DESIGN APPROACH FOR MODELING MANUFACTURING SYSTEMS

Myriam Noureddine

Department of Computer Science
University of Science and Technology of Oran
BP1505, El M’naouer, 31000, Oran, Algeria.
myriam.noureddine@univ-usto.dz

Abstract:
This paper deals with a design approach for modeling any manufacturing system, formalized through the notion of software architecture used in the software engineering domain. First, we describe the adopted method of modeling manufacturing system and next the building of software architecture.

The proposed modeling of manufacturing system begins with the description of the concerned system until the analysis of the obtained final model. The approach follows sequential steps grouped in two modules: generation of conceptual model and analysis step. From the physical description of the system, the conceptual model is building through a linear succession of entities that represents resources of the manufacturing system. Entities are connected by links that allows the representation of the synchronization between the two adjacent entities. The Petri Nets (PN) have been chosen as analysis tool because PN model offers qualitative properties that characterize the dynamic behavior of the system. In the context of a modular modeling, the class of Controllable-Output Petri Nets (COPN) is well suited, because of the integration operation, preserving the qualitative properties. In this context, each link is modeled following a COPN and these links are integrated through a specify rule.

The software architecture of a system allows describing elements of the concerned system and the relationships among them. We use the software engineering concepts for modeling the integration principle of sub Petri nets, which are issued from the modeling approach proposed above. The integration rule of Petri nets is specified following a hierarchical of description units. Each description unit represents a data and/or an operation of the integration rule. The software architecture is obtained through a top-down strategy where the top-level description unit represents exactly the system functionality. The hierarchy of description units is built through data decomposition at the beginning and functional decomposition after that. The building strategy allows creating relations between different description units situated in adjacent levels, through the notion of imported operations that are defined in their corresponding description units. The methodology is supported by a formal language to write description units.

As the UML formalism (Unified Modeling Language) is an unquestionable standard of modeling around an object approach, and to validate the obtained software architecture, we generate the corresponding UML model of the integration rule through a translation process.

We conclude that the design approach presents a theoretical framework for the realization of the integration operation. The principle is applied recursively and we obtain an integrated final PN that will be analyzed, providing information on the state of the manufacturing system.

Keywords:
Manufacturing system, design, software architecture, integration, Petri net, description unit.

Introduction
The success of software development is conditioned by the upstream design step. This fundamental step can be formalized through the notion of software architecture [1]. The software architecture of a system allows describing elements of the concerned system and the relationships among them [2] and it expresses formally the subordinate links between entities that represent data or operations.
This paper deals with a design approach for modeling any manufacturing system, formalized through the software architecture. First, we describe the adopted method of modeling manufacturing system and next the building of software architecture. As the UML formalism (Unified Modeling Language) is an unquestionable standard of modeling around an object approach, widely used to describe the software architecture of a system [3], [4], and to validate the obtained software architecture, we generate the corresponding UML model of the integration rule through a translation process.

This paper is organized as follows: in the next section, we present the adopted method of modeling manufacturing system. The third section gives the software architecture for the integration principle of submodels, which are issued from the modeling approach proposed above, and its corresponding UML model.

**Method of modeling manufacturing system**

The proposed modeling of manufacturing system follows sequential steps grouped in two modules: generation of conceptual model and analysis step.

A. Generation of conceptual model

From the physical description of the system, the conceptual model is building through a linear succession of entities (E) connected by links (L). Entities represent resources (processing resources or machines and transportation resources) of the manufacturing system and links represent the synchronization between the two adjacent entities. The obtained conceptual model shows a clear and linear vision of the manufacturing system, maintaining a high abstraction level.

For example, consider a manufacturing system with two processing resources (machines M1 and M2) and a conveyer (C) as transportation resource. In this case, we note entities Ei, Ei’ for machines and Ej for the conveyer: the link Lk assures the transfer of the piece from M1 (Ei) to the adjacent entity Ej=Ei+1 (C) to transport the piece to the other machine (M2) for realizing the next manufacturing operations. The same example can show a machine (Ei) with its driven manipulator (Ej=Ei+1).

![Figure 1. Entities and links for manufacturing systems](image)

B. Analysis tool

The Petri Nets (PN) have been chosen as analysis tool because PN model offers qualitative properties that characterize the dynamic behavior of the system. In the context of the natural approach of modular modeling (decomposition of the whole system into subsystems), the class of Controllable-Output Petri Nets (COPN) or CO nets [5] is well suited, because of the integration operation, preserving the qualitative properties.

Formally, a COPN \( N \) is an ordinary PN with input transitions (source transitions, without input places) and output transitions (sink transitions, without output places): \( N=(P\cup R,T,F,M_0) \), where \( P \) and \( R \) are two disjointed sets of process places and resource places; the restricted subnet, where the set of places is \( P \), is an acyclic graph without isolated nodes.
In our context, each link is modeled following a COPN (Figure 2) [6]: the link between Ei and Ej is modeled with primitives of the generic model of the synchronization (VD, D, VS, S and Trt) that ensure the transfer of a piece from Ei to Ej. Places are deducted for having a comprehensive and coherent PN.

**Figure 2.** COPN model for a link Ei-Ej

We can simplify the obtained COPN by removing resource places (data place and entity place). Removing these places does not change the behavior of the Petri net. As we are not concerned by the internal functioning of the PN, we model this COPN by a generic COPN box, specified only by the input transitions (tei, i=1 à 3) and output transitions (tsi, i=1 à 3). We obtain thus a sequence of COPN models (or PN boxes) that can therefore be integrated following the integration operation of COPN [5], preserving the qualitative properties. So, we define the next integration rule: each output transition ‘End of Store’ (ts2) on a COPN Pk will be connected to the input transition ‘Deliver demand’ (te2) of the adjacent COPN Pk+1.

**Software architecture of the integration rule**

We use the software engineering concepts for modeling the integration principle of sub Petri nets, which are issued from the modeling approach proposed above.

A. Concept of software architecture

We associate the design entity to the concept of the description unit or Unit of Description (UD) [7], where each UD represents a data and/or an operation of the specified system. As building software architecture of a system is a complex activity, an incremental approach will be preferred: the software architecture is a hierarchy of UD, obtained through a top-down strategy where the top-level is UD0, representing exactly the system functionality. Others UDi are obtained through data decomposition at the beginning and functional decomposition after that. The building strategy allows creating relations between UD, following the notion of imported or exported operations. The methodology is supported by a formal language to describe UD into variables and defined operations. The software architecture can be represented according a graphical form grouped all UDi (i>0), where the representation on the graph is a simple link for a decomposition link and an arrow link for the exported operation.
B. Software architecture of Integration rule

The software architecture of the integration rule is building on four levels. The level 1 gives the UD0, representing the main data: the final Integrated PN (IPN), with the main operation, specified following the operation ‘integration’ defined in another UD. We build the UD hierarchy according to the top-down strategy: the data decomposition gives UD1 (data PNB) representing the PN box, as the COPN defined above with only input and output transitions. We simulate this data by a graph, which is the concatenation of two graphs: a left graph with three arcs from te to b and a right graph with three arcs from b to ts.

In this context, the realization of the previously integration principle (Figure 3) introduces one interface place ‘pl’ and inter-modules arcs, and allows to obtain a new graph as COPN box.

The data decomposition gives UD2 as the left graph (LGR), with the defined operation which specifies the input arcs (from nodes te to the node b), and UD3 as the right graph (RGR), with the defined operation which specifies the output arcs (from the node b to nodes ts). These graphs are decomposed following UD4 as a sequence of arcs (SEQ[ARC]). UD4 is defined like a Cartesian product (CARDPROD) of the abstract data ‘nodes’ with first and second projection operations. We consider the UD4 as elementary data and so it is useless to decompose it. After that, the main operation of the top-level UD0 is decomposed into the UD5 (OPT) with the ‘integration’ operation.

Following the specification of each defined operation, the exported operations are deduced immediately and the figure 4 gives the complete software architecture of the integration process.

**Figure 3.** Principle representation of the integration rule

**Figure 4.** Software architecture of the integration principle

---

**Definition of the software architecture**

**Variables:**
- \( i = \text{integer} \)
- \( ipn = \text{SEQ[pn]} \)
- \( pnb = \text{SEQ[graph]} \)
- \( graph = \text{SEQ[arc]} \)
- \( arc = \text{CARPROD[node,node]} \)

**Operations:**
- **building_final_PN_in_UD0** (p: pnb) r: ipn, such as \( r = \Sigma_i \text{(integration}(p_i,p_{i+1})) \)
- **input_arc_in_UD2** (lgr: graph) ai : arc, such as \( ai = \text{creation_arc}(te,b) \land 1 \leq i \leq 3 \)
- **output_arc_in_UD3** (rgr: graph) ai : arc, such as \( ai = \text{creation_arc}(b,ts) \land 1 \leq i \leq 3 \)
- **creation_arc_in_UD4** (n1, n2 : nodes) a: arc, such as \( \text{projection1}(a) = n_1 \land \text{projection2}(a) = n_2 \)
The integration operation in UD5 formalizes exactly the integration rule, by giving a new graph \( g'' \) from graphs \( g \) and \( g' \), using an intermediate graph (\( ig \)). The graph \( ig \) is formed by two inter-modules arcs and a new node (interface place ‘pl’ in a COPN). Note than the two operations ‘input_arc’ and ‘output_arc’ on the graph \( ig \) are defined in the same that previously, using the operation ‘creation_arc’:
- \( \text{input}_\text{arc}(ig: \text{graph}) \ a: \text{arc}, \) such as \( a = \text{creation}_\text{arc}(ts_2 \ , \text{pl}) \)
- \( \text{output}_\text{arc}(ig: \text{graph}) \ a: \text{arc}, \) such as \( a = \text{creation}_\text{arc}(\text{pl} \ , \text{te}_2) \).

C. Validation of the software architecture
As the UML formalism (Unified Modeling Language) is an unquestionable standard of modeling around an object approach, and to validate the obtained software architecture, we generate the corresponding UML model of the integration rule through a translation process on UD and links. This translation process deals with the equivalency principle [7] between the both concepts of UD and UML class diagram [8] and consequently equivalency of the links in software architecture and UML association relationships [8].

The following figure gives the complete software architecture and its corresponding UML model of the integration process. As UD1 and UD5 have both imported operations (input_arc and output_arc), we add the operation ‘integration’ of UD5 in the class PNB, which represents the COPN models to be integrated.

**Definition of UML elements**

**Abstract Classes:**
- from Translation rules on UD
  1. Delete UD0 (IPN) and UD5 (OPT)
  2. Translate UD\( i \) in abstract classes with defined operations of UD
    \( \rightarrow \) PNB, LGR, RGR, ARC

**Association relationships:**
- from Translation rules on UD links
  3. Remove exportation links between LGR and OPT, RGR and OPT
  4. **UML aggregation links** from links both Decomposition and Exportation
    \( \rightarrow \) PNB and LGR
    \( \rightarrow \) PNB and RGR
    \( \rightarrow \) LGR and ARC
    \( \rightarrow \) RGR and ARC

**Figure 5.** The UML model of integration principle

**Conclusion**
We proposed in this article the software architecture of the integration rule of Petri nets, issued from a modeling approach of manufacturing systems. As building software architecture with all the entities is a complex activity, we adopt an incremental approach. The obtained software architecture is specified through a hierarchy of elements and treatments that intervene in the integration rule.

To validate this software architecture, we generate the corresponding UML model of the integration rule through a translation process. The UML formalism specifies semantics and a complete graphical notation but no process methodology is defined within it [9]. In this context, we can offer a systematic procedure to generate the UML model of the integration process, by translation the obtained software architecture.
This design approach presents a theoretical framework for the realization of the integration process. In our context of manufacturing system modeling and for all COPN boxes, the principle described in the two models is applied recursively: the obtained new graph $g''$ will be considered like a simple graph to integrate with the following adjacent graph (corresponding COPN box).

The building of the final graph allows identifying cases where problems be underline during the integration step: existence of cycles (not authorized in COPN) by the presence of more than an occurrence of an entity or a link, and deadlocks due to the utilization of common transportation resources.

For example, consider the manufacturing system in Figure 1 and a manufacturing process $M1M2M1M2$. We obtain a linear succession of entities connected by links: $I-L1-E1-L2-E2-L3-E3-L4-E2-L5-E1-L2-E2-L3-E3-L6-O$ where $I$ and $O$ represent the input (beginning of process) and the output (end of manufacturing process). The final graph is obtained by the integration of links: linear integration for links $L1, L2, L3, L4, L5$ on the one hand, and $L3, L6$ on the other hand. But, the integration of links $L5$ and $L2$ introduces a cycle because the output $ts_2$ in $E2$ (conveyer C) will be connected via an interface place to the input transition $te_2$ in $E1$ (machine $M1$). At the same, the study of the final graph shows the common use of resources, giving a deadlock. For example, if we have the same manufacturing process $M1M2M1M2$ with many products to manufacture, a potential deadlock will be identified induced by the final graph, where entities or links appear many times.

We conclude that the design approach presents a theoretical framework for the realization of the integration operation. The principle is applied recursively and we obtain an integrated final PN that will be analyzed, providing information on the state of the manufacturing system.

References