AN AUTOMATED MODELLING AND VERIFICATION OF DISTRIBUTED MANUFACTURING PROCESS

AMEL MELIOUH*,  ELHILLALI KERKOUCHE **,  ALLAOUACHAOUI ***

* Departement of Computer Science, University of Biskra Algeria
  a_meliouh@yahoo.fr

** Departement of Computer Science University of Oum El Bouaghi Algeria
  elhillalik@yahoo.fr

*** Departement of Computer Science, University Mentouri Constantine, Algeria
  a_chaoui2001@yahoo.com

ABSTRACT

The design of a supervisory controller for distributed manufacturing process, demands modular modelling and formal analysis. The objective of this paper is to present an automated design based on UML (Unified Modelling Language) modelling and Petri nets verification.

First UML use cases and class diagrams are used to model the manufacturing process. Then a transformation into their equivalent Petri net is carried. The transformation is done based on the combined use of meta modelling and graph grammars supported by ATOM3 tool. After that INA is used for verification purposes. The work is illustrated by a water bottling line.

Key Words: Distributed Manufacturing systems, Petri nets, UML, supervisory controller, graph grammars, ATOM3.

1. INTRODUCTION

A manufacturing system (MS) is a system that aims to produce items. The production processes of MS are composed of production steps. A production step is an activity or set of activities that transform the state of the item undergoing the process. The objective of MS is the execution of these production processes [4].

MS fall into the category of discrete Event Systems (DES), since the dynamical evolution is driven by asynchronous events. It means events or actions that occur in a finite time interval; provoke a discrete state transition of the system. This class of systems can present characteristics of parallelism, synchronism, and. conflict, because of the lack of a time dependent and predefined sequence of event occurrence [4]. As MS are dynamic systems with discrete events, they require complex supervising controller [11].

The supervisory controller of a MS consists, on the basis of a specification of the structure and behaviour of the physical system, finding a policy of Control. This controller must be able to enable, disable and synchronize the production process operations. This ability is according to the occurrence of the events occurring in the physical system.

The design of a supervisory controller for such systems rests on the use of models and formal verification. The purpose is to select the optimal design alternative, and operational policy [13]. These models must account for the structure and the behaviour of the systems and allow the analysis of their properties.

Effective modelling of complex concurrent systems requires a formalism that can capture essential properties such as nondeterminism, synchronization and parallelism. The most known formalisms of representation of (DES) are finite state automata, Petri nets, temporal logic..etc. As Petri nets offer a clean formalism for concurrency and represent the most effective method for modelling DES, they are used to model MS [8] and converted to a Token Passing Ladder Logic (TPLL) methodology to ensure the supervision of the MS. Also [5] describes how to apply timed Petri nets and existing production data to model and verify production systems.

Indeed these techniques suffer from problems of states explosion, i.e. the number of states grows in an exponential way with the size of the system. It is then necessary to have recourse to modular approaches. A modular approach consists in identifying the basic components of a MS, represents each one separately, to facilitate the design of complex systems, and to make them more comprehensible.
For that the several researchers have advocated a paradigm shift towards object oriented (OO) techniques [3]. UML, being the industry standard as a common OO modelling language. UML takes the designer through the design life cycle, starting from the description provided by users or experts down to the final software product. UML provides several kinds of graphical diagrams for modelling different views on a system [13].

Both, UML and model checking have been applied in [13], the approach has applied model checking verification to UML designs in the domain of time-dependent manufacturing systems.

Petri nets are a discrete event model [11]. Their theory can be used to analyse DEDS characteristics such as synchronisation, concurrency, conflicts, resource sharing, precedence relations, event sequences, nondeterminism and system deadlocks [2]. The work presented in [3] has proposed a design method of a verifiable discrete event controller for a production process. The design has used an UML based method for static modelling and Petri net to provide analytic Dynamic Models; the method is simulated by Matlab.

Based on this work we propose an automated modelling and verification method for designing a supervisory controller for a distributed manufacturing process using ATOM3 and INA tools.

Modelling of the distributed manufacturing process is based on UML design procedure and ATOM3 meta-modelling. We start by analysing UML Use Cases which is a detailed description of the system objectives and component behaviours. Then we generate a tool for modelling the distributed manufacturing process by a class diagram using ATOM3.

The dynamics of UML objects are described using Petri nets model. It is generated from the class diagram by transformation techniques of ATOM3. The Petri net will be analysed using Petri net theory offered by INA tool. A graph for desirable states of the production process will be generated suited to the supervisory policy used.

The work is demonstrated by modelling and verifying a typical water bottling line.

The rest of the paper is organized as follows. In section 2 we recall some basic notions about modeling using UML. In section 3 we present Petri nets for behavior modeling and analysis. In section 4 we give an overview of ATOM3. In section 5 we propose our approach and illustrate it through an example. Section 6 concludes the paper and give some perspectives of this work.

2. MODELLING WITH UML

UML (Unified Modeling Language) is an oriented object modeling language constructed by G.Booch, I.Jacobson and J.Rumbaugh [10]. The goal of UML is to become a common language for creating models of object oriented computer software [12]. UML is a set of diagrams [10]; the last version UML 2.0 contains thirty-three diagrams. The UML diagrams are comprised of two major subdivisions. There are those for modelling the static aspect of a system and others for modelling the dynamic aspect of a system [12].

Use cases diagrams are descriptions of what the objectives are and how the job is carried out. Studying the use cases enables the designer to recognize different key objects of the system [2].

The purpose of a class diagram is to depict the classes within a model. The classes are represented graphically by rectangular boxes containing three elements [10]. The first is the name of the class. The second is a list of attributes (member variables). The last is a set of operations (methods). Classes are connected together by lines or links that can be either of association type or of generalization type [2].

In this work, the class diagram for a manufacturing process is composed of classes representing the actors of the use cases diagram.

The classes have four Boolean variables indicating the state of a production component at a precise instant and five operations that move the production process components from a state to another:

- **dp:** (decision point) decision of what’s the future activity to do.
- **wait:** stopping to wait for another activity.
- **out:** the process activity is terminated.
- **work:** the process activity is executed.
- **new:** a new process activity is going to start.
- **ab:** (abort) stop the process step and wait.
- **start:** restart the process step after a waiting phase.
- **go:** no waiting phase, the process activity takes place.
- **exit:** the process step has been terminated normally.
3. PETRI NETS FOR THE BEHAVIOR MODELING AND MODEL ANALYSIS

Petri nets are a graphical and mathematical modelling tool applicable to many systems [15]. A Petri net is a particular kind of bipartite directed graphs populated by three types of objects: places, transitions, and directed arcs, which connect places to transitions or transitions to places. In order to study the dynamic behaviour of a Petri net modelled system in terms of its states and state changes, each place may potentially hold either none or a positive number of tokens. The presence or absence of a token in a place can indicate whether a condition associated with this place is true or false, for instance.

A Petri net is formally defined as a 5-tuple \( N = (P, T, I, O, M_0) \), where
- \( P = \{p_1, p_2, \ldots, p_m\} \) is a finite set of places;
- \( T = \{t_1, t_2, \ldots, t_n\} \) is a finite set of transitions, \( P \cup T \neq \emptyset \), and \( P \cap T = \emptyset \);
- \( I: P \times T \rightarrow N \) is an input function that defines directed arcs from places to transitions, where \( N \) is a set of nonnegative integers;
- \( O: T \times P \rightarrow N \) is an output function that defines directed arcs from transitions to places; and
- \( M_0: P \rightarrow N \) is the initial marking [7].

The dynamic of a model describes behavioural aspects of the object classes, in the sense that they describe the sequence of operations that occur without regard for what the operations do, what they operate on, or how they are implemented [2].

In this work, the dynamic of the UML class diagram for the manufacturing process is illustrated by a Petri net for each class. Where places represent the state of a class in a certain instant (expressed by Boolean attributes) and transitions represent the operations applied on a class (expressed by class operations).

The next step in the behavioural modelling with Petri net is model verification after designing the synchronization logic as defined in the use cases diagram which enforces the mutual synchronization heuristic for the classes, by adding a set of synchronization places. The undesirable states are prevented by creating the scenario of desirable cases for the composition of the global Petri net.

4 ATOM3 OVERVIEW

The name AToM3 is shorthand of “A Tool for Multi-formalism and Meta-Modelling”. The tool was developed at the Modelling, Simulation and Design Lab in the School of Computer Science of McGill University. It is written in Pyton. The two main tasks of AToM3 are meta-modelling and model transforming [1].

In AToM3, formalisms and models are described as graphs. From a meta-model (presented in the Entity-Relationship formalism or class diagram), AToM3 generates a tool to process (create and edit) models described in the specified formalism. Transformations between formalisms are performed by graph rewriting system, which uses graph Grammar rules to visually guide the procedure of the transformation [6].

As string grammars, graph grammars can be used to describe graph transformation or to generate sets of valid graphs. They are composed of production rules; each having graphs in their left and right hand sides (LHS and RHS), conditions to verify and actions to execute. Rules are compared with an input graph called host graph. If a matching is found between the LHS of a rule and a subgraph in the host graph, the rule can be applied and the matching subgraph of the host graph is replaced by the RHS of the rule.

5. AUTOMATED MODELLING AND VERIFICATION OF A DISTRIBUTED MANUFACTURING PROCESS

The design of a supervisory controller for a distributed manufacturing process is based on a multi-formalisms modelling in AToM3 tool, and accomplished in three steps:

- Generation of a visual tool for Modelling distributed manufacturing process as UML class diagram using AToM3 meta-modelling.
- Generation of a global Petri net of the class diagram using a graph transformation grammar.
- Verification of the properties of the global Petri net model and generation of desirable states using a verification tool for Petri nets.

5.1 META-MODELLING OF THE DISTRIBUTED MANUFACTURING PROCESS

The modelling of manufacturing process in AToM3 needs the definition of a meta-model. The meta-formalism used in this work is the UML Class Diagrams and the constraints are expressed in Python code. The meta model contains one class representing a component of the manufacturing process, each component is linked to another by an arc representing a relationship between the classes, as shown in figure 1.
Based on this meta model, ATOM³ generate a visual tool to models the class diagram of the distributed manufacturing process. This tool is illustrated in figure 2.

5.2. META-MODELLING OF PETRI NETS

The meta model of Petri nets contains four classes since Petri nets consist of places, transitions, and arcs. Classes describe places, transitions, input arcs, and output arcs as shown in Figure 3.

Based on this meta-model, ATOM³ generates a visual tool to process the Petri net as illustrated in figure 4.

5.3. Graph grammar for transforming class diagram to Petri Nets formalisms

For the transformation of the class diagram of manufacturing process to the Petri net generated by ATOM3, we have proposed a graph grammar that has:

- An initial action that initiates the values of global variables used in actions and constraints code of each rule.
- Ten rules which will be applied in ascending order. They transform the class diagram to Petri net and create synchronization places. The creation follow the policy used by the supervisor of the distributed manufacturing process.
- A final action that deletes all the global variables used in the graph grammar.

We present here the most important rules.

a. Rule1: ClasstoRdpRule (priority 1)

This rule is applied to transform each component of the manufacturing process (a class of the class diagram) to a Petri net formed of four places and five transitions and specify names for the places and the transitions relating to the name of the corresponding manufacturing process component.

b. Rule2: IdentifyAssociationRule (priority 2)

This rule is applied to identify if there exist a relationship between two classes, it omit the link and saves the names of the two connected classes.
c. Rule 4: AddloopRule (priority 4)

This rule is a synchronization rule. It is applied to search two transitions and two places in the Petri net by their names and add two arcs to make each place as an input/output place to each transition and an output place to others. The goal is to ensure the mutual exclusion between two manufacturing components.

5.4 VERIFICATION OF THE MANUFACTURING PROCESS AND GENERATION OF GRAPH OF DESIRABLE STATES

Properties of the Petri net generated are analysed using the INA tool. The analysis takes place after a generation of a textual description file [14] of the global Petri net. The Graph of Desirable States (GDS) is a graph that enumerates all desirable states and their relationships. The GDS maps the Use Case information into the Petri net domain, i.e. sentences in the Use Case are translated in sets of rules in terms of places and transitions. The word “desirable” reflects the fact that the graph embraces all we expect the system to do, and any unwanted or undesirable behaviour is prohibited. The GDS is equivalent to the reachability graph, so we use also INA tool to calculate this graph.

5.5 APPLICATION OF THE APPROACH ON AN EXAMPLE OF A DISTRIBUTED MANUFACTURING PROCESS: EXAMPLE OF A WATER BOTTLING LINE

The figure 5 illustrates a simple water bottling line. It is made of four machines: a Feeder provides empty bottles on a belt conveyor named the Conveyor. The conveyor is the input of a washing machine named the Washer. The Washer is connected to the Rotary-Filler, named the "Filler" that fills the bottles with water.

Figure 5: Example of a water bottling Line [13].

5.5.1 UML MODELING

The two diagrams corresponding to the example of figure 5 are:

a. Use cases Diagram

When the feeder start to provides empty bottles, if the conveyor is at decision point (dp), The bottles are evacuated to the conveyor and carries of bottles can take place (go) at speeds of 1 or 2 meters/second. The bottles quantity on the conveyor stays between limits detected by accumulation sensors. However, if the Conveyor is still outside the bottling area, the feeder will stop, waiting for the conveyor to arrive at its decision point (abort operation). When the Conveyor arrives at decision point (new), the feeder is restarted. When the two machines are both in the working state, the bottles may be accumulated at the output of the conveyor. If the output is closed, the washer starts Washing. If the filler is at decision point (dp), then it puts a constant amount of freeze water into each bottle (go). To do that, the bottles must be always filled at the same speed. So, when the Filler is to stop, the bottles which are inside must be evacuated and cannot accumulate. This is done by closing the input, of the filler. The bottle filled exit from the bottling area (exit leading to the state out). The bottles then will be stocked.

b. Class Diagram

Since the water bottling line is composed of four machines so we have four classes as shown in figure 6.
5.5.2 PETRI NET MODEL

Each class of the class diagram is represented by a Petri net, composed of four places and five transitions. We present here Petri net of the class feeder, as shown in figure 7.

Due to great size of the Petri net of the class diagram illustrated above, we have chosen to show only a part of the generated Petri net concerned the conveyor and the feeder classes.

5.5.3 AUTOMATED MODELLING AND VERIFICATION OF THE WATER BOTTLING LINE

The class diagram model of the water bottling line and its equivalent Petri net after the application of our graph grammar are illustrated in the following figures.

For the analysis of the properties of the Petri net of figure 10, we have used the tool INA that accept a net description file of a place transition net as input and provide the properties of the Petri net as shown in figure 11. The input file is generated by our graph grammar.

Each class is defined as shown in figure 9

We can see from INA screen that the net is bounded
and live and has no dead transitions. The GDS of the Petri net of figure 10 is generated by INA in textual format as shown in the following figures and we have transformed it to a graphical format to facilitate its comprehension.

![Diagram](image_url)

**Figure 11:** Analysis of the structural properties of the Petri net generated.

### 6. CONCLUSION

The requirement for more efficient manufacturing systems increases the need for an automated method for the design of supervisory controller.

In this work, we have presented our approach based on combining Meta-modelling and Graph Grammars to automatically designing a supervisory controller for a distributed manufacturing process. The design is based on UML modelling and Petri nets analysis. UML dynamic is modelled by Petri net where graph transformation is applied. UML use cases information can be used to build the supervisor controller policy and to generate the graph of desirable states of the system.

The approach has been illustrated by its application to a water bottling line and has generated the best policy to have a safe working and good synchronization between the machines.

This work has satisfied our requirements, and facilitates the automated design of a supervisory controller for a distributed manufacturing process, however it needs future improvements that concern the size of the global Petri net generated. As the system components enlarge, the number of places and transitions augments and the time of execution of the graph grammar augment.

Related issues to be investigated and developed, in this case is the optimisation of the Petri net generated by adding other rules in the graph grammar to reduce the Petri net based on its reduction techniques. Another future work is the application of our approach on a large manufacturing system composed of great number of manufacturing process in cooperation.

### 7. REFERENCES


[9] N Audv, F Pmnet., “Comparison of extended Petri...
Nets for high throughput production line accurate modelling and simulation”, Laboratoire d’Informatique, de Robotique, et de Microélectronique de Montpellier 161 rue Ada - 34 392 Montpellier - Cedex 5 – France.


