras08] C. Cassandras and S. Lafortune, "*Introduction to Discrete Event Systems*", 2nd ed. Springer Science+Business Media, LLC, 2008.

[Cerami02] E. Cerami, "*Web Services Essentials Distributed Applications with XML-RPC, SOAP, UDDI & WSDL*". O'Reilly & Associates, Inc., 2002.

[Chachkov01] S. Chachkov and D. Buchs, "*From formal specifications to ready-to-use software components: The concurrent object oriented Petri Net approach.*" In Proceedings of the International Conference on Application of Concurrency to System Design, pp. 99-110, 2001.

[Cheesman01] J. Cheesman and J. Daniels, "*UML Components: A Simple Process for Specifying Component-Based Software*". Addison-Wesley, 2001.

[Chen08] L. Chen, Z. Shao, G. Fan, and X. Wang, "*Modeling and Analyzing Distributed Real-time and Embedded Systems with High-Level Petri Nets.*" IEEE/ASME International Conference on Mechtronic and Embedded Systems and Applications, MESA 2008, pp. 476-481, 2008.

[Chhabra07] A. Chhabra and G. Singh, "*Parametric Identification for Comparing Performance Evaluation Techniques in Parallel Systems.*" Proceedings the 1st National Conference on Challenges and Opportunities in Information Technology, COIT, pp. 92-97, 2007.

[Cho99] K. Cho and J. Lim, "*Mixed centralized/decentralized supervisory control of discrete events dynamic systems.*" Automatica, Vol. 35, No. 1, pp. 121-128, 1999.

[Cinderella07] Cinderella Official Website : *http://www.cinderella.dk/* - 2007.

[Corin07] R. Corin, S. Etalle, P. Hartel, and A. Mader, "*Timed Analysis of Security Protocols.*" Journal of Computer Security, Vol. 15, No. 6, pp. 619-645, 2007.

[Coulouris01] G. Coulouris, J. Dollimore, and T. Kindberg, "*Distributed Systems: Concepts and Design*", 3rd ed. Pearson Education, 2001.

[CPN07] Computer Tool for Coloured Petri Nets . CPN Tools: *http://wiki.daimi.au.dk/cpntools/cpntools.wiki* - 2007.

[DaSilveira02a] M. Da Silveira and M. Combacau, "*Supervision and Control of Heterarchical Discrete Event Systems: The Laas Approach.*" CBA - Congresso Brasileiro de Automática,, 2002.

[DaSilveira02b] M. Da Silveira, M. Combacau, and A. Subias, "*From centralized to distributed models: A systematic procedure based on Petri nets*." SMC - IEEE International Conference, 2002.

[DeLamotte05] F. De Lamotte, P. Berruet, and J.-L. Philippe, "*Using model transformation for the analysis of the architecture of a reconfigurable system*." IMACS 2005 world congress, 2005.

[Diaz87] M. Diaz, "*Petri Nets Based Models in the Specification and Verification of Protocols*." Lecture Notes In Computer Science, Vol. 255, pp. 135-170, 1987.

[Diaz91] M. Diaz and B. Berthomieu, "*Modeling and Verification of Time Dependent Systems Using Time Petri Nets*." IEEE Transactions on Software Engineering, Vol. 17, No. 3, 1991.

[Diaz06] G. Diaz, J. Pardo, M. Cambronero, V. Valero, and F. Cuartero, "*Verification of Web Services with Timed Automata*." Electronic Notes in Theoretical Computer Science, Vol. 157, No. 2, pp. 19-34, 2006.

[Díaz08] M. Díaz, D. Garrido, L. Llopis, and J. Troya, "*Designing distributed software with RT-CORBA and SDL*." Computer Standards & Interfaces, 2008.

[Dong04] J. Dong, P. Hao, S. Qin, J. Sun, and W. Yi, "*Timed patterns: TCOZ to Timed Automata*." Lecture Notes in Computer Science, ICFEM'04 , Vol. 483-498, p. 3308, 2004.

[Drosos01] C. Drosos, M. Zayadine, and D. Metafas, "*Real-Time Communication Protocol Development Using SDL for an Embedded System On Chip Based on ARM Microcontroller*." 13th Euromicro Conference on Real-Time Systems, pp. 89-94, 2001.

[DSouza99] D. D'Souza and A. Wills, "*Objects, Components, and Frameworks with UML: The Catalysis Approach*". Addison-Wesley, 1999.

[Dulaney09] E. Dulaney, "*CompTIA Security+™ Study Guide*", 4th ed. Wiley Publishing, Inc., 2009.

[Eddon99] G. Eddon and H. Eddon, "*Inside COM+ Base Services*". Microsoft Press , 1999.

[Edwards97] S. Edwards, D. Gibson, B. Weide and S. Zhupanov, "*Software Component Relationships*", In Proc. 8th Annual Workshop on Software Reuse, 1997.

[Eichner05] C. Eichner, H. Fleischhack, R. Meyer, S. U, and C. Stehno, "*Compositional Semantics for UML 2.0 Sequence Diagrams Using Petri Nets*." Lecture Notes in Computer Science, Vol. 3530, pp. 133-148, 2005.

[Engels02] G. Engels, J. Hausmann, R. Heckel, and S. Sauer, "*Testing The Consistency of Dynamic UML Diagrams*." Proceedings of the 6th International Congress of Integrated Design and Process Technology , IDPT, 2002.

[Erl05] T. Erl, "*Service-Oriented Architecture: Concepts, Technology, and Design*". Prentice Hall PTR, 2005.

[Ethier86] S. Ethier and T. Kurtz, "*Markov Processes Characterization and Convergence*". John Wiley & Sons, Inc. , 1986.

[Evangelista05] S. Evangelista, "*High Level Petri Nets Analysis with Helena*." Lecture Notes in Computer Science, Vol. 3536/2005, pp. 455-464, 2005

[Fadali09] M. Fadali and A. Visioli, "*Digital Control Engineering : Analysis and Design*". Elsevier Inc., 2009.

[Fishman01] G. Fishman, "*Discrete-Event Simulation: Modeling, Programming, and Analysis*". Springer Series in Operations Research, Springer-Verlag, 2001.

[Fortier03] P. Fortier and H. Michel, "*Computer Systems Performance Evaluation and Prediction*". Digital Press , 2003.

[Foster95] I. Foster, "*Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*". Addison-Wesley Longman Publishing Co., Inc., 1995.

[FWang08] F. Wang and D. Liu, "*Networked Control Systems: Theory and Applications*". Springer-Verlag London Limited, 2008.

[Gan06] B. Gan, L. Chan, and S. Turner, "*Interoperating Simulations of Automatic Material Handling Systems and Manufacturing Processes*." Proceedings of the Winter Simulation Conference, WSC 06, pp. 1129-1135, 2006.

[Geisterfer06] C. Geisterfer and S. Ghosh, "*Software Component Specification: A Study in Perspective of Component Selection and Reuse*." Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems, pp. 100-108, 2006.

[Giordano06] V. Giordano, J. Zhang, D. Naso, M. Wong, F. Lewis, and A. Carbotti, "*Matrix-based discrete event control of automated material handling systems*." Proceedings of the 45th IEEE Conference on Decision & Control, 2006.

[Glabbeek08] R. Glabbeek, U. Goltz, and J. Schicke, "*On Synchronous and Asynchronous Interaction in Distributed Systems*." Proceedings of the 33rd international symposium on Mathematical Foundations of Computer Science, Vol. 5162, pp. 16-35, 2008.

[Godary04] K. Godary, I. Augé-Blum, and A. Mignotte, "*SDL and timed petri nets versus UPPAAL for the validation of embedded architecture in automotive*." Forum on specification and Design Language, FDL, 2004.

[Gössler03] G. Gössler and J. Sifakis, "*Composition for Component-Based Modeling*." In Proceedings of the First International Symposium on Formal Methods for Components and Objects, FMCO 2002, Vol. LNCS 2852, pp. 443-466, 2003.

[Gössler07] G. Gössler, S. Graf, M. Majster-Cederbaum, M. Martens, and J. Sifakis, "*An Approach to Modelling and Verification of Component Based Systems*." Lecture Notes in Computer Science, SOFSEM 2007: Theory and Practice of Computer Science, Vol. 4362, pp. 295-308, 2007.

[Gouyon04] D. Gouyon, "*Product Driven Control of Manufacturing Execution Systems: synthesis techniques contribution*." PhD Thesis, Nancy-I, 2004.

[Haugen04] Ø. Haugen, B. Møller-Pedersen, and T. Weigert, "*Structural Modeling with Uml2.0: Classes, Interactions and State Machines*." UML for Real, pp. 53-76, 2004.

[Haverkort98] B. Haverkort, "*Performance of Computer Communication Systems: A Model-Based Approach*". John Wiley & Sons, Ltd, 1998.

[Heck03] B. Heck, L. Wills, and G. Vachtsevanos, "*Software Technology for Implementing Reusable, Distributed Control Systems*." IEEE Control Systems Magazine, Vol. 23, pp. 21-35, 2003.

[Hendriks06] M. Hendriks and M. Verhoef, "*Timed Automata Based Analysis of Embedded System Architectures*." 20th International Parallel and Distributed Processing Symposium, IPDPS, 2006.

[Henry05] S. Henry, "*Synthèse de Lois de commande pour la configuration et la reconfiguration des systèmes industriels complexes*." Institut National Polytechnique de Grenoble - INPG, 2005.

[Heusse03] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda, "*Performance anomaly of 802.11 b*." INFOCOM, 2003.

[Hinz05] S. Hinz, K. Schmidt and C. Stahl, "*Transforming BPEL to Petri Nets*." Lecture Notes in Computer Science, Vol. 3649/2005, pp. 220-235, 2005.

[Hong08] D. Hong and Y. Seo, "*A Hybrid Simulation Model for Manufacturing Systems Using Event-State-Operation Transitions*." IEEE International Conference on Industrial Engineering and Engineering Management, IEEM, pp. 1413-1417, 2008.

[Huang98] C. Huang and A. Kusiak, "*Modularity in Design of Products and Systems*." IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans, Vol. 28, No. 1, 1998.

[Huvenoit93] B. Huvenoit, E. Craye, and J. Bourey, "*Implantation oriented methodology in design control of flexiblemanufacturing systems*." Proceedings of Computers in Design, Manufacturing, and Production, CompEuro 93, pp. 125-131, 1993.

[IBM09] IBM Cell SDK. Cell Broadband Engine Resource Center: *http://www.ibm.com/developerworks/power/cell/* - 2009.

[IEC09] The International Engineering Constorium. Specification Description Language: *http://www.iec.org/online/tutorials/acrobat/sdl.pdf* - 2009.

[IEEE02] IEEE Std 802.3™, "*Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*." 2002.

[IEEE07] IEEE Computer Society, "*Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*." IEEE Std. 802.11™, 2007.

[IEEE09] IEEE. IEEE 802.3 Ethernet Working Group: *http://www.ieee802.org/3/* - 2009.

[IEEE802] 802 - Overview & Architecture : *http://www.ieee802.org/1/pages/802.html* - 2007.

[ISO09] International Organization for Standardization : *http://www.iso.org/iso/home.htm* - 2009.

[ITUT94] International Telecommunication Union, ITU-T, "*Open System Interconnection - Model and Notation*." ITU-T Recommendation X.200, 1994.

[JADE99] JADE Official Website : *http://homepages.dcc.ufmg.br/~coelho/jade.html* - 1999.

[Jain91] R. Jain, "*The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling* ". John Wiley & Sons,Inc., 1991.

[Jensen91] Kurt Jensen, "*Coloured Petri nets: A high level language for system design and analysis*", Lecture Notes in Computer Science, Vol. 483/1991, pp. 342-416, 1991.

[Jia05] W. Jia and W. Zhou, "*Distributed Network Systems: From Concepts to Implementations*". Springer Science + Business Media, Inc., 2005.

[JWang07] J. Wang, "*Petri nets for dynamic event-driven system modeling*." Handbook of Dynamic System Modeling, 2007.

[Kamrani02] A. Kamrani and S. Salhieh, "*Product Design for Modularity* ." Springer, 2002.

[Khomenko03] V. Khomenko and M. Koutny, "*Branching Processes of High-Level Petri Nets.*" Proceeding of International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS, Vol. 2619/2003, pp. 458-472, 2003.

[Kim07] T. Kim, Q. Yang, S. Park, and Y. Shin, "*SDL Design and Performance Evaluation of a Mobility Management Technique for 3GPP LTE Systems*." Lecture Notes in Computer Science, Vol. 4745, pp. 272-288, 2007.

[Kohler05] M. Kohler and J. Ortmann, "*Formal Aspects for Semantic Service Modeling Based on High-Level Petri Nets*." International Conference on Computational Intelligence for Modelling, Control and Automation, Vol. 1, pp. 107-112, 2005.

[Kozierok05] C. Kozierok, "*The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference*", 3rd ed. No Starch Press, 2005.

[Kristensen04] L. Kristensen, J. Jørgensen, and K. Jensen, "*Application of Coloured Petri Nets in System Development*." Lecture Notes in Computer Science, Vol. 3098, pp. 626-685, 2004.

[KRONOS02] VERIMAG. The tool Kronos:

http://www-verimag.imag.fr/TEMPORISE/kronos/ - 2002.

[Kryvyy08] S. Kryvyy and L. Matvyeyeva, "*Algorithm of Translation of MSC-specified System into Petri Net*." Special Issue on Concurrency Specification and Programming (CS&P), Vol. 79, No. 3-4, pp. 431-445, 2008.

[Kummer04] O. Kummer, et al., "*An Extensible Editor and Simulation Engine for Petri Nets: Renew*." Lecture Notes in Computer Science, Vol. 3099, pp. 484-493, 2004.

[Kumar07] P. Kumar and S. Radha, "*Parallel Discrete Event Simulation of IEEE 802.11 MAC Layer using Cell Processor*." International Conference on Emerging Trends in High Performance Architecture Algorithms & Computing, HiPAAC, 2007.

[Kumar09] N. Kumar and R. Sridharan, "*Simulation modelling and analysis of part and tool flow control decisions in a flexible manufacturing system* ." Robotics and Computer-Integrated Manufacturing, 2009.

[LAGIS09] Le Laboratoire d'Automatique, Génie Informatique et Signal. LAGIS : *http://lagis.ec-lille.fr/* - 2009.

[Lai02] R. Lai, "*A survey of communication protocol testing*." The Journal of Systems and Software, Vol. 62, No. 1, p. 21–46, 2002.

[Lakos95] C. Lakos, "*From Coloured Petri Nets to Object Petri Nets*." Lecture Notes in Computer Science, Vol. 935, PATPN, 1995.

[Lakos02] C. Lakos, "*The Challenge of Object Orientation for the Analysis of Concurrent Systems*." Lecture Notes in Computer Science, Vol. 2360, pp. 59-67, 2002.

[Lallican07] J-L. Lallican, "Proposition d'une approche composant pour la conception de la commande des systèmes transitiques." PhD Thesis University of Bretagne-Sud, 2007.

[Lammle08] T. Lammle, "*CCENT: Cisco Certified Entry Networking Technician Study Guide*". Wiley Publishing, Inc.,, 2008.

[Langlois02] R. Langlois, "*Modularity in technology and organization*." Journal of Economic Behavior & Organization, Vol. 49, p. 19–37, 2002.

[Larsen05] K. Larsen, M. Mikucionis, B. Nielsen, and A. Skou, "*Testing real-time embedded software using UPPAAL-TRON: an industrial case study*." Proceedings of the 5th ACM international conference on Embedded software , pp. 299-306, 2005.

[Latkoski07] P. Latkoski and L. Gavrilovska, "*Analysis of Bluetooth Protocol in Presence of Bursty Traffic*." Journal of Communications, Vol. 2, No. 6, pp. 38-45, 2007.

[Lee04] J. Lee and P. Hsu, "*Design and Implementation of the SNMP Agentsfor Remote Monitoring and Control via UMLand Petri Nets*." IEEE transactions on control systems technology, Vol. 12, No. 2, pp. 293-302, 2004.

[Lejri08] O. Lejri and M. Tagina, "*Modeling Hybrid Reconfigurable Manufacturing Systems Using Petri Nets*", Communications of SIWN, Vol. 3, pp. 130-134, 2008.

[Lerner02] M. Lerner, G. Vanecek, N. Vidovic, and D. Vrsalovic, "*Middleware Networks Concept, Design and Deployment of Internet Infrastructure*". Kluwer Academic Publishers, 2002.

[Lewis98] G. Lewis and C. Lakos, "*Towards Incremental Analysis*." First IEEE Workshop on Formal Methods for Dependable Systems, FMDS'98, 1998.

[Liang06] H. Liang, J. Dingel, and Z. Diskin, "*A Comparative Survey of Scenario-based to State-based Model Synthesis Approaches*." Proceedings of the international workshop on Scenarios and state machines: models, algorithms, and tools, *SCESM'06, pp.* 5 – 12, 2006.

[Lim08] H. Lim, B. Wang, C. Fu, A. Phull, and D. Ma, "*A Middleware Services Simulation Platform forWireless Sensor Networks*." The 28th International Conference on Distributed Computing Systems Workshops, ICDCS, pp. 168-173, 2008.

[Lin90] F. Lin and W. Wonham, "*Decentralized supervisory control of discrete event systems with partial observation*." IEEE transaction in Automatic Control, Vol. 35, pp. 1330-1337, 1990.

[Liu08] X. Liu, G. Yin, and Z. Zhang, "*A kind of Object-Oriented Petri Net and its Application*." International Conference on Internet Computing in Science and Engineering, ICICSE '08, pp. 541-544, 2008.

[Mallet06] F. Mallet, M. Peraldi-Frati, and C. André, "*From UML to Petri Nets for non functional Property Verification*." First IEEE Symposium on Industrial Embedded Systems (IES'06), pp. 1-9, 2006.

[Masri08a] A. Masri, T. Bourdeaud'huy, and A. Toguyeni, "*Performance Analysis of IEEE 802.11b Wireless Networks with Object Oriented Petri Nets*", Proceedings of First International Workshop on Formal Methods for Wireless Systems FMWS'08/CONCUR'08, Toronto, Canada, August 2008. (A special version for ENTCS is at August 2009).

[Masri08] A. Masri, T. Bourdeaud'huy, and A. Toguyeni, "*Network Protocol Modeling: A Time Petri Net Modular Approach*." 16th International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2008, pp. 274-278, Split - Dubrovnik, Croatia, September 2008.

[Masri09a] A. Masri, T. Bourdeaud'huy, and A. Toguyeni, "*A Component Modular Modeling Approach Based on Object Oriented Petri Nets for the Performance Analysis of Distributed Discrete Event Systems*", The Fifth International Conference on Networking and Services ICNS'09 Valencia, Spain, April, 2009. (**Best Paper Award**)

[Masri09b] A. Masri, T. Bourdeaud'huy, and A. Toguyeni, "A component-based approach based on High-Level Petri Nets for modeling Distributed Control Systems", International Journal on Advances in Intelligent Systems, Vol. 2, No. 3, Online publication End September 2009.

[Masri09] A. Masri, T. Bourdeaud'huy, and A. Toguyeni, "*Performance Evaluation of Distributed Systems: A Component-Based Modeling Approach based on Object Oriented Petri Nets*". In Book "***Petri nets***", ISBN 978-953-7619-X-X, Publication End November 2009.

[Matena03] V. Matena, S. Krishnan, L. DeMichiel, and B. Stearns, "*Applying Enterprise JavaBeans: Component-Based Development for the J2EE Platform*", 2nd ed. The Java Series. Addison-Wesley, 2003.

[Melia06] T. Melia, R. Aguiar, A. Sarma, and D. Hogrefe, "*Case study on the use of SDL for Specifying an IETF micro mobility protocol*." First International Conference on Communication System Software and Middleware, Comsware , 2006.

[Mendez02] H. Mendez, "*Synthèse de lois de surveillance pour les procédés industriels complexes*." Institut National Polytechnique de Grenoble - INPG , 2002.

[Merlin76] P. Merlin and D. Farber, "*Recoverability of communication protocols: Implications of a theoretical study*." IEEE Tr. Comm., Vol. 24, No. 9, pp. 1036-1043, 1976.

[Meyer97] B. Meyer, "*Object-Oriented Software Construction "*, 2nd ed. Prentice Hall, 1997.

[Microsoft96] Microsoft Corporation. DCOM Technical Overview: *http://msdn.microsoft.com/en-us/library/ms809340.aspx* - 0996.

[Mieghem06] P. Mieghem, "*Performance Analysis of Communications Networks and Systems "*. Cambridge University Press, 2006.

[Mir07] N. Mir, "*Computer and Communication Networks*". Prentice Hall, Inc, 2007.

[Moldt03] D. Moldt and H. Rolke, "*Pattern Based Workflow Design Using Reference Nets*." Lecture Notes in Computer Science, Vol. 2678, p. 246–260, 2003.

[Moraes06] R. Moraes, P. Portugal, and F. Vasques, "*A Stochastic Petri Net Model for the Simulation Analysis of the IEEE 802.11e EDCA Communication Protocol*." IEEE Conference on Emerging Technologies and Factory Automation, ETFA '06, pp. 38-45, 2006.

[Moreno08] R. Moreno, D. Tardioli, and J. Salcedo, "*Distributed Implementation of Discrete Event Control Systems based on Petri Nets*." IEEE International Symposium on Industrial Electronics, ISIE 2008, pp. 1738-1745, 2008.

[Murata89] T. Murata, "*Petri nets: Properties, Analysis and Applications*." Proc. of the IEEE, Vol. 77, No. 4, 1989.

[Nepomniaschy08] V. Nepomniaschy, D. Beloglazov, T. Churina, and M. Mashukov, "*Using Coloured Petri Nets to Model and Verify Telecommunications Systems*." Lecture Notes in Computer Science, Vol. 5010, pp. 360-371, 2008.

[Newcomer03] E. Newcomer, "*Understanding Web Services: XML, WSDL, SOAP, and UDDI*". Addison Wesley, 2003.

[NS208] NS2. Official Website : *http://nsnam.isi.edu/nsnam/index.php/Main_Page* - 2008.

[OMA09] Object Management Group OMG. Object Management Architecture: *http://www.omg.org/oma/* - 2009.

[OMG09a] The Object Management Group OMG. CORBA Component Model CCM : *http://www.omg.org/* - 2009.

[OMG09] The Object Management Group. OMG : *http://www.omg.org/* - 2009.

[OMNet09] OMNeT++ Community Site. *http://www.omnetpp.org/* - 2009.

[OPNET09] OPNET Technologies, Inc. *http://www.opnet.com/* - 2009.

[P2PSim05] P2PSim. Simulator for Peer-To-Peer Protocols: *http://pdos.csail.mit.edu/p2psim/index.html* - 2005.

[Pahl07] G. Pahl, W. Beitz, J. Feldhusen, and K. Grote, "*Engineering Design A Systematic Approach*", 3rd ed. Springer-Verlag London Limited, 2007.

[Paraskevopoulos02] P. Paraskevopoulos, "*Modern Control Engineering*". Marcel Dekker, Inc., 2002.

[Penix98] J. Penix and P. Alexander, "*Using formal specifications for component retrieval and reuse*." In Proc. of the 31st Hawaii International Conference on System Sciences, Vol. 3, pp. 356-365, 1998.

[Peterson03] L. Peterson and B. Davie, "*Computer Networks: A Systems Approach*", 3rd ed. Morgan Kaufmann Publishers, 2003.

[Peterson81] J. Peterson, "*Petri Net Theory and the Modeling of Systems*". Prentice-hall International, 1981.

[Petin05] J. Petin, P. Berruet, A. Toguyeni, and E. Zamaï, "*Impact of information and communication emerging technologies in automation engineering : outline of the intica proj*." 1st Workshop on Networked Control System and Fault Tolerant Control, 2005.

[Petri66] C. A. Petri, "*Communication with Automata*." Technical Report RADC-TR-65-377 Rome Air Dev. Center, New York, 1966.

[Petrucci05] L. Petrucci, "*Modularity and Petri Nets.*" 7th International Symposium on Programming and Systems, ISPS, 2005.

[PNW09] Petri Nets World: http://www.petrinetz.de/ - 2009.

[Puder06] A. Puder, K. Römer, and F. Pilhofer, "*Distributed Systems Architecture: A Middleware Approach* ". Morgan Kaufmann Publishers, Elsevier Inc., 2006.

[Ramadge87] P. Ramadge and W. Wonham, "*Supervisory control of a class of discrete event processes*." SIAM Journal on Control and Optimization, Vol. 25, No. 1, pp. 206-230, 1987.

[Ramadge89] P. Ramadge and W. Wonham, "*The Control of Discrete Event Systems*." Proceedings of the IEEE, Vol. 77, No. 1, 1989.

[Ramchandani74] C. Ramchandani, "*Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*." Project MAC, TR120, M.I.T., 1974.

[Renew08] The Reference Net Workshop. Renew : *http://www.renew.de/* - 2008.

[RFC1180] T. Socolofsky and C. Kale, " *RFC 1180 - A TCP/IP Tutorial*." Spider Systems Limited, 1991.

[Rofail99] A. Rofail and Y. Shohoud, "*Mastering COM and COM+ "*. Sybex, Inc. , 1999.

[Rumbaugh99] J. Rumbaugh, I. Jacobson, and G. Booch, "*The Unified Modeling Language Reference Manual*". Addison Wesley, Inc., 1999.

[Sametinger97] J. Sametinger, "*Software engineering with reusable components*", Springer, 1997.

[Sankar05] K. Sankar, S. Sundaralingam, A. Balinsky, and D. Miller, "*Cisco Wireless LAN Security*". Cisco Press, 2005.

[Sarjoughian05] H. Sarjoughian, W. Wang, K. Kempf, and H. Mittelmann, "*Hybrid discrete event simulation with model predictive control for semiconductor supply-chain manufacturing*." Proceedings of the 37th Conference on Winter Simulation, pp. 256 – 266, 2005.

[Schoop00] R. Schoop and R. Neubert, "*Agent-oriented material flow control system based on DCOM*", Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, ISORC, pp. 342-345, 2000

[Schriber05] T. Schriber and D. Brunner, "*Inside discrete-event simulation software: How it works and why it matters*." Proceedings of the Winter Simulation Conference, pp. 167-177, 2005.

[SDL09] SDL Forum Society : *http://www.sdl-forum.org/* - 2009.

[Seyler02] F. Seyler and P. Aniorte, "*A Component Meta Model for Reused-Based System Engineering*." Workshop in Software Model Engineering, Dresden, German, 2002.

[SOCRADES09] The SOCRADES Project: *http://www.socrades.eu/Home/default.html* - 2009.

[Spurgeon00] C. Spurgeon, "*Ethernet The Definitive Guide*", 1st ed. O'Reilly & Associates, Inc, 2000.

[Srinivasan95a] R. Srinivasan, "*RFC1831 - RPC: Remote Procedure Call Protocol Specification Version 2*." Sun Microsystems, 1995.

[Stallings07] W. Stallings, "*Data and Computer Communications*", 8th ed. Prentice-Hall, Inc, 2007.

[Subramanian05] R. Subramanian and B. Goodman, "*Peer-to-Peer Computing: The Evolution of a Disruptive Technology*". Idea Group Publisher,, 2005.

[Sun09a] Sun Developer Network (SDN). Java Sun : *http://java.sun.com/* - 2009.

[Sun09b] Sun Microsystems. JavaTM Remote Method Invocation (RMI): *http://java.sun.com/j2se/1.3/docs/guide/rmi/* - 2009.

[Tanenbaum03] A. Tanenbaum, "*Computer Networks*", 4th ed. Prentice Hall, 2003.

[Tanenbaum95] A. Tanenbaum, "*Distributed Operating Systems*". Prentice Hall, 1995.

[Tari01] Z. Tari and O. Bukhres, "*Fundamentals of Distributed Object Systems: The CORBA Perspective*". John Wiley & Sons, Inc., 2001.

[Thai02] T. Thai and H. Lam, ".*NET Framework Essentials*", 2nd ed. O'Reilly Media, 2002.

[Tina09] Laas - CNRS. TIme Petri Nets : *http://www.laas.fr/tina* - 2009.

[Toguyeni06] A. Toguyeni, "*Design of Modular and Hierarchical Controllers for Reconfigurable Manufacturing Systems*." IMACS Multiconference on Computational Engineering in Systems Applications, Vol. 1, pp. 1004-1011, 2006.

[Tsai05] M. Tsai and L. Lin, "*Web-based distributed manufacturing control systems*", The International Journal of Advanced Manufacturing Technology , Vol. 25, No. 5-6, pp. 608-618, 2005.

[UML09] OMG. Unified modeling language UML : *http://www.omg.org/uml* - 2009.

[UPPAAL09] UPPAAL Official Website : *http://www.uppaal.com/* - 2009.

[Verma04] D. Verma, "*Legitimate Applications of Peer-to-Peer Networks*". John Wiley & Sons, Inc., 2004.

[Vojnar01] T. Vojnar, "*Towards Formal Analysis and Verification over State Spaces of Object-Oriented Petri Nets.*", PhD thesis, 2001.

[W3C01] W3C. URIs, URLs, and URNs: Clarifications and Recommendations 1.0 : *http://www.w3.org/TR/uri-clarification/* - 2001.

[W3C09] W3C. World Wide Web : *http://www.w3.org/* - 2009.

[Weber07] D. Weber, J. Glaser, and S. Mahlknecht, "*Discrete Event Simulation Framework for Power Aware Wireless Sensor Networks*." 5th IEEE International Conference on Industrial Informatics, Vol. 1, pp. 335-340, 2007.

[Werner05] C. Werner, X. Fu, and D. Hogrefe, "*Modeling Route Change in Soft-State Signaling Protocols Using SDL: A Case of RSVP*." Lecture Notes in Computer Science, Vol. 3530, pp. 174-186, 2005.

[Werner06] C. Werner, S. Kraatz, and D. Hogrefe, "*A UML Profile for Communicating Systems*." Lecture Notes in Computer Science, Vol. 4320, pp. 1-18, 2006.

[Weyuker98] E. Weyuker, "*Testing Component-Based Software: A CautionaryTale*." IEEE Software, Vol. 15, No. 5, pp. 54-59, 1998.

[Woodside95] M. Woodside, J. Neilson, D. Petriu, and S. Majumdar, "*The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software*." IEEE Transactions on Computers, Vol. 44, No. 1, pp. 20-34, 1995.

[Wu07] Y. Wu, K. Zhang, X. Wang, and J. Tian, "*Extending metadata with scenarios in adaptive distributed system*." Journal of Network and Computer Applications, Vol. 30, No. 4, pp. 1283-1294, 2007.

[Xu07] T. Xu and T. Tang, "*The modeling and Analysis of Data Communication System (DCS) in Communication Based Train Control (CBTC) with Colored Petri Nets*." Eighth International Symposium on Autonomous Decentralized Systems, ISADS '07, pp. 83-92, 2007.

[Yovine97] S. Yovine, "*KRONOS: a verification tool for real-time systems*." International Journal on Software Tools for Technology Transfer (STTT), Vol. 1, No. 1-2, pp. 123-133, 1997.

[Zamaï98] E. C.-S. A. Zamaï and M. Combacau, "*An architecture for control and monitoring of discrete events systems*." Computers in Industry, Vol. 36, No. 1 - 2, pp. 95-100, 1998.

[Zamaï06] E. Zamaï, "*Contribution à la ConduiteRéactive de Flux en Contexte incertain*." Institut National Polytechnique de Grenoble - INPG, 2006.

[Zhang01] W. Zhang, M. Branicky, and S. Phillips, "*Stability of Networked Control Systems*." IEEE Control Systems Magazine, Vol. 21, pp. 84-99, 2001.

[Zimmermann80] H. Zimmermann, "*OS1 Reference Model-The IS0 Model of Architecture for Open Systems Interconnection*." IEEE Transactions on Communications, Vol. COM-28, No. 4, 1980.

[ZWang07] Z. Wang, X. Peng, and Z. Ji, "*Interactive multimedia synchronization model based on Petri Nets*." Wuhan University Journal of Natural Sciences, Vol. 12, No. 6, pp. 1019-1023, 2007.

Abstract: *Manufacturing systems* belong to the class of *distributed discrete event systems.* Their size requires distributing the software to control them on architecture of several industrial computers connected by networks. In this context, it becomes crucial to be able to evaluate the impact of a specific architecture on the manufacturing systems services both in terms of performance and quality. The performance of the underlying network can notably affect the productivity of the system. In traditional methodology proposed in literature, this aspect is not taken into account in the design stage. Thus, modeling such systems is important to verify some properties at that stage. In this thesis, we propose a *component-based modeling approach* with High Level Petri nets based method for modeling some network protocols in order to evaluate the manufacturing systems as being distributed systems. The selection of Petri nets is justified by their expression power with regard to the modeling of distributed and concurrent systems. *Component-based approach* can decrease modeling complexity and encourages genericity, modularity and reusability of ready-to-use components. This allows building new models easily and reducing the systems development cost. Moreover, this can help in better managing services and protocols and to easily change/modify a system element. Finally, this modeling enables us to evaluate discrete event systems by means of centralized simulations.
Key-words: Communication Networks, Distributed Systems, Protocols, Manufacturing Systems, High-Level Petri Nets, Component-based Modeling

Résumé : *Les systèmes de production manufacturiers* sont une classe des *systèmes à événements discrets*. Leur taille nécessite de distribuer le logiciel de contrôle sur une architecture industrielle de plusieurs ordinateurs reliés en réseau. Dans ce contexte, il devient essentiel d'être capable d'évaluer l'impact d'une architecture réseau spécifique sur les services des systèmes manufacturiers en termes de la performance et la qualité. Les performances du réseau sous-jacent peuvent notamment nuire à la productivité du système. Dans la méthodologie traditionnelle proposée dans la littérature, cet aspect n'est pas pris en compte au niveau conception. Cependant, la modélisation de tels systèmes est importante pour vérifier certaines propriétés. Dans cette thèse, nous proposons une *approche de modélisation par composants* à l'aide *des réseaux de Petri haut niveau* pour la modélisation de certains protocoles de réseaux afin d'évaluer les systèmes manufacturiers comme étant des systèmes distribués. La sélection des réseaux de Petri est justifiée par leur pouvoir d'expression en ce qui concerne la modélisation des systèmes distribués et concurrents. L'approche par composants permet de diminuer la complexité de la modélisation et encourage la généricité, la modularité et la réutilisabilité des *composants prêt-à-utiliser*. Cela permet de construire facilement de nouveaux modèles et de réduire les coûts de développement de systèmes. En outre, cela peut aider à une meilleure gestion des services et des protocoles et à changer facilement/modifier un élément du système. Notre modélisation permet enfin d'évaluer ces systèmes par le biais de simulations centralisées.

Mots-clés *:* Réseaux de Communication, Systèmes Distribués, Protocoles, Systèmes de Production Manufacturés, Réseaux de Petri Haut Niveau, Modélisations par Composants.