UNIVERSITÀ DEGLI
STUDI DI SALERNO

.DIEM

Ministero dell'Istruzione, dell'Università e della Ricerca

Facoltà di Ingegneria
Dipartimento di Ingegneria dell'Informazione ed Elettrica e Matematica Applicata

Dottorato di Ricerca in Ingegneria dell'Informazione
XIV Ciclo – Nuova Serie

TESI DI DOTTORATO

An approach to task coordination for hyperflexible robotic workcells

CANDIDATO: **DIEGO GERBASIO**

COORDINATORE: **PROF. MAURIZIO LONGO**

TUTOR: **PROF. PASQUALE CHIACCHIO**

Anno Accademico 2014 – 2015

Contents

Chapter 1

Preface

The manufacturing industry is very diverse and covers a wide range of specific processes ranging from extracting minerals to assembly of very complex products such as planes or computers, with all intermediate processing steps in a long chain of industrial suppliers and customers. Extracting from this wide variety of businesses and processes, some general trends that could be meaningful to the whole industry chain is far from easy and is bound to give rather general fields [EAG].

It is well know that the introduction of robots in manufacturing industries has many advantages. Basically, in relation to human labor, robots work to a constant level of quality. For example, waste, scrap and rework are minimized. Furthermore they can work in areas that are hazardous or unpleasant to humans. Robots are advantageous where strength is required, and in many applications they are also faster than humans. Also, in relation to special-purpose dedicated equipment, robots are more easily reprogrammed to cope with new products or changes in the design of existing ones [ASV08].

In the last 30-40 years, large enterprises in high-volume markets have managed to remain competitive and maintain qualified jobs by increasing their productivity, through, among others, the incremental adoption and use of advanced ICT and robotics technologies. In the 70s, robots have been introduced for the automation of a wide spectrum of tasks such as: assembly of cars, white goods, electronic devices, machining of metal and plastic parts, and handling of work

pieces and objects of all kinds. Robotics has thus soon become a synonym for competitive manufacturing and a key contributing technology for strengthening the economic base of Europe [EUR]. So far, the automotive and electronics industries and their supply chains are the main users of robot systems and are accounting for more than 60% of the total annual robot sales. Robotic technologies have thus mainly been driven by the needs of these high-volume market industries.

The degree of automation in the automotive industries is expected to increase in the future as robots will push the limits towards flexibility regarding faster change-over-times of different product types (through rapid programming generation schemes), capabilities to deal with tolerances (through an extensive use of sensors) and costs (by reducing customized work-cell installations and reuse of manufacturing equipment). These immediate challenges lead to the following current RTD trends in robotics:

- Expensive single-purpose transport and fixing equipment is replaced by standard robots thus allowing continuous production flows. Remaining fixtures may be adjusted by the robot itself.

- Cooperative robots in a work-cell coordinate handling, fixing and process tasks so that robots may be easily adjusted to varying work piece geometries, process parameters and task sequences. Short change-over times are reached by automated program generation which takes into account necessary synchronization, collision avoidance and robot-to-robot calibration.

- Measuring devices mounted on robots and increased use of sensor systems and RFID-tagged parts carrying individual information contributes to better dealing with tolerances in automated processes.

- Human-robot-cooperation bridges the gap between fully manual and fully automated task execution. People and robots will share sensing, cognitive and physical capabilities.

There are numerous new fields of applications in which robot technology is not widespread today due to its lack of flexibility and high

costs involved when dealing with varying lot sizes and variable product geometries. In such cases, *hyper-flexible robotic work cells* can help in providing flexibility to the system and making it adaptable to the different dynamic production requirements. Hyper-flexible robotic work cells, in fact, can be composed of sets of industrial robotic manipulators that cooperate to achieve the production step that characterize the work cell; they can be programmed and re-programmed to achieve a wide class of operations and they may result versatile to perform different kind of tasks [GA12].

1.1 Thesis contribution

Related key technology challenges for pursuing successful long-term industrial robot automation are introduced at three levels: basic technologies, robot components and systems integration. On a systems integration level, the main challenges lie in the development of methods and tools for instructing and synchronising the operation of a group of cooperative robots at the shop-floor. Furthermore, the development of the concept of hyper flexible manufacturing systems implies soon the availability of: consistent middleware for automation modules to seamlessly connect robots, peripheral devices and industrial IT systems without reprogramming everything ("plug-and-play") [EUR].

In this thesis both innovative and traditional industrial robot applications will be analyzed from the point of view of task coordination. In the modeling environment, contribution of this dissertation consists in presenting a new methodology to obtain a model oriented to the control the sequencing of the activities of a robotic hyperflexible cell. First a formal model using the Colored Modified Hybrid Petri Nets (CMHPN) is presented. An algorithm is provided to obtain an automatic synthesis of the CMHPN of a robotic cell with detail attention to aircraft industry. It is important to notice that the CMHPN is used to model the cell behaviour at a high level of abstraction. It models the activities of each cell component and its coordination by a supervisory system. As more, an object oriented approach and supervisory control are proposed to implement industrial automation control systems (based on

Programmable Logic Controllers) to meet the new challenges of this field:capability to implement applications involving widely distributed devices and high reuse of software components. Hence a a method is proposed to implement both controllers and supervisors designed by Petri Nets on Programmable Logic Controllers (PLCs) using Object Oriented Programming (OOP). Finally preliminary results about a novel cyber-physical approach to the design of automated warehouse systems is presented.

1.2 Thesis Overview

The thesis is organized as follow:

Chapter 2 describes the state of the art of hyperflexible robotic workcells and introduces the issues of a complex but essential task in aeronautical industries

Chapter 3 contains a brief background on Petri Net (PN), Colored PN and Hybrid PN formalisms, necessary to understand the others chapters. Because of their importance in the developing of the contribute of this thesis, a brief literature review about the Hybrid PNs is also presented.

Chapter 4 introduces an approaches to model complex Hyper-flexible robotic work cells based on a new Petri net formalism that merges the concepts of Hybrid Petri Nets and Colored Petri Nets.

Chapter 5 presents a method to implements controllers and supervisors designed by Petri Nets on PLCs using Object Oriented Programming

Chapter 6 summarizes the previous chapter and presents a cyber-physical perspective view about the control of automated warehouse systems.

Chapter 2

State of the art

There are numerous new fields of applications in which robot technology is not widespread today due to its lack of flexibility and high costs involved when dealing with varying lot sizes and variable product geometries. New robotic applications will soon emerge from new industries and from SMEs, which cannot use today's inflexible robot technology or which still require a lot of manual operations under strenuous, unhealthy and hazardous conditions. In this chapter the state of the art for robotic applications will be described and the issues of automation for aeronautical industries will be introduced.

2.1 Introduction on hyperflexible cells

The manufacturing industry is very diverse and covers a wide range of specific processes ranging from extracting minerals to assembly of very complex products such as planes or computers, with all intermediate processing steps in a long chain of industrial suppliers and customers. Extracting from this wide variety of businesses and processes, some general trends that could be meaningful to the whole industry chain is far from easy and is bound to give rather general fields [EAG].

It is well know that the introduction of robots in manufacturing industries has many advantages. Basically, in relation to human labor, robots work to a constant level of quality. For example, waste, scrap and rework are minimized. Furthermore they can work in ar-

eas that are hazardous or unpleasant to humans. Robots are advantageous where strength is required, and in many applications they are also faster than humans. Also, in relation to special-purpose dedicated equipment, robots are more easily reprogrammed to cope with new products or changes in the design of existing ones [ASV08].

In the last 30-40 years, large enterprises in high-volume markets have managed to remain competitive and maintain qualified jobs by increasing their productivity, through, among others, the incremental adoption and use of advanced ICT and robotics technologies. In the 70s, robots have been introduced for the automation of a wide spectrum of tasks such as: assembly of cars, white goods, electronic devices, machining of metal and plastic parts, and handling of work pieces and objects of all kinds. Robotics has thus soon become a synonym for competitive manufacturing and a key contributing technology for strengthening the economic base of Europe [EUR].

So far, industrial robot technology and products have largely been driven by the requirements of the automotive and the electronics (light assembly) industry. It is foreseen that future manufacturing paradigms in these industries will still be largely depending on future robotic products, solutions and services. However, the emergence of other applications from non-automotive industries opens up new technological horizons and market opportunities for robotic technologies.

Relieving people from bad working conditions (e.g., operation of hazardous machines, handling poisonous or heavy material, working in dangerous or unpleasant environments) leads to many new opportunities for applying robotics technology. Examples of bad working conditions can be found in foundries or the metal working industry. Besides the need of handling objects at very high temperatures, work under unhealthy conditions takes place in manual fettling operations, which contribute to about 40% of the total production cost in a foundry. Manual fettling means heavy lifts, strong vibrations, metal dust and high noise levels, resulting in annual hospitalization costs of more than €150m in Europe. Bad working conditions can also be found in slaughterhouses, fisheries and cold stores where beside low temperatures also the handling of sharp tools makes the work unhealthy and hazardous. Other examples where robots can improve the working

environment are painter workshops, glazier workshops and garbage handling plants.

If sensor information can be reliably used for robot control and if robot instruction schemes may be intuitive (e.g., by using more intuitive interaction mechanisms, built-in process knowledge and automatic motion generation), many other applications where present robot technology has failed can be envisioned:

- **Assembly and disassembly** (vehicles, airplanes, refrigerators, washing machines, consumer goods). Obviously challenges address cost-effective robot systems which are able to cope with a wide range of processes, tasks and objects. In many cases fully automatic task execution by robots is impossible. Cooperative robots should support the worker in terms of force augmentation, parallelization or sharing of tasks. Cost-effectiveness can only be reached by drastically reducing the health hazards for the worker or increasing the productivity of the manual workplace by typically 50 - 100

- **Aerospace industry** currently uses customized NC machines for machining, drilling, assembly, quality testing operations on structural parts. In assembly and quality testing, the automation level is still low due to the variability of configurations and insufficient precision of available robots. Identified requirements for future robots call for higher accuracy, adaptivity towards workpiece tolerances, flexibility to cover different product ranges, and safe cooperation with operators.

- **SME manufacturing**: Cutting, fettling, deflashing, drilling, deburring, milling, grinding and polishing of products made of metal, glass, ceramics, plastics, rubber and wood.

- **Food and consumer good industries**: Processing, assembly, filling, handling and packaging of food and consumer goods

- **Construction**: Cutting, drilling, grinding and welding of large beams and other construction elements for buildings, bridges, ships, trains, power stations, wind mills etc. In most of these

applications robots would have to cope with products having big variations in geometry and material properties and often produced in small batches. In order to achieve the required adaptability, the robot controller must be able to make use of information from different types of sensors, where the most important input will come from vision- and force sensing systems. Simultaneously, it must be very simple to instruct the robot to perform different tasks more or less autonomously in response to the sensor information and built-in process knowledge. These requirements are similar to service robots and a closer R&D collaboration between industrial and service robotics will be crucial to obtain the radical innovations needed to obtain a widespread use of industrial robot automation [EAG].

Industrial Robot Applications [ele] can be divided into:
Material-handling applications:

- Involve the movement of material or parts from one location to another.
- It includes part placement, palletizing and/or depalletizing, machine loading and unloading.

Processing Operations:

- Requires the robot to manipulate a special process tool as the end effectors.
- The application include spot welding, arc welding, riveting, spray painting, machining, metal cutting, deburring, polishing.

Assembly Applications:

- Involve part-handling manipulations of a special tools and other automatic tasks and operations.

Inspection Operations:

- Require the robot to position a work part to an inspection device.

- Involve the robot to manipulate a device or sensor to perform the inspection.

In this thesis most of industrial robot applications will be analyzed from the point of view of task coordination. In detail Material-handling applications will be discussed in chapter 6 whereas Assembly,Processing and Inspection applications will be analyzed in chapter 4.

2.2 System Integration issues

Related key technology challenges for pursuing successful long-term industrial robot automation are introduced at three levels: basic technologies, robot components and systems integration [EUR]. On a systems integration level, the main challenges are as follows:

- **Fine manipulation/high precision**. Installation and change-over times of robot work-cells are highly dependent on negotiating tolerances in processes, product geometries and product position/presentation. This aspect is even more emphasized as product components decrease in size (micro, nano...). A goal is to account for required precision on the basis of existing (non-precision) machines by use of increased numbers of sensors and improved sensor data processing.

- **Human-robot-collaborative work-cells**. A cooperative task execution between robot and worker can increase the overall productivity through a perfect split of capabilities This idea also extends to the vision of making robots a commodity in manufacturing and crafts.

- **Cooperating robots**. As unit prices drop at increasing rates, the cost of typical robot peripherals (conveyors, feeders, positioning devices, fixtures ...) can be drastically reduced and at the same time provide more flexibility. The result would be a network of interlinked robots which cooperatively transport, machine, handle and assemble work-pieces. A typical, simple

scenario is a robot presenting a work-piece and positioning it so that a second robot can work on that piece. RTD tasks especially comprise scalable/distributed architectures for multiple robots, so that synchronization, sensor data processing, programming, task allocation, decision making and diagnosis can be organized and managed in a distributed system.

- **Hyper flexible manufacturing systems**. Product volumes and life-times are especially uncertain for consumer goods (electrical appliances, mobile communication). An immediate change-over may give additional opportunities to react to market developments and receptivity. The adaptation to new batches, product variants or new products should be shortened by typically one order of magnitude compared to today. This should result in a consistent modularization of manufacturing systems both in terms of software (components, interfaces) and hardware (interfaces, signal, energy transmission):
  - A consistent middleware of automation modules to connect robots, peripheral devices and industrial IT systems (in a mechanical, electrical and especially logical way) without reprogramming (“Plug and Play “)
  - The “wireless shop floor “. Signal transmission should be detached from wiring and switching cabinets. Closely associated to this challenge are aspects of data security.
  - Mobile work-cells should facilitate the change-over of manufacturing lines to new compositions or, in a more advanced way to “abandon“the robot work-cell in favour of installing robots temporarily at the workplace/workbench.
  - Establishment of a life-cycle oriented consideration of production equipment (procurement, financing, planning)
- **Micro and nano-manufacturing**. As products become smaller manufacturing technology has to be scaled. However, as materials, manufacturing, processes and design principles for micro-systems differ from traditional products and manufacturing, the

development of lowcost, dependable micro manufacturing equipment constitutes a major challenges. These systems generally incorporate rich sensor capabilities for optimized process control, robotic devices for automated handling, assembly and machining of microparts. It is expected that the manufacturing of nano-systems will follow radically new and fully automated processes (from solid state physics, generative processes from biology) requiring new robotics devices, possible based on completely different motion generating principles

2.3 Implementation of industrial automation system

Nowadays, programming of automation systems is largely based on the International Electrotechnical Commission (IEC) 61131 standard [IEC03]. This standard does not meet Object Oriented Programming (OOP) requirements making programming of large distributed systems quite difficult. As a matter of fact, the standard is currently being revised to overcome this deficiency. The IEC 61131 already contains a simple class concept, the Function Block (FB) which has an internal state, a routine manipulating this state, and it may be instantiated several times [TF11]. So, extending the existing FB with object-oriented features is a natural way of introducing object orientation in the IEC 61131. This is the path that the current working draft of IEC 61131 revision is following: adding methods, inheritance and interface abstraction (polymorphism) to FB [Wer09].

OOP has demonstrated its capability in handling complex software development problems and producing flexible and reusable software components. Despite the advantages listed before, software engineering industrial experience shows that OOP technologies fail to provide all the expected benefits, particularly, in the areas of management of complexity, and performance. The reusability and adaptability of a framework or a class library can be increased by adding new features and variation points. Unfortunately, as the number of variation points increases, the complexity of the framework rapidly grows while its

performance usually deteriorates [CE00]. In industrial automation, objects are often related to physical devices built in a certain company or specific industrial domain context. Thus, in industrial automation the application of OOP is expected to be more intuitive than it has been in software engineering.

Industrial control systems are not generic computer systems and then OOP tools oriented to industrial automation should satisfy the additional requirements [Sch07]: I/O configuration and direct access to I/O signals; Multi-paradigm programming, i.e. object-programming should be optional to offer a stepwise and reversible transition to OOP; OOP should be introduced as an addition to current IEC 61131, to avoid a steep learning curve; OOP should be supported in all languages provided by IEC 61131 so that both textual and graphical languages can be used, the latter being very useful when programming sequences. Real time software requirements are implicit in industrial automation [Dou99]. From a control engineering point of view, industrial control systems are usually equipped with real time operative systems, which guarantee the meeting real time constraints.

Some programming environments have been introduced to provide OOP for industrial automation programming, see for instance references [GDF+10, 3S 12]. At best of our knowledge, only CoDeSys V3 developed by a medium-size vendor of software tools can be used for programming a significant number of industrial devices (see [3S 12] for further details). Moreover, it supports PLCopen XML import/export functionality that allows to interface such a tool with other environments to improve the design of OOP applications [PSK11]. CoDeSys meets the additional requirements listed above for OOP programming tools; it extends the IEC 61131 FB to a class construct by the addition of methods, inheritance; it introduces the INTERFACE-construct for the declaration of abstract FBs with polymorphic reference semantics.Finally, it allows dynamic memory allocation by means of __New operator.

On the other hand, there exists another IEC standard, the IEC 61499 [IEC05]. This standard, built on IEC 61131 and on the FB concept, provides an implementation-independent distributed control standard. A good introduction to the development of controllers us-

ing IEC 61499 is provided by [Lew01], while [Vya07, Vya11, DV12] give a practical introduction to using IEC 61499 controllers in industry settings. An application model in the 61499 standard is a network of interconnected FBs that are different from those in IEC 61131. FBs have now clearly defined distinct interfaces for event and data inputs and outputs. Event inputs are used to activate the FB while event outputs are used to propagate events to other FBs. Data inputs and outputs are used to pass data values between FBs. An FB can have internal variables that are fully protected, i.e. not directly accessible from the outside. The internal FB behavior is defined by an Execution Control Chart (ECC) with its own states, transitions, and actions for a basic FBs. Composite FB behavior is defined by a network of basic FBs or other composite FBs. An ECC basically describes a Moore-type finite state machine. Transition conditions between its states are evaluated whenever the FB receives an input event. Such conditions are expressed using an input event and/or a Boolean conditions. A transition between two states happens when the associated condition is satisfied. Thus, whenever an ECC is triggered by an input event, a sequence of transitions may take place within the ECC in response to that event. Each ECC state can be associated with actions, i.e. algorithms written in any of IEC 61131 languages or high level programming languages like C or Java, and an output event can be associated with signal actions' completion. The actions related to a given state are executed once upon entry to the state.

One of the main achievements of the IEC 61499 standard is the introduction of the **event-based execution order** for program organization unit (e.g., a FB or a program). Main drawbacks of 61499 are: it is very different from IEC 61131 thus requiring a steep learning curve; it supports partially OOP (inheritance is not supported); there are few commercial implementations [Thr09]. Moreover, a huge amount of manpower and money has been spent in industry in order to establish applications and libraries using IEC 61131 and this makes difficult to adopt new standards or new paradigms, at least in the next future, unless a certain compatibility with respect to 61131 is preserved [ZSSB09].

In IEC 61131 when a program organization unit (e.g., a FB or a

program) is invoked, it executes its code sequentially from beginning to end. A task is associated to each program organization unit. Each task can be configured to control the execution of a program organization unit to execute periodically or upon the occurrence of a specified trigger, i.e. an event. More precisely, events must be derived from boolean-valued signals by detecting rising and falling edges between two consecutive scan cycles (e.g. the rising of a photocell signal). Then, an event-based invocation method is available in IEC 61131. However, the invocation mechanism of a FB by a task is not supported by all IEC 61131 programming environments. Moreover, the term event-based execution order introduced by 61499 is not only related to the invocation mechanism but also to the I/O behavior. A FB may consist of several algorithms, but just some of them would be selected (even just one) to be executed if the input signals have a certain set of values (i.e., a certain event occurs), and consequently a value is given to each output signal. This is different from the fact to invoke a FB when the input signals have a certain set of values. Another problem is how to manage the concurrent behavior of the (many) FBs that constitute the automation software. Each time an event occurs (a photocell detects something, an FB completes an action, etc.) it is necessary to describe how it should be handled. A formal approach is needed since the concurrency and the logical constraints on the desired sequences of events make this a not easy task. To the purpose, a service-oriented approach can be useful. In a Service-oriented-Architecture (SoA) distributed resources provide their functionalities in form of services that can be accessed externally by clients without knowing the underlining implementation [MMLCR08, JS05]. Thus, the challenge of SoA is to reconcile the opposing principles of autonomy and interoperability. Requesters of services only need to know the external visible interface (description of a service) and rules on how to access them; the internal structure and functionality represented by a service are hidden.

2.4 Automated warehouse systems

Nowadays, it is well recognized that flexibility, modularity and reconfigurability are the main challenges in the design of manufacturing systems. Automated warehouse systems play a key role in such systems and are currently controlled using hierarchical and centralized control architectures and conventional automation programming techniques. The automated warehouse systems has been investigated since 1990 [Van99] and performs the short-term optimization of handling sequences. In particular, it usually has the objective to minimize the time to complete a little number of picking or storage operations and is based on the current state of the system.

A general real warehouse architecture, consisting of a number of aisles, each one served by a crane, an Interface System (IS) and picking positions (see Fig. 2.1), is considered in this chapter.

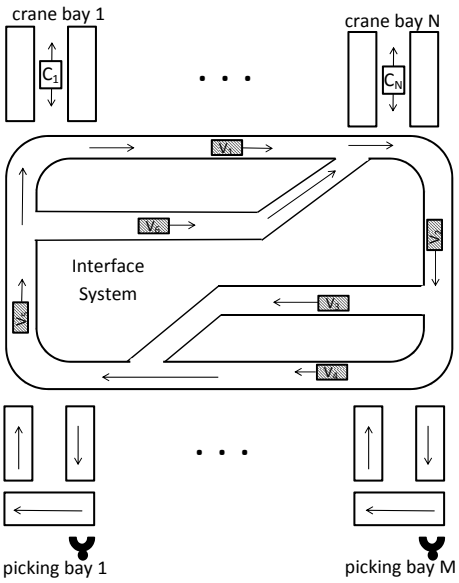


Figure 2.1 Scheme of a general real warehouse architecture: at the top aisles and crane bays (blocks $C_1 \dots C_N$ represent the cranes serving the respective aisles), at the bottom picking bays, in the center Interface System routes with running vehicles (blocks V_i, with $i = 1 \dots 6$ represent the vehicles).

On both sides of each aisle there is a storage rack composed of

n_r rows and n_k columns; moreover, as said, each aisle is served by a crane, capable of moving both vertically and horizontally at the same time, which is in charge of picking a Stock Unit (SU) at the input buffer/bay of the aisle, storing it in the assigned location, as well as retrieving and moving it to the output buffer/bay of the aisle.

The IS consists of vehicles which can move a number of SUs. The vehicles move along a mono-dimensional guidepath placed orthogonally with respect to the aisle axis. They perform picking actions (from the aisles output bays and from the picking area output bays) and deposit actions (into the aisles input bays and into the picking area input bays).

The picking area represents the output point of warehouse systems. A picking bay consists of a picking location connected via conveyors to the IS input and output interfaces, so that a SU can be partially emptied by a human operator and then carried back to an aisle rack location.

An input buffer, not reported in Fig. 2.1, represents the interface of the warehouse with the incoming area. It is used to load full SUs in the warehouse.

A set of missions is given as input to this kind of systems. Each mission requires that a certain quantity of an item, which can be stored in more than one aisle, is moved to a picking bay. Hence, the execution of a mission requires the choice of the SU to move among those containing the desired item (this choice includes also the choice of the crane since there is one crane in each aisle), the choice of a vehicle to transfer the SU to the picking area, the choice of the picking bay, again the choice of a vehicle to return the SU in the storage area and the choice of the location where the SU must be stored among those available.

The control problem consists in assigning each available resource (a location, a picking bay, a crane or a vehicle) to a mission. When one resource is available for a set of missions, a conflict occurs. The output of the control problem consists in determining Who has to do What and in Which Order in a manner that a certain objective is reached over a certain time horizon. In other words, the control must solve these conflicts. A detailed model is needed since it is important to detect in

which order these conflicts occur.

The typical control architecture for these systems consists of three different levels [ABCC05]:

- MS (Management System - level 3)
- OS (Optimizer System - level 2)
- HS (Handling System - level 1)

In this context, MS receives the loading/unloading planning from the Planning System (PS) and, based also on the *location map* (i.e., the list of the SUs stored in the warehouse and the coordinates of their location), provides to OS a set of complex handling requests, hereafter named *missions*, to be executed. It also communicates to PS the inventory items.

PS does not know where a SU is, but it only knows whether a SU is or not in the warehouse, thus it can order to pick it by sending a retrieval mission to MS; PS does not know where a SU has to be stored but it only knows that there is at least an empty location in the warehouse, thus it can send a storage mission to MS. PS works on a statistical characterization of the system performance. Its main task is material planning and lot-sizing keeping in mind the constraints concerning the inventory levels, buffer capacities and production times. Thus, PS is outside the control architecture.

MS at level 3 knows only the warehouse location map, but not the actual state of the whole plant (crane and shuttle positions, tracking position state). Therefore MS schedules the missions to be performed exclusively on the basis of the state of the warehouse location map by using its optimal management algorithm.

The movements in an automated warehouse are carried out by subdividing the given mission into several *basic handling sequences*. In this context, HS is used for the sole basic handling sequences (for example `crane_goes_to_position_X`, `shuttle_picks_SU_from_buffer`, etc.) while OS takes charge of the real-time optimization based on the current state of the plant. Therefore OS receives the list of missions from MS, reschedules and segments them into basic sequences

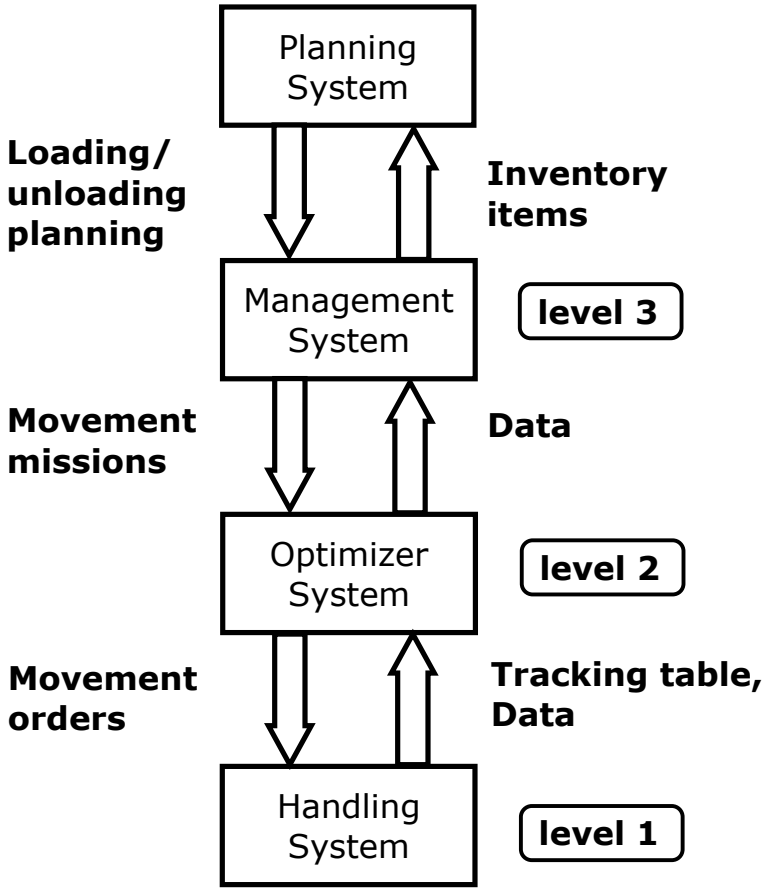


Figure 2.2 Typical control architecture of an automated warehouse system.

according to the plant state and the appropriate optimization policy; such sequences are then communicated to HS to be executed. To perform the movement, simple functions which implement those basic sequences are needed.

HS is typically implemented on a PLC, and OS can be physically separated from HS and runned on a a different machine, typically a workstation.

The hierarchical control architecture benefits from the separation of functionalities but limits the modularity and flexibility of the control system, being MS and OS implemented in a centralized way. In the

last chapter will be explained that the recent development of intelligent networked embedded systems and technologies, could help to change this.

2.5 Aircraft industry

One of the most important challenges for the next aircraft assembly lines is the increase of the level of automation. There are several reasons to pursue such an objective as the high quality standards allowed by automatized solutions or the high production rates and flexibility. These features are more and more important since aerospace production volumes have been increasing steadily over the last three years. For instance, Boeing Commercial Airplanes built more than 700 airliners in 2014 while about 650 in 2013 and 600 in 2012. For this reason, main aeronautics manufacturers are investing heavily in flexible systems to reduce costs, improve quality and boost productivity, mainly by adopting robots, automated guided vehicles and other technologies.

2.5.1 Assembly Task

As previously mentioned the use of robots in aircraft structural assembly tasks is a challenge. As for example, dedicated riveting machines are used to assemble small parts, while the assembly of large fuselage sections is usually done manually. In [SBMB09] Sahr et al. conclude that the next step in today's high-level assembly, in aerospace, is to exchange the manual labor with flexible, adaptable and affordable semi-automated systems.

The main problem in implementing semi-automated assembly solutions is the need to integrate human operators and robots together with auxiliary systems, such as a vision camera, since the strong precision requirements cannot be satisfied using industrial robots that have a limited accuracy. The current airframe assembly process of composites, metals and hybrid structures is affected by an important number of non-added value operations, which strongly causes disruptions and prevents fast ramp-up and high production rates. For example, it

can be considered one of the case study defined in the ongoing European project LOCOMACHS (LOw COst Manufacturing and Assembly of Composite and Hybrid Structures, http://www.locomachs.eu/). In particular, it consists in the simplified assembly of a generic air-craft wing-box. In such a task, spars, ribs, hinge ribs and skin panels must be properly assembled in order to reach the final configuration (see Figure 2.3). The actual wing-box assembly process begins with joining the internal structures of ribs and spars usually in huge floor-mounted fixtures. After ribs and spars have been drilled and riveted, the upper skin panels on the upper side are loaded and drilled. The structure is, then, closed by assembling the opposite panels forming the wing-box.

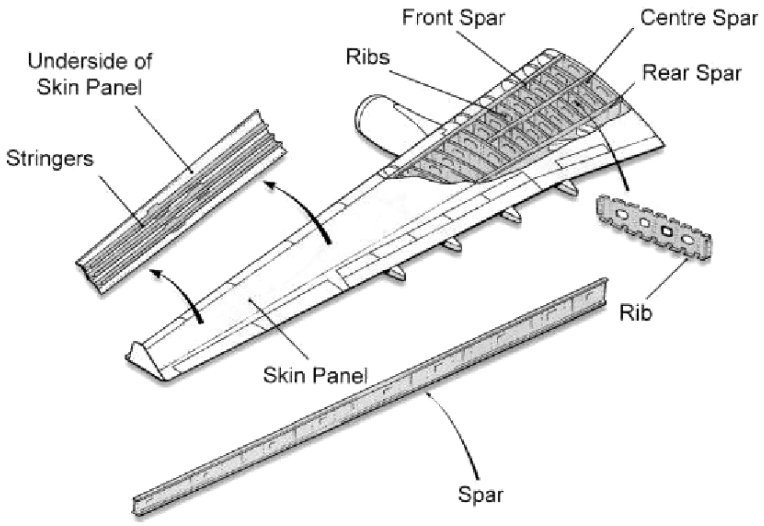


Figure 2.3 Common components in an air-craft wing-box.

In particular, spars and ribs must be accurately positioned one with respect to the other and hold while drilling and riveting. To this aim, reconfigurable fixtures are used as the one shown in Figure 2.4. Attached to this fixtures, there are several electric driven pick-ups that are basically high precision 6-DOF parallel kinematic machines. The correct part positioning is obtained by means of external metrology systems as the 6-DOF laser tracker in Figure 2.4.

In addition to fixtures and pick-ups, standard serial 6-DOF indus-

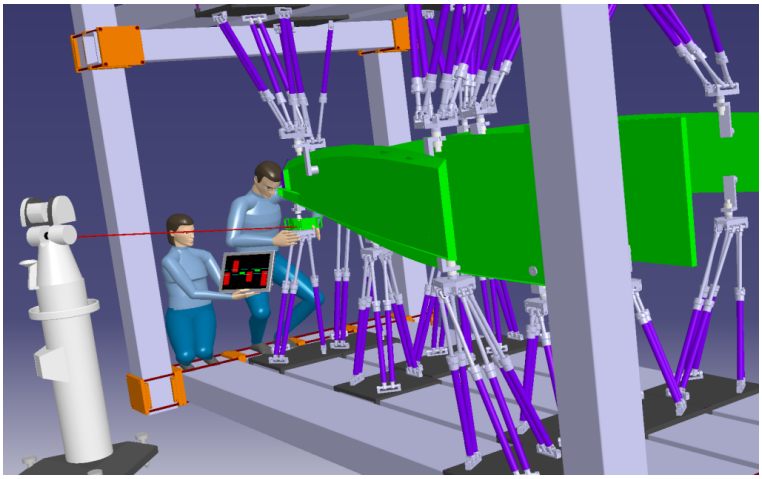


Figure 2.4 Simplified picture of a wing assembly line with fixture, pick-ups, hinges ans ribs and a laser tracker.

trial robots are used to perform the drilling/riveting operations once the positioning task is completed. Each manipulator is equipped with a drilling/riveting end-effector and other sensors useful to correctly locate the exact point to drill. In order to increase the flexibility and avoid complete off-line programming, it is supposed that a human operator roughly places the manipulator in front of the part to drill even by physically grasping the end-effector and carrying it by means of a built-in force control algorithm. By using sensor as high precision cameras and reference markers, the manipulator is able to accurately position itself at the exact point where to drill.

Moreover, quality check is required, as for example the diameter of the drilled holes must meet the admitted tolerances. To this aim, a human operator accesses to the working area and checks the hole's diameter by means of a proper probe.

2.5.2 Drilling Task

Drilling,fastener insertion, riveting, sealing, coating and painting applications, in addition to material handling, are the most recurrent operations in aircraft assembly lines. Therefore, it is clear that the automation of such operations would lead great and immediate bene-

fits to aircraft industry in terms of production rate. However, mainly because of safety motivations and government regulations, hard constraints are requested to be met, especially concerning the process tolerances.

In this section, the focus is on the drilling task since the drilling process is probably the most crucial phase of assembling airplane components due to the large number of needed holes, making it the bottleneck of the whole manufacturing process. Differently from other industrial contexts, the tighter constraints make the drilling operations a not trivial task by using low-cost industrial robots, thus, huge and expensive machines are used [OHK+10]. Recently, different approaches have been presented in order to cope with aeronautics requirements and many robotic drilling solutions have been developed. Mainly, the research activities and the industrial efforts were focused on the development of all-in-one robotic drilling systems. For example, the Electroimpact developed a robotized drilling end effector for Airbus UK Ltd. and a Kuka KR350 robot. The end effector is referred to as DDEE (Drill and Drive End Effector), and incorporates four main functions: push-up of components, drilling with panel detection, hole inspection and bolt insertion [HS01] . Furthermore, the Electroimpact proposed ONCE (ONe-sided Cell End effector), a more complete system to drill, countersink, and measure fastener holes in the wing trailing edge flaps of the Boeing F/A-18E/F Super Hornet [DSFI02]. The second generation of Electroimpact ONCE robotic drilling system [DeV09], successfully deployed in production, strove toward "oneup" assembly, whereby the product was assembled one time (drilled, countersink, inspected, and ultimately fastened) without removal of components for deburring, cleaning, sealing, etc. [DF08] . A robotic drilling system, which uses orbital hole-drilling technology, was developed by Novator AB in collaboration with Boeing, to overcome the obstacles of drilling holes in a combination of both hard metals and composites [WE09]. Another robotic drilling system for titanium structures was presented in [BL11] . The system functions include locating workpiece with a calibration stick or the vision system, weld mark inspection, one-sided clamping, drilling and reaming hole in material stack combinations of titanium and aluminum, and real-time thrust force

feedback. In [OHK+10] and [ORJ07] , it is proposed a method for high-precision drilling using an industrial robot with high-bandwidth force feedback, which is used to eliminate the sliding movement (skating) of the end effector during the clamp-up to the workpiece surface. In [BL11] , an heavy off-line programmed robot and a complex drilling tool are adopted for drilling titanium components. As mentioned before, all these solutions involve complex CNC machines or very heavy, customized robots in order to accurately counter-balance the forces involved in the drilling process and provide the necessary rigidity to the system [Isa09] . These systems fulfill all the requirements but are very expensive and limited to a particular production. Moreover, although all of the aforementioned systems are valid solutions, it is necessary for the aviation industry of different countries to develop their own robotic drilling systems because of intellectual property and various products to assembly.

2.5.3 Low-cost and flexible architecture for robotized drilling

This section shows a possible architecture to perform the drilling task by using low-cost components. In particular, a low-accuracy robot (with respect to the tolerances admitted by the aeronautics industry) equipped with a force sensor and an off-the-shelf drilling tool is adopted. The adoption of a force/torque sensor in the task at hand might be required for several reasons [AD94] . In the case reference jigs are not used, it allows to control the force in the drilling direction while minimizing the tangential forces to prevent skating phenomena. On the other hand, force control algorithms are needed in presence of reference jigs in order to take into account the imperfect knowledge of the drilling tool position with respect to the drilling mask and properly control the force along the drilling direction. The architecture is described both in terms of required hardware and designed software, providing all the necessary details to lead to the final solution starting from off-the-shelf robots.

The setup described below is installed at the Automatic Laboratory of the University of Salerno. A SmartSix robot manifactured by Co-

Figure 2.5 Robot with Drilling tool.

mau is adopted to drill aluminium components. This is a small size 6 Degrees Of Freedoms (DOFs) 6 kg payload serial chain robot with an anthropomorphic structure as shown in Figure 2.5. This robot is mainly used in automotive industry for arc-welding, sealing or painting operations. The robot is equipped with a C4G controller that allows, among the other things, the control of the robot by using a standard external PC. This is nowadays a very common feature even for industrial robots (like the considered one), thus, not underminining the generality of The part to be drilled, it is an aluminum panel held by a flexible fixture.

Figure 2.6 shows an overview of the software architecture adopted for the drilling system. The main part of the architecture is represented by the control unit that is in charge of gathering all the information coming from the different peripherals (including the robot), elaborating the control strategy and sending back the control commands. The control unit is a standard PC running a Linux operative system that has been patched with a freely available real-time extension in order to get a real time operative system [MDP00] . In order to produce efficient and reusable code, all the software for the control of the system was

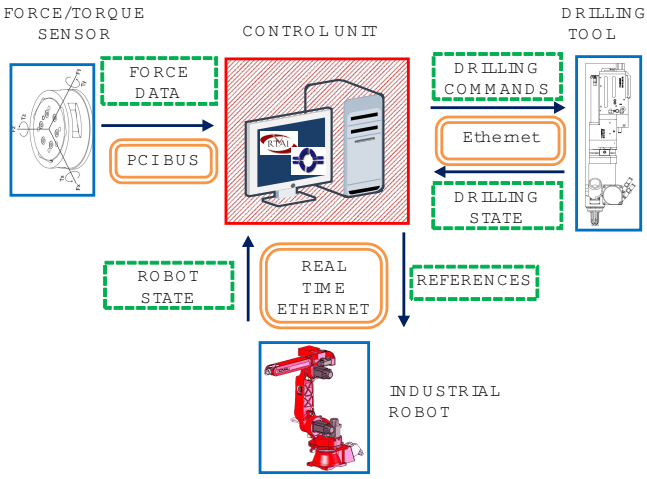


Figure 2.6 Software architecture.

written by using the Open Robot Control Software project (OROCOS) framework [Bru01] , [oro] released under Free Software licenses. The aim of OROCOS is to develop a general-purpose, free software, and modular framework for robot and machine control. It is sometimes referred to as middle-ware because it sits between the application and the operating system and it takes care of the real-time communication and execution of software components. OROCOS is designed in order to naturally allows the modularity and re-usability of the software. In fact, each one of the devices shown in Figure 2.6 is associated to an OROCOS component that gathers information from it and sends back commands. Information among the different components is ensured by statically or dynamically defining components connectivity. This choice allows to easily add new functionalities to the system. For example, it is planned the adoption of a camera in order to locate the position of the reference jig or of a thermal imaging camera for monitoring the temperature during the drilling process, to on-line adapt the process parameters and improve the quality of the final hole. As stated above, the robot chosen for this setup is a Comau SmartSix (Figure 2.5), the smallest among the Comau robots. The C4G control unit communicates with the control PC through a real-time communication on an Ethernet network, at a frequency of 500Hz. At each time step, the state of each robot axis (position, velocity, motor current and other

monitoring information) are sent to the control PC which computes the reference input. Concerning the latter, several operation modes are allowed; in particular it is possible to send:

- joint position references and use the built-in Comau position controllers (position mode);
- velocity references and use the built-in Comau velocity control controllers (velocity mode);
- current references by allowing the user to design its own control strategy (current mode).

Since one of the objective of the activity was to develop an as general as possible solution, the first operation mode was selected. In fact, almost all industrial robot controllers allow to externally generate joint position commands, while only few controllers allow the velocity and the current mode. The drilling tool is commanded through the control PC that sends commands to the tool control box via Ethernet. A single command that activates the concentric collet, starts the drilling task and deactivates the concentric collet is available in the standard operation mode and with a cycle's duration of about 10 seconds. Finally, the force sensor is connected to the Control Unit by using a FTD-DAQ-M1PCI-6220 DAQ card by National Instruments working on a PCI bus.

2.5.4 Elements of a generic hyperflexible cell for aircraft industry

As describes in the previous section, industrial robots usually work with other things: processing equipment, work parts, conveyors, tools and perhaps human operators. One of the contributions of this thesis consists in presenting a new methodology to obtain a model oriented to the control the sequencing of the activities of a robotic hyperflexible cell. The cell behavior results to be hybrid since many components, including human operators, can be abstracted as a discrete event system, but the behavior of the robots should be represented in a 3D animation environment and modeled as a continuous variable system. Many

commercial tools can easily model the robot kinematics and dynamics, such as Delmia from Dassault Systems and Robcad by Siemens, but they are not adequate to represent discrete event dynamics. To overcome the problem, in the approach proposed in [dAVJ11] the information flows in both directions, from discrete event simulation environment to 3D animation one and viceversa. In detail, a Coloured Petri Net(CPN) is used to model the cell behavior at a high level of abstraction. It models the activities of each cell component and its coordination by a supervisory system. A CPN model can command the robot, but the robot behavior is also subject to physical restrictions, such as the position, velocity, acceleration and torque joint limits. The successful execution of a command or the occurrence of a collision must be informed by the 3D animation environment to the CPN simulator and changes the CPN evolution. Flordal et al. in [HMKD07] presents a method that makes use of information in a 3D robot simulation environment in order to automatically extract finite state models. These models are used in the design of supervisory system for multiple robots. The workspace is divided into zones and the supervisory system coordinates the access of each robot to the zone. The finite state model generates supervisors that avoid deadlocks. Differently from [dAVJ11] the approach proposed in this thesis aims to obtain a single model, based on hybrid CPNs, able to capture the discrete event as well as the continuous time phenomena involved in the cell. This allows to obtain more compact models with respect to approaches based on automata and to take explicitly into account the robot trajectories avoiding the extraction of information from other frameworks as in [HMKD07]. Branicky and Chhatpar in [BC01] propose a single model, based on hybrid automata, to model a force-guided robotic assembly systems. However, the focus is on the interaction between robots and environment, and so, hybrid automata with few states are obtained, while in this thesis the focus is between robots and other cell components, that lead to very complex models. In [CL12] a framework is introduced to represent robot task plans based on Petri nets (PNs). The overall model is obtained from the composition of simple models, leading to a modular approach. The framework models single and multi-robot tasks, controllable (e.g., decision to start an action) and un-

controllable (e.g., failure to track a moving object, reaching a given location) events. The approach is oriented to autonomous robot systems whose actions can have several uncertain effects. Therefore, ordinary PNs and Generalised Stochastic PNs views of the model are used to retrieve logical/qualitative and (probabilistic) performance/quantitative properties of the robot task plans, respectively. With respect to the work [CL12], this thesis focuses on resource assignment, simulation and it is based on a hybrid model.

According to the taxonomy developed in [BCC+12a], the multi-arm work-cell configuration considered in this thesis is composed by **positioners** (elements moving the workpiece) and **workers** (elements executing the work on the workpiece). Usually, workers are robotic manipulators but they can be human operator, while positioners could be either specialized devices (e.g., rotating tables, conveyors) or general-purpose robotic manipulators as well. The availability of low cost of wireless communication devices allows to know in real time when a manual operation is started or completed. Notice that the moving of a man in the cell is uncontrollable, and it may happen that a man enters in a zone where the simultaneous presence of men and manipulators is forbidden, thus requiring robots' trajectory replanning. Moreover, robots carrying sensors are also considered. The term **watchers** is used to denote these robots, whose motion must be planned according to the cooperative task in addition to that of workers and positioners.

The elements of the architecture are very different and with specific time characteristics. For this reasons, high level supervisors is needed. An approach to coordination for this kind of task will be introduced in the chapter 4.

Chapter 3

Petri Nets: Notations and Definitions

In this Chapter the formalisms used in this dissertation are briefly recalled.

At first PNs are introduced. Then, a brief overview on CPNs is furnished. For further details on PNs and on simulation of Petri Nets (PNs), the reader can refer to [Mur89] and to [BCC07a].

Finally a background on the Hybrid Petri Nets (HPNs) and on the Colored Modified Hybrid Petri Nets (CMHPN) is presented.

3.1 Petri Nets (PNs) and Colored Petri Nets (CPNs)

PNs are widely used to model and analyze manufacturing systems. The reasons are their formal semantics, graphical nature, expressiveness in representing explicitly sequential behavior and concurrency, the availability of analysis techniques to prove structural properties (invariance properties, deadlock, liveness, etc.).

A *Place/Transition* (*P/T*) net is a 4-tuple $\mathcal{N} = \rangle P, T, \mathbf{Pre}, \mathbf{Post}|$, where P is a set of w places (represented by circles), T is a set of n transitions (represented by black bars), $\mathbf{Pre} : P \subseteq T \not\in \mathbb{N}$ ($\mathbf{Post} : P \subseteq T \not\in \mathbb{N}$) is the *pre* (*post-*) *incidence* matrix. $\boldsymbol{C} = \mathbf{Post} \quad \mathbf{Pre}$ is

the incidence matrix. The net *marking* is represented as a vector $\boldsymbol{m}$ / $\mathbb{N}^m$. The marking of a place p is a scalar value m_p / $\mathbb{N}$. A transition t is enabled at $\boldsymbol{m}$ iff $\boldsymbol{m} \rightarrow \mathbf{Pre}(\cdot, t)$ and this is denoted as $\boldsymbol{m}[t\rangle$. An enabled transition t may fire yielding the marking $\boldsymbol{m}' = \boldsymbol{m} + \boldsymbol{C}(\cdot, t)$ and this is denoted as $\boldsymbol{m}[t\rangle \boldsymbol{m}'$. The symbols ${}^\bullet p$ (${}^\bullet t$) and $p^\bullet$ ($t^\bullet$) are used for the *pre-set* and *post-set* of a place p / P (transition t / T), respectively, e.g. ${}^\bullet t = \{p \;/\; P \;\; \mathbf{Pre}(p,t) = 0\}$.

As shown in Fig. 3.1(a), there is a *structural* conflict when ${}^\bullet t_i \cap {}^\bullet t_j = \emptyset$. If t_i and t_j are both enabled, the conflict becomes a *behavioral* conflict.

Let $\mathcal{S} = \langle \mathcal{N}, \boldsymbol{m}_0 \rangle$ be a *Petri net system*, where $\mathcal{N}$ is a PN and $\boldsymbol{m}_0$ is its initial marking. Marking of $\mathcal{S}$ can be (partially) observable. In such a case, it can be divided in $\boldsymbol{m} = [\boldsymbol{m}_O, \boldsymbol{m}_{uO}]$, where $\boldsymbol{m}_O(\boldsymbol{m}_{uO})$ is the marking of observable (unobservable) places. We call $P_O(P_{uO})$ the set of observable (unobservable) places.

A *firing sequence* from $\boldsymbol{m}$ is a sequence of transitions $\sigma = t_1 \dots t_k$ such that $\boldsymbol{m}[t_1\rangle \boldsymbol{m}_1[t_2\rangle \boldsymbol{m}_2 \dots [t_k\rangle \boldsymbol{m}_k$, and this is denoted as $\boldsymbol{m}[\sigma\rangle \boldsymbol{m}_k$. An enabled sequence σ is denoted as $\boldsymbol{m}[\sigma\rangle$, while t_i / σ denotes that the transition t_i belongs to the sequence σ. The function $\boldsymbol{\sigma} : T \in \mathbb{N}$, where $\boldsymbol{\sigma}(t)$ represents the number of occurrences of t in σ, is called *firing count vector* of the firing sequence σ. As it has been done for the marking of a net, the firing count vector is often denoted as a vector $\boldsymbol{\sigma}$ / $\mathbb{N}^n$. Note that, if a sequence is made by a single transition, i.e., $\sigma = t_i$, then the corresponding firing count vector is the i-th canonical basis vector denoted as $\boldsymbol{e}_i$.

A marking $\boldsymbol{m}'$ is said to be *reachable* from $\boldsymbol{m}_0$ iff there exists a sequence σ such that $\boldsymbol{m}_0[\sigma\rangle \boldsymbol{m}'$. $R(\mathcal{N}, \boldsymbol{m}_0)$ denotes the set of reachable markings of the net system $\langle N, \boldsymbol{m}_0 \rangle$.

A net system $\mathcal{S}$ is *bounded* if there exists a positive constant K such that $\boldsymbol{m}(p) \geq K, \; \emptyset \; \boldsymbol{m}(p) \;/\; R(\mathcal{N}, \boldsymbol{m}_0)$.

A net system $\mathcal{S}$ is *live* if all its transitions are live. A transition t is live under the initial marking $\boldsymbol{m}_0$ if for every marking $\boldsymbol{m}$ reachable from $\boldsymbol{m}_0$, it exists a sequence σ, fireable from $\boldsymbol{m}$, which contains transition t. In other words, whatever the net evolution, a possibility always remains for firing t.

A PN system $\mathcal{S} = \langle \mathcal{N}, \boldsymbol{m}_0 \rangle$ is said to be *reversible* if, for each

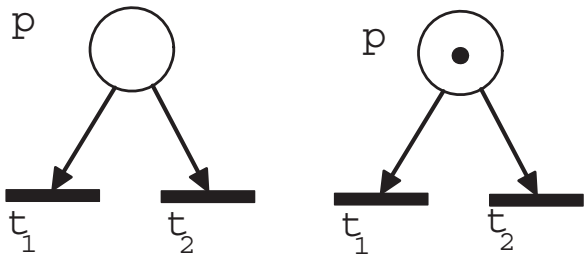
marking $\boldsymbol{m}\ /\ R(\mathcal{N}, \boldsymbol{m}_0)$, $\boldsymbol{m}_0$ is reachable from $\boldsymbol{m}$. Thus, in a reversible net one can always get back to the initial marking.

The *reachability graph* of a bounded net $\mathcal{S} = \rangle \mathcal{N}, \boldsymbol{m}_0 |$ is a directed graph RG such that: i) the root node of RG is the initial marking of the net; ii) the other nodes of RG are associated to the reachable markings of $\mathcal{S}$; iii) an arc labeled t between two nodes X and Y of RG represents that the firing of transition t leads the net system from the marking associated to the X to the marking associated to Y

A net system $\mathcal{S} = \rangle \mathcal{N}, \boldsymbol{m}_0 |$ is bounded, live and reversible iff its reachability graph is finite, strongly connected and each transition t labels at least one arc [DHP+93].

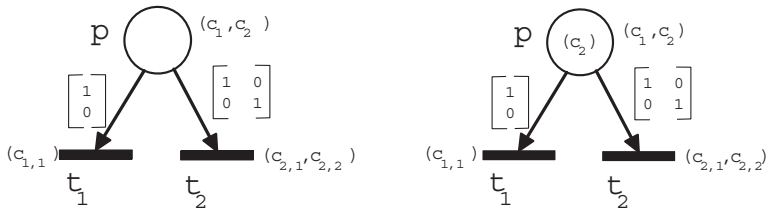
All the formal definitions given for PNs can be naturally extended to Colored PNs (CPNs). Formally, a *CPN* is a 6-tuple $\mathcal{C} = \rangle P, T, \mathbf{Pre}, \mathbf{Post}, Cl, Co |$. As in PNs, P is a set of m places (represented by circles), T is a set of n transitions (represented by bars). *Cl* is the set of colors. *Co*: $P \cap T \ \in\ Cl$ is a color function that associates to each element in $P \cap T$ a non-empty ordered set of colors in the set of possible colors *Cl*. Forall $p\ /\ P, Co(p_i) = \}a_{i,1}, a_{i,2}, ..., a_{i,u_i}\langle \leq Cl$ is the ordered set of possible colors of tokens in p_i, and u_i is their number. Forall $t\ /\ T, Co(t_j) = \big\}b_{j,1}, b_{j,2}, ..., b_{j,v_j}\Big(\leq Cl$ is the ordered set of possible occurrence colors in t_j, and v_j is$\Big($their number. For each place $p_i\ /\ P$, the marking m_i is defined as a non-negative multi-set over $Co(P_i)$. The mapping $m_i : Co(P_i) \in\ \mathbb{N}$ associates to each possible token color in P_i a non-negative integer representing the number of tokens of that color that is contained in the place p_i. The column vector of u_i non-negative integers, whose h-th component $m_{p_i}(h)$ is equal to the number of tokens of color $a_{i,h}$ that are contained in p_i, is denoted as $\mathbf{m}_{p_i}$. The marking of a CPN is an m-dimensional column vector of multisets: $\mathbf{m} = [\mathbf{m}_{p_1} ... \mathbf{m}_{p_m}]^T$. For the sake of simplicity, a token of color "$c1$" contained in a place p_i will be indicated with the symbol (c_1).

In literature, more than one formal definition for CPNs exist, depending on how the incidence matrix and transition colors are defined. In the formalism chosen in this work, matrix entries are represented by matrices. **Pre** and **Post** are the pre-incidence and post-incidence $w \subseteq n$-sized matrices, respectively. $\mathbf{Pre}(p_i, t_j)$ is a mapping from the set of



(a) (b)

Figure 3.1 Conflict in Petri Nets: (a) structural conflict and (b) behavioral conflict.



(a) (b)

Figure 3.2 (a) Unmarked CPN ; (b) Marked CPN.

occurrence colors of t_j to a non-negative multiset over the set of colors of p_i, namely, **Pre**$(p_i, t_j) : Co(t_j) \in \mathbb{N}(Co(p_i))$, for $i = 1, ..., w$ and $j = 1, ..., n$. **Pre**(p_i, t_j) represents a matrix of $u_i \subseteq v_j$ non-negative integers whose generic element $Pre(p_i, t_j)(h, k)$ is equal to the weight of the arc from place p_i w.r.t color $a_{i,h}$ to transition t_j w.r.t color $b_{j,k}$. **Post**$(p_i, t_j) : Co(t_j) \in \mathbb{N}(Co(p_i))$, for $i = 1, ..., m$ and $j = 1, ..., n$. **Post**(p_i, t_j) represents a matrix of $u_i \subseteq v_j$ non-negative integers whose generic element $Post(p_i, t_j)(h, k)$ is equal to the weight of the arc from transition t_j w.r.t color $b_{j,k}$ to place p_i w.r.t color $a_{i,h}$. The incidence matrix C is a $m \subseteq n$ matrix, whose generic element $C(p_i, t_j) : Co(t_j) \in \mathbb{Z}(Co(p_i))$, for $i = 1, ..., w$ and $j = 1, ..., n$, is the $u_i \subseteq v_j$ matrix of integer numbers $C(p_i, t_j) =$ **Post**(p_i, t_j) **Pre**(p_i, t_j). The concepts of *pre-set* and *post-set* of a place $p \ / \ P$ or a transition $t \ / \ T$ are naturally inherited from PNs, but colors must be also considered: ${}^\bullet t_i c_j = \{ t_i c_j \ / \ T \ \ \mathbf{Pre}(p_h c_k, t_i c_j) \models 0 \}$.

In Fig. 3.2(a) a CPN with a structural conflict is shown. it is made

up of a place *p*, having $C_o(p) = \}c_1, c_2\langle$, and of two transitions, t_1 and t_2 with $C_o(t_1) = c_{1,1}$ and $C_o(t_2) = \}c_{2,1}, c_{2,2}\langle$. When t_1 fires, one token, corresponding to color $c_{1,1}$, is removed from place *p*; t_2 can fire both under color $c_{2,1}$ and $c_{2,2}$ and when it fires, one $c_{2,1}$ or $c_{2,2}$ token, respectively, is removed from *p*. In Fig. 3.2(b) a token is added to the CPN in Fig. 3.2(a); notice that the conflict is still structural (not behavioral), since no $c_{1,1}$ token is present in *p* and, consequently, transition t_1 cannot fire.

When time is added to PNs and CPNs a time function is defined, which associates to each transition t_i in the case of PNs, or to each transition color $t_i c_j$ in the case of CPNs, a time duration from enabling to firing. In this case the PNs and CPNs become TPNs and CTPNs. Notice that timed and un-timed (also said immediate in the next) transitions will be represented with empty filled boxes and black bars, respectively.

3.2 Hybrid Petri Nets (HPNs)

A hybrid system is defined like a system consisting of a mixture of a continuous time system and a discrete event system (DES), having each one an own state space. These two systems are not independent but they influence each other. For the continuous time system, the influence of DES results in abrupt changes in the dynamic and can occur either as switches in the vector field or as jumps in the state. Reversely, the continuous evolution influences the DES one by generating events that affect the discrete states [PL95].

A continuous system can be described by differential equations

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t), \quad \boldsymbol{x}(0) = \boldsymbol{x}_0 \tag{3.1}$$

$$\boldsymbol{y}(t) = \boldsymbol{g}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \tag{3.2}$$

where $\boldsymbol{x} \ / \ \mathbb{R}^n$ is the state vector, $\boldsymbol{u} \ / \ \mathbb{R}^m$ is the input vector and $\boldsymbol{y} \ / \ \mathbb{R}^r$ is the output vector. In particular, if the interest is focused on the class of hybrid systems having autonomous commutations, i.e. systems for which changes in the dynamic occur if an analytical boundary

condition about the instantaneous state value is reached, the equation

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{3.3}$$

with

$$\boldsymbol{f} = \begin{cases} \boldsymbol{f}_1(\boldsymbol{x}(t), \boldsymbol{u}(t)) & for \quad \boldsymbol{h}(\boldsymbol{x}(t)) \geq 0 \\ \boldsymbol{f}_2(\boldsymbol{x}(t), \boldsymbol{u}(t)) & for \quad \boldsymbol{h}(\boldsymbol{x}(t)) > 0 \end{cases} \tag{3.4}$$

can be used, where it has been supposed the system can switch only between two possible dynamics ($\boldsymbol{f}_1$ and $\boldsymbol{f}_2$) and $\boldsymbol{h}$ is the boundary condition.

For systems having linear, time-invariant, continuous part, like the ones treated in this thesis, each dynamic in (3.4) can be written as:

$$\boldsymbol{f}_i(\boldsymbol{x}(t), \boldsymbol{u}(t)) = \boldsymbol{A}_i \times \boldsymbol{x}(t) + \boldsymbol{B}_i \times \boldsymbol{u}(t) \tag{3.5}$$

where $\boldsymbol{A}_i$ is a constant n-order square matrix and $\boldsymbol{B}_i$ is a (n$\subseteq$m)-order matrix.

To model hybrid systems behavior HPNs can be used [PL95, GU98, DA05, DPP09].

In more general hybrid systems, switching between different dynamics is caused not only by the boundary conditions but also by external input events, also called exogenous events. An exogenous event, as the term suggests, is an event originating from the outside world; by opposition, a change in internal state, as the occurrence of a boundary condition, can be called endogenous event or internal event. The external events can be "controllable" (i.e. their occurrence can be forced/disabled by an external agent, for example by a controller) or not controllable (i.e. their occurrence cannot be forced/disable by an external agent); an endogenous event is always not controllable. When changes in dynamic are ruled also by exogenous events, the HPNs used to model the system behavior are said *synchronized*, as those used in this dissertation: for these HPNs, an external event is associated with some transitions and the firing of these transitions occurs when the transition is enabled and the associated event occurs. Transitions whose firing is controlled by the occurrence of an external or internal event are called "synchronized". If the external event is a controllable event, then also transitions synchronized to such an event

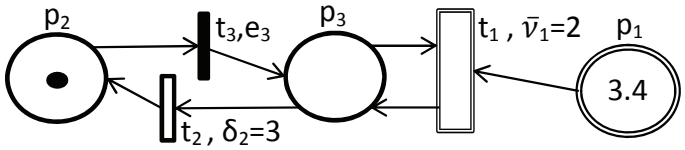


Figure 3.3 A basic HPN.

are called controllable, otherwise if a transition is synchronized to an uncontrollable event, then such a transition is said uncontrollable.

A HPN can be view as the combination of a "discrete" PN and a "continuous" PN.

Literature about HPNs is wide: a their complete presentation is given in [DA05]; in [PL95] it is shown how HPNs can be used to describe a general hybrid system having jumps in the state space and switches in its dynamic. Application of HPNs to oil refinery can be found in [WZC08a, WCZ09, WCCZ10].

Several variants of HPNs have been proposed. Differential Petri Nets (DPNs) are introduced the first time in [DK98]; in these nets the marking of a differential place may be negative as well as the weights of arcs to or from a differential place. In [DA05], it has been shown how the behavior of DPNs can be obtained using HPNs whose transitions firing speeds is a function of the net marking, and for this reason they are called Modified HPNs (MHPNs) [DA05]. Then, it is not a limitation the use of no-negative markings and weights, as it is done in this dissertation.

To model systems having first-order continuous behavior, which can be studied by linear algebraic tools, Balduzzi et al. introduce the First-Order HPNs (FOHPNs) [BGM00] and use them to model manufacturing systems [BGS01]. In FOHPNs continuous transition firing speeds are constant values, chosen by a control agent in a fixed range. When an event occurs, the net state changes, and a controller can decide to vary speed values, while between two event occurrences the firing speeds remain constant. In this thesis firing speed values are not chosen in a fixed set but they are function of the marking of the net.

In formal way, a HPN is a 7-tuple $\mathcal{H} = \rangle P, T, \mathbf{Pre}, \mathbf{Post}, h, \boldsymbol{\delta}, \boldsymbol{\nu}|$ such that: $P = P^D \bigcap P^C$, with $P^D \bigwedge P^C = \cup$, where P^D (P^C) is

the set of w_d discrete (w_c continuous) places, drawn like one (two) line circles; $T = T^D \bigcap T^C$, with $T^D \bigwedge T^C = \cup$, where T^D is the set of n_d discrete transitions, which can be both immediate (drawn like black bars) and timed (drawn like white bars) and T^C is the set of n_c continuous transitions, drawn as a two lines boxes; $\mathbf{Pre} : P \subseteq T \in \mathbb{R}^+$ is the pre-incidence matrix; $\mathbf{Post} : P \subseteq T \in \mathbb{R}^+$ is the post-incidence matrix; $h : P \bigcap T \in \}D, C\langle$, called "hybrid function", indicates for every node whether it is a discrete node (sets P^D and T^D) or a continuous one (sets P^C and T^C); $\boldsymbol{\delta} : T^D \in (\mathbb{R}^+)^{n_d}$ is the firing delay vector, whose element δ_i is the firing delay associated to each discrete transition t_i^D: if $\delta_i = 0$ then the transition t_i^D is immediate, else if $\delta_i > 0$ then t_i^D is timed. Function $\boldsymbol{\nu} : T^c \in (\mathbb{R}^+)^{n_c}$ is the firing speed vector. Note that in case of discrete nodes, $\mathbf{Pre}$ and $\mathbf{Post}$ assume integer positive values. The incidence matrix of the net is defined as $\boldsymbol{C} = \mathbf{Post} \quad \mathbf{Pre}$ and it can be written as the block matrix:

$$\boldsymbol{C} = \Big) \begin{array}{c|c} \boldsymbol{C}_{CC} & \boldsymbol{C}_{CD} \\ \hline \boldsymbol{C}_{DC} & \boldsymbol{C}_{DD} \end{array} \Big[\tag{3.6}$$

where $\boldsymbol{C}_{CC}$ is the block regarding connections between continuous nodes, $\boldsymbol{C}_{DD}$ is the block regarding connections between discrete nodes, $\boldsymbol{C}_{CD}$ is the block regarding connections between continuous places and discrete transitions and $\boldsymbol{C}_{DC}$ is the block regarding connections between discrete places and continuous transitions.

HPN marking is a function $\boldsymbol{m} = \}\boldsymbol{m}^C, \boldsymbol{m}^D($, with $\boldsymbol{m}^C : P^C \in \mathbb{R}^+$, $\boldsymbol{m}^D : P^D \in \mathbb{N}$, that assigns to each continuous place a real number and to each discrete place a nonnegative integer number of tokens (graphically represented as black dots in the discrete places). The notation $\boldsymbol{m}(\tau_k)$ is used to denote the value of the marking of the net at the instant τ_k. The marking of a place p at a time τ_k is denoted by $m_p(\tau_k)$. The symbols ${}^\bullet p$ (${}^\bullet t$) and $p^\bullet$ ($t^\bullet$) are used for the *preset* and *postset* of a place $p \ / \ P$ (transition $t \ / \ T$), respectively, e.g. ${}^\bullet t = \}p \ / \ P \ \ \mathbf{Pre}(p,t) > 0($.

A discrete transition t^D is enabled at time τ_k if $m_p(\tau_k) \ \rightarrow \mathbf{Pre}(p, t^D), \ \emptyset p \ / \ {}^\bullet t^D$. A transition t^D can be either autonomous or synchronized to a logical expression, function of an external control input g and/or of an internal condition e. Both g and

e are boolean functions $g, e : T^D \in \ \}0, 1\langle$. The former becomes true, so generating an exogenous event, when a controller sets to true the external event it is associated to; the latter becomes true, so generating an endogenous event, when the internal event it is associated to is verified. A discrete transition t^D can fire if it is enabled and the associated logical expression becomes true, i.e. both the endogenous and the exogenous events its firing is synchronized to occur. As for example, in a system formed by two masses traveling along a guidepath, an internal condition can be associated to the reaching of a threshold distance that makes masses decelerate; an external control input for the same system is an asynchronous stop command arriving from an external controller; a logical expression can be the logic function AND between g and e, e.g. $g \{ \ e$.

A continuous transition $t^C \ / \ T^C$ is enabled at time τ_k if
i) $m^D_{p^D}(\tau_k) \rightarrow \mathbf{Pre}(p^D, t^C), \ \emptyset p^D \ / \ {}^\bullet t^C$ and
ii) $m^C_{p^C}(\tau_k) \ \rightarrow 0 \ \emptyset p^C \ / \ {}^\bullet t^C$. To each continuous transition t^C_i is associated the instantaneous firing speed (in the following also called simply firing speed) ν_i: if t^C_i is disabled $\nu_i = 0$; when t^C_i is enabled ν_i is equal to the maximal firing speed $\bar{\nu}_i$, indicated near the transition. The firing of continuous transitions cannot change the marking of discrete places, consequently $\boldsymbol{C}_{DC}(p^D, t^C) = 0, \ \emptyset p^D \ / \ P^D$, thus $\boldsymbol{C}_{DC} = \mathbf{0}$. The time derivative of the marking of a continuous place p^C, $\frac{d\boldsymbol{m}_{p^C}}{dt}$, is called *balance* and it is defined as: $\dot{\boldsymbol{m}}_{p^C} = I \quad O$ where $I = \int_{t^C_j \in {}^\bullet p^C} Post(p^C, t^C_j)\nu_j$ is the *feeding speed* of the place p^C, while $O = \int_{t^C_k \in p^{C\bullet}} Pre(p^C, t^C_k)\nu_k$ is the p^C *draining speed*. The evolution of the net can be described by its fundamental equation (written in a way pointing out the continuous part and the discrete part):

$$\begin{bmatrix} \boldsymbol{m}^C(\tau_k) \\ \boldsymbol{m}^D(\tau_k) \end{bmatrix} = \begin{bmatrix} \boldsymbol{m}^C(\tau_{k-1}) \\ \boldsymbol{m}^D(\tau_{k-1}) \end{bmatrix} + \\ + \begin{bmatrix} \boldsymbol{C}_{CC} & \boldsymbol{C}_{CD} \\ \mathbf{0} & \boldsymbol{C}_{DD} \end{bmatrix} \Big(\begin{bmatrix} \mathbf{0} \\ \boldsymbol{\sigma}(\tau_k) - \boldsymbol{\sigma}(\tau_{k-1}) \end{bmatrix} + \int_{\tau_{k-1}}^{\tau_k} \begin{bmatrix} \boldsymbol{\nu} \\ \mathbf{0} \end{bmatrix} \Big) \tag{3.7}$$

where $\boldsymbol{\sigma}(\tau_k) : T^D \in \ \mathbb{N}^{n_d}$ is the discrete firing vector whose component $\sigma_{t^D_i}(\tau_k)$ represents the number of times the discrete transition t^D_i is fired up to the current time τ_k.

For the sake of clarity, from now on the term "synchronized transition" will not be used any more and synchronized transitions will be called just controllable or uncontrollable.

A basic HPN is shown in Fig. 3.3, having:

- $P^C = \{p_1\}$, $P^D = \{p_2, p_3\}$;
- $T^C = \{t_1\}$, $T^D = \{t_2, t_3\}$ where t_3 is an immediate uncontrollable discrete transition, with associated the internal condition e_3 and t_2 is a discrete timed transition;
- $\delta = \{\delta_2\}$;
- $\boldsymbol{C} = \left(\begin{array}{c|cc} 1 & 0 & 0 \\ \hline 0 & 1 & 1 \\ 0 & 1 & 1 \end{array}\right)$

For basic HPNs, the maximal firing speed of continuous transitions is a constant value, but powerful modifications have been proposed where continuous transition maximal firing speed is a function of the input places marking, of the input vector and of the time:

$$\nu_t(\tau) = f(\boldsymbol{m}(\tau), \boldsymbol{u}(\tau), \tau) \tag{3.8}$$

These kind of HPNs are called Modified HPNs (MHPNs).

3.3 Colored Modified Hybrid Petri Nets

Colored Modified Hybrid Petri Nets (CMHPNs) have been presented in [BCC12c]. A CMHPN is a four-tuple $\{\mathcal{H}, Cl, Co, \boldsymbol{\nu}\}$ where $\mathcal{H}$ is a HPN; Cl is the set of colors. Co: $P \cap T \in Cl$ is a color function that associates to each element in $P \cap T$ a set of colors; for all $t_i^C \in T^C$, $\boldsymbol{\nu}$ is the mapping $Co(t_i^C) \in \mathbb{R}^+$ that associates an instantaneous firing speed to each color of the continuous transition t_i^C. For all $p_i \in P$, $Co(p_i) = \{a_{i,1}, a_{i,2}, ..., a_{i,u_i}\} \leq Cl$ is the set of possible colors of p_i, and u_i is their number. For all $t_j \in T, Co(t_j) = \{b_{j,1}, b_{j,2}, ..., b_{j,v_j}\} \leq Cl$ is the set of possible occurrence colors of t_j and v_j is their number. For all $p_i \in P^D$, the marking

$\boldsymbol{m}_{p_i}^D$ is defined as the mapping $Co(p_i) \in \mathbb{N}$ that associates to each possible color of p_i a non-negative integer representing the number of tokens of that color contained in place p_i. For the sake of simplicity, a discrete marking w.r.t. color r is indicated as c_r. Logical expressions associated to the discrete transitions are column vectors of size v_j: their r-th element corresponds to the logical function associated to the transition firing color r.

For all $p_i \ / \ P^C$, the structured marking $\boldsymbol{m}_{p_i}^C$ is defined as the mapping $Co(p_i) \in (\mathbb{R}^+)^{(q)}$, thus, at each place $p_i \ / \ P^C$, w.r.t. the color r, a vector of q non-negative real numbers,$< x_1 \dots x_q >_r$, is associated. The q values of the marking are called "attributes" and they completely describe the state of the system.

For all $t_i^C \ / \ T^C$, $\boldsymbol{\nu}(t_i^C) = \boldsymbol{\nu}_i = (\nu_{i,1}, \nu_{i,2}, \dots, \nu_{i,v_c})^T$ is the vector of firing speeds of the continuous transition t_i^C. Its r-th element $\nu_{i,r}$, is the firing speed of t_i^C w.r.t. the color r and, when t_i^C is enabled, it is a linear, time-invariant function of the marking of the t_i^C input places and it can be written as $\boldsymbol{f}_i(\boldsymbol{x}(t), \boldsymbol{u}(t)) = \boldsymbol{A}_i\boldsymbol{x}(t) + \boldsymbol{B}_i\boldsymbol{u}(t)$ where $\boldsymbol{A}_i$ is a constant q-order square matrix and $\boldsymbol{B}_i$ is a (q$\subseteq$z)-order matrix, where z is the dimension of the input vector $\boldsymbol{u}(t)$.

Similarly, $\emptyset t_i^D \ / \ T^D$, $\boldsymbol{\delta}_i = (\delta_{i,1}, \dots, \delta_{i,v_d})^T$ is the column vector of the discrete transition t_i^D firing delays. The r-th element $\delta_{i,r}$ is the firing delay associated to the color r.

$\mathbf{Pre}(p_i, t_j)$ is a mapping $\mathbf{Pre}(p_i, t_j) : Co(t_j) \in \mathbb{R}^+(Co(p_i))$, for $i = 1, \dots, w = w_d + w_c$ and $j = 1, \dots, n = n_d + n_k$. At the same way $\mathbf{Post}(p_i, t_j)$ is defined as the mapping $\mathbf{Post}(p_i, t_j) : Co(t_j) \in \mathbb{R}^+(Co(p_i))$, for $i = 1, \dots, w$ and $j = 1, \dots, n$. $\mathbf{Pre}(\mathbf{Post})(p_i, t_j)$ is a matrix of dimensions $u_i \subseteq v_j$; the element $Pre(p_i, t_j)(r, s) = Pre_{ij,rs}$ $(Post(p_i, t_j)(r, s) = Post_{ij,rs})$ is the weight of the arc connecting p_i (t_i) w.r.t. the color r (color s) to t_j (p_i), w.r.t. the color s (color r). The nature of the element depends on the kind of nodes it connects, e.g. weighs of arcs connecting transitions to discrete places are non-negative integer numbers, while weighs of arcs connecting transitions to continuous places are row vectors of non-negative real numbers, with dimension equal to q. When weights are diagonal matrices, having all the diagonal elements equal to 1, weights are not reported near the arcs. Elements of $\boldsymbol{C}$ are the matrices $\boldsymbol{C}(p_i, t_j) =$

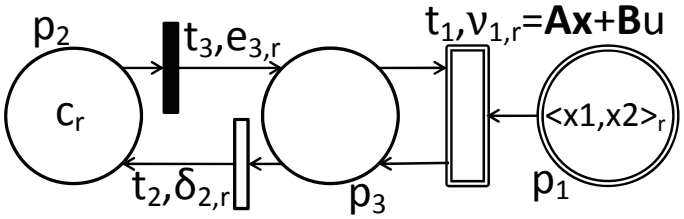


Figure 3.4 A colored modified HPN.

$\mathbf{Post}(p_i, t_j)$ $\mathbf{Pre}(p_i, t_j)$ with dimension $u_i \subseteq v_j$.

An example of colored modified HPN is shown in Fig. 3.4, where:

- continuous place p_1 has a structured marking made up of two attributes, $x1$ and $x2$;
- colors have been introduced: in Fig. 3.4 the marking of the net with respect to the generic color r is shown;
- firing delay δ_2 is a vector, having as many components as many are the occurrence colors of transition t_2;
- e_3 is a vector since a different internal condition must be defined for each occurrence color of t_3;
- firing speed of continuous transition t_1 is a linear function of the marking of p_1 preset. In particular for the example proposed, u is a scalar input and $\boldsymbol{A}$ and $\boldsymbol{B}$ are two matrices with dimensions $2 \subseteq 2$ and $2 \subseteq 1$, respectively.

Under the initial marking of p_1, continuous transition t_1 is not enabled: it becomes enabled when internal condition e_3 is satisfied and t_3 fires. Once t_1 is enabled it decreases the marking of p_1 with respect to the law 3.4 $\bigcup_{\tau_k-1}^{\tau_k} 2d\tau$ until p_1 marking becomes zero or until the firing of t_2.

Chapter 4

Automated synthesis of HCMPN models

This chapter focuses on the automated model generation of semi-automated hyperflexible robotic cells for aircraft industry. One of the contributions of this thesis consists in presenting a new methodology to obtain a model oriented to the control the sequencing of the activities of a robotic hyperflexible cell.

4.1 Definition of a robotic cell in aircraft industry

A robotic cell can be modeled from many points of view. In this chapter, the focus is on the sequencing of the activities of a semi-automated robotic cell for aircraft industry. This chapter presents a methodology that allows to obtain a formal model of such a robotic cell, automatically, from high level specifications and the behavior of available resources in terms of states, events and state transition function. A *resource* is a dynamic system that can execute elementary activities called *services*. A service is executed after the occurrence of a *service request* event, if some *conditions* depending on *resource state* are verified. A *service completion* event is generated as soon as the service has been completed. The cell specifications are expressed as *tasks* that

specify in which order to execute a set of services to complete an action, providing a desired sequence of *phases* where a subset of service request and service completion events is admissible.

The evaluation of conditions may require the availability of data (e.g. the destination position, the value of a measurement). *Input data* are available when the associated input event occurs. Input data handling is specified by *actions* that are logical expressions associated to a discrete state (task phase). Actions are evaluated when the resource (task) leaves a discrete state (task phase), that is when a state (phase) transition event occurs. *Output data* handling is specified assigning an output event to an output data, that can be a data produced by an action or an input data to be passed to another block as input. The output event occurrence time indicates when the output data is available.

Resources and tasks are considered as blocks that can be connected each others (see Fig. 4.11) by means of proper inputs and outputs.

A resource can be of two different kinds:

1. Discrete Resource (DR) - a resource that can assume only discrete states (e.g. idle or busy); for example, it can model a human operator or a buffer position.

2. Hybrid Resource (HR) - a resource that has discrete and continuous states; for example, it can model a vehicle or a robotic arm.

Definition 1. A DR is a block $DRB = (Q, q_0, I_D, O_D, \Lambda, f)$, shown in Fig. 4.11(a).

Q is the set of discrete states.

q_0 is the discrete initial state.

I_D is the input set composed of the *service request events* and of *data signals*, if they are needed for the execution of services (e.g. the position to reach at the end of a movement).

O_D is the output set composed of *service completion events*, *data signal*, if they must be provided by resources as result of the completed service (e.g. the value of a measurement), and the *current discrete state* q.

Λ is the set of conditions that are logical expressions depending on the occurrences of service request events s_r, service completion events

Figure 4.1 hmn_f

current state q	condition λ	duration $\delta(q,\lambda)$	next state q'
idle	acquisition request	0	assigned
assigned	go to pos	0	moving
assigned	start disassembling	0	inspecting
assigned	release request	0	idle
moving	at final position	0	assigned
disassembling	disassembling completed	0	assigned

s_c, internal events e (e.g., the reaching of a particular dynamic mode, the reaching of a particular discrete state, excess of a threshold by a dynamic state).

The internal behavior of a DRB is defined by means of the transition function $f : (Q \subseteq \Lambda \subseteq \mathbb{R}) \in Q$ that, for each discrete state $q \ / \ Q$ yields the next state $q' = f(q, \lambda, \delta(q, \lambda))$ that is reached when the condition $\lambda \ / \ \Lambda$ is true and the time duration $\delta(q, \lambda)$ is elapsed. If $\delta(q, \lambda)$=0 the state transition from q to q' is said *immediate* otherwise is called *timed*.

The following instructions are used to define a discrete resource:

- *DefineDiscreteResource(name, Q, f, q_0, I_{s_r}, O_{s_c})*

  where *name* is the symbolic name that univocally identifies the resource; I_{s_r} is the set of the input service request events and O_{s_c} is the set of the output service completion events;

- *AssignData(name, eventDataFunction)*
  where eventDataFunction: $\Lambda \in \mathfrak{D}$, *where $\mathfrak{D}$ is the set of data signals, is a function that associates input and output data to the corresponding transition event;*

- *AssignActions(name, actionTable)*
  where *actionTable* is the table that specifies for each discrete state the associated actions and the input data required for its evaluation.

As an example, consider the problem of modeling a human operator that can enter in a robotic cell to move the arms close to a desired

Figure 4.2 hmn_eDFunction

event		data
input event	output event	
go to pos		posDes

position or to execute some manual operations. Assume that its discrete states are hmn_Q=}idle, assigned, moving, disassembling⟨; its input service request set is hmn_I_{s_r}=}acquisition request, go to pos, release request⟨ while its set of the output service completion events is hmn_O_{s_c}=}at final position⟨; the transition function f and eventDataFunction are shown in table of figure 4.1 and 4.2, namely respectively hmn_f and hmn_eDFunction.

Then the discrete resource Human can be defined with the instructions

DefineDiscreteResource(Human, hmn_Q, hmn_f, "*idle*", hmn_I_{s_r}, hmn_O_{s_c})

AssignData(Human, hmn_eDFunction).

Since human operator must not perform any action, there are not any actions associated to any state, table actionTable is not defined and the instruction *AssignActions(name, actionTable)* is not used.

HRs have discrete states but they have also a continuous state $\boldsymbol{x}$ and a continuous linear time invariant dynamic law $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}_i\boldsymbol{x}(t) + \boldsymbol{B}_i\boldsymbol{u}(t)$ for each continuous mode, where $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ are matrices and $\boldsymbol{u}(t)$ is an external input. As example, usually a robot trajectory has three continuous modes (constant speed, constant acceleration or constant deceleration) and so a HR state can be represented by the attributes vector $\boldsymbol{x} =< \text{position}, \text{velocity} >$, and the input $u(t)$ is the constant acceleration, a.

Definition 2. A HR is a block $HRB = (DRB, X, \boldsymbol{x}_0, C_i, \boldsymbol{u}, \boldsymbol{x})$, shown in Fig. 4.11(b).

$DRB = (Q, q_0, I_D, O_D, \Lambda, f)$ is a discrete resource corresponding to the discrete states of the HR.

X is the set of the continuous states.

$\boldsymbol{x}_0$ is the initial continuous state.

C_i is the triple $(\boldsymbol{A}_i, \boldsymbol{B}_i, Q_i)$, where $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ are the matrices associated to the i-th continuous mode and $Q_i \leq Q$ is the set of discrete states where the i-th continuous mode is admissible.

$\boldsymbol{u}$ is the continuous time *input*.
$\boldsymbol{x}$ is the *continuous state*.
The following instructions are used to define a hybrid resource:

- *DefineHybridResource(name, DRB, X,* $\boldsymbol{x}_0$ *continuousModes, hybridTable);*

  where *name* is the resource name, *continuousModes* is the set of symbolic names that univocally identify each continuous mode, *hybridTable* is the table that associates a dynamic law to each continuous mode (i.e. for each continuous mode it specifies the triple $C_i = (\boldsymbol{A}_i, \boldsymbol{B}_i, Q_i)$ and the input vector $\boldsymbol{u}$).

  Data and actions assignment for the HR is accomplished by using the same two instructions introduced for discrete resources.

Figure 4.3 wtc_f

current state q	condition λ	duration $\delta(q,\lambda)$	next state q'
idle	acquisition request	0	assigned
assigned	go to pos	0	moving
assigned	start inspection	0	inspecting
assigned	release request	0	released
released	(acquisition request $\bigvee$ traveling at constant speed)	0	assigned
released	at home position	0	idle
moving	at final position	0	assigned
inspecting	inspection completed	0	assigned
accelerating	go at constant speed	0	constant speed
accelerating	at deceleration position	0	decelerating
constant speed	accelerate	0	accelerating
constant speed	(too fast at deceleration position)	0	decelerating
decelerating	(at final position at home position)	0	stopped
stopped	(go to pos release request)	0	accelerating

As an example, consider the problem of defining a watcher as a resource that can move with constant speed, as well as accelerating or decelerating and that, as more, is able to execute inspections at predefined positions.

Assume that its discrete states are *wtc_Q* = }idle, assigned, released, moving, inspecting, accelerating, constant speed, decelerating,

stopped$\langle$, while its continuous modes are *wtc_cModes* = }acc, const, dec$\langle$. The continuous state is described by the vector $\boldsymbol{x} =< x1, x2 >$, where continuous variable $x1$ and $x2$ are associated to watcher position and velocity, respectively. The initial continuous state is $\boldsymbol{x}_0 =< HomePos, 0 >$, where $HomePos$ is the watcher home position. The resource input events are *wtc_*I_{s_r} = }go to pos, start inspection, acquisition request, release request$\langle$ while its output events are *wtc_*O_{s_c} = }inspection completed, at final pos$\langle$. Moreover, the transition function f, represented by the table named *wtc_*f, and the hybridTable, named *wtc_hTable*, are shown respectively in table of figure 4.3 and in table of figure 4.4, where $\boldsymbol{A}_1 = \boldsymbol{A}_2 = \boldsymbol{A}_3 = \left] \begin{matrix} 0 & 1 \\ 0 & 0 \end{matrix} \right\{$, $\boldsymbol{B}_1 = \boldsymbol{B}_3 = \left] \begin{matrix} 0 \\ 1 \end{matrix} \right\{$, since the continuous mode equations are respectively $(\dot{x_1} = 0, \dot{x_2} = a)$, $(\dot{x_1} = 0, \dot{x_2} = 0)$ and $(\dot{x_1} = 0, \dot{x_2} = \quad a)$ for acceleration, constant speed and deceleration mode.

Figure 4.4 wtc_hTable

continuous mode	discrete state q	$A_i,\ B_i,\ u$
acc	accelerating	$\boldsymbol{A}_1,\ \boldsymbol{B}_1,\ a$
const	constant speed	$\boldsymbol{A}_2,\ \boldsymbol{0}, 0$
dec	decelerating	$\boldsymbol{A}_3,\ \boldsymbol{B}_3,\quad a$

Then, a watcher can be defined using the instruction

DefineHybridResource(Watcher, wtc_Q, wtc_f, "*idle*$\bigvee$*stopped*", wtc_I_{s_r}, wtc_O_{s_c}, wtc_cModes, $< HomePos, 0 >$, wtc_hTable)

AssignData(Watcher, wtc_eDFunction)

AssignAction(Watcher, wtc_aTable)

In table of figure 4.5 the eventDataFunction associated to resource Watcher, named *wtc_eDFunction*, is reported. In table of figure 4.6, the actionTable associated to resource Watcher, named *wtc_aTable*, is shown.

Figure 4.5 wtc_eDFunction

event		data
input event	output event	
go to pos		posDes
release request		homePos
	inspection completed	result

Figure 4.6 wtc_aTable

$q[> q'$	actions	data
assigned [> moving	posDes:=destPos	destPos
assigned [> released	posDes:=homePos	homePos
accelerating [> constant speed	posDec:=f(posDes,x_2,v_{max})	
constant speed [> accelerating	posDec:=f(posDes,x_2,v_{max})	
stopped [> accelerating	posDec:=f(posDes,x_2,v_{max})	

Definition 3. A task is a block $TB = (S, s_0, I_T, O_T, \Lambda, \phi)$, shown in Fig. 4.11(c).

S is the set of phases.

s_0 is the initial phase.

I_T is the input set composed of *service completion events*, resources *current discrete and continuous states*, *data*, when provided at the end of a service.

O_T is the outputs set composed of *service request events* and *data*, if needed to execute services.

Λ is the set of conditions.

The internal behavior of a TB is defined by means of the transition function $\phi : (S \subseteq \Lambda) \in S$ that specifies the task internal evolution: for each phase $s \ / \ S$ identifies the next phase $s' = \phi(s, \lambda)$ that is reached when a condition $\lambda \ / \ \Lambda$ is true.

The following instructions are used to define a task:

- *DefineTask(taskName, S, ϕ, s_0, I_{s_c},O_{s_r})*;

  The fields have the following meaning: *taskName*, the symbolic name of the task that univocally identifies it; I_{s_c}, the input service completion events set; O_{s_r}, the output service request events set.

- *DefineLinks(taskName, resourceFunction)*;

  resourceFunction: $\Lambda_s \in (\mathfrak{R} \subseteq \mathbb{N})$, with $\Lambda_s \ / \ \Lambda$= set of service request and completion events and $\mathfrak{R}$=set of (discrete and hybrid) defined resources, is the function that indicates for each service request and completion which are the involved resources and how many resources of that kind are necessary to complete the service.

Data and actions assignment for the task is accomplished by using the same two instructions introduced for the resources, with the only difference that field *name* of both instructions contains the task name and that *actionTable* indicates the associated actions and the relative required input data for each task phase s.

As for example consider the case to model a drilling task, described as follow:

Drilling task. A drill hole has to be done by the worker in an unconformable position. Desired position is first reached by the watcher that, when arrived, provides to the worker the final destination. The watcher waits until the worker reaches the drilling position.

The drilling task can be defined using the instructions:

DefineTask(Drilling, drl_S, drl_ϕ, "*ready*", drl_I_{s_c}, drl_O_{s_r})

DefineLinks(Drilling, drl_rFunction).

The sets of Drilling task phases, service request and completion events are the following: drl_S=}ready, wtc aquired, wtc moving, waiting for wrk, wrk acquired, wrk moving, wrk arrived, wtc released, drilling, drilling done, end⟨

drl_I_{s_c}=}wtc at final position, wrk at final position, drilling completed⟨;

drl_O_{s_r}=}wtc acquisition request, wtc go to pos, wrk acquisition request, wrk go to pos, wtc release request, start drilling, wrk release request⟨.

The transition function ϕ and the resourceFunction associated to Drilling task are shown in table of figure 4.7 and 4.8, and namely drl_ϕ and drl_rFunction, respectively.

The occurrence of conditions, in the general case, depends on the reaching of particular discrete states q (phases s) or on the reaching of particular values of the continuous state $\boldsymbol{x}$ of a HR.

At each condition λ, a logical expression has to be assigned.

The entering in a discrete state, or in a particular phase, corresponds to the occurrence of the boolean event *discreteState=true* (*phase=true*) and the associated logical expression can be written using the formalism nameresource.q=true (nametask.s=true) , where notation nameresource.q (nametask.s) is used to refer to the particular state q (phase s) of the indicated resource (task).

Figure 4.7 drl_ϕ

current phase s	condition λ	next phase s'
ready	wtc acquisition request	wtc acquired
wtc aquired	wtc go to pos	wtc moving
wtc moving	wtc at final position	waiting 4 wrk
waiting 4 wrk	wrk acquisition request	wrk acquired
wrk acquired	wrk go to pos	wrk moving
wrk moving	wrk at final position	wrk arrived
wrk arrived	wtc release request	wtc released
wtc released	start drilling	drilling
drilling	drilling completed	drilling done
drilling done	wrk release request	end

Figure 4.8 drl_rFunction

event		resource	number
input event	output event		
wtc at final position		Watcher	1
wrk at final position		Worker	1
drilling completed		Worker	1
	wtc acquisition request	Watcher	1
	wtc go to pos	Watcher	1
	wtc release request	Watcher	1
	wrk acquisition request	Worker	1
	wrk go to pos	Worker	1
	wrk release request	Worker	1
	start drilling	Worker	1

Also the reaching of a particular continuous state is a boolean event. Possible conditions to test are $=, >, <, \not\to, \geq, \not\geq, \not\leq$.

As example, let R_j and R_k be the name of two HRs, event $e_1 = (< x_1, \ldots, x_n >^j \quad < T_{h1}, \ldots, T_{hn} >) \to \mathbf{0}$ occurs when each attribute $R_j.x_i$[1] are equal to or greater than the corresponding threshold value T_{hi}, while event $e_2 = (< x_1, x_2, \ldots, x_n >^j \quad < x_1, x_2, \ldots, x_n >^k < T_{h1}, 0 \ldots, 0 > \not\leq \mathbf{0})$ occurs when $R_j.x_1 \quad R_k.x_1 \geq T_{h1}$.

Logical expressions associated to service request or service completion events correspond to the condition itself. Assignment of a logical expression to each condition λ is done by means of the following

[1]Notation $R_j.x_i$ is used to indicate the single i-th attribute of R_j while notation $R_j.\boldsymbol{x}$ indicates the whole vector.

Figure 4.9 wtc_condTable

condition λ	definition
go to pos	go to pos
inspection completed	inspection completed
at home position	$< x_1, x_2 >^{Watcher} =< HomePos, x_2 >$
go const	$< x_1, x_2 >^{Watcher} =< x_1, vDes >$
traveling at constant speed	$Watcher.const = True$

instruction:

- *AssignLogicalExpression(name, conditionTable);*

Field *name* identifies the resource (task) the logical expressions have to be assigned to; field *conditionTable* is a table that associates logical expressions to the conditions. As example, in table of figure 4.9 a part of the conditionTable, defined for the resource Watcher and named *wtc_condTable*, is reported.

Note that a logical expression may reduce to a single event as is the case of condition *go to pos* in table of figure 4.9.

- *InstantiateBlocks(nameBlock, name1, . . . , nameI);*

This instruction allows to define instances of resource and task blocks. Field *nameBlock* is the name of the resource (task) block while field *name1, . . . , nameI* are the names of the I instances of the block.

4.2 Creation of the CMHPN model

After the definition of the robotic cell, the corresponding CMHPN model can be automatically created in two steps: at step 1, CMHPN of each task T_i and each resource R_j is created; at step 2, such nets are merged together to obtain the final model.

For the sake of readability, the list of the notations used in this section is shown in table of figure 4.1.

4.2.1 Algorithm to obtain tasks and resources CMHPN model

Let $(q, \lambda, \delta(q, \lambda))$ be a triple belonging to transition function f domain, each time a DR of kind j is defined, a timed PN is built as indicated:

1. add a discrete place p_q^{Rj} for each state $q \ / \ Q$;
2. add an immediate transition t_h^{Rj} for each triple $(q, \lambda, \delta(q, \lambda))$ and synchronize it with the corresponding condition λ; if $\delta(q, \lambda) > 0$, add an additional place $p_q^{*Rj}$and a timed transition t_h^{*Rj};
3. associate to t_h^{*Rj} the time duration $\delta(q, \lambda)$;
4. associate to t_h^{Rj} the logical expression reported in table conditionTable, associated to λ;
5. associate to t_h^{Rj} the data eventDataFunction(λ), if it exists;
6. for each triple $(q, \lambda, \delta(q, \lambda))$ add an arc going from place p_q^{Rj}, associated to state q, to transition t_h^{Rj}, associated to λ;
7. for each state $q' = f(q, \lambda, \delta(q, \lambda))$ with $\delta(q, \lambda) = 0$, add an arc going from transition t_h^{Rj}, associated to condition λ, to place $p_{q'}^{Rj}$ associated to state q';
8. for each state $q' = f(q, \lambda, \delta(q, \lambda))$ with $\delta(q, \lambda) > 0$ add an arc going from transition t_h^{Rj}, associated to condition λ, to the additional place p_q^{*Rj}, associated to state q, an arc going from place p_q^{*Rj} to transition t_h^{*Rj} associated to $\delta(q, \lambda)$ and an arc going from transition t_h^{*Rj} to place $p_{q'}^{Rj}$ associated to state q'.

Each time a HR of kind j is defined, a MHPN is built as indicated:

1. execute all the steps shown for the construction of the DR net.
2. add a continuous place p_c^{Rj};
3. add a continuous transition t_h^{Rj} for each continuous mode in the set continuousModes;

4. add a self loop connecting each continuous transition t_h^{Rj} with continuous place p_c^{Rj};

5. associate weighs $\mathbf{Pre}(p_c^{Rj}, t_h^{Rj})$, $\mathbf{Post}(p_c^{Rj}, t_h^{Rj})$ to each arc: they are vectors of dimension equal to the number of the attributes of continuous state $\boldsymbol{x}$ with all elements equal to 1.

6. associate the firing speed $\boldsymbol{\nu}_h^{Rj} = \boldsymbol{A}_i\boldsymbol{x} + \boldsymbol{B}_i\boldsymbol{u}$ at each transition t_h^{Rj} as indicated in table hybridTable;

7. for each continuous mode indicated in hybridTable, add a self loop between transition t_h^{Rj}, corresponding to the continuous mode, and place p_q^{Rj}, corresponding to the discrete state associated to the continuous mode.

Each time a task of kind i is defined a discrete PN is built as indicated:

1. execute steps 1-4 and 6 of the algorithm for the construction of DR model, substituting symbols "p_q^{Rj}", "t_h^{Rj}", "q", "Q" and "f with "p_s^{Ti}", "t_k^{Ti}", "s", "S" and "ϕ", respectively;

2. for each phase $s' = \phi(s, \lambda)$ of transition function ϕ add an arc going from transition t_k^{Ti}, corresponding to condition λ, to place $p_{s'}^{Ti}$, associated to phase s';

3. for each couple
(resource, number)= resourceFunction(s_r) resourceFunction(s_c)[
associate to transition t_k^{Ti}, corresponding to the service request s_r (service completion s_c), a number $w_{kj}^{T_i}$ of value equal to the number of resources R_j involved in the service. Notice that $w_{kj}^{T_i}$ is not an arc weight: it will be used later, during the merging procedure.

Add as many colors to each place and each transition of the corresponding net, as many time R_j (T_i) has been instantiated and add a token of color c_r (c_t) to the place corresponding to the initial state q_0 (phase s_0) of R_j (T_i). If a HR is instantiated, then add a structured

marking at the corresponding place p_c^{Rj}, corresponding to the initial continuous state $\boldsymbol{x}_0$ of the resource.

Applying the algorithm defined above to the HR Watcher, the MHPN of Fig. 4.17(b) is obtained. It has 9 discrete places, one for each discrete state of set *wtc_Q* (e.g., p_{12} and p_{13} correspond to states *idle* and *assigned*, respectively) and it has 14 discrete transitions, one for each triple $(q, \lambda, \delta(q, \lambda))$ of function wtc_f domain, and no timed transition is present in the final model since no timed state transition has been specified (e.g., t_{11} corresponds to the first row of table of figure 4.3 and it is associated to condition s_{r1} =*acquisition request*; no data are associated to t_{11}). The continuous place p_{c1} and one continuous transition for each one of the 3 continuous mode of set *wtc_cModes*, with the respective firing speeds, are added; these transitions are connected by means of self loops to p_{c1} and to the corresponding discrete places as indicated in table of figure 4.4 (e.g., t_{CON1} is associated to the continuous mode *const*, and, as specified in table of figure 4.4, the firing speed $\boldsymbol{\nu}_2 = \boldsymbol{A}_2\boldsymbol{x}$ is associated to t_{CON1}; finally since p_{18} corresponds to the state *constant speed*, a self loop between t_{CON1} and p_{18} is inserted). Note that, since HRB Worker is instantiated just once, an uncolored net is obtained.

4.2.2 Merging of PNs

The final model of the robotic cell is obtained merging together transitions of tasks with transitions of resources, when they are synchronized with the same event.

Assume that N_T different kinds of task have been defined, each kind indicated with notation T_i, with $i = 1, \ldots, N_T$ and that N_R different kinds of resource have been defined, each one indicated with notation R_j, with $j = 1, \ldots, N_R$. Each task (resource) of kind T_i (R_j) has been instantiated v_{T_i} (v_{R_j}) times, consequently, the places and transitions of the CMHPN modeling T_i (R_j) have v_{T_i} (v_{R_j}) colors; each instance of T_i (R_j) is identified as T_i^t (R_j^r).

The merging algorithm is shown in Fig. 4.12. It is composed by 3 steps: at Step 1 each transition t_k^{Ti} of T_i, synchronized with service request s_r (service completion s_c) is merged with each transition t_h^{Rj} of

R_j, synchronized with the same service request (service completion); Step 2 consists in appropriately changing the color sets of such places and transitions of T_i that have not been involved in merging with R_j; finally during Step 3, the final model is obtained, deleting all merged transitions of R_j.

Step 1 is executed by means of the merging subroutine, shown in Fig. 4.13, that is composed by the following 5 steps:

Step 1.1: Task transition t_k^{Ti} and each transition t_h^{Rj}, synchronized with the same event, are merged together obtaining a unique transition $t_{k'}^{Ti}$. Preset and postset of new transition are obtained by the union of preset and postset of t_k^{Ti} and each t_h^{Rj}.

Step 1.2: The set of occurrence colors of $t_{k'}^{Ti}$ is defined. Its number depends on the number v_{R_j} of instances of R_j and on the number w_{kj}^{Ti} of its instances involved in the service s_r (s_c), as well as on the occurrence color number[2] $v_{t_k^{Ti}}$, and it is devised as $v_{t_{k'}^{Ti}} = v_{t_k^{Ti}} \times {}^{v_{R_j}}\!\big[_{w_{kj}^{Ti}}$.

Step 1.3: The matrices of incidence of places ${}^\bullet t_h^{Rj}$, as well as of places $t_h^{Rj\bullet}$, are opportunely defined. Since colors of such places are not changed after the merging, $\mathbf{Pre}({}^\bullet t_h^{Rj}, t_{k'}^{Ti})$, as well as $\mathbf{Post}(t_h^{Rj\bullet}, t_{k'}^{Ti})$, is a matrix obtained duplicating matrix $\boldsymbol{M}_R$ $v_{t_k^{Ti}}$ times.

Matrix $\boldsymbol{M}_R$ has dimension $v_{R_j} \subseteq {}^{v_{R_j}}\!\big[_{w_{kj}^{Ti}}$, each one of its column has w_{kj}^{Ti} element equal to 1 and v_{R_j} w_{kj}^{Ti} elements equal to 0 and all columns are different from each others.

Step 1.4: Colors of places of task T_i are changed on the base of the kind of event $t_{k'}^{Ti}$ is synchronized with. When $t_{k'}^{Ti}$ is synchronized with an acquisition request (a release request), colors of p / ${}^\bullet t_k^{Ti}$ (p / $t_k^{Ti\bullet}$) do not change. Consequently, $\mathbf{Pre}(p, t_{k'}^{Ti})$ ($\mathbf{Post}(p, t_{k'}^{Ti})$) is set equal to $\boldsymbol{M}_T$ (see table of figure 4.1). When $t_{k'}^{Ti}$ is synchronized with any other event

[2]Before merging, $v_{t_k^{Ti}} = v_{T_i}$ = number of instances of T_i.

color set of a place $p \ / \ {}^{\bullet}t_k^{Ti}$ ($p \ / \ t_k^{Ti\bullet}$) is changed after the merging and consequently $\mathbf{Pre}(p, t_{k'}^{Ti})$ ($\mathbf{Post}(p, t_{k'}^{Ti})$) is set equal to $\boldsymbol{I}_{v_{t_{k'}^{Ti}}}$ (see table of figure 4.1).

Running merging subroutine, colors of only merged transitions and their preset and postset are changed, all the other places and transitions conserve their original color set. Consequently, after that all possible merges between T_i and R_j have been executed, the *colors refinement procedure* is accomplished by means of the subroutine show in Fig. 4.14.

To this aim the concept of *Influenced Subnet* is introduced.

Definition 4. Consider transitions t_f^{Ti} and t_l^{Ti} of task T_i, synchronized respectively with service request event "*resource* R_j *acquisition*" and with service request event "*resource* R_j *release*".

The *Influenced Subnet of* T_i *on the base of resource* R_j, $IS(T_i, R_j, t_f^{Ti}, t_l^{Ti})$ is defined as the PN composed of the only places and transitions, belonging to T_i, that set up a directed path[3], having t_f^{Ti} as the first transition and t_l^{Ti} as the last one.

T_i can require to acquire and release R_j more than once during its execution, hence, in general, given a couple (T_i, R_j), it is possible to build Z Influenced Subnets, each one delimited by a different couple of transitions $(t_{f\ z}^{Ti}, t_{l\ z}^{Ti})$, where $t_{l\ z}^{Ti}$ is the first transition of T_i, synchronized with the event "*resource* R_j *release*", that is reached starting from $t_{f\ z}^{Ti}$ (synchronized with the event "*resource* R_j *acquisition*"), following the direct path that joins them.

The *color refining subroutine*, executed at Step 2 of merging algorithm, is the following:

For each transitions t_k^{Ti} of $IS(T_i, R_j, t_{f\ z}^{Ti}, t_{l\ z}^{Ti})$ that has not been merged with any transition of R_j

Step 2.1: assign to t_k^{Ti} the same occurrence colors of the merged transitions.

[3] A path is a sequence $q_1 q_2 \ldots q_n$ such that $(q_i, q_{i+1}) \ / \ P \subseteq T \bigcap T \subseteq P$ and $q_{i+1} \ / \ q_i^{\bullet}$ or $q_i \ / \ q_{i+1}{}^{\bullet}$, i.e., it is a sequence of places and transitions that alternatively joins two distinct nodes, places and transitions, with an arc (if the arc is directed, i.e.,$q_{i+1} \ / \ q_i^{\bullet}$, the path is called directed).

Step 2.2: if places of t_k^{Ti} preset belong to $IS(T_i, R_j, t_{f\ z}^{Ti}, t_{l\ z}^{Ti})$ and their color set has not been changed during merging procedure, then assign to ${}^{\bullet}t_k^{Ti}$ the same color set of t_k^{Ti} and set $\mathbf{Pre}({}^{\bullet}t_k^{Ti}, t_k^{Ti})$ equal to the identity matrix $\boldsymbol{I}_{v_{t_k^{Ti}}}$, otherwise do not change their color set and set $\mathbf{Pre}({}^{\bullet}t_k^{Ti}, t_k^{Ti}) = \boldsymbol{M}_T$.

Step 2.3: if places of t_k^{Ti} postset belong to $IS(T_i, R_j, t_{f\ z}^{Ti}, t_{l\ z}^{Ti})$ and their color set has not been changed during merging procedure, then assign to $t_k^{Ti\bullet}$ the same color set of t_k^{Ti} and set $\mathbf{Post}(t_k^{Ti\bullet}, t_k^{Ti})$ equal to the identity matrix $\boldsymbol{I}_{v_{t_k^{Ti}}}$, otherwise do not change their color set and set $\mathbf{Post}(t_k^{Ti\bullet}, t_k^{Ti}) = \boldsymbol{M}_T$.

Now an example is provided to show how the merging procedure works.

Example 1. Consider Fig. 4.15(a) where the models of one task, T_1, and of two DRs, R_1 and R_2, are shown. Assume that $v_{T_1} = 1$, $v_{R_1} = v_{R_2} = 2$, i.e., each place and transition of T_1 has 1 color while each place and transition of both resources has 2 colors. Note that for the sake of clarity arcs weights are not reported in any figure, while colors are omitted in Fig. 4.15(a).

Task T_1 requires R_1 acquisition (s_{r1}), then it waits for the occurrence of an internal event (e_1) and then it requires R_2 acquisition (s_{r2}). Soon after, the service s_{r3}, implying both kinds of resource, starts. After service completion s_{c3}, first R_2 and then R_1 is released (s_{r4} and s_{r5}).

As more, assume that T_1 needs acquisition of 2 instances of R_2 at the same time and just 1 instance of R_1 to complete the activities. At the occurrence of s_{r4} both instances of R_2 are released: i.e., with reference to the merging algorithm, $w_{k1} = 1$ and $w_{k2} = 2,\ \emptyset k$.

Transition of T_1 is merged with transitions of R_1: the result is shown in Fig. 4.15(b). All task transitions have been merged but t_2^{T1}, t_3^{T1} and t_6^{T1}, since they are not synchronized with events involving R_1.

Since for each task transitions t_k^{T1} the occurrence color number is $v_{t_k^{T1}} = 1$, while for each transition t_h^{R1} of R_1 it is $v_{t_h^{R1}} = 2$, and since $w_{kj}^{Ti} = 1,\ \emptyset k$, as indicate at Step 1.2 of merging subroutine, $v_{t_{k'}^{T1}} = 1 \times {}_1^2[= 2$ (i.e., the service can be executed in two way: acquiring R_1^1

or R_1^2; each way corresponds to a different occurrence color of transition $t_{k'}^{T1}$). A possible choice for matrix $\mathbf{Pre}({}^{\bullet}t_h^{R1}, t_{k'}^{T1})$, as well as for $\mathbf{Post}(t_h^{R1\bullet}, t_{k'}^{T1})$, is $\boldsymbol{M}_R = \left.\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right\{$(i.e., acquiring of R_1^1 (R_1^2) is associated to color c_1 (c_2)). As more, $\mathbf{Pre}({}^{\bullet}t_1^{T1}, t_{1'}^{T1}) = \mathbf{Post}(t_7^{T1\bullet}, t_{7'}^{T1}) = \begin{bmatrix} 1 & 1 \end{bmatrix}$. For all the other merged transitions $\mathbf{Pre}({}^{\bullet}t_k^{T1}, t_{k'}^{T1}) = \mathbf{Post}(t_k^{T1\bullet}, t_{k'}^{T1}) = \left.\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right\{$(see Step 1.3 of merging subroutine).

After the merging algorithm is completed, the color refining subroutine starts.

$IS(T_1, R_1, t_1^{T1}, t_7^{T1})$ is the subnet in the dotted square of Fig. 4.15(b); transitions interested by the color refining are t_2^{T1}, t_3^{T1} and t_6^{T1}: their occurrence color sets will be changed from $\}c_1\langle$ to $\}c_1, c_2\langle$. Consequently, also the color set of place p_3^{T1} and its relative incidence matrices will be changed. Merging algorithm goes on with the merge of transitions belonging to T_1 and R_2: the result is shown in Fig. 4.15(c). Transitions t_3^{T1}, t_4^{T1}, t_5^{T1} and t_6^{T1} are merged respectively with t_1^{R2}, t_3^{R2}, t_4^{R2} and t_2^{R2}.

Since after the merging with transitions of R_1 the occurrence color number of each task transitions t_k^{T1} has become $v_{t_k^{T1}} = 2$, and since for each transition t_h^{R2} of R_2 $v_{t_h^{R2}} = 2$ and $w_{k2} = 2$, $\emptyset k$, then $v_{t_{k'}^{T1}} = 2 \times \begin{matrix}2\\2\end{matrix}\Big[= 2$. The only possible choice for $\mathbf{Pre}({}^{\bullet}t_h^{R2}, t_{k'}^{T1})$ and $\mathbf{Post}(t_h^{R2\bullet}, t_{k'}^{T1})$ is $\boldsymbol{M}_R = \left.\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}\right\{$(i.e., both instances R_2^1 and R_2^2 are acquired by the task T_1 whatever the way in which it is performing – using R_1^1 or using R_1^2). For all the merged transitions $\mathbf{Pre}({}^{\bullet}t_k^{T1}, t_{k'}^{T1}) = \mathbf{Post}(t_k^{T1\bullet}, t_{k'}^{T1}) = \left.\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right\{$.

After the merging subroutine is completed the color refining subroutine starts. $IS(T_1, R_2, t_3^{T1}, t_6^{T1})$ is the subnet in the dotted square of Fig. 4.15(c); transitions interested by the color refining are t_4^{T1} and t_5^{T1}: however they color set, for this specific example, rests unchanged and the same occurs for their preset and postset and relative incidence matrices. ◇

The CMHPN model can be implemented and used to monitor online the system state, as well as to detect conflicts and to implement a

control based on dispatching rules. Moreover, an example for showing the effectiveness of the automated model generation, to perform the online reconfiguration of the robotic cell model, is presented. For the sake of brevity, only the case when a resource breaks and it must be replaced with another one is considered.

Assume that during a task execution, R_j^x breaks and it must be replaced by R_j^y: the algorithm to update online the robotic cell model is presented in Fig. 4.16. For each task T_i that use resources of kind j, algorithm consists in changing the marking of each influenced subnet $IS(T_i, R_j, {t_f^{Ti}}_z, {t_l^{Ti}}_z)$, substituting tokens of color c_x, associated to R_j^x, with tokens of color c_y, associated to R_j^y (steps from 1 to 3 in Fig. 4.16). After that for all the tasks the marking of all the influenced subnets has been changed, the continuous state of R_j^y (i.e., its state vector and continuous mode) are set equal to the restarting ones (step 4 in Fig. 4.16).

4.3 Case Study

Consider a robotic cell composed of two arms, one *worker* and one *watcher*. The worker executes a drilling operation while the watcher is able to execute job verification and to indicate to the worker the position where to execute the job. *Drilling task* is the same task defined in Section 4.1. In addition, a human operator can enter in the cell to inspect the execution without any direct interaction with the arms. If the distance between human operator and arms goes beyond a given threshold, the arms must be stopped.

The robotic cell can be defined by means of the following instructions.

DefineHybridResource(Worker, wrk_Q, wrk_f, "*idle*$\bigvee$*stopped*", wrk_cModes, $< x1, x2 >$,
wrk_hTable,
$< wrk\_HomePos, 0 >$, wrk_I_{s_r}, wrk_O_{s_c})
AssignData(Worker, wrk_eDFunction)
AssignAction(Worker, wrk_aTable)
AssignLogicalExpression(Worker, wrk_condTable)
DefineHybridResource(Watcher, wtc_Q, wtc_f, "*idle*$\bigvee$*stopped*", wtc_cModes, $< x1, x2 >$,

wtc_hTable,
$< wtc\_HomePos, 0 >$, wtc_I_{s_r}, wtc_O_{s_c})
AssignData(Watcher, wtc_eDFunction)
AssignAction(Watcher, wtc_aTable)
AssignLogicalExpression(Watcher, wtc_condTable)
DefineTask(Drilling, drl_S, drl_ϕ, "*ready*", drl_I_{s_c}, drl_O_{s_r})
DefineLinks(Drilling, drl_rFunction)
AssignData(Drilling, drl_eDFunction)
AssignLogicalExpression(Drilling, drl_condTable).
InstantiateBlock(Worker,wrk1)
InstantiateBlock(Watcher,wtc1)
InstantiateBlock(Drilling,drl1)

Details of resource Watcher, as well as the ones of task Drilling, except its eventDataFunction and conditionTable, have been presented in Section 4.1; details of resource Worker are omitted for the sake of brevity.

The eventDataFunction associated to Drilling task is shown in Table 4.10 and namely
drl_eDFunction.

Drilling task conditionTable is not shown, since all the logical expressions coincide with the corresponding conditions (i.e., w.r.t. Table4.9, λ=go to pos, definition=go to pos).

Figure 4.10 drl_eDFunction

event		data
input event	output event	
	wtc go to pos	posDes
	wrk go to pos	posDes
	start drilling	dTime

Nets modeling each resource and task of the cell are shown in Fig. 4.17(a), Fig. 4.17(b) and Fig. 4.17(d). Note that, for the sake of clearness, places and transitions of the task and of all the resources are numbered in progressive way. Drilling is modeled by the net in Fig. 4.17(a); Watcher and Worker are respectively modeled by the nets of Fig. 4.17(b), and Fig. 4.17(d). Under the initial marking the task is ready to start (a token is in p_1) and both the watcher and the worker are

idle (a token is in p_{12} and p_{25}) and stopped in their current position (a token in p_{20} and p_{31}).

Places p_{17}, p_{18} and p_{19} (p_{28}, p_{29} and p_{30}) model the different dynamics that the watcher (worker) can have during its motion (*acc, const, dec*); transitions connecting such places model the change of dynamic, e.g. when the event e_4 (e_8), corresponding to the reaching of the maximal allowed speed, occurs, t_{22} (t_{39}) fires and the resource transits from *acc* to *const*. Place p_{c1} (p_{c2}) is the continuous place modeling the continuous state of the resource: the structural marking indicates current position and speed of the resource.

Transitions t_{24}, t^*_{24} and t_{25} model respectively the starting of the worker drilling service, its time duration represented by the value δ associated to t^*_{24}, and its completion.

In Fig. 4.18, it is shown how the proposed model is able to capture the hybrid behavior of the interaction between the arm and the human operator. The worker is initially stopped and the human operator is out of the cell, consequently arm speed rises until the maximal speed is reached. The human operator enters in the cell. The worker is moving with constant speed when its distance with human operator passes a fixed threshold; consequently the worker begins to decelerate (transition t_{21} fires) until it completely stops (firing of t_{23}). It starts to move again (firing of t_{24}) only when the human goes away, over the safety distance. The model has been simulated using PNetLab [BCC12c].

In order to prove the effectiveness of method in terms of integration of a formal modeling approach with modern 3D simulation environments, as state previously, one of them was used to implement the case study described above. A video of the corresponding 3D animation is available at [vid]. There are many tools that make this possible and one of them is V-REP [vre]. This environment can simulate the continuous dynamics of any complex object one may think about and, in particular, of watchers, workers and human operators. It would be, however, required to translate the behavior of the discrete event part of the model (discrete places and discrete transitions) in V-REP's proprietary language (namely, LUA language). Alternatively, it is also possible to remotely interact with the simulator (that would be in charge of the only continuous dynamics) and to use proper dedicated tool for the

discrete dynamics, that would be, thus, independent from the specific 3D environment.

Consider now a slightly different cell, obtained adding an assembling task to the previous one and where the human operator can enter to move the arms close to a desired position or to execute some manual operations.

Assembling task. Two parts have to be joined by using rivets; the human operator drives the worker as close as possible to the two parts, then the worker, by using high precision cameras and reference markers, is able to position itself at the exact point where to introduce the rivet. After the assembling of the two parts, the worker remains in the final position waiting for the watcher that has to check the quality of the assembling, inspecting the parts and deciding if the task is completed or if it must be repeated. Only in the first case the worker becomes idle, otherwise first the human operator takes off rivets and then the worker starts to assemble parts again.

Note that human operator activities must now be explicitly taken into account. The cell can be defined adding the following instructions to those used for the previous cell model.

DefineDiscreteResource(Human, hmn_Q, hmn_f, "*idle*", hmn_I_{s_r}, hmn_O_{s_c})
AssignData(Human, hmn_eDFunction)
AssignLogicalExpression(Human, hmn_condTable)
DefineTask(Assembling, asb_S, asb_ϕ,"*ready*",asb_I_{s_c}, asb_O_{s_r})
DefineLinks(Assembling, asb_rFunction)
AssignData(Assembling, asb_eDFunction)
AssignLogicalExpression(Assembling, asb_condTable)
InstantiateBlock(Human, hmn1)
InstantiateBlock(Assembling, asb1)

All details about the resource Human have been presented in Section 4.1. The conditionTable associated to Human is not shown since all the logical expressions coincide with the corresponding conditions. For the sake of brevity, details of task Assembling are omitted. Nets modeling the new cell are shown in Fig. 4.17. Assembling task is modeled by the net in Fig. 4.17(e); human operator is modeled by the net of Fig. 4.17(c). Under the initial marking also the assembling task is ready to start and the human operator is idle (there is a token in p_{32} and

in p_{63}).

The presented approach is a first effort on the direction to create tools, addressed to users that do not have any knowledge of PNs, for modeling and analyzing robotic cell in aircraft industry.

Nowadays, it is possible to use tools operating on CMHPNs, as the one presented in [BCC12d], to execute model simulation and analysis. In particular, deadlock situations must be carefully checked. In [BCC12d] a methodology is presented to devise a deadlock prevention policy that acts directly on the PNs. Deadlock prevention is out of the scope of this thesis, but, simulating the CMHPN model obtained applying the presented approach, it is possible to obtain information useful also for users that are not confident with PNs, thanks to the expressiveness of the net colored marking.

Indeed, analyzing a deadlocked colored marking, it is possible to know

- each task's instance T_i^t and each resource's instance R_j^r involved in the deadlock;
- which phase (state) each T_i^t (R_j^r), involved in the deadlock, is blocked in;
- which events have not occurred because of the deadlock;
- what is the current continuous state of R_j^r.

Such information, in the case of simple robotic cells, can be sufficient to redefine tasks for regulating the resource assignment in the way of preventing the deadlock. Consider the following situation: the assembling task and the drilling task are being executed at the same time. Assembling task has acquired the unique worker of the cell and, once it has completed the assembling procedure, it is stopped waiting for a watcher inspection. Contemporaneously, the drilling task has acquired the unique watcher of the cell that, once it is arrived to the drilling position, it is stopped waiting for the arrive of the worker. As shown in Fig. 4.19, where for the sake of clearness only a part of the final model is reported, such a situation correspond to a deadlock. Tokens in places p_{23} and p_{13} indicate, respectively that worker

and watcher have been acquired by a task, while tokens in places p_{31} and p_{20} indicate that they are stopped at position x_1. Tokens in places p_4 and p_{41} indicate that assembling task and drilling task are respectively waiting for acquiring the watcher and the worker. Transitions t_4 and t_{51} cannot fire because the first requires that place p_{25} (watcher idle) or both the places p_{26} and p_{29} (watcher released that is moving with constant speed) are marked, and the second requires that place p_{12} (worker idle) or both the places p_{16} and p_{18} (worker released that is moving with constant speed) are marked. The proposed model allows to take into account the continuous-time dynamic of the robots position (e.g. the variable x_1) during the execution of tasks (e.g. the condition s_{c2} is true when the $x_1 = posDest$ causing the transition of task Drilling from phase wtc moving, place p_2, to phase waiting 4 wrk, place p_3). From the above analysis it results that deadlock occurs since Watcher (Worker) is acquired by Drilling (Assembling) task and at same time the Assembling (Drilling) task is waiting for it. To avoid this situation tasks can be properly redefined. As example, the release of Worker during the assembling task can be anticipate and executed before than Watcher begins inspection: Assembling can acquire it again successively in the case that inspection gives negative results.

The chapter contribution can be summarized in three main points.

1) To provide an automatic synthesis of the hybrid CPN of a robotic cell in aircraft industry, starting from an intuitive and simple specification of resources and tasks, considered the main cell components. Then, an algorithm is provided to obtain their model as separate net modules, finally an algorithm is given to obtain a unique hybrid CPN integrated model. Hence, the approach is conceived to be used without any knowledge of hybrid CPN modeling, since resource and task behaviors are specified in terms of states, events and state transition function or from an instance of a predefined block in a library. The robot trajectories are assumed to be available as an external input, but they are explicitly taken into account in the model. The control problem which the proposed model is addressed to consists in assigning a resource (a camera, a robot, etc.), when it becomes available, to a task. If one resource is available for a set of tasks, a conflict occurs. The control must solve these conflicts in a manner that all the assigned

tasks are completed. Simulation schemes have been presented to implement dispatching rule based controls for these systems. These simulation schemes in a pure discrete event context have been obtained by extending the classical scheme for the simulation of discrete event systems based on the scheduled events list [CL08, BCC12c] using implementation in standard programming languages to speed-up these simulations, while avoiding the use of commercial software. This approach has been adopted for material handling systems in [JGL12] using a matrix-based discrete event controller and in [BCC12c] using a hybrid CPN modeling framework named Colored Modified Hybrid Petri Nets (CMHPNs), introduced by the authors.

2) To provide a formal model, a CMHPN model, that, as any PN model, provides local state representation so that each activity is modeled by a transition that is graphically connected to a place that models a resource state [ZK98, WBC07, WZC08b, WCCZ11, DF05]. This permits to formally characterize the system's state, to use standard simulation schemes as well as to realize halt and recovering mechanisms so that the state of the system and all the simulation outputs are saved at a certain time and restored when necessary. The use of colors helps to obtain compact models and allows to represent the identity of resources and tasks. This is very useful in robotic cell for aircraft industry, since they are characterized by heterogeneous resources and tasks. Moreover, the automatic synthesis procedure allows one to add and remove a resource and/or a task during an online simulation.

3) To provide a way to integrate modern simulation environments for robotic cells (e.g. Gazebo, V-REP) with a formal modeling approach of these systems. These simulation environments well simulate the continuous time behavior of robots and other devices but tasks and discrete resources behavior must be implemented in proprietary programming languages available in these environments. Literature is full of approaches to code generation from PN models that can be used to this purpose [CG03]. Then, our model offers, by means of the translation of the discrete event part of the CMHPN model related in such languages, a formal way to implement tasks and manage discrete resources in these simulation environments. In this way, the benefits of a formal modelling approach are in addition to those of accurate 3D

animations and environment interactions.

Table 4.1: Table of notations of Section 4.2.

Notation	Meaning
$\mathbf{1_{n_c}}(\mathbf{0_{n_c}})$	row vector of dimension n_c having all elements equal to 1 (0).
c_r (c_t)	color associated to the r-th (t-th) instance of R_j (of T_i).
$\boldsymbol{I}_{v_{t_k^{T_i}}}$	identity matrix of dimension $v_{t_k^{T_i}}$.
$IS(T_i, R_j, {t_f^{Ti}}_z, {t_l^{Ti}}_z)$	z-th Influenced Subnet of T_i on the base of resource R_j.
$\boldsymbol{M}_R$	matrix of dimension $v_{R_j} \subseteq \binom{v_{R_j}}{w_{kj}^{T_i}}$ s.t. each column has $w_{kj}^{T_i}$ element equal to 1 and $v_{R_j} \quad w_{kj}^{T_i}$ elements equal to 0 and all columns are different from each others.
$\boldsymbol{M}_T$	$\left[\begin{array}{c\|cccc} & c_1 \dots c_{n_c} & c_{(n_c+1)} \dots c_{(2n_c)} & \dots & c_{(v_{t_{k'}^{Ti}}-n_c)} \dots c_{v_{t_{k'}^{Ti}}} \\ \hline c_1 & \mathbf{1_{n_c}} & \mathbf{0_{n_c}} & \dots & \mathbf{0_{n_c}} \\ c_2 & \mathbf{0_{n_c}} & \mathbf{1_{n_c}} & \dots & \mathbf{0_{n_c}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{v_{t_k^{Ti}}} & \mathbf{0_{n_c}} & \mathbf{0_{n_c}} & \dots & \mathbf{1_{n_c}} \end{array}\right]$
N_R (N_T)	number of kinds of resource (task).
$n_c = \binom{v_{R_j}}{w_{kj}^{T_i}} = \frac{v_{R_j}!}{w_{kj}^{T_i}!(v_{R_j}-w_{kj}^{T_i})!}$	binomial coefficient.
$p_q^{R_j}$	place associated to state q of resource R_j.
$p_q^{*R_j}$	additional place associated to state q of resource R_j.
$p_c^{R_j}$	continuous place associated to the HR R_j.
$p_s^{T_i}$	place associated to phase s of task T_i.
R_j	j-th resource.
s_r (s_c)	service request (compliant) event.
T_i	i-th task.
$t_h^{R_j}$	h-th immediate transition associated to resource R_j.
$t_h^{*R_j}$	h-th timed transition associated to resource R_j.
$t_k^{T_i}$	k-th immediate transition associated to task T_i.

Table 4.1: Table of notations (continued)

Table 4.1: Table of notations (continued)

Notation	Meaning
$t_k'^{T_i}$	transition obtained after the merging of transition $t_k^{T_i}$ with each transition $t_h^{R_j}$, synchronized with the same event.
$t_{f\ z}^{Ti}$	first transition of the z-th Influenced Subnet of task T_i on the base of resource R_j.
$t_{l\ z}^{Ti}$	last transition of the z-th Influenced Subnet of task T_i on the base of resource R_j.
v_{R_j} (v_{T_i})	number of instances of resource R_j (of task T_i).
$v_{t_k^{Ti}}$	occurrence color number of transition t_k^{Ti}.
$\boldsymbol{\nu}_h^{Rj}$	firing speed associated to transition t_h^{Rj}.
$w_{kj}^{T_i}$	number of instances of R_j involved in the service s_r, synchronized with t_k^{Ti}.
Z	number of Influenced Subnets of T_i with respect to R_j.

Table 4.1: Table of notations

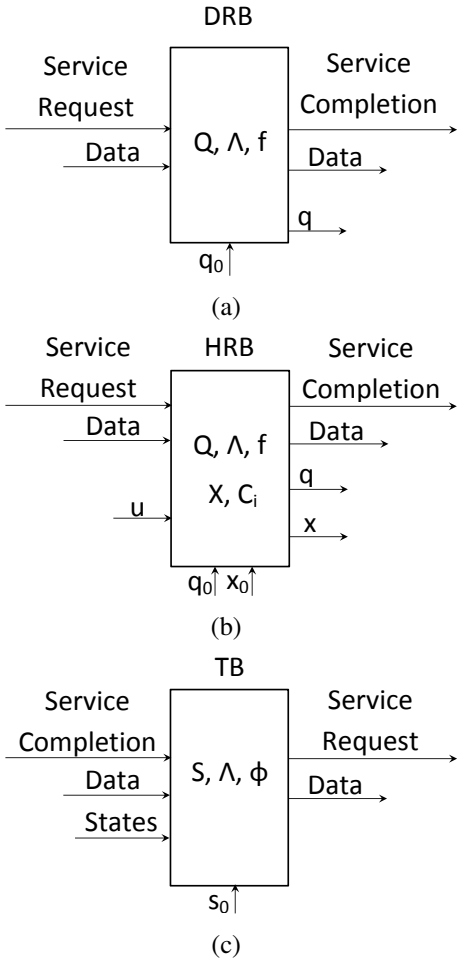


Figure 4.11 (a) Discrete Resource Block; (b) Hybrid Resource Block; (c) Task Block.

$T_i = i$-th task type.
N_T = number of task types.
$n_{T_i} = T_i$ transitions number.
$T_i^t = t$-th instance of i-th task type.
v_{T_i}= number of instance of i-th task type.
$t_k^{T_i} = k$-th transition of task T_i.
$R_j = j$-th resource type.
N_R = number of resource types.
$n_{R_j} = R_j$ transition number.
$R_j^r = r$-th instance of j-th resource type.
v_{R_j}= number of instance of j-th resource type.
$IS(T_i, R_j, t_f^{Ti}{}_z, t_l^{Ti}{}_z) = z$-th Influenced subnet of T_i on the base of resource R_j.
Z= number of Influenced Subnet of T_i with respect to R_j.

for $i = 1$ to N_T
  for $j = 1$ to N_R
Step 1: Merging.
    change $\leftarrow$ 0
    for $k = 1$ to n_{T_i}
      Merged_Transitions$_{jk} \leftarrow \emptyset$
      if t_k^{Ti} synchronized to a service request s_r or to a service confirmation s_c
      then
        change $\leftarrow$ 1
        Merge t_k^{Ti} with transition of resource R_j
        Merged_Transitions$_{jk} \leftarrow$ Merged_Transitions$_{jk} \bigcup t_k^{Ti}$
      end
    end
Step 2: Color Refining.
    if change=1
      for $z \leftarrow 1$ to Z
        $IS(T_i, R_j, t_f^{Ti}{}_z, t_l^{Ti}{}_z) \leftarrow$ Build z-th Influenced Subnet of T_i with respect to R_j
        Start Color Refine Procedure for $IS(T_i, R_j, t_f^{Ti}{}_z, t_l^{Ti}{}_z)$
      end
    end
  end
end
Step 3: Final model.
for $j = 1$ to N_R
  for $h = 1$ to n_{R_j}
    Delete R_{jh}_Merged_Transition
  end
end

Figure 4.12 Merging Algorithm.

t_k^{Ti} = k-th transition of task T_i.
t_h^{Rj} = h-th transition of resource R_j.
v_{R_j} = number of instance of j-th resource type.
$w_{kj}^{T_i}$ = number of resource R_j involved in the service s_r synchronized to transition t_k^{Ti}
$n_c = \binom{v_{R_j}}{w_{kj}^{T_i}} = \frac{v_{R_j}!}{w_{kj}^{T_i}!(v_{R_j}-w_{kj}^{T_i})!}$ = binomial coeff cient.
$\cdot \bowtie \cdot$ = merging operator.
$\boldsymbol{I}_d$ = identity matrix of dimension d.
$\mathbf{1_d}(\mathbf{0_d})$ = row vector of dimension d having all elements equal to 1 (0).

$$\boldsymbol{M}_T = \left[\begin{array}{c|cccc} & c_1 \dots c_{n_c} & c_{(n_c+1)} \dots c_{(2n_c)} & \cdots & c_{(v_{t_{k'}^{Ti}}-n_c)} \dots c_{v_{t_{k'}^{Ti}}} \\ \hline c_1 & \mathbf{1_{n_c}} & \mathbf{0_{n_c}} & \dots & \mathbf{0_{n_c}} \\ c_2 & \mathbf{0_{n_c}} & \mathbf{1_{n_c}} & \dots & \mathbf{0_{n_c}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{v_{t_k^{Ti}}} & \mathbf{0_{n_c}} & \mathbf{0_{n_c}} & \dots & \mathbf{1_{n_c}} \end{array}\right].$$

$\boldsymbol{M}_R$ = matrix of dimension $v_{R_j} \times \binom{v_{R_j}}{w_{kj}^{T_i}}$ s.t. each column has $w_{kj}^{T_i}$ element equal to 1 and $v_{R_j} - w_{kj}^{T_i}$ elements equal to 0 and all columns are different from each others.

Step 1.1: Creation of new transition $t_{k'}^{Ti}$:
for $h = 1$ to n_{R_j}
  R_{jh}_Merged_Transition $\leftarrow \emptyset$
end
for $h = 1$ to n_{R_j}
  if t_h^{Rj} is synchronized with s_r then
    if R_{jh}_Merged_Transition is empty then
      $t_{k'}^{Ti} \leftarrow t_k^{Ti} \bowtie t_h^{Rj}$
      $t_{k'}^{Ti\bullet} = t_k^{Ti\bullet} \bigcup t_h^{Rj\bullet}$
      ${}^\bullet t_{k'}^{Ti} = {}^\bullet t_k^{Ti} \bigcup {}^\bullet t_h^{Rj}$
    else
      $t_{k'}^{Ti} \leftarrow t_{k'}^{Ti} \bowtie t_h^{Rj}$
      $t_{k'}^{Ti\bullet} = t_{k'}^{Ti\bullet} \bigcup t_h^{Rj\bullet}$
      ${}^\bullet t_{k'}^{Ti} = {}^\bullet t_{k'}^{Ti} \bigcup {}^\bullet t_h^{Rj}$
    end
    R_{jh}_Merged_Transition $\leftarrow R_j$_Merged_Transition$\bigcup t_h^{Rj}$
  end
end
Step 1.2: Def nition of $t_{k'}^{Ti}$ occurrences color set:
$v_{t_{k'}^{Ti}} \leftarrow v_{t_k^{Ti}} \cdot \binom{v_{R_j}}{w_{kj}^{T_i}}$
$Co(t_{k'}^{Ti}) \leftarrow \{c_1,\ \dots,\ c_{v_{t_{k'}^{Ti}}}\}$
Step 1.3: Def nition of arc weights between $t_{k'}^{Ti}$ and places of R_j :
$\mathbf{Pre}({}^\bullet t_h^{Rj}, t_{k'}^{Ti}) \leftarrow [\underbrace{\boldsymbol{M}_R \dots \boldsymbol{M}_R}_{v_{t_k^{Ti}} \text{times}}]$
$\mathbf{Post}(t_h^{Rj\bullet}, t_{k'}^{Ti}) \leftarrow \mathbf{Pre}({}^\bullet t_h^{Rj}, t_{k'}^{Ti})$
Step 1.4: Def nition of arc weights between $t_{k'}^{Ti}$ and places of T_i :
if s_r is a resource acquisition then
  $Co(t_k^{Ti\bullet}) \leftarrow Co(t_{k'}^{Ti})$
  Changed_Places$_{ij}$ $\leftarrow$ Changed_Places$_{ij} \bigcup t_k^{Ti\bullet}$
  $\mathbf{Pre}({}^\bullet t_k^{Ti}, t_{k'}^{Ti}) \leftarrow \boldsymbol{M}_T$
  $\mathbf{Post}(t_k^{Ti\bullet}, t_{k'}^{Ti}) \leftarrow \boldsymbol{I}_{v_{t_{k'}^{Ti}}}$
elseif s_r is a resource release then
    $Co({}^\bullet t_k^{Ti}) \leftarrow Co(t_{k'}^{Ti})$
    Changed_Places$_{ij}$ $\leftarrow$ Changed_Places$_{ij} \bigcup {}^\bullet t_k^{Ti}$
    $\mathbf{Pre}({}^\bullet t_k^{Ti}, t_{k'}^{Ti}) = \boldsymbol{I}_{v_{t_{k'}^{Ti}}}$
    $\mathbf{Pre}(t_k^{Ti\bullet}, t_{k'}^{Ti}) \leftarrow \boldsymbol{M}_T$
else
    $Co({}^\bullet t_k^{Ti}) \leftarrow Co(t_{k'}^{Ti})$
    $Co(t_k^{Ti\bullet}) \leftarrow Co(t_{k'}^{Ti})$
    Changed_Places$_{ij}$ $\leftarrow$ Changed_Places$_{ij} \bigcup {}^\bullet t_k^{Ti} \bigcup t_k^{Ti\bullet}$
    $\mathbf{Post}(t_k^{Ti\bullet}, t_{k'}^{Ti}) = \boldsymbol{I}_{v_{t_{k'}^{Ti}}}$
    $\mathbf{Pre}({}^\bullet t_k^{Ti}, t_{k'}^{Ti}) = \boldsymbol{I}_{v_{t_{k'}^{Ti}}}$
end
$t_k^{Ti} \leftarrow t_{k'}^{Ti}$

Figure 4.13 Merging Subroutine.

$IS(T_i, R_j, t^{T_i}_{f\ z}, t^{T_i}_{l\ z})$= z-th Inf uenced subnet of T_i on the base of resource R_j.
Z= number of Inf uenced Subnet of T_i with respect to R_j.
$firstOf(IS(T_i, R_j, t^{T_i}_{f\ z}, t^{T_i}_{l\ z}))$ = function that returns the index of transition synchronized to R_1 acquisition
(i.e. if t_f is such a transition, $firstOf(IS(T_i, R_j, t^{T_i}_{f\ z}, t^{T_i}_{l\ z})) = f$).
$lastOf(IS(T_i, R_j, t^{T_i}_{f\ z}, t^{T_i}_{l\ z}))$ = function that returns the index of transition synchronized to R_1 release
(i.e. if t_l is such a transition, $releaseOf(IS(T_i, R_j, t^{T_i}_{f\ z}, t^{T_i}_{l\ z})) = l$).
$nextOf(t^{T_i}_k, IS(T_i, R_j, t^{T_i}_{f\ z}, t^{T_i}_{l\ z}))$ = function that given a transition $t^{T_i}_k$ belonging to $IS(T_i, R_j, t^{T_i}_{f\ z}, t^{T_i}_{l\ z})$ returns the set of index of the successive transitions;
i.e. if $(t^{T_i}_k{}^{\bullet})^{\bullet} \in IS(T_i, R_j, t^{T_i}_{f\ z}, t^{T_i}_{l\ z}) = \{t_n, t_m\}$, $nextOf(t^{T_i}_k, IS(T_i, R_j, t^{T_i}_{f\ z}, t^{T_i}_{l\ z})) = \{n, m\}$.
$n_c = \binom{v_{R_j}}{w^{T_i}_{kj}} = \frac{v_{R_j}!}{w^{T_i}_{kj}!(v_{R_j} - w^{T_i}_{kj})!}$ = binomial coeff cient.
$\boldsymbol{I}_d$ = identity matrix of dimension d.
$\mathbf{1_d}(\mathbf{0_d})$ = row vector of dimension d having all elements equal to 1 (0).

$$\boldsymbol{M}_T = \left[\begin{array}{c|cccc} & c_1 \dots c_{n_c} & c_{(n_c+1)} \dots c_{(2n_c)} & \dots & c_{(v_{t^{T_i}_{k'}} - n_c)} \dots c_{v_{t^{T_i}_{k'}}} \\ \hline c_1 & \mathbf{1_{n_c}} & \mathbf{0_{n_c}} & \dots & \mathbf{0_{n_c}} \\ c_2 & \mathbf{0_{n_c}} & \mathbf{1_{n_c}} & \dots & \mathbf{0_{n_c}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{v_{t^{T_i}_k}} & \mathbf{0_{n_c}} & \mathbf{0_{n_c}} & \dots & \mathbf{1_{n_c}} \end{array}\right].$$

change ← 1
While change=1
  change ← 0
  $f \leftarrow firstOf(IS(T_i, R_j, t^{T_i}_{f\ z}, t^{T_i}_{l\ z}))$
  $K \leftarrow f$
  While $K \neq lastOf(IS(T_i, R_j, t^{T_i}_{f\ z}, t^{T_i}_{l\ z}))$
    for $q \leftarrow 1to|K|$
      $k \leftarrow K(q)$
Step 2.1: def ning of occurrences colors of unmerged transitions
      if $t^{T_i}_k \notin$ Merged_Transitions$_{ij}$ then
        $v_{t^{T_i}_k} \leftarrow v_{t^{T_i}_k} \cdot \binom{v_{R_j}}{w_{fj}}$
        $Co(t^{T_i}_k) \leftarrow \{c_1, \dots, c_{v_{t^{T_i}_k}}\}$
        change ← 1
Step 2.2: def ning of occurrences colors of preset of unmerged transitions
      if $\bullet t^{T_i}_k$ belongs to $IS(T_i, R_j, t^{T_i}_{f\ z}, t^{T_i}_{l\ z})$ and $\bullet t^{T_i}_k \notin$ Changed_Places$_{ij}$ then
        $v_{(\bullet t^{T_i}_k)} \leftarrow v_{(\bullet t^{T_i}_k)} \cdot \binom{v_{R_j}}{w_{fj}}$
        $Co(\bullet t^{T_i}_k) \leftarrow \{c_1, \dots, c_{v_{t^{T_i}_k}}\}$
        $\mathbf{Pre}(\bullet t^{T_i}_k, t^{T_i}_k) \leftarrow \boldsymbol{I}_{v_{t^{T_i}_k}}$
      else
        $\mathbf{Pre}(\bullet t^{T_i}_k, t^{T_i}_k) \leftarrow \boldsymbol{M}_T$
      end
Step 2.3: def ning of occurrences colors of postset of unmerged transitions
      if $t^{T_i\bullet}_k$ belongs to $IS(T_i, R_j, t^{T_i}_{f\ z}, t^{T_i}_{l\ z})$ and $t^{T_i\bullet}_k \notin$ Changed_Places$_{ij}$ then
        $v_{(t^{T_i}_k\bullet)} \leftarrow v'_{(t^{T_i}_k\bullet)} \cdot \binom{v_{R_j}}{w_{fj}}$
        $Co(t^{T_i\bullet}_k) \leftarrow \{c_1, \dots, c_{v_{t^{T_i}_k}}\}$
        $\mathbf{Post}(t^{T_i\bullet}_k, t^{T_i}_k) \leftarrow \boldsymbol{I}_{v_{t^{T_i}_k}}$
      else
        $\mathbf{Post}(t^{T_i\bullet}_k, t^{T_i}_k) \leftarrow \boldsymbol{M}_T$
      end
    end
    $K \leftarrow nextOf(t^{T_i}_k, IS(T_i, R_j, t^{T_i}_{f\ z}, t^{T_i}_{l\ z}))$
  end
end

Figure 4.14 Color Refining Subroutine.

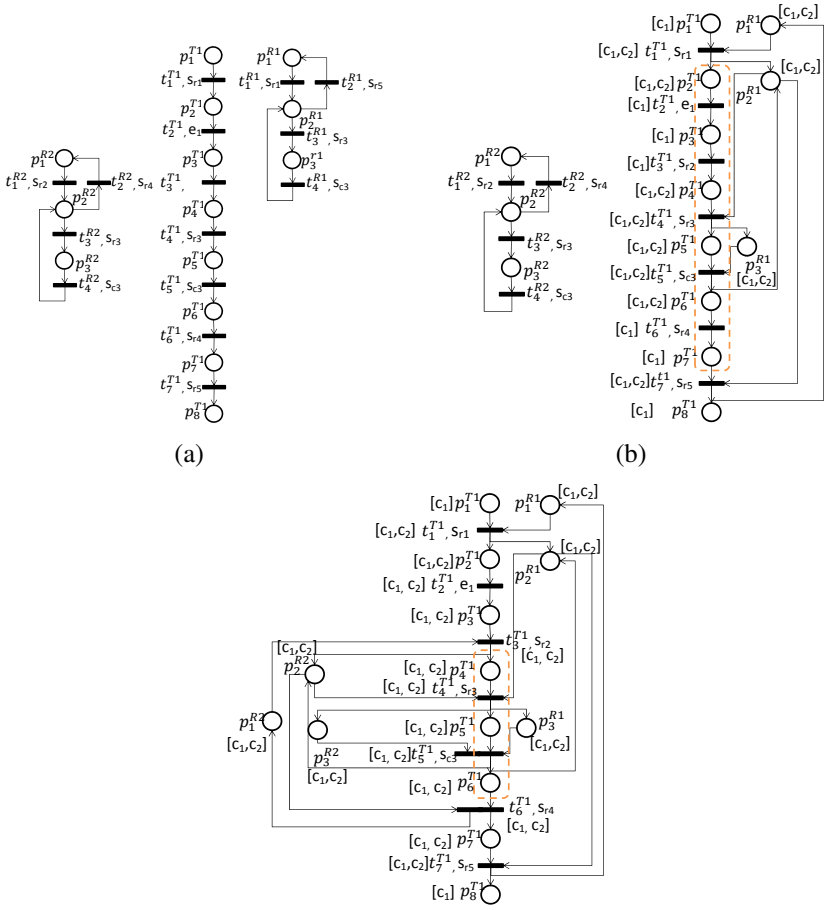


Figure 4.15 (a) Nets of Task T_1 (in the center) and of resources R_1 (on the right) and R_2 (on the left); (b) resulting net after T_1 and R_1 merging but before the start of color refine procedure; (c) final net.

R_j^x = broken resource of type j.
R_j^y = replacement resource of type j.
c_x=color associated to R_j^x.
c_y=color associated to R_j^y.
$P_{R_j}^D$ =discrete place set of R_j.
$< \mathbf{x} >_y$=state of R_j^y
$< \mathbf{x} >_{restarting}$=restarting state of R_j^y
Restarting_Continuos_Mode= restarting continuous mode of R_j^y
HybridResorce($\cdot$)=function that returns "True" if its argument is a hybrid resource.

```
for i = 1 to N_T
    for z ← 1 to Z
        f ← firstOf(IS(T_i, R_j, t_f^{Ti}_z, t_l^{Ti}_z))
Step 1: Change of Marking Color of •t_f^{Ti} ∈ P_{R_j}
        for r ← 1 to |•t_f^{Ti}|
            if p_r ∈ P_{R_j}^D then
               delete the token of color c_x from p_r
               add the token of color c_y in p_r
            end
        end
        K ← f
Step 2: Change of Marking of t_k^{Ti•} marking:
        While K ≠ lastOf(IS(T_i, R_j, t_f^{Ti}_z, t_l^{Ti}_z))
              for q ← 1to|K|
                  k ← K(q)
                  delete the token of color c_x from t_k^{Ti•}
                  add one token of color c_y int_k^{Ti•}
              end
              K ← nextOf(t_k^{Ti}, IS(T_i, R_j, t_f^{Ti}_z, t_l^{Ti}_z))
        end
Step 3: Change of Marking of t_l^{Ti•} ∈ P_{R_j}:
        l ← lastOf(IS(T_i, R_j, t_f^{Ti}_z, t_l^{Ti}_z))
        for r ← 1 to |t_l^{Ti•}|
            if p_r ∈ P_{R_j} then
               delete the token of color c_x from p_r
               add one token of color c_y in p_r
            end
        end
    end
end
if HybridResorce(R_j) then
Step 4: Set of the continuous state of R_j^y:
  < x >_y ← < x >_{restarting}
  for r ← 1 to |P_{R_j}^D|
      delete tokens of color c_x and c_y from p_r
      if p_r is associated to Restarting_Continuos_Mode then
         add one token of color c_y to p_r
      end
  end
end
```

Figure 4.16 Algorithm to update online the model, replacing a resource.

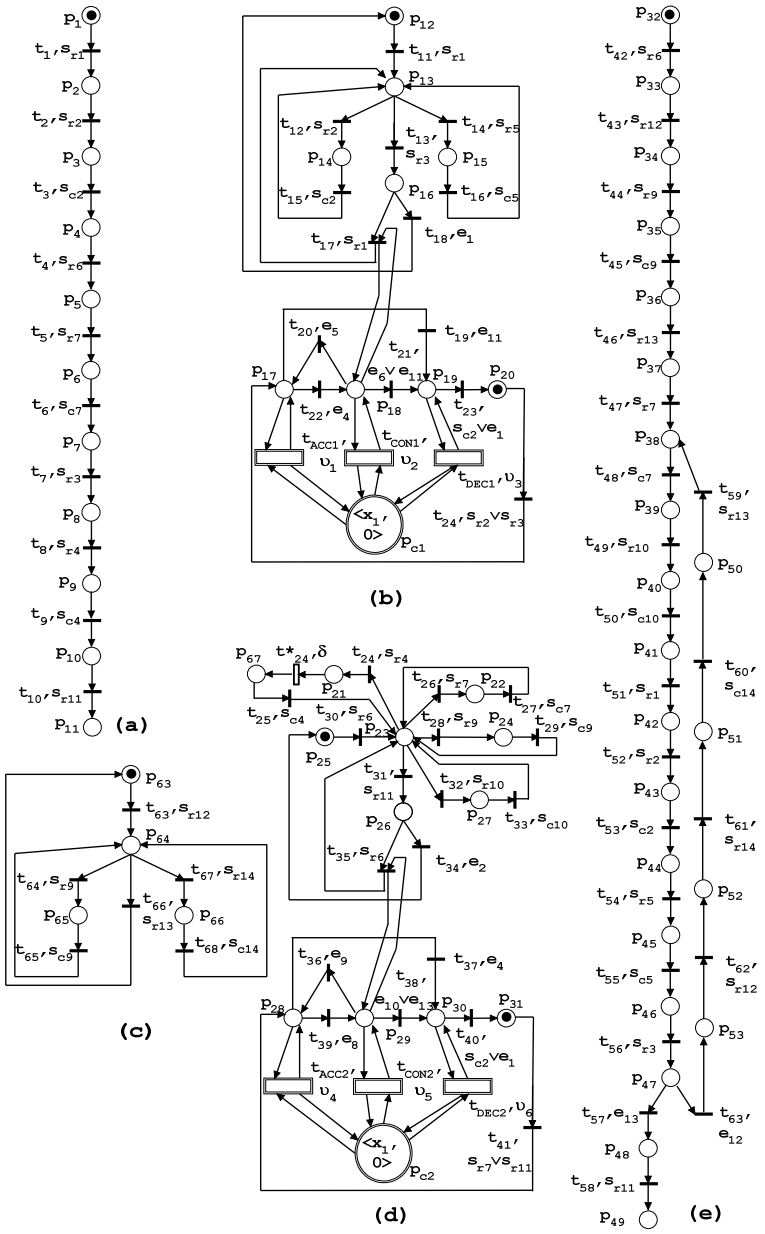


Figure 4.17 Nets modeling case study tasks and resources.

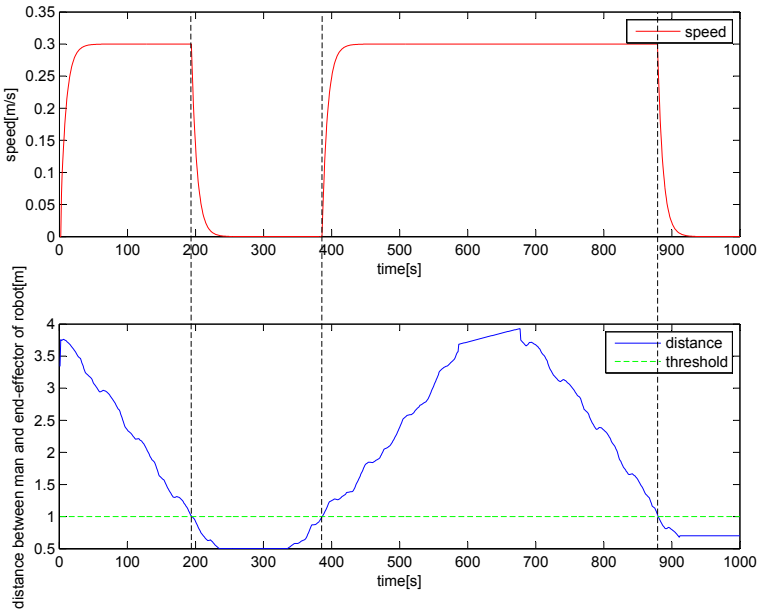


Figure 4.18 Variation of watcher speed on the base of the distance between the arm and the human operator.

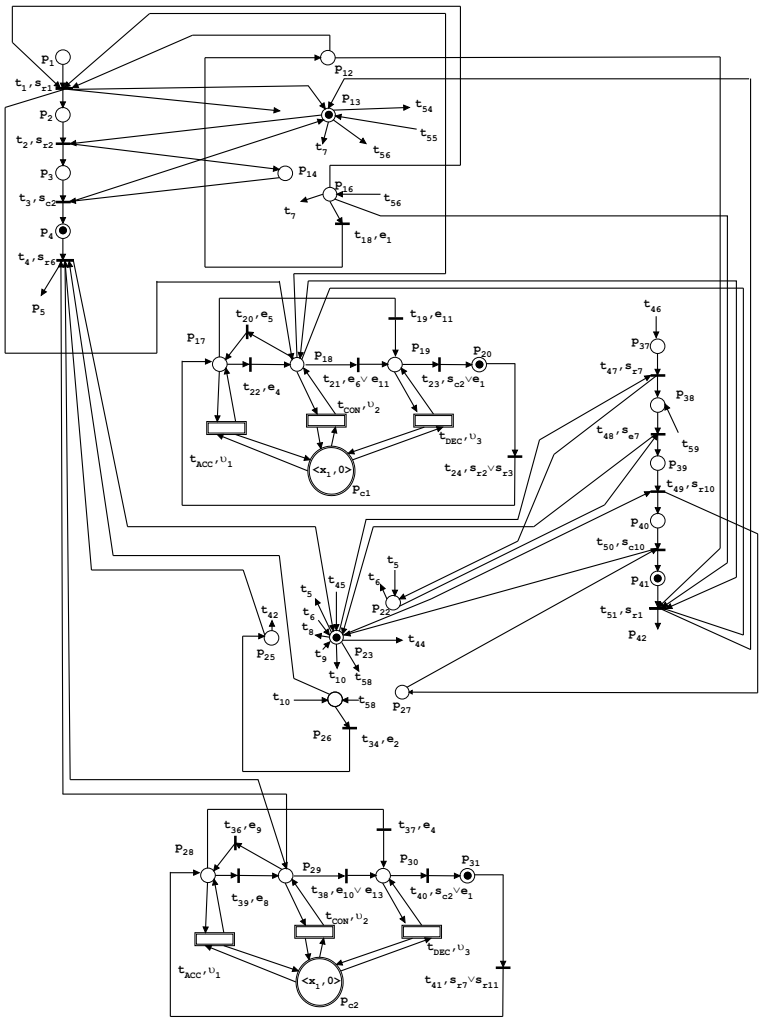


Figure 4.19 Deadlock marking in the final model.

Chapter 5

On the implementation of industrial automation system based on PLC

Today industrial automation software requirements include capability to implement applications involving widely distributed devices, high reuse of software components, formal verification that specifications are fulfilled. In this chapter, an object oriented approach, the programming language SFC together with a proper way to organize the inputs and outputs of FBs and supervisory control are proposed to implement industrial automation control systems to meet the new challenges of this field.

5.1 Function Block Model

A first contribution of this chapter is to show how an event-based execution order for industrial automation software (as suggested by IEC 61499) can be achieved even using IEC 61131 languages extended with OOP. An exact event-based execution order requires that the operative system implements an event-triggered execution of FBs as in IEC 61499. In this thesis it is shown how this can be achieved in terms of keeping activated a FB only when required. OOP has been implemented by one medium-sized vendor of software tools which is cur-

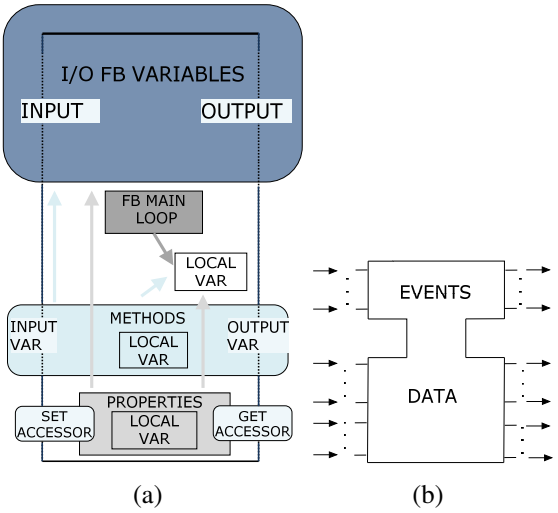


Figure 5.1 (a) FB model in OOP; (b) proposed FB model.

rently suggesting its inclusion into the next revision of the IEC 61131 standard. At this aim, a new FB model is proposed and is derived by its definition in the IEC 61131 standard. The extension is based on two main concepts: FBs are objects; FBs can have an event-based execution order. In the following these two concepts will be discussed.

5.1.1 FB instances are objects

It is assumed that a FB instance is an object in the OOP framework according to ongoing revision of the standard [Wer09] and available commercial products (as [3S 12]). So, methods and properties can be defined additionally to the known 61131 FB elements (declaration, implementation). These methods and properties will form the external view of an FB, see Fig.5.1(a). This extension will provide FBs with the possibility of exhibiting different behaviors (in IEC 61131 FBs can only have one behavior). Moreover, if an FB is seen as an object, its variables can be accessed only by the FB interface, thus emphasizing the aspect of data encapsulation.

Every FB's method contains an own declaration and implementation part. Each method acts as event handler, when invoked it modifies

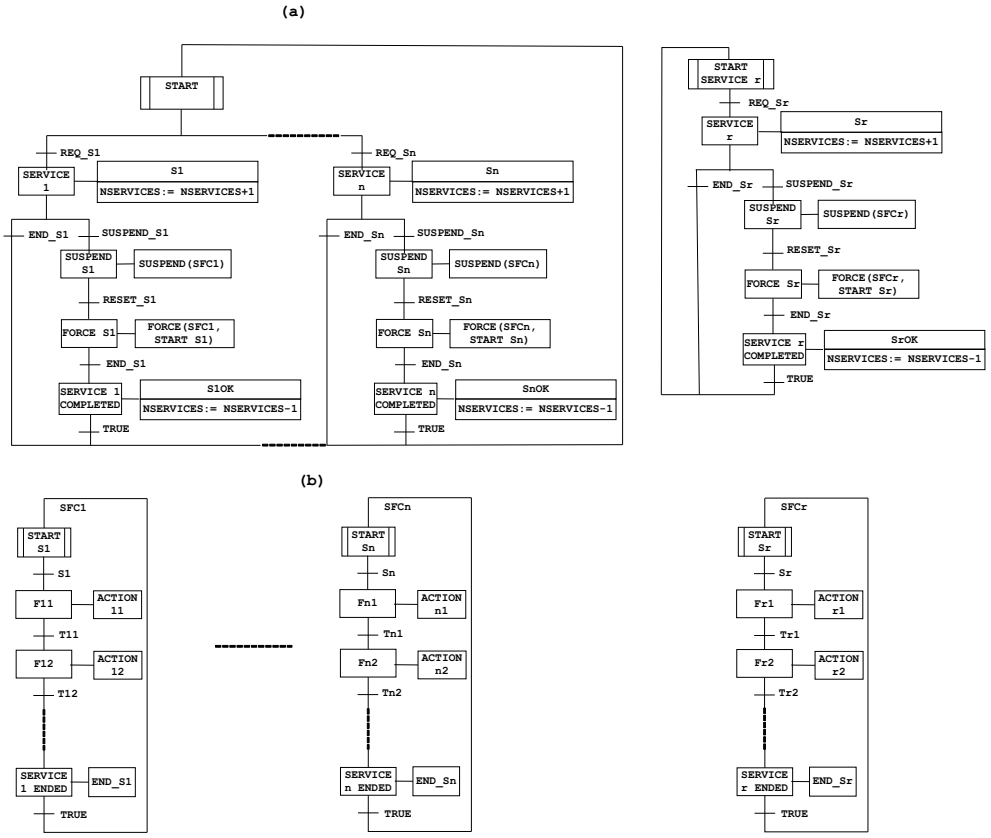


Figure 5.2 (a) Execution Control SFC; (b) SFC implementing service requests.

FB's state and it may produce new values of output variables. This permits to consider the FB model as the one depicted in Fig. 5.1(b): the input/output variables are divided in events and data. Remarkably, this is the FB model as depicted in IEC 61499. Notice that a method has local variables but it does not store data between two consecutive calls.

Properties enable a controlled access to internal variables of the FB. A property has a data type and it is not a variable of the FB but it can access its internal variables. Properties offer an abstract, external view of the internal data of the FB object. When the property is read/written from outside of FB the get/set accessor method is called implicitly. These methods permit to perform operations on the data, for instance to convert their values in some measure units or to obtain derived data.

Methods, properties and input/output variables can be used by the standard dot notation (e.g. FB_Instance_name.namemethod).

Differently both from IEC 61131 and IEC 61499, since FBs are objects they may be constructed by inheritance. A new FB inherits all variables and methods of the old FB and additional variables and/or methods can be defined.

5.1.2 Event-based execution order

In this chapter it is proposed to use the FB body to achieve the event-based execution order. This is achieved by including in the body a program written in SFC, called Execution Control SFC (ECSFC), which acts as the ECC required by IEC 61499 FBs. SFC is one of the programming languages available in IEC 61131; a program written in SFC is a set of graphs; the reader is referred to [IEC03] for a detailed description of SFC.

The ECSFC must change its active steps according to only the event inputs which are seen as service requests. Each service request will be implemented in the body by an algorithm written in one of the available languages. The actions associated with the ECSFC active steps must enable executions of the corresponding algorithm, thus fulfilling the request. The completion of an algorithm is an event that

must be output (it is an output event). The number of active services at the current time is available by means of the output data variable FB_Instance_name.NSERVICES.

As an example, consider the SFC in Fig. 5.2(a) which activates the algorithms (still implemented in SFC) in Fig. 5.2(b). This is done by setting the Boolean variable S_i, which is associated with the condition of the unique output transition of the initial step of SFCi. SFCi represents the algorithm which performs the service S_i. Thus, the SFC in Fig. 5.2(a) acts as an ECSFC; the SFC in Fig. 5.2(a) together with those in Fig. 5.2(b) make possible to achieve the event-based execution order, as required.

An improvement can be obtained if it is possible to utilize hierarchical SFC to program the ECSFC. Hierarchical SFC permits a graph to directly modify the status of another graph. Hierarchy improves SFC modeling capabilities and it solves some very important problems in the design of control algorithms [DA92]. In [Chi98] it is presented how to implement hierarchical SFC even if this is not available in the programming environment: an algorithm to translate SFC programs in other IEC 61331 programming languages is presented and a detailed example of translation of a SFC program into ladder diagram, which is a very common language for automation systems programmers, is also shown.

In particular, the two macroactions available in hierarchical SFC, SUSPEND and FORCE, are very useful since their use permits to suspend and reset a service; in Fig. 5.2(a) the two macroactions are used.

Remark 1. If a FB consists of several algorithms, the use of the ECSFC makes clear and readable which algorithms are activated for a given set of input values. The same result could be obtained also using a case or an if statement, even if the resulting code would be less elegant and readable. However, using ECSFC, it results much easier the implementation of macroactions such as the suspension of an algorithm (a service). Indeed, in industrial automation context, an algorithm is interfaced with the field, i.e. sensors and actuators. This means that to suspend an algorithm, it is not enough to suspend the execution of an algorithm but some output data should be properly set, to ensure that also the effects of the algorithm on the field are suspended (e.g. to turn

off motors, to close a valve, etc.). This can be done easily at ECSFC level, where the suspension can be treated by a sequence of actions.

Remark 2. Note that an exact event-based execution order requires a real event-triggered algorithm execution. In 61131 (also in 3rd edition) each FB is triggered in every scan cycle allowing it to perform its execution. In 61499 a FB would typically only be activated when it gets an event. This allows to control the execution sequence of the FBs but the operative system provides this.

However, if the calling of a FB in IEC 61131 is done by means of an IF statement,

```
IF ((REQ_S1 OR REQ_S2 OR ...REQ_Sn) OR FB_Instance_name.NSERVICES>0)
    THEN FB_Instance_name.NSERVICES(inputs,outputs)
```

the FB is triggered only when at least one of its input events is true and it is kept activated while at least one service is being executed.

This allows to achieve an event-based execution as for the FB calling.

As far as the execution order of the algorithms coded in the FB, in IEC 61131 all the program organization units execute their code sequentially by beginning to end. Hence, using an ECSFC, when the scanning process of a FB is performed only the actions of the algorithms enabled by the ECSFC are executed while the other algorithms remain in the initial step which has no actions associated (see Fig.5.2).

Note that also ISaGRAF [VC08], a commercial tool that allows the development of programs that are IEC 61131 or IEC 61499 compliant, supports a hierarchy structure for SFCs. In particular, SFCs placed on the top of hierarchy, called fathers, are activated by the system. Those placed at lower level, called child SFCs, are started, killed, frozen or restarted by their father. In this way it is also possible to implement ECSFCs. However, the approach proposed in this thesis requires standard SFC programming language just plus two macroactions, called SUSPEND and FORCE, and it can be implemented also on platform where SFC is not supported by using the translation algorithm shown in [Chi98].

The main idea behind the algorithm is that of allocating a bit

(marker) for each step and for each transition. The marker of a step indicates if it is active or not; the marker of a transition indicates if it is fireable or not. An algorithm can be composed of more not connected SFCs (i.e. the term SFC is referred to a single connected graph), but a unique Ladder Diagram program composed of four sections can be used to implement all the SFCs (i.e. all the connected graphs), to be coded in the right sequence: Initializing section; Action Execution section; Transition Evaluation section; Condition Update section. The Initialing section must be executed only once at the first scan of the program and it sets all the initial steps as active, i.e. set their marker to 1. The second section must execute actions for all SFCs. It will contain one rung for each action describing when the action must be executed. The third section contains evaluation of the markers of transitions. For each transition, it is evaluated if it is fireable or not and the corresponding marker set. The fourth and last section realizes the condition update. For each transition, depending on the state of its marker, the marker of connected steps are set or reset. Macroactions can be easily introduced in the second section, adding one or more implementing rungs, depending on the macroation, where one or some markers (forcing) or all the markers (suspension) of a certain SFC are set or reset.

In addition of the known advantages of OOP (reusability, data encapsulation, etc.), the benefits of the proposed FB model are:

- A service (i.e. the associated algorithm) is executed only when it is explicitly invoked by means of the associated event as required by IEC 61499. The execution environment influences only the time required to execute an algorithm, not when an algorithm must be executed.

- Once the programming environment allows OOP, the coding is still fully IEC 61131 compliant

- The new FB model permits to delegate the coordination of service requests to an external agent. This will be the subject of one of the next sections.

- Different execution modes of an FB can be easily managed by input events.

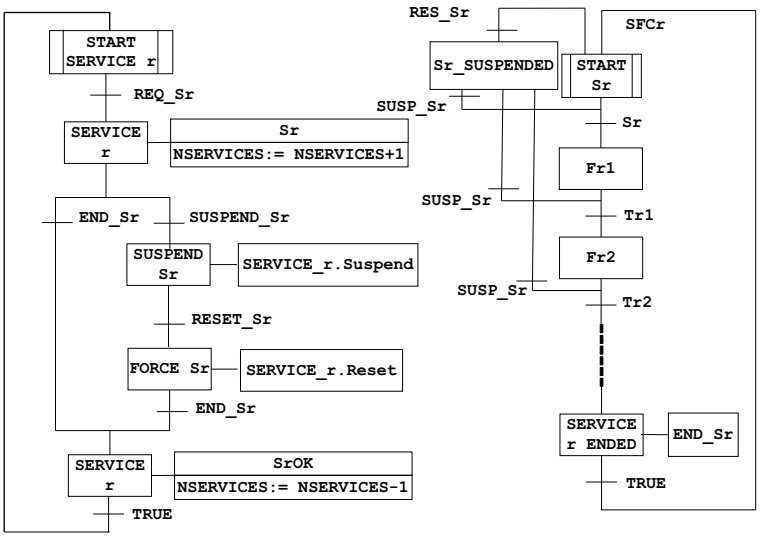


Figure 5.3 Implementation of suspension and reset of a sequence.

Note that the execution of SFC consists of a number of tasks that in each scan cycle are performed in a certain order [HFL01]. In particular, each IEC 61131 commercial implementation executes the evaluation of transition conditions leading from active steps and their firing in a different way. This may result in a different behavior. However, this is a well known problem and in the literature it is clearly shown how to solve it [Lew98]. This is not the scope of this thesis.

As for our approach, in the ECSFC the unique problem concerning the evolution is that it has diverging paths, then there may be simultaneously enabled alternative transitions. According to the standard, transitions to the left have precedence over transitions to the right. However, a specific IEC 61131 implementation may implement a different priority rule, or the programmer can change it by rewriting the logical conditions.

As for example, in the SFC in Fig. 5.2(a) by replacing "REQ_S1" by "REQ_S1 AND NOT(REQ_Sn)", the n-th service is executed before the first service, if they are both enabled.

Finally, the proposed FB model can be improved by using another useful OOP tool, the Interface, which is an abstract specification for FB methods. An Interface defines signatures for the methods without specifying their implementation. A method declared by an Interface can be implemented in an FB, and its implementation (related to that particular FB) must be coded. For instance, consider the suspension and the reset of a certain sequence, a common requirement in industrial automation. An Interface could be defined with the signature of the two methods: suspension and reset; then, each FB can use the suspension and the reset in a different way by implementing the interface and describing what the two methods actually do in that particular FB:

```
INTERFACE SUSPEND_AND_RESET
  METHOD Suspend
        ...
  END_METHOD
  METHOD Reset
        ...
  END_METHOD
END_INTERFACE
```

As told, the suspension and the reset of a sequence are common requirements in industrial automation so some more considerations are in order. The suspension and the reset of a sequence is a state-dependent action and thus a method is not sufficient to implement them, since method cannot depend on the current state. A possible solution is to use a method to set a boolean variable which triggers a unique step in the SFC, which remains active until the method RESET sets to true the value variable causing the transition into the initial step. A method is used since some actions on the field (turn off a motor, close a valve, etc.) may be required in addition to the cited variable setting. An example is shown in Fig. 5.3: the method SERVICE_r.Suspend set to true SUSP_Sr causing the transition of SFCr into the step Sr_SUSPENDED.

It is worth noticing that suspension of a sequence (and forcing a sequence in a given step) are very useful macroactions [Chi98] which are not provided by the revision of the standard. Their implementation in an OOP environment is one of the contribution of this chapter.

OMAC PackML [OMA] state model, now part of ISA 88 standard, introduces a standard state representation of what a machine is doing to

improve system programming efficiency. SUSPEND and RESET can help in implementing such a state model since these methods allow to define clearly the functioning state of a FB, and then the state of a device, also when something wrong occurs.

5.2 FB Design

The goal now is to design a FB that performs an event-based execution order and that is capable to fulfill the service requests associated with events.

In an OOP context an object is characterized by data and methods. It must be noticed that the program of sequences, one of the fundamental things in industrial automation, cannot be coded as a method since methods have local variables but they do not store data between two consecutive calls. Then, to implement a sequence there are two solutions: a method that uses variables declared in the body of FBs; the implementation in the object body. Both solutions would not be a good practice, since adding new operations would imply the redefinition of the class (i.e. adding new local variables or writing a different body).

In this section, it is proposed to implement each operation as a separate class called *Operation FB* (OFB) while the basic functionalities that requires access to I/O field signals are implemented in a class called *Device FB* (DFB). Operation FB uses such functionalities to perform a certain working cycle. If more complex behaviors are needed, a new DFB can be defined that inherits all methods and properties of an existing DFB and extends it by new methods and properties. Notice that both OFBs and DFBs are FB objects as provided by the revision of the standard, they are named differently only to point out their role.

To link an OFB with a DFB, pointers can be used. In the revised version of the standard, pointers are provided. They can point to any data types, any user-defined types, programs, methods, FBs, functions. Then, the pointer to a DFB can be given as input to an OFB to indicate the DFB it has to perform its operations on.

As an example, assume to have a DFB named SILO, which implements, for a certain real silo, basic behaviors as opening and closing of its input and output valves and conversion of its sensor measures; assume, moreover, to have an OFB named FILLING, which implements the whole sequence of silo filling.

The following code

```
VAR
    S1:SILO;
    FIL1:FILLING(PTSILO:=ADR(S1));
END_VAR
```

allows to create an instance of the OFB FILLING named FIL1 which receives in input the pointer to S1, that is an instance of the DFB SILO (ADR stays for address operator and PTSILO is a pointer to a SILO object). This allows the use of methods or properties of S1 inside FIL1 by standard dot notation (e.g. PTSILO^.namemethod).

In this way all the benefits of OOP are used: an operation can be added simply by defining a new OFB class just for the new operation. Moreover, the distinction between DFB and OFB helps to reuse DFBs which have direct interface with the field. Notice that a DFB certainly has field events since it provides field services (e.g. open valve, start motor), while an OFB may have no field inputs/outputs but it certainly has input/output events since it consists of a sequence of basic operations (e.g. filling of a silo, transfer of a pallet) which are started by means of input events.

5.2.1 Example

Consider the layout shown in Fig. 5.4. It consists of a set of silos connected by a pipe. Each silo is cyclically filled and emptied with a liquid. However, before emptying silo S2 the liquid is heated by a resistance until a given temperature is reached, while in silo S3 it is mixed for a given time. In silo S4 the liquid is heated and mixed before silo is emptied.

The liquid of silo S1 (S2) is poured into silo S4 (S3). Silos S3 and S4 cannot be filled at the same time since there is only a single pipe to do this operation. Mixing the liquid in silos S3 and S4 at the same time is not possible because it requires too much power.

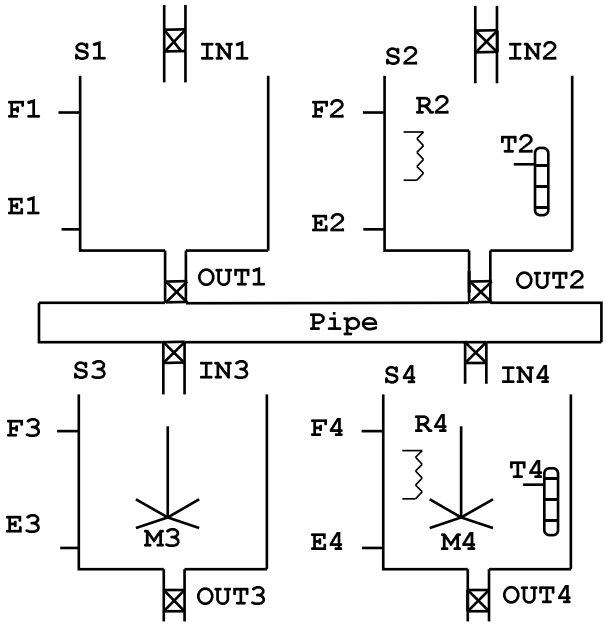


Figure 5.4 fig/Plant layout.

According to the approach described in this section, a DFB class named SILO has been created for the basic silo without mixer and heater, see Fig. 5.5 for the graphical I/O definition.

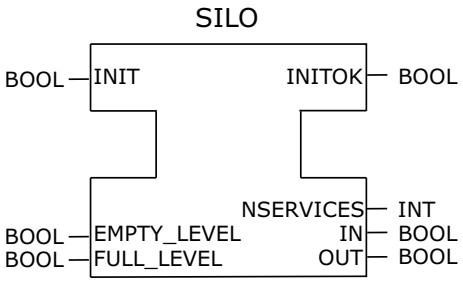


Figure 5.5 DFB SILO.

```
FUNCTION_BLOCK SILO
  VAR_INPUT
        EMPTY_LEVEL:BOOL; (* sensor of empty level *)
        FULL_LEVEL:BOOL; (* sensor of full level *)
        INIT:BOOL; (* initialization event *)
  END_VAR
  VAR_OUTPUT
        IN:BOOL; (* input valve *)
        OUT:BOOL; (* output valve *)
        N.SERVICES:INT; (* number of active services *)
        INITOK:BOOL; (* initialization completed event *)
  END_VAR
  VAR
        ...
  END_VAR
  METHOD Set_Init
        ...
  END_METHOD
  METHOD Set_In
        ...
  END_METHOD
  METHOD Set_Out
        ...
  END_METHOD
END_FUNCTION_BLOCK
```

Then, DFB SILO_WITH_MIXER_AND_HEATER has been created. It inherits by the DFB SILO all methods, properties, interfaces and it adds those related to the mixer and heater, as shown in Fig. 5.6 for the graphical I/O definition.

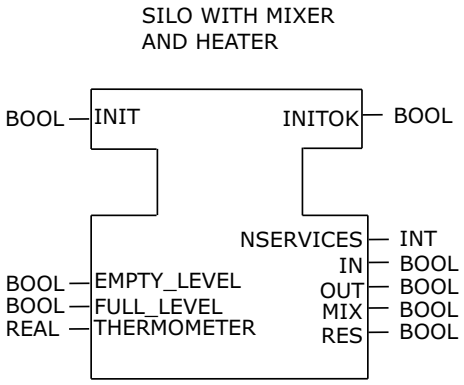


Figure 5.6 DFB SILO with mixer and heater.

```
FUNCTION_BLOCK SILO_WITH_MIXER_AND_HEATHER EXTENDS SILO
  VAR_INPUT
        THERMOMETER:REAL; (* temperature sensor *)
  END_VAR
  VAR_OUTPUT
        MIX:BOOL; (* mixer motor *)
        RES:BOOL; (* heater resistance *)
  END_VAR
  VAR
        ...
  END_VAR
  METHOD Set_Mixer
        ...
  END_METHOD
  METHOD Set_Res
        ...
  END_METHOD
  PROPERTY TEMPERATURE : REAL
        GET
          TEMPERATURE:=(THERMOMETER*1.8)+32;
        END GET
  END_PROPERTY
  PROPERTY TEMPERATURE : BOOL
        GET
          END_EMPTY:=NOT(EMPTY_LEVEL);
        END GET
  END_PROPERTY
END_FUNCTION_BLOCK
```

From the specifications, four working sequences must be defined: filling, emptying, heating and mixing. For all these sequences, the methods SUSPEND and RESET have been defined by means of INTERFACE construct and properly implemented in the respective FBs.

As for example, in this chapter the OFB implementing the heating sequence has been reported.

```
FUNCTION_BLOCK HEATING IMPLEMENTS SUSPEND_AND_RESET
  VAR_INPUT
        REQ_HEAT:BOOL; (* request of service HEAT *)
        INIT:BOOL; (* initialization event *)
        T_HEAT:REAL; (* temperature set point *)
  END_VAR
  VAR_OUTPUT
        HEATOK:BOOL; (* service HEAT completed; *)
        INITOK:BOOL; (* initialization completed event *)
        N.SERVICES:INT; (* number of active services *)
  END_VAR
  VAR
        PTSILO:POINTER TO SILO_WITH_MIXER_AND_HEATHER;
        SUSPEND_HEATING:BOOL; (* event alarm *)
        RESET_HEATING:BOOL; (* reset command *)
        SUSPEND_INITIALIZING:BOOL; (* event alarm *)
        RESET_INITIALIZING:BOOL; (* reset command *)
  END_VAR
  METHOD Suspend
        ...
  END_METHOD
  METHOD Reset
        ...
  END_METHOD
  FB body consists of SFC in Fig.5.7
END_FUNCTION_BLOCK
```

Notice that each FB usually has an initialization procedure, since in practice some checks are needed before the starting of the normal running of a device and of a working cycle. This is why initialization input and output events are always present in the FBs shown in the chapter.

5.3 A formal approach to FBs coordination

After FBs have been declared and instantiated according to OOP approach, the body (main loop) of the automation program must be developed. In the proposed approach, an event-based approach, the body must consist of FB callings by means of methods. The problem of how to formally define this automation program is still open.

In industrial automation the purpose of a program is, known the occurred events, to produce events (i.e. the starting of an operation) in

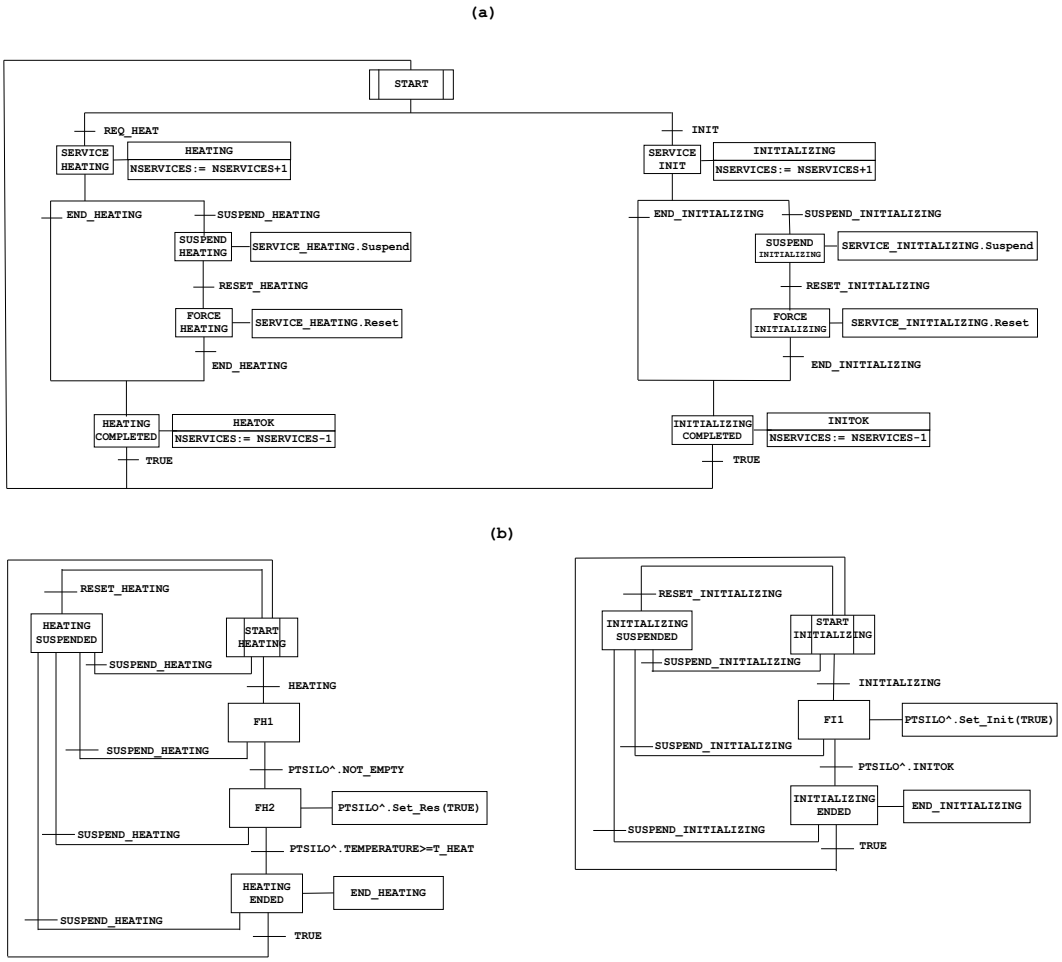


Figure 5.7 (a) ECSFC of OFB HEATING; (b) SFC implementing service requests of OFB Heating.

such a manner that a desired behavior is obtained. The desired behavior can be expressed in terms of sequences and in terms of constraints on these sequences.

Consider for instance a working cell composed by machines, conveyors and buffers: input events for the program can be the end of a working sequence, the arriving of a part at the end of a conveyor, the end of the loading of a part from a buffer; output events can be the start of a working sequence, the loading of a part on a conveyor, the loading of a part from the buffer.

The desired behavior for such a cell can be described by the correct loading sequences and by the logical constraints on these sequences, e.g. avoiding buffer overflows, mutual exclusion when using shared resources, etc.

Usually the automation program is used both to enforce desired sequences and to enforce logical constraints. Moreover, SFC language is utilized [R.D95] thanks to its ability to graphically represent sequential behavior and concurrency, and to offer a clear understanding of the input-output behavior of the controller. The problem is that this approach, common to industrial automation community, is not efficient when the system complexity grows. In such a case a formal method to design the program is needed.

Here it is proposed to model the sequences of the desired behavior by means of a PN, called PN controller, which has the role of calling FBs. The PN controller, which is implemented in the body of the automation program, sends to FBs the order to start a certain service (i.e. it forces event occurrences), and it receives from FBs the events of service completion.

This permits to look to the FBs and the PN controller as an extended process which, from an external point of view, spontaneously generates events [BC07]. On this extended process, which already enforces desired sequences, the logical constraints must be imposed; in this thesis it is proposed to use supervisory control for this purpose.

Supervisory control theory provides powerful results [RW89] for event systems where events are spontaneously generated (event occurrence cannot be forced) and, some events, named controllable events, can be disabled, while the other ones are called uncontrollable events.

As for uncontrollable events, their occurrence can be just observed. An external agent, called supervisor, disables the occurrence of controllable events in order to enforce logical constraints on the sequence of events generated by the system. In the following, it will be shown that the extended process, formed by FBs as controlled by the PN controller, generates controllable and uncontrollable events according to the point of view of supervisory control. Thus, in this thesis logical constraints are delegated to the supervisor, which is also implemented in the body of the automation program.

Notice that if the supervisor can be modeled by a PN (this is frequent in this context), a unique PN model is obtained for the automation program.

As for the supervisor synthesis, nowadays several tools can be found on the web: Ramadge-Wonham theory is implemented in Supremica [AFFM06] and in the tools available on the web page of the Systems Control Group of The University of Toronto [Sys]; Petri net based supervisory control is implemented in the tools described in [Ior], [Amb01] and [BCC07a].

The modeling and analysis of concurrent programs using PNs has been considered before [BDK98, BCD09]. The approach presented here has similarities with the problem of finding and correcting potential deadlock situations in software [WKK+08] and with the problem of automated code generation for general requirements expressed in terms of supervisory control specifications [IA10]. The main difference with the approach presented in [WKK+08] is the fact that we use PNs to write the program and to compute a supervisor that could be implemented by a PN too and not to obtain a model of the existing program to analyze deadlocks. In the program synthesis developed in [IA10] a PN model and supervisory control specification are both extracted from specification given in high level language, then a PN supervisor is computed and finally both model and supervisor are coded in a target programming language. Here, apart from the extraction of a PN from a high level language, a similar approach is used but in industrial automation context. On the one hand, that approach is made difficult by the presence of I/O signals from the field, by the fact that IEC languages only can be used and by the necessity to reuse

tested FBs. On the other hand, the specifications are simpler than the ones for the general software engineering applications or, at least, it is easier to express them directly in PN form instead of using high level languages.

Another important issue is the implementation of both the PN model and supervisor in the automation program. In the following, an algorithm is proposed for the case when a unique PN model is obtained, but the approach can be extended also to the case when the supervisor has a different model [BCC07b].

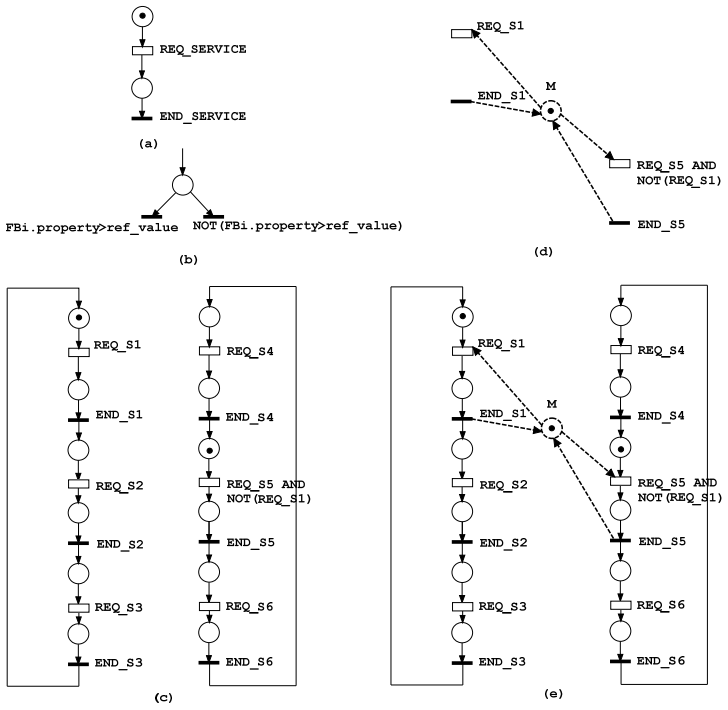


Figure 5.8 (a) Net structure implementing the call of a single service, (b) conflicting transitions, (c) an example of PN controller used to coordinate the service requests, (d) an example of PN supervisor to enforce mutual exclusion in using a common resource, (e) an example of PN model.

5.3.1 PN representation of FB services

A PLC program could be represented as a network of FBs that run concurrently to execute desired services' sequences.

According to the FB model presented in Section 5.1, each FB can execute more than one service at a time, if this is allowed by its ECSFC. Each service can be modeled as a place of a PN. If this place is marked, it means that a FB is running the associate service.

Events are related to net transitions. In particular, one event is associated with each transition and vice-versa. The calling of a service can be represented by a chain transition-place-transition, where the input transition models the service request, the output transition models the service completion, and the place the service execution. The structure in Fig. 5.8(a) implements the calling of a single service.

A PN model of the program is obtained by connecting the places according to a desired service's sequence. Hence, the aforementioned network of FBs can be seen as a PN. As a token moves from one place to another, the corresponding sequence progresses from one service to another. Thus, the various places of the PN correspond to different stages in the execution of a desired services' sequence.

5.3.2 The PN controller

Event set is divided into two disjoint sets: controllable transitions (drawn as empty boxes) and uncontrollable transitions (drawn as filled boxes).

Service requests are assumed to be controllable events, i.e. they can be disabled by a supervisor. When a controllable event occurs a command is sent by the PN controller in input to the proper FB to force the start of the associated service. Here, it is assumed that a controllable event fires as soon as it is enabled by the supervisor.

The service completion event is modeled by an uncontrollable event since its occurrence cannot be disabled. These events are generated by FBs when the sequence responsible of a certain service has been completed. Moreover, condition involving properties of a FB can be associated with uncontrollable transitions (e.g. a temperature has reached

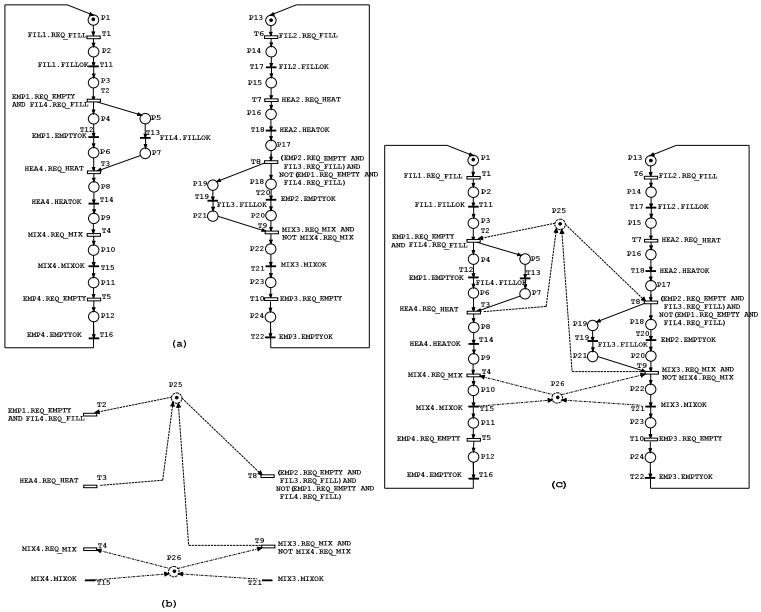


Figure 5.9 (a) PN controller; (b) PN supervisor; (c) PN model.

a reference value) as in Fig. 5.8(b).

In general, PN places may have more than one output transition. In this case a conflict occurs and the conflicting transitions must be labeled by conditions, indicating which transition should be taken when the conflict is enabled. This label can consist of a logical condition as in Fig. 5.8(b) implementing a predefined priority rule or it may be the output of a scheduler system, that is an upper level system designed to solve conflicts. The scheduler design is not a goal of this thesis.

The net in Fig. 5.8(c) is an example of a PN controller: two sequences are specified; each sequence involves three services (the first one S1, S2 and S3, the second one S4, S5 and S6).

Consider again the example presented in Subsection 5.2.1. The control of this system can be designed by properly sending the service requests to each FB according to the desired sequences and according to the logical constraints on these sequences.

From the specifications, two sequences must be implemented: 1) fill S1, pour the liquid of S1 into S4, heat S4, mix S4, empty S4; 2) fill

S2, heat S2, pour the liquid of S2 into S3, mix S3, empty S3.

Then, to implement these services' sequencing the PN model shown in Fig. 5.9(a) can be used.

5.3.3 The Supervisor

The supervisor acts on the extended process, i.e. the event set generated by the PN controller and by FBs.

If the supervisor is modeled by a PN, called PN supervisor, it consists of places and transitions, since it is a PN model. In practice, its event set is a subset of the PN controller.

Consider again the PN controller in Fig. 5.8(c). Assume two services (S1 and S5) require a common resource. A PN supervisor, as the one shown in Fig. 5.8(d) can be used. It reduces to the place M and the transition (event) set consists of REQ_S1, END_S1, REQ_S5, END_S5. Finally, the PN model in Fig. 5.8(e) is obtained.

In the general case, the supervisor could not be a PN but a logical predicate or an algorithm [LBG97]. In these cases, the supervisor is implemented as a separate program which sends to the PN controller the enabling signals for controllable events.

Consider again the example presented in Subsection 5.2.1 and the related PN controller shown in Fig. 5.9(a).

Two mutual exclusion constraints must be enforced to share the pipe and to avoid to mix the liquid of silos S3 and S4 at same time. These two constraints can be modeled by two linear inequalities

$$\begin{aligned} &\boldsymbol{m}(p_4) + \boldsymbol{m}(p_6) + \boldsymbol{m}(p_{18}) + \boldsymbol{m}(p_{20}) \geq 1 \\ &\boldsymbol{m}(p_{10}) + \boldsymbol{m}(p_{22}) \geq 1 \end{aligned} \tag{5.1}$$

where the notation $\boldsymbol{m}(p_i)$ is used to denote the marking of the place p_i. By using the approach described in [MA00] and [BCG06], the supervisor Fig. 5.9(b) can be obtained. It consists of places p_{25} and p_{26} with respective input and output arcs.

5.3.4 PN model implementation

Industrial controllers work typically by cyclically reading and storing inputs, executing user program(s), and finally writing the outputs Fig. 5.10(a). The READ-EXECUTE-WRITE cycle is called the scan cycle.

In Fig. 5.10(b) the evolution algorithm proposed for the implementation of the PN controller and the PN supervisor is reported. In the following the main data structures used by the algorithm are discussed.

The PN model can be translated in any IEC 61131 programming language according to the evolution algorithm shown in Fig. 5.10(b). In the following, Structured Text (ST) is used just for the sake of readability. The approach can be easily extended to other IEC 61131 languages. As for Instruction List, the reader can refer to the implementation of Automation PN (APN) [JU96], an extension of PN, or to the implementation of Signal Interpreted Petri Nets (SIPN) [Fre00]. As for the Ladder Diagram, a good survey is presented in [PZ04]. A good list of references about the problem of implementing PNs on industrial control systems can be found also in [BC07], where the problems of implementing supervisory control are discussed.

A vector of integer variables, named M is needed to store the net marking. A vector of Boolean input variables, named UCT, is allocated for the set of signals associated with uncontrollable transitions. Events, i.e. uncontrollable transitions firings, are derived from boolean-valued signals by detecting rising and falling edges between two consecutive scan cycles using R_TRIG that is a predefined function block in IEC 61131 standard. A vector named UTRIG is associated with the set of uncontrollable transitions. If at least one transition firing occurred, the net marking is updated and the enabling of controllable transitions is evaluated. A vector of Boolean output variables, named CT, is allocated for the set of controllable transitions. For each enabled controllable transition, the corresponding element in CT is set to true and the corresponding service is requested. Note that controllable transitions fire as soon as they are enabled since they represent service requests.

PN controller and supervisor can be both implemented in the main loop of the PLC program. The body of such a program, named SI-

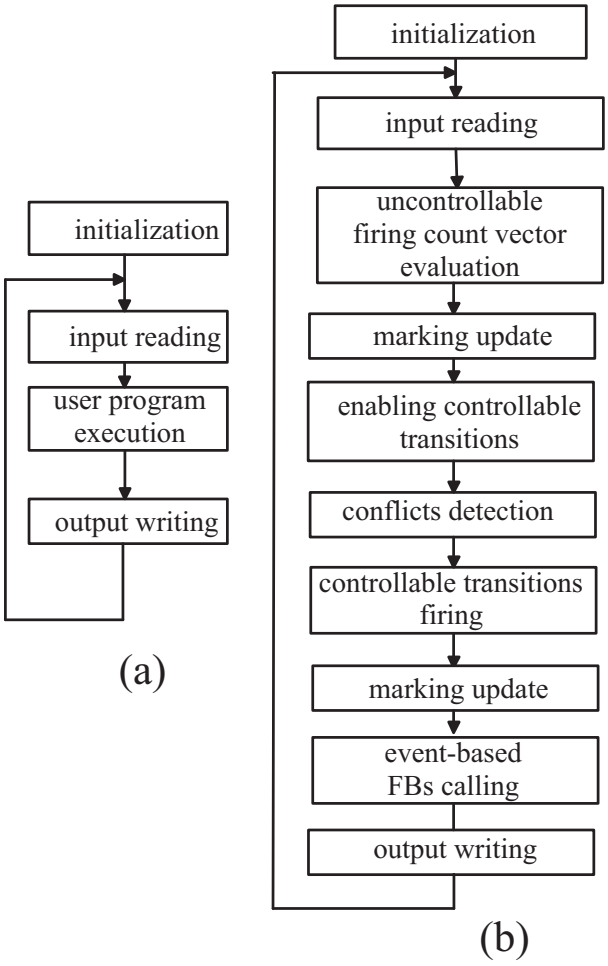


Figure 5.10 (a) Scan cycle. (b) Evolution algorithm for a controller and a supervisor modeled by net structures if state equation is used to update marking.

LOS COORDINATION, consists in implementing the PN model in Fig. 5.9(c) and in connecting properly each declared FB. For the sake of brevity, the variable declaration section and only some parts from each section of the program are reported below, as well as the section where each FBs is initialized is omitted.

```
PROGRAM SILOS COORDINATION
  VAR_INPUT
        (* physical inputs declaration *)
  END_VAR
  VAR_OUTPUT
        (* physical outputs declaration *)
  END_VAR
  VAR
        S1:SILO;
        S2:SILO_WITH_MIXER_AND_HEATHER;
        S3:SILO_WITH_MIXER_AND_HEATHER;
        S4:SILO_WITH_MIXER_AND_HEATHER;
        FIL1:FILLING(PTSILO:=ADR(S1));
        EMP1:EMPTYING(PTSILO:=ADR(S1));
        FIL2:FILLING(PTSILO:=ADR(S2));
        EMP2:EMPTYING(PTSILO:=ADR(S2));
        HEA2:HEATING(PTSILO:=ADR(S2));
        FIL3:FILLING(PTSILO:=ADR(S3));
        EMP3:EMPTYING(PTSILO:=ADR(S3));
        MIX3:MIXING(PTSILO:=ADR(S3));
        FIL4:FILLING(PTSILO:=ADR(S4));
        EMP4:EMPTYING(PTSILO:=ADR(S4));
        MIX4:MIXING(PTSILO:=ADR(S4));
        HEA4:HEATING(PTSILO:=ADR(S4));
        UCT : ARRAY [1..12] OF BOOL;
        UTRIG : ARRAY [1..12] OF R_TRIG;
        CT : ARRAY [1..10] OF BOOL;
        UPDATE:BOOL;
        M: ARRAY [1..26] OF INT; (* PLACE MARKING *)
        T: ARRAY [1..10] OF BOOL; (* TRANSITIONS *)
        dummy,dummy2:BOOL;
  END_VAR
(* INITIALIZING SECTION*)
  M[1]:=1; M[13]:=1; M[25]:=1; M[26]:=1;
(* EVENT DETECTION SECTION *)
  FOR COUNTER:=1 TO 12 BY 1 DO
        UTRIG[COUNTER](CLK:= UTC[COUNTER]);
        IF UTRIG[COUNTER].Q
          THEN UPDATE := TRUE;
        END_IF;
  END_FOR;
(* UPDATE SECTION *)
  IF UPDATE THEN
        (* STATE UPDATE SECTION: UNCONTROLLABLE TRANSITIONS *)
        IF UTRIG[1].Q THEN
          M[3]:=M[3]+1;
          M[2]:=M[2]-1;
```

```
        END_IF
        .......
        (* OUTPUT EVALUATION SECTION *)
        IF M[1]≥ 1 THEN
          CT[1]:=TRUE;
          T[1]:=TRUE;
        END_IF
        .......
        (* STATE UPDATE SECTION: CONTROLLABLE TRANSITIONS *)
        IF T[1] THEN
          M[2]:=M[2]+1;
          M[1]:=M[1]-1;
        END_IF
        .......
  END_IF
(* event-based FB CALLS *)
  .......
IF (CT[3] OR HEA4.NSERVICES) THEN HEA4(INIT:= 0, REQ_HEAT:= CT[3],
        T_HEAT:= 60, INITOK:= dummy,
        HEATOK:=UCT[4], N.SERVICES:=dummy2);
IF (CT[9] OR MIX3.NSERVICES) THEN MIX3(INIT:= 0, REQ_MIX:=CT[9],
        T_MIX:=15, EMPTY:=%IX5, INITOK:= dummy,
        MIXOK:=UCT[11], N.SERVICES:=dummy2);
  .......
END_PROGRAM
```

For the sake of simplicity, transitions are ordered in such a way that t_1 , ..., t_{10} are controllable transitions, while t_{11} , ..., t_{22} are uncontrollable transitions.

The values can be assigned to the parameters in parentheses after the instance name of the function block.

```
.....
FB_Instance_name(Start_Service1:=REQ_S1, ..., Start_Servicen:= REQ_Sn, Data1:=A1, ..,
          ...,Datan:=An , End_Service1:=END_S1 , ..., End_Servicen:= END_Sn);
.....
```

Function blocks are always called through a function block instance using an IF statement, as discussed in Subsection 5.1.2, to ensure that a FB is activated when at least one controllable event occurs and is kept activated while at least one service is being executed.

Note that for each enabled controllable transition the corresponding service is requested. Finally, FBs are called by properly linking service requests and service completion variables. In this work start service values are associated to controllable transitions, end service values are associated to uncontrollable transitions, while data can be

auxiliary data or field I/O data. As for example, the controllable transition t_3 corresponds to REQ_HEAT of S4, HEAT_OK of S4 corresponds to the uncontrollable transition t_{14}, T_HEAT is a data related to the heating temperature.

5.4 Implementation on PLCs using OOP

In this section a method is proposed to implement controllers and supervisors designed by Petri Nets on PLCs using OOP .

An algorithm is here proposed to implement a unique PN model formed by both a PN controller and a PN supervisor. It is based on two main class objects: place and transition. As more, the PLCopen XML format [PLC09] is used to code the PN model and import it in a development environment which supports PLCopen XML and OOP (e.g. CoDeSys).

It is worth noticing that the proposed method preserves the structure of PNs. In the resulting program each instruction is directly related to the evaluation of a transition, to the update of the marking, or to the enabling of a transition. This preserves the auto-documentation features of PNs and allows the user to easily modify the program directly, starting from the modifications made to the PN design. The method is presented for Structured Text (ST) language but it can be adapted to other programming language for PLCs supporting OOP.

PLCopen is an independent organization providing efficiency in industrial automation based on the needs of users. PLCopen members have concentrated on technical specifications around IEC 61131, creating specifications and implementations in order to reduce cost in industrial engineering. The outcome is standardized libraries for different application fields, harmonized language conformity levels and engineering interfaces for exchange, as for example the PLCopen XML.

In Fig. 5.11 steps to implement PNs in IEC 61131 languages extended with OOP are shown. Step 1 consists in the PN design by the user. The goal of this section is the discussion of steps 2 and 3.

Step 2. A MATLAB® script has been implemented. It requires the number of the PN places and transitions, and the pre and post-

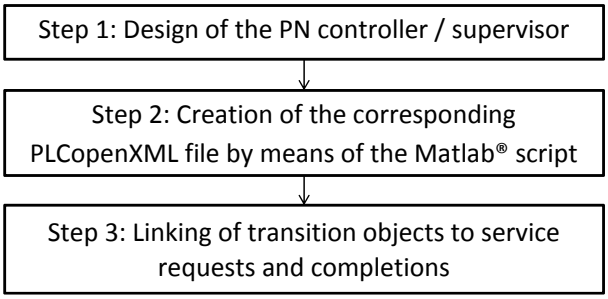


Figure 5.11 Workflow to implement PNs on PLCs.

incident matrices and generates a PLCopen XML file that implements the structure of the PN and its evolution. The file is composed by two parts: variables declaration and body of program. The body of program, where the evolution of the PN is coded, is subdivided in the following 3 sections: *Initializing Section* where the structure of the net is created and its initial marking is set; *Event Detection Section* where the enabling of each transition is evaluated; *Update Section* where the marking of the net is updated based on the fired transitions.

Note that the user has not to be confident with PLCopen XML format.

The script requires the existence of a library composed by two basic classes (as extension of standard FBs), Place and Transitions, presented in the following.

In Fig. 5.12(a) FB Place is shown. Its inputs are: EVENT_IN (EVENT_OUT), the pointer to the array of events associated with place preset (postset) transitions; PRE (POST), the pointer to the array of weights associated with the input (output) arcs. Its output is the value of the marking of the place.

Methods of the FB Place are: *setArcs*, (it defines the place preset and postset), *setInitialMarking* (it sets its initial marking), *updateMarking* (it updates its marking), *update* (it allows to add a new input - output- arc).

FB Transition (Fig. 5.12(b)) has three input variables: EVENT, a boolean variable equal to 1 when the event (i.e., a service request/completion, or a supervisor enabling) associated with the transition occurs, otherwise it is 0; PRE, the pointer to the pre-incident matrix row related to

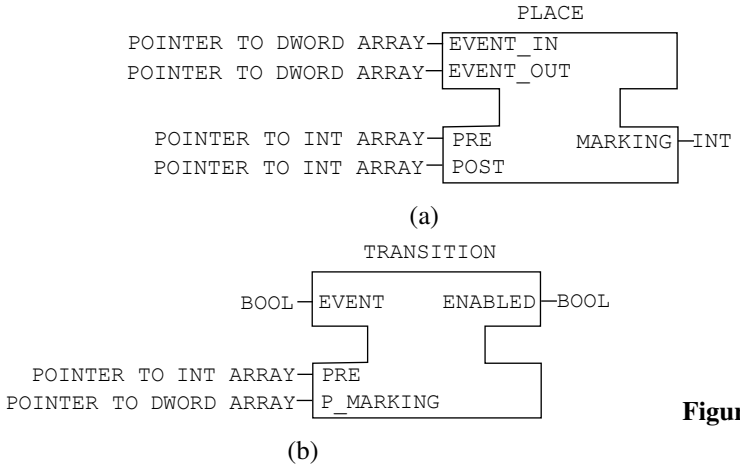


Figure 5.12 (a) Function Block Place and (b) Function Block Transition.

the transition; P_MARKING, the pointer to the marking of the input places. Its output is ENABLED, the boolean variable that is equal to 1 if the transition is enabled, otherwise it is 0.

Methods of FB Transitions are *setArcs* (it defines the transition preset), *value* (it evaluates the transition) and *update* (it allows to add a new place to the transition preset).

The code implementing the FB Transition is shown in Fig. 5.13.

The key idea of this code is the use of pointers and of the operator __New: it allocates memory for FB instances or arrays of standard data types. The operator returns a suitably typed pointer to the object. In this way the memory should be allocated dynamically for the application.

Memory dynamic allocation is not a new concept in the computer science but it is not used in PLC field. Indeed, such an operator is not prescribed by the IEC 61131 standard, it is only supported by CoDeSys V3.

The use of __New allows the definition of an unique FB Transition, since no limitation to the maximum number of input arcs have to be introduced: the transition can have any number of arcs (only limited to the memory of the device).

Step 3. The program automated generated by means of the PLCopen XML file must be integrated by the user, linking the transition objects

to the FB service requests (completions).

Consider the example presented in 5.2.1.Silos S3 and S4 cannot be filled at the same time since there is only a single pipe to do this operation. It can be decided if executing the mixing of the liquid in silos S3 and S4 contemporaneously or not, setting the variable *power* equal to 0 or 1, respectively. the PN modeling the supervised process is shown in Fig. 5.14: when contemporaneous mixing is allowed, place P26 is not connected to the rest of net, it is linked when the mixing in S3 and S4 can not occur at the same time.

The ST code, implementing such a PN is shown in Fig. 5.15: the bold text lines are the ones manually added by the user to link the firing of transitions to the service requests and completions; *ADDITIONAL CONSTRAINTS SECTION* is added to allow the two different configurations of the net.

When the constraints about mixing is activated, calling the methods *update* of the classes Place and Transition, new arcs can be added, linking the place P26 to net transitions: without the dynamic allocation, since each place (transition) must have a fixed number of arcs, two different programs would have been implemented.

Notice that it is easy to modify the PN if the system specifications are changed: as example the adding (delating) of a node only requires to change the corresponding parameters (number of places (transitions), incidence matrices) and to execute again the Matlab® script.

The approach can be applied to large and complex systems, with the only limitation deriving from the physic limits of PLCs.

```
FUNCTION_BLOCK TRANSITION
  VAR_INPUT
        P_MARKING:POINTER TO DWORD; (* marking of input places *)
        PRE:POINTER TO UINT; (* pre-incindent matrix row related to transition *)
        EVENT:BOOL;
  END_VAR
  VAR_OUTPUT
        ENABLED:BOOL;
  END_VAR
  VAR
        N_P_IN:UINT; (*number of input places *)
  END_VAR
  METHOD setArcs
        VAR_INPUT
          N_P_INPUT:UINT;
        END_VAR
        N_P_IN :=N_P_INPUT;
        PRE:=__NEW(INT, N_P_IN);
        P_MARKING:=__NEW(DWORDS, N_P_IN);
  END_METHOD
  METHOD value
        VAR
          EN,COUNTER: UINT;
          PT:POINTER TO INT;
        END_VAR
        EN:=0;
        ENABLED:=FALSE;
        FOR COUNTER:=1 TO N_P_IN BY 1 DO
          PT :=TOKENS[COUNTER];
          IF PT^ >= PRE[COUNTER] THEN
            EN:=EN+1;
          END_IF
        END_FOR;
        IF EN=N_P_IN AND EVENT THEN
          ENABLED:=TRUE;
        END_IF;
  END_METHOD
  METHOD PUBLIC update
        VAR_INPUT
          WEIGHT:UINT;
          PLACE_MARKING:DWORD;
        END_VAR
        VAR
          p: UINT;
          PRE_temp:POINTER TO UINT;
          P_MARKING_temp:POINTER TO DWORD;
          New_DIM: UINT;
        END_VAR
        PRE_temp:=__NEW(INT, N_P_IN);
        P_MARKING_temp:=__NEW(DWORD, N_P_IN);
        FOR p:=1 TO N_P_IN BY 1 DO
          PRE_temp[p]:=PRE[p];
          P_MARKING_temp[p]:=P_MARKING[p];
        END_FOR
        __DELETE(PRE);
        __DELETE(P_MARKING);
        New_DIM:=N_P_IN+1;
        PRE:=__NEW(INT, New_DIM);
        P_MARKING:=__NEW(DWORD, New_DIM);
        FOR p:=1 TO N_P_IN-1 BY 1 DO
          PRE[p]:=PRE_temp[p];
          P_MARKING[p]:=P_MARKING_temp[p];
        END_FOR
        PRE[New_DIM]:=WEIGHT;
        P_MARKING[New_DIM]:=PLACE_MARKING;
        N_P_IN:=N_P_IN+1;
        __DELETE(PRE_temp);
        __DELETE(P_MARKING_temp);
  END_METHOD
  (* FB body is empty *)
END_FUNCTION_BLOCK
```

Figure 5.13 ST code implementing FB Transition.

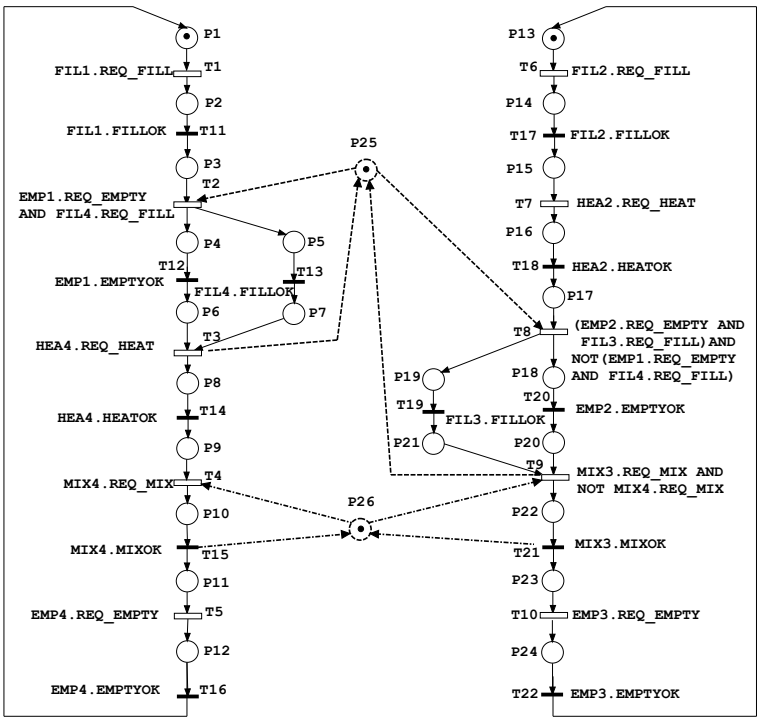


Figure 5.14 Petri Net model.

```
PROGRAM SILOS COORDINATION
  VAR_INPUT
        (* physical inputs declaration *)
  END_VAR
  VAR_OUTPUT
        (* physical outputs declaration *)
  END_VAR
  VAR_CONSTANT
        nPlaces: UINT :=26; (* places of net *)
        nTransitions: UINT :=22; (* transition of net *)
  END_VAR
  VAR
        S1, S2, S3, S4:SILO_WITH_MIXER_AND_HEATHER;
        FIL1:FILLING(PTSILO:=ADR(S1));
        ...
        MIX3:MIXING(PTSILO:=ADR(S3));
        HEA4:HEATING(PTSILO:=ADR(S4));
        PRE:ARRAY[1 .. nPlaces, 1 .. nTransitions] OF UINT :=[ ... ]
        POST:ARRAY[1 .. nPlaces, 1 .. nTransitions] OF UINT:=[ ... ]
        places:ARRAY [1 .. nPlaces] OF PLACE;
        transitions:ARRAY[1 .. nTransitions] OF TRANSITION;(* transitions *)
        marking:ARRAY[1 .. nPlaces] OF UINT:=[ ... ]
        temp,temp2, temp3: UINT:=1;
        p,t,index,index2:: UINT := 0;
        step0,dummy,dummy2:BOOL:=1;
        power:BOOL:=0;
  END_VAR
(* INITIALIZING SECTION*)
  IF step0 THEN
        FOR p:=1 TO nPlaces BY 1 DO
          temp,temp2:=0;
          FOR t:=1 TO nTransitions BY 1 DO
            IF PRE[p,t]> 0 THEN
                temp:=temp+1;
            END_IF
            IF POST[p,t]> 0 THEN
                temp2:=temp2+1;
            END_IF
          END_FOR
          places[p].setArcs(temp2,temp); (* set the number of arcs *)
        END_FOR
        FOR t:=1 TO nTransitions BY 1 DO
          temp:=0;
          FOR p:=1 TO nPlaces BY 1 DO
            IF PRE[p,t]> 0 THEN
                temp:=temp+1;
            END_IF
          END_FOR
          transitions[t].setArcs(temp); (* set the number of arcs *)
        END_FOR
        FOR p:=1 TO nPlace BY 1 DO
          index,index2:=1;
          FOR t:=1 TO nTransitions BY 1 DO
            IF PRE[p,t]> 0 THEN
                places[p].POST[index]:= PRE[p,t];
                places[p].EVENT_OUT[index] := ADR(transitions[t].ENABLED) ;
                index:=index+1;
            END_IF
            IF POST[p,t]> 0 THEN
                places[p].PRE[index2]:= POST[p,t];
                places[p].EVENT_IN[index2] := ADR(transitions[t].ENABLED) ;
                index2:=index2+1;
            END_IF
          END_FOR
        END_FOR
        FOR t:=1 TO nTransitions BY 1 DO
          index:=1;
          FOR p:=1 TO nPlaces BY 1 DO
            IF PRE[p,t]> 0 THEN
                transitions[t].PRE[index]:=PRE[p,t];
                transitions[t].P_MARKING[index]:=ADR(places[p].MARKING);
                index:=index+1;
            END_IF
          END_FOR
        END_FOR
        FOR p:=1 TO nPlaces BY 1 DO
          places[p].setInitialMarking(marking[p]);
        END_FOR
        step0:=0;
  END_IF
(* event-based FB calls *)
  .......
IF ((transitions[3].ENABLED OR HEA4.NSERVICES> 0)) THEN
  HEA4(INIT:= 0, REQ_HEAT:= transitions[3].ENABLED, T_HEAT:= 60,
   INITOK:= dummy,  HEATOK:=transitions[14].EVENT, NSERVICES:=dummy2);
  END_IF
IF((transitions[9].ENABLED OR MIX3.NSERVICES> 0)) THEN
  MIX3(INIT:= 0, REQ_MIX:=transitions[9].ENABLED, T_MIX:=15,EMPTY:=%IX5,
   INITOK:= dummy, MIXOK:=transitions[21].EVENT, NSERVICES:=dummy2);
  END_IF
(* EVENT DETECTION SECTION *)
  IF temp3 THEN
        FOR t:=1 TO nTrans BY 1 DO
          transitions[t].value();
        END_FOR
        FOR t:=1 TO nTransitions BY 1 DO
          IF transitions[t].ENABLED THEN temp3:=0; END_IF
        END_FOR
  ELSE
(* UPDATE SECTION *)
        FOR p:=1 TO nPlace BY 1 DO
          places[p].updateMarking();
        END_FOR
        temp3:=1;
  END_IF
 (* ADDITIONAL CONSTRAINTS SECTION *)
  IF power THEN
        .....
        transitions[4].update(1,ADR(places[26].MARKING ) );
        transitions[9].update(1,ADR(places[26].MARKING) );
  END_IF
  .....
END_PROGRAM
```

Figure 5.15 ST code, implementing PN of Fig. 5.14.

Chapter 6

Conclusion and further research

The results of the previous chapters are here summarized. Several observations regarding them are made and some possible extensions of this research are discussed. More in detail a preliminary results in developing a flexible, modular and distributed control architecture using a cyber-physical system perspective are presented.

6.1 What has been accomplished

In this dissertation a modeling framework for semi-automated hyper-flexible robotic cells in aircraft industry has been proposed.

The goal of chapter 4 is the synthesis of a model for the sequencing of the activities of a robotic cell in aircraft industry. The robotic cell is built, starting from the models of resource and task blocks, that are the cell elementary components. A systematic algorithm that yields a Colored Modified Hybrid Petri Net model of the whole cell, starting from resource and task models, has been presented.

In the chapter 4, an object oriented approach, the programming language SFC together with a proper way to organize the inputs and outputs of FBs and supervisory control are proposed to implement industrial automation control systems to meet the new challenges of this field. FBs are assumed to be objects with methods and properties.

Methods together with a SFC help to make event-based the execution order of FBs. Moreover, FBs are seen as service providers, according to a service-oriented paradigm. Furthermore, it has been shown that supervisory control can be adopted to solve the coordination, in the context of industrial control, of the concurrent behavior of the several FBs which constitute a typical automation software application. Indeed, the desired behavior of such an application results to be expressed in terms of desired sequences of events. PNs have been chosen since efficient methodologies are available to apply supervisory control using this formal model and it has been shown that they can be easily implemented on commercial PLCs using Object Oriented Programming. The approach can be applied to large and complex systems, with the only limitation deriving from the physic limits of PLCs.

The last part of this dissertation focuses on the control of automated warehouse systems which are at service or are integral part of hyperflexible robotic workcells .

6.2 Cyber-Physical Systems

The Industrial Revolution is a concept and a development that has fundamentally changed our society and economy. The term *development* may seem to indicate some tardiness in the context of a revolution, which really signifies a rapid and fundamental change, but there is no doubt that major alterations occurred within a relatively short period. Industries arose and replaced small-scale workshops and craft studios. Textile and pottery factories were the first to recognize the new dawn, and a new infrastructure of canals and railway lines enabled efficient distribution. It was the transition from industrious to industrial, and the start of a boom for both. From the first mechanical loom, dating from 1784, exactly 230 years ago, we can distinguish four stages in the ongoing process called the Industrial Revolution. That is the way we currently look at it. The first acceleration occurred toward the end of the 18th century: mechanical production on the basis of water and steam. We place the Second Industrial Revolution at the beginning of the 20th century: the introduction of the conveyor belt and

mass production, to which the names of icons such as Henry Ford and Frederick Taylor are linked. Number three is the digital automation of production by means of electronics and it. At present, we find ourselves at the beginning of this fourth stage, which is characterized by so-called "Cyber-Physical Systems" (CPS). These systems are a consequence of the far-reaching integration of production, sustainability and customer-satisfaction forming the basis of intelligent network systems and processes [vin].

CPS is a promising new class of systems that deeply embed cyber capabilities in the physical world, either on humans, infrastructure or platforms, to transform interactions with the physical world. Advances in the cyber world such as communications, networking, sensing, computing, storage, and control, as well as in the physical world such as materials, hardware, and renewable green fuels, are all rapidly converging to realize this class of highly collaborative computational systems that are reliant on sensors and actuators to monitor and effect change. A core differentiator of CPS is the tight conjoining of and coordination between cyber and physical resources, which yields unprecedented capabilities. Traditional cyber systems were usually considered to be the passive, dumb part in the physical world, but with CPS, we have to now take into account what is being moved or changed in the physical world. A major difference between CPS and a regular control system or an embedded system is the use of communications, which adds reconfigurability and scalability as well as complexity and potential instability. Furthermore, CPS has significantly more intelligence in sensors and actuators as well as substantially stricter performance constraints. Today, examples of nascent CPS are emerging across sectors, such as flight control and electrochromic cabin windows in airplanes, adaptive cruise control and antitheft devices in cars, location services in cell phones, field devices in power grids, pacemakers in humans, robotic vacuum devices at homes, entertainment, gaming, and haptic systems. However, many existing systems either do not focus on cyber-physical interactions or are far more capable of richer cyber-physical interactions [Poo10].

The economic and societal potential of CPS is believed to be tremendously greater than what has been achieved by existing systems in

terms of autonomy, adaptability, efficiency, flexibility, and versatility. However, many technical challenges need to be addressed before we can take full advantages of CPS. One of the major challenges is just-in-time assembly of networked physical entities into desired capabilities. To facilitate rapid integration of physical entities to achieve desired tasks, it is necessary to model these entities and support discovery and composition of their capabilities[HBYZ10]. Existing SOA can be used for the modeling, management, and integration of physical entities in CPS. The capabilities of the physical entities can be wrapped as services and SOA technologies, such as service discovery and composition, can be applied. However, SOA models do not provide full solution and some new concepts and extensions are required. Also, the SOA model for CPS should include a (unified) modeling approach to facilitate automated composition. Specifically, we need to consider

- **Specification of the physical entity (PE) and the services it provides**. In existing models, the service provider (the PE) is not well defined. The providers of software services may not be important. But the provider of physical services has critical physical meanings. For example, a PE can only be at one location at one time and cannot provide its services out of its context.
- **Specification of tasks**. Task specification is critical in automated composition. The specification model should allow easy correlation of tasks and services.

6.2.1 A cyber-physical approach to automated warehouse systems

The contribution of this section is to present a cyber-physical perspective view about the control of automated warehouse systems, where the tradition multilevel architecture is replaced by a set of cyber-physical components interacting each other and implementing management, optimization and execution of handling sequences in a very distributed/modular way. The hierarchical control architecture, explained in the second chapter, benefits from the separation of functionalities but limits the

modularity and flexibility of the control system, being MS and OS implemented in a centralized way. However, the recent development of intelligent networked embedded systems and technologies, ranging from components and software to CPS could help to change this.

The umbrella paradigm underpinning novel collaborative systems is to consider the set of intelligent system units as a conglomerate of distributed, autonomous, intelligent, proactive, fault-tolerant and reusable units, which operate as a set of cooperating entities. These entities are capable of working in a proactive manner, initiating collaborative actions and dynamically interacting with each other to achieve both local and global objectives.

This evolution towards global service-based infrastructures indicates that new functionality will be introduced by combining services in a cross-layer form, i.e. services relying on the enterprise system, on the network itself and at device level will be combined. New integration scenarios can be applied by orchestrating the services in scenario-specific ways. In addition, sophisticated services can be created at any layer (even at device layer) taking into account and based only on the offered functionality of other entities that can be provided as a service.

The cyber part of the control architecture in automated warehouse systems reduces to the control algorithms implemented according to International Electrotechnical Commission (IEC) programming standard. Indeed, in industrial automation the whole control functionalities are assigned to software executed on general purpose embedded boards or PLCs, while no functionalities are assigned to the electronic parts [Thr14].

As explained in the previous chapter, programming of automation systems is largely based on the IEC 61131 standard and on 61499 standard. FBs, as offered by both the standards 61499 and 61131, can be considered as an emerging architectural framework for the design of distributed industrial automation systems making possible the implementation of multiagent and holonic control systems [Thr13]. As an example, in [BV10] a very interesting multiagent control approach is proposed for a baggage handling system (BHS) using IEC 61499 FBs. In particular, it focuses on demonstrating a decentralized control system that is scalable, reconfigurable, and fault tolerant. The design fol-

lows the automation object approach, and produces a FB component representing a single section of conveyor. In accordance with holonic principles, this component is autonomous and collaborative, such that the structure and the behavior of a BHS can be entirely defined by the interconnection of these components within the FB design environment.

As for the physical part, it can be assumed that, at a certain level of abstraction, there is a software component for each physical unit, e.g. a conveyor, an elevator, and so on. However, the complexity of modern warehouse systems, like the real one considered in [BCC12b], requires big interfaces (e.g. a carousel, shuttles, rail guided vehicles) between cranes and picking area and so many vehicles must be used. Furthermore, when each crane cycle involves more than one picking and deposit, the number of SUs moved by vehicles at a time in the interface area grows, and then a significant time is required to cover the interface guidepath. In practice, vehicle as well as conveyors can be moved with constant speed, can stop at the interface points to load (unload) a SU from (to) another subsystem bay, and can vary their speed with constant acceleration if a particular condition occur (distance between two vehicles goes under a threshold, a particular point is reached...)[BCC12b]. To conclude, physical dynamic influence the optimization, developed essentially in the cyber part.

Such a feedback loop between physical processes and computations encompasses sensors, actuators, physical dynamics, computation, software scheduling, and networks with contention and communication delays.

6.2.2 Cyber component model

In this section the focus is on the cyber part: a perspective view of automated warehouse system components, implementable by FB models, is presented and the effects of the physical part reduces to the prediction of SU position through time assuming a motion with constant speed or acceleration. Communication delays and software scheduling are not taken into account.

To obtain a Cyber-Physical Component (CPC) each FB is associ-

ated to a physical component (a conveyor, an elevator, etc.) to implement its cyber part where traditional and collaborative/intelligent functionalities are implemented.

As for example, traditional functionalities are:

- Ability to move objects from one end to the other. At this aim each component is equipped at least with a photocell at its beginning plus one at the end, and a motor drive.
- Ability to indicate readiness to receive/deliver SUs in order to avoid inappropriate SU transfers from/to upstream/downstream component.
- To manage the shifting of tracking data according to the real SU movements in the plant.

Notice that conveyors implementing divert have an additional photocells, elevators are equipped with additional motors to move along different levels. These functionalities can be easily added in a OOP framework using inheritance.

As for example, collaborative/intelligent functionalities are:

- Ability to change speed/acceleration according to the SU weight to save energy and/or to avoid breaking the loaded SUs.
- Prediction of SUs position on the component through time.
- Ability to cooperate with other components to implement a dynamic path building algorithm (a first contribution can be found in [BV10]).
- Ability to choose an optimal path for a certain SUs.

Traditional functionalities represent what is called HS, while collaborative/intelligent functionalities make possibile to implement OS and MS at a device level in a completely distributed way. Just the warehouse location map must be centralized.

6.2.3 Case study

The proposed approach has been used to move SUs in the automated warehouse system prototype installed at the Automatic Control and Robotics Laboratory of the University of Salerno, shown in Fig. 6.1(a).

It is a modular system, made up of conveyor belts and elevators which move SUs in horizontal as well as in vertical way. The handling system is subdivided in three zones named, respectively, Zone A, Zone B and Zone C and each zone is subdivided in four level, (Level 1-Level 4 in Fig. 6.1(b)). The system is controlled by a PLC Siemens S300 able to acquire sensors values and command actuators by means of three remote I/O modules, each one associated to a different zone, which communicate with the PLC by means of a field bus.

Some conveyor belts of each zone are used as storage locations (A3, A4, B4, B5, B6, B10, B11, C3 and C4 in Fig. 6.1(b)). Each physical node of the system (i.e. locations, elevators, belts) together with its FB model represents a CPC capable to define its states (*running* or *out of order*), to know its neighbours for a given direction, its length and to set its speed/acceleration.

As for the speed/acceleration setting, in a cyber-physical perspective a lot of functionalities can be implemented in each CPC. As for example, energy saving and SU breaking avoidance have been considered.

Energy cost varies during the working-day a node accelerates/slows down when the energy cost decrease/rise. At this aim, each component receives as input info about energy cost and computes the right speed.

CPC A1, which is the input point of the system, is equipped with an RFID reader, so SUs are identified and a set of technical data, like its weight and form factor, are acquired. Weight and form factor are key factors in setting the speed of component to prevent the breaking of the SU they carry on. As a consequence, each component receives these data and set the speed to the maximum value with respect to the SUs carried on.

For sure, dynamic path planning is another functionality that can be implemented in a CPC of an automated warehouse. When a storage mission has to be performed, CPC A1 sends to MS the ID number,

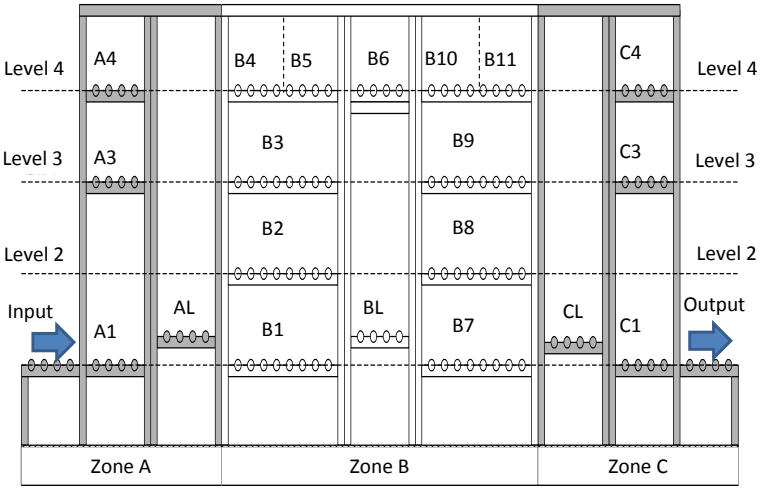
in order to build the minimum length path in a distribute way; MS identifies some possible storage locations for the item and returns back the list to the input point. Soon after, CPC A1 sends a storage request to each location on the list. Provided that the state of the locations is *running*, they back propagate the request to all their neighbours. Unless a node is out of order, it adds the information about its length to the message and, in turn, back propagates the request to its adjacent nodes, different from the sender. When the storage requests return to CPC A1, all the feasible paths are defined: CPC A1 selects the best one depending on its length, given by the sum of each segment it is made of.

Fig. 6.2 shows the possible twenty-seven paths for the storage of a SU, assumed that MS indicates B4, B5, B6, B10 and B11 as possible storage locations to CPC A1. As more, it has been assumed that locations B4 and B5 are out of order, elevators AL, BL and CL are placed at first, second and fourth level, respectively.

Two different methodologies have been used to develop the FBs modeling the nodes of the system: the Codesys Development System, compliant to IEC 61131 standard, third version, so benefiting from the OOP; Forte, which is an IEC 61499 Compliant Runtime Environment.

Preliminary results about a novel cyber-physical approach to the design of automated warehouse systems has been presented in this section. The results show that a lot of basic and interesting tasks can be implemented at a device level. These tasks can include, as for example, dynamic path planning and speed setting. Future research efforts will focus on a full integration of cyber-physical systems in the hyper-flexible robotic work cell.

(a)



(b)

Figure 6.1 (a) The prototype of automated handling system installed at the University of Salerno; (b) its layout.

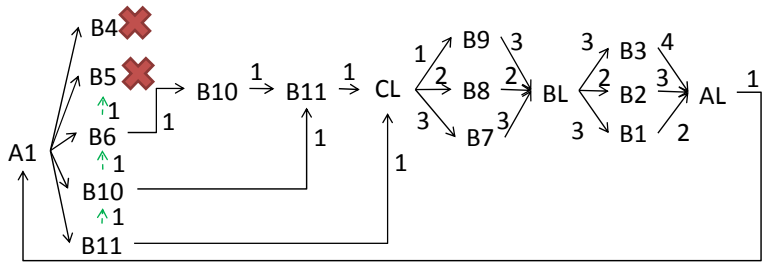


Figure 6.2 Message exchange between the input point and the candidate storage locations. Numbers on the arcs represent the length of each segment of the path. The shorter path is obtained storing the SU in B11, passing by B1 or B2.

Bibliography

[3S 12] 3S Smart Software Solutions. Codesys v3 - web page. *http://www.3s-software.com*, 2012.

[ABCC05] Francesco Amato, Francesco Basile, Ciro Carbone, and Pasquale Chiacchio. An approach to control automated warehouse systems. *Control Engineering Practice*, 13(10):1223 – 1241, 2005.

[AD94] G. Alici and R.W. Daniel. Robotic drilling under force control: execution of a task. In *Intelligent Robots and Systems '94. 'Advanced Robotic Systems and the Real World', IROS '94. Proceedings of the IEEE/RSJ/GI International Conference on*, volume 3, pages 1618–1625 vol.3, September 1994.

[AFFM06] K. Akesson, M. Fabian, H. Flordal, and R. Malik. Supremica - an integrated environment for verification, synthesis and simulation of discrete event systems. 8^{th} *International Workshop on Discrete Event Systems (WODES'06)*, pages 384–385, July 2006.

[Amb01] M. Ambu. Petri net toolbox. *http://www.diee.unica.it/giua/ARP/SOFT/*, 2001.

[ASV08] Adriano José Cunha de Aguiar, Alex Sandro de Araújo Silva, and Emília Villani. Graphic robot simulation for the design of work cells in the aeronautic industry. In *ABCM Symposium Series in Mechatronics*, volume 3, pages 346–354, 2008.

[BC01] M.S. Branicky and S.R. Chhatpar. A computational framework for the simulation, verification, and synthesis of force-guided robotic assembly strategies. *Int. Conf. on Intelligent Robots and Systems (IROS'01)*, 3:1471–1476, 2001.

[BC07] F. Basile and P. Chiacchio. On the implementation of supervised control of discrete event systems. *IEEE Transactions on Control Systems Technology*, 15(4):725–739, July 2007.

[BCC07a] F. Basile, C. Carbone, and P. Chiacchio. Simulation and analysis of discrete-event control systems based on Petri nets using PNetLab. *Control Engineering Practice*, 15(2):241–259, February 2007.

[BCC07b] F. Basile, P. Chiacchio, and C. Carbone. Feedback control logic for backward conflict free choice nets. *IEEE Transactions on Automatic Control*, 52(3):387–400, March 2007.

[BCC+12a] F. Basile, F. Caccavale, P. Chiacchio, J. Coppola, and C. Curatella. Task oriented motion planning for multi arm robotic systems. *Robotics and Computer Integrated Manufacturing*, 29:569–582, 2012.

[BCC12b] F. Basile, P. Chiacchio, and J. Coppola. A hybrid model of complex automated warehouse systems - part I: Modeling and simulation. *IEEE Transactions on Automation Science and Engineering*, 9(4):640–653, 2012.

[BCC12c] Francesco Basile, Pasquale Chiacchio, and Jolanda Coppola. A Hybrid Model of Complex Automated Warehouse Systems - Part I: Modeling and Simulation. *IEEE Trans. on Automation Science and Engineering*, 9(4):640–653, 2012.

[BCC12d] Francesco Basile, Pasquale Chiacchio, and Jolanda Coppola. A Hybrid Model of Complex Automated Warehouse Systems - Part II: Analysis and experimental results. *IEEE Trans. on Automation Science and Engineering*, 9(4):654–668, 2012.

[BCD09] F. Basile, P. Chiacchio, and D. Del Grosso. A two-stage modelling architecture for distributed control of real-time industrial systems: Application of UML and Petri net. *Computer Standards & Interfaces*, 31(3):528–538, March 2009.

[BCG06] F. Basile, P. Chiacchio, and A. Giua. Suboptimal supervisory control of Petri nets in presence of uncontrollable transitions via monitor places. *Automatica*, 42(6):995–1004, 2006.

[BDK98] Eike Best, Raymond Devillers, and Maciej Koutny. Petri nets, process algebras and concurrent programming languages. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Lectures on Petri Nets II: Applications*, volume 1492 of *Lecture Notes in Computer Science*, pages 1–84. Springer Berlin Heidelberg, 1998.

[BGM00] F. Balduzzi, A. Giua, and G. Menga. First-order Hybrid Petri Nets: a model for optimization and control. *IEEE Transactions on Robotics and Automation*, 16(4):382 – 399, August 2000.

[BGS01] F. Balduzzi, A. Giua, and C. Seatzu. Modelling and simulation of manufacturing systems with first-order hybrid Petri nets. *International Journal of Production Research*, 39(2):255 – 282, 2001.

[BL11] Shusheng Bi and Jie Liang. Robotic drilling system for titanium structures. *The International Journal of Advanced Manufacturing Technology*, 54(5–8):767–774, 2011.

[Bru01] H. Bruyninckx. Open robot control software: the orocos project. In *ICRA IEEE International Conference on Robotics and Automation, 2001. Proceedings 2001.*, volume 3, pages 2523–2528 vol.3, 2001.

[BV10] G. Black and V. Vyatkin. Intelligent component-based automation of baggage handling systems with IEC 61499. *IEEE Tran. on Automation Science and Eng.*, 7(2):337–351, April 2010.

[CE00] Krysztof Czarnecki and Ulrich Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley Professional, 2000.

[CG03] R. Valk C. Girault. *Petri-Nets for Systems Engineering*. Springer, Berlin, 2003.

[Chi98] P. Chiacchio. Implementation of hierarchical sequential functional charts for programmable logic controllers. 9^{th} *IFAC Symposium on Information Control in Manufacturing (INCOM 98)*, 2:59–64, June 1998.

[CL08] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, 2008.

[CL12] Hugo Costelha and Pedro Lima. Robot task plan representation by Petri nets: modelling, identification, analysis and execution. *Autonomous Robots*, 33(4):337–360, 2012.

[DA92] R. David and H. Alla. *Petri Nets and Grafcet*. Englewood Cliff, New Jersey, 1992.

[DA05] R. David and H. Alla. *Discrete, Continuous and Hybrid Petri Nets*. 2005.

[dAVJ11] Adriano Josè Cunha de Aguiar, Emília Villani, and Fabrício Junqueira. Coloured Petri nets and graphical simulation for the validation of a robotic cell in aircraft

industry. *Robotics and Computer-Integrated Manufacturing*, 27(5):929 – 941, 2011.

[DeV09] R. DeVlieg. Robotic trailing edge flap drilling system. In *SAE Technical Paper 2009-01-3244*, 2009.

[DF05] M. Dotoli and M.P. Fanti. A coloured Petri net model for automated storage and retrieval systems serviced by rail-guided vehicles: a control perspective. *Int. Journal on Comp. Integrated Manufacturing*, 18(2-3):122–136, May 2005.

[DF08] R. DeVlieg and E. Feikert. One-up assembly with robots. In *SAE Technical Paper 2008-01-2297*, 2008.

[DHP+93] F. DiCesare, G. Harhalakis, J.M. Proth, M. Silva, and F.B. Vernadat. *Practice of Petri Nets in Manufacturing*. Chapman and Hall, 1993.

[DK98] I. Demongodin and N.T. Koussoulas. Differential Petri nets: representing continuous systems in a discrete-event world. *IEEE Transactions on Automatic Control*, 43(4):573 –579, April 1998.

[Dou99] Bruce Powell Douglass. *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*. Addison-Wesley Professional, 1999.

[DPP09] M.A. Drighiciu, A.P. Petrisor, and M. Popescu. A Petri Nets approach for hybrid systems modeling. *International Journal of Circuits, Systems and Signal Processing*, 2009.

[DSFI02] R. DeVlieg, K. Sitton, E. Feikert, and J. Inman. Once (one-sided cell end effector) robotic drilling system. *SAE Technical Paper*, 26(1):24–38, 2002.

[DV12] Wenbin Dai and V. Vyatkin. Redesign distributed plc control systems using IEC 61499 function blocks. *IEEE*

Transactions on Automation Science and Engineering, 9(2):390–401, 2012.

[EAG] *http://ec.europa.eu/research/industrial_technologies/pdf/nmp-expert-advisory-group-reporten.pdf*.

[ele] *http://elearning.vtu.ac.in/11/enotes/CompIntManf/unit8-nan.pdf*.

[EUR] *http://www.euron.org/miscdocs/docs/euron2/year2/dr-14-1-industry.pdf*.

[Fre00] G. Frey. Automatic implementation of Petri net based control algorithms on PLC. *Proceedings of the 2000 American Control Conference (ACC'00), Chigago, Illinois*, 4:2819–2823, 2000.

[GA12] S. Chiaverini G. Antonelli, F. Arrichiello. Toward control of mobile multi-robot systems in hyper-flexible work cells. In *Control Themes in Hyperflexible Robotic Workcells*, pages 145–162, 2012.

[GDF+10] V.M. Gonzandez, A.L. Sierra Diaz, P. Garcia Fernandez, A. Fernandez Junquera, and R. Mayo Bayon. MIOOP. An object oriented programming paradigm approach on the IEC 61131 standard. *IEEE Conference on Emerging Technologies and Factory Automation (ETFA 2010), Bilbao, Spain*, September 2010.

[GU98] A. Giua and E. Usai. Modeling hybrid system by high-level Petri nets. *API - JESA*, 32(9):9–10, 1998.

[HBYZ10] Jian Huang, F.B. Bastani, I-Ling Yen, and Wenke Zhang. A framework for efficient service composition in cyber-physical systems. In *Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on*, pages 291–298, June 2010.

[HFL01] A. Hellgren, M. Fabian, and B. Lennartson. On the execution of discrete event systems as sequential function charts. *Proceedings of the 2001 IEEE International Conference on Control Applications (CCA '01), Mexico City, Mexico*, pages 428–433, September 2001.

[HMKD07] H.Flordal, M.Fabian, K.Akesson, and D.Spensieri. Automatic model generation and plc-code implementation for interlocking policies in industrial robot cells. *Control Engineering Practice*, 15(11):1416–1426, 2007.

[HS01] R. Mistry R. Hempstead, B. DeVlieg and M. Sheridan. Drill and drive end effector. In *SAE Technical Paper 2001-01-2576*, 2001.

[IA10] M.V. Iordache and P.J. Antsaklis. Concurrent program synthesis based on supervisory control. *American Control Conference (ACC '10), Baltimore, Maryland, USA*, pages 3378–3383, June 2010.

[IEC03] IEC. *International Standard IEC 61131-3: Programmable Controllers–Part 3: Programming Languages*. International Electrotechnical Commission, 2003.

[IEC05] IEC. *International Standard IEC 61499-1: Function Blocks–Part 1: Architecture, first ed.* International Electrotechnical Commission, 2005.

[Ior] M. Iordache. SPNBOX - Matlab toolbox for the supervisory control of Petri nets. *http://www.letu.edu/people/marianiordache/abs/spnbox/index.html*.

[Isa09] Robert Isaksson. Master thesis drilling with force feedback, 2009.

[JGL12] Yen Yen Joe, Oon Peen Gan, and Frank L. Lewis. Multi-commodity flow dynamic resource assignment

and matrix-based job dispatching for multi-relay transfer in complex material handling systems (mhs). *Journal of Intelligent Manufacturing*, pages 1–17, 2012.

[JS05] F. Jammer and H. Smit. Service-oriented paradigms in industrial automation. *IEEE Transaction on Industrial Informatics*, 1(1):62–70, February 2005.

[JU96] A.H. Jones and M. Uzam. Design of sequential control systems in statement lists using TPL: Part II - Token Passing Statement List. 2^{nd} *Portuguese Control Conference, Controlo'96, Porto, Portugal*, 14:716–728, September 1996.

[LBG97] L.E. Holloway, B.H. Krogh, and A. Giua. A survey of Petri nets methods for controlled discrete event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 7(7):151–190, 1997.

[Lew98] R. Lewis. *Programming Industrial Control Systems using IEC 1131-3: Revised edition*. Institution of Electrical Engineers: London, UK, 1998.

[Lew01] R. Lewis. *Modelling control systems using IEC 61499: Applying function blocks to distributed systems*. Institution of Electrical Engineers: London, UK, 2001.

[MA00] J. Moody and P. Antsaklis. Petri net supervisors for DES with uncontrollable and unobservable transitions. *IEEE Trans. on Aut. Control*, 45(3):462–476, March 2000.

[MDP00] P. Mantegazza, E. L. Dozio, and S. Papacharalambous. Rtai: Real time application interface. *Linux J.*, 2000(72es), April 2000.

[MMLCR08] J. Marco Mendes, P. Leitao, A.W. Colombo, and F. Restivo. High-level Petri nets control modules for service-oriented devices: A case study. 34^{th} *IEEE*

Annual Conference of Industrial Electronics (IECON 2008), pages 1487–1492, November 2008.

[Mur89] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of IEEE*, 77(4):541–580, April 1989.

[OHK+10] Tomas Olsson, Mathias Haage, Henrik Kihlman, Rolf Johansson, Klas Nilsson, Anders Robertsson, Mats Björkman, Robert Isaksson, Gilbert Ossbahr, and Torgny Brogårdh. Cost-efficient drilling using industrial robots with high-bandwidth force feedback. *Robotics and Computer-Integrated Manufacturing*, 26(1):24–38, 2010.

[OMA] OMAC. PackML - web page. *http://www.omac.org/content/packml*.

[ORJ07] T. Olsson, A. Robertsson, and R. Johansson. Flexible force control for accurate low-cost robot drilling. In *IEEE International Conference on Robotics and Automation, 2007*, pages 4770–4775, April 2007.

[oro] The Orocos Project. http://www.orocos.org.

[PL95] S. Pettersson and B. Lennartson. Hybrid modelling focused on Hybrid Petri Nets. *in 2nd European Workshop on Real-time and Hybrid systems*, pages 303–309, 1995.

[PLC09] PLCopen Technical Committee 6 (TC6). *XML Formats for IEC61131-3, Version 2.01 - Official Release*, 2009.

[Poo10] R. Poovendran. Cyber physical systems: Close encounters between two parallel worlds [point of view]. *Proceedings of the IEEE*, 98(8):1363–1366, Aug 2010.

[PSK11] N. Papakostantinou, S. Sierla, and K. Koskinen. Object oriented extensions of IEC 61131-3 as an enabling

technology of software product lines. *IEEE Conference on Emerging Technologies Factory Automation (ETFA 2011)*, September 2011.

[PZ04] S.S. Peng and M.C. Zhou. Ladder diagram and Petri-net-based discrete event control design methods. *IEEE Transaction on Systems, Man and Cybernetics - Part C: Application and Reviews*, 34(4):523–531, November 2004.

[R.D95] R.David. Grafcet: A powerful tool for specification of logic controllers. *IEEE Trans. On Control System Technology*, 3(3):253–268, September 1995.

[RW89] P.J. Ramadge and W.M. Wonham. The control of vector discrete-event systems. *Proc. of IEEE*, 77(1):81–98, January 1989.

[SBMB09] B. Sahr, J. Buttrick, C. Munk, and R. Bossi. Aircraft manufacturing and assembly. In Shimon Y. Nof, editor, *Springer Handbook of Automation*, pages 893–910, Berlin, Germany, 2009.

[Sch07] Ulf Schunemann. Programming Plcs with an object-oriented approach. *Automation Technology in Practice*, (2):59–63, November 2007.

[Sys] Systems Control Group of The University of Toronto. Design software for supervisory control. *http://www.control.utoronto.ca/DES/*.

[TF11] K. Thramboulidis and G. Frey. An MDD process for IEC 61131-based industrial automation systems. *IEEE Conference on Emerging Technologies Factory Automation (ETFA'11), Toulouse, France*, September 2011.

[Thr09] K. Thramboulidis. Different perspectives. Face to face; IEC 61499 function block model: Facts and fallacies.

IEEE Industrial Electronics Magazine, 3(4):7–26, December 2009.

[Thr13] Kleanthis Thramboulidis. IEC 61499 as an enabler of distributed and intelligent automation: A state of the art review - a different view. *Journal of Engineering*, 2013.

[Thr14] Kleanthis Thramboulidis. A cyber-physical system-based approach for industrial automation systems. *CoRR*, abs/1407.2077, 2014.

[Van99] J.P. Van den Berg. A literature survey on planning and control of warehousing systems. *IIE Transactions*, 31:1–13, 1999.

[VC08] V. Vyatkin and J. Chouinard. On comparisons of the ISaGRAF implementation of IEC 61499 with fbdk and other implementations. 6^{th} *IEEE International Conference on Industrial Informatics (INDIN 2008)*, pages 289–294, July 2008.

[vid] *http://www.automatica.unisa.it/video/RobotsCooperativeTskVrep.avi*.

[vin] vint-research-3-the-fourth-industrial-revolution http://www.fr.sogeti.com/globalassets/global/downloads/reports/vint-research-3-the-fourth-industrial-revolution.pdf.

[vre] *http://www.coppeliarobotics.com*.

[Vya07] V. Vyatkin. *IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design*. ISA, 2007.

[Vya11] V. Vyatkin. IEC 61499 as enabler of distributed and intelligent automation: State-of-the-art review. *IEEE Transactions on Industrial Informatics*, 7(4):768 –781, 2011.

[WBC07] Naiqi Wu, Liping Bai, and Chengbin Chu. Modeling and conflict detection of crude oil operations for refinery process based on controlled colored timed Petri net.

IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 37(4):461 –472, July 2007.

[WCCZ10] N. Q. Wu, F. Chu, C. B. Chu, and M. C. Zhou. Hybrid Petri net modeling and schedulability analysis of high fusion point oil transportation under tank grouping strategy for crude oil operations in refinery. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 40(2):159–175, 2010.

[WCCZ11] NaiQi Wu, Chengbin Chu, Feng Chu, and MengChu Zhou. Schedulability analysis of short-term scheduling for crude oil operations in refinery with oil residency time and charging-tank-switch-overlap constraints. *IEEE Transactions on Automation Science and Engineering*, 8(1):190 –204, January 2011.

[WCZ09] N. Q. Wu, F. Chu, and M. C. Zhou. Short-term schedulability analysis of multiple distiller crude oil operations in refinery with oil residency time constraint. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 39(1):1–16, 2009.

[WE09] G. Whinnem, E. Lipczynski and I. Eriksson. Development of orbital drilling for the boeing 787. In *SAE Int. J. Aerosp. 1(1):811-816, 2009*, 2009.

[Wer09] B. Werner. Object-oriented extensions for IEC 61131-3. *IEEE Industrial Electronics Magazine*, 3(4):36 –39, December 2009.

[WKK+08] Y. Wang, T. Kelly, M. Kudlur, S. Mahlke, and S. Lafortune. The application of supervisory control to deadlock avoidance in concurrent software. 9^{th} *International Workshop on Discrete Event Systems (WODES 98), Goteborg, Sweden*, pages 287–292, May 2008.

[WZC08a] N. Q. Wu, M. C. Zhou, and F. Chu. A Petri net-based heuristic algorithm for realizability of target refining schedule for oil refinery. *IEEE Transactions on Automation Science and Engineering*, 5(4):661–676, 2008.

[WZC08b] Naiqi Wu, MengChu Zhou, and Feng Chu. A Petri net-based heuristic algorithm for realizability of target refining schedule for oil refinery. *IEEE Transactions on Automation Science and Engineering*, 5(4):661 –676, October 2008.

[ZK98] M.C. Zhou and K.Venkatesh. *Modelling, Simulation and Control of Flexible Manufacturing Systems: A Petri Net Approach*. World Scientific, Singapore, 1998.

[ZSSB09] A. Zoitl, T. Strasser, C. Sunder, and T. Baier. Is IEC 61499 in harmony with IEC 61131-3? *IEEE Industrial Electronics Magazine*, 3(4):49–55, December 2009.