Contribution to the integration of supervisory control theory in a model-based system engineering method

Xiaoshan Lu

N°d'ordre NNT : 2019LYSEI072

THESE de DOCTORAT DE L'UNIVERSITE DE LYON
opérée au sein de
INSA Lyon

Ecole Doctorale N° 160
Electronique Electrotechnique Automatique (EEA)

Spécialité Automatique

Soutenue publiquement le 18/09/2019, par :
Xiaoshan Lu

Contribution to the Integration of Supervisory Control Theory in a Model-Based System Engineering Method

---

Devant le jury composé de :

Berruet, Pascal	Professeur des Universités	Université de Bretagne Sud	Rapporteur
Carré-Ménétrier, Véronique	Professeur des Universités	Université de Reims	Rapporteuse
Niel, Eric	Professeur	INSA de Lyon	Directeur de thèse
Piétrac, Laurent	Maître de Conférences HDR	INSA de Lyon	Co-directeur de thèse

Département FEDORA – INSA Lyon - Ecoles Doctorales – Quinquennal 2016-2020

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
CHIMIE	**CHIMIE DE LYON** http://www.edchimie-lyon.fr Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage secretariat@edchimie-lyon.fr INSA : R. GOURDON	**M. Stéphane DANIELE** Institut de recherches sur la catalyse et l'environnement de Lyon IRCELYON-UMR 5256 Équipe CDFA 2 Avenue Albert EINSTEIN 69 626 Villeurbanne CEDEX directeur@edchimie-lyon.fr
E.E.A.	**ÉLECTRONIQUE, ÉLECTROTECHNIQUE, AUTOMATIQUE** http://edeea.ec-lyon.fr Sec. : M.C. HAVGOUDOUKIAN ecole-doctorale.eea@ec-lyon.fr	**M. Gérard SCORLETTI** École Centrale de Lyon 36 Avenue Guy DE COLLONGUE 69 134 Écully Tél : 04.72.18.60.97 Fax 04.78.43.37.17 gerard.scorletti@ec-lyon.fr
E2M2	**ÉVOLUTION, ÉCOSYSTÈME, MICROBIOLOGIE, MODÉLISATION** http://e2m2.universite-lyon.fr Sec. : Sylvie ROBERJOT Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 INSA : H. CHARLES secretariat.e2m2@univ-lyon1.fr	**M. Philippe NORMAND** UMR 5557 Lab. d'Ecologie Microbienne Université Claude Bernard Lyon 1 Bâtiment Mendel 43, boulevard du 11 Novembre 1918 69 622 Villeurbanne CEDEX philippe.normand@univ-lyon1.fr
EDISS	**INTERDISCIPLINAIRE SCIENCES-SANTÉ** http://www.ediss-lyon.fr Sec. : Sylvie ROBERJOT Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 INSA : M. LAGARDE secretariat.ediss@univ-lyon1.fr	**Mme Sylvie RICARD-BLUM** Institut de Chimie et Biochimie Moléculaires et Supramoléculaires (ICBMS) - UMR 5246 CNRS - Université Lyon 1 Bâtiment Curien - 3ème étage Nord 43 Boulevard du 11 novembre 1918 69622 Villeurbanne Cedex Tel : +33(0)4 72 44 82 32 sylvie.ricard-blum@univ-lyon1.fr
INFOMATHS	**INFORMATIQUE ET MATHÉMATIQUES** http://edinfomaths.universite-lyon.fr Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage Tél : 04.72.43.80.46 infomaths@univ-lyon1.fr	**M. Hamamache KHEDDOUCI** Bât. Nautibus 43, Boulevard du 11 novembre 1918 69 622 Villeurbanne Cedex France Tel : 04.72.44.83.69 hamamache.kheddouci@univ-lyon1.fr
Matériaux	**MATÉRIAUX DE LYON** http://ed34.universite-lyon.fr Sec. : Stéphanie CAUVIN Tél : 04.72.43.71.70 Bât. Direction ed.materiaux@insa-lyon.fr	**M. Jean-Yves BUFFIÈRE** INSA de Lyon MATEIS - Bât. Saint-Exupéry 7 Avenue Jean CAPELLE 69 621 Villeurbanne CEDEX Tél : 04.72.43.71.70 Fax : 04.72.43.85.28 jean-yves.buffiere@insa-lyon.fr
MEGA	**MÉCANIQUE, ÉNERGÉTIQUE, GÉNIE CIVIL, ACOUSTIQUE** http://edmega.universite-lyon.fr Sec. : Stéphanie CAUVIN Tél : 04.72.43.71.70 Bât. Direction mega@insa-lyon.fr	**M. Jocelyn BONJOUR** INSA de Lyon Laboratoire CETHIL Bâtiment Sadi-Carnot 9, rue de la Physique 69 621 Villeurbanne CEDEX jocelyn.bonjour@insa-lyon.fr
ScSo	**ScSo*** http://ed483.univ-lyon2.fr Sec. : Véronique GUICHARD INSA : J.Y. TOUSSAINT Tél : 04.78.69.72.76 veronique.cervantes@univ-lyon2.fr	**M. Christian MONTES** Université Lyon 2 86 Rue Pasteur 69 365 Lyon CEDEX 07 christian.montes@univ-lyon2.fr

*ScSo : Histoire, Géographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Motivations

In the field of automatic control system, one of the interesting scientific branches is the study on the control of Discrete Event System (DES), which is extensively involved in different application domains such as manufacture system, electric system, traffic system, etc. DES is an area concerned with systems usually discrete in time and state space, driven by instantaneous events other than (or in addition to) the tick of a clock and 'nondeterministic' in the sense of making state-transitional choices by internal chance or other mechanisms not necessarily modeled by the system analyst [1].

Supervisory Control Theory (SCT), firstly introduced by Ramadge and Wonham in 1987 [2], is one of the most important academic theories for Automatic Control (AC) of DES. With SCT, the requirements which are checked afterward in traditional engineering are used as input for generation of the design of the controller that is correct by construction. By the scientific achievements within the past several decades, the framework of SCT forms a systematically formal paradigm to synthesize controllers for DESs and a series of concepts and methods are proposed, such as supervisor synthesis [3], controllability [4], observability [5], and supremal sublanguage [6] [7]. Attempts to apply SCT to industrial problems encountered the barrier notorious as exponential state space explosion, different control architectures such as modular control and distributed control are consequently proposed by related research works [8-11]. Nowadays, there are still a lot of emerging contributions which continuously push forwards the front edge of SCT. Some contributions focus on the extending SCT to other formal models rather than automata, such as Extended Finite Automata (EFAs) [12] [13], Binary Decision Diagrams (BDDs) [14] and Petri Nets [15] [16], etc. Other contributions are devoted to the study on application of SCT in real systems. The results show that SCT can be widely applied in the control on DESs of different industrial domains, which include a number of case studies such as autonomous robot [17] [18], power management [19] and health care system [20]. In addition,

some researchers study on the implementation architecture of SCT. As PLC (Programmable Logic Controller) plays an important role in industrial automation applications, efforts are made to achieve the implementation by PLC controller [21] [22]. Besides, a part of contributions focus on performing control simulation based on SCT paradigm supported by toolkits such as STATEFLOW [23] [24].

However, despite the academic achievements of SCT, there are still gaps between the theoretical development and applications of SCT in engineering practice. The state-of-the-art SCT-based modeling process for AC is composed of three main phases: formalization, supervisor/controller synthesis and implementation [25]. In this process, SCT provides a theoretical basis for behavior analysis of components and constraints in the form of formal models. On the other hand, the typical system development process in engineering context is usually based on V-model [26-28], which can be divided into a series of phases: requirements analysis, functional analysis, design synthesis, implementation traceability and validation. For each modeling phase of V-model, a variety of models are built to present the different aspects of the studied system including structure, requirements, function and behavior, etc. Compared with V-model, several issues can be found in the typical SCT-based process:

- The formal models of SCT just represents the behavior aspect of the system and the structure models such as concepts, components, the interfaces or interaction can hardly be identified; Besides, there is lack of models representing the traceability within SCT paradigm.

- The SCT doesn’t provide the global modeling process from analyzing and decomposing the informal requirements to sub-systems to transform models to implementation.

In fact, the issues above attribute to the limitation of modeling scope of formal model within the SCT paradigm. Therefore, it becomes our focus point in this study that how the modeling scope of SCT paradigm can be extended with the help of other modeling languages and methods so as to form an development framework for AC in engineering practice.

The Model-Based System Engineering (MBSE) provides the possibility to deal with the limitations of SCT. MBSE is the actual state-of-the-art global design process in engineering practice. Some case studies and scientific projects show that MBSE is practical and prospective system development technology [29] [30]. On the other hand, MBSE are supported by a number of methods, modeling languages and toolkits. Compared with conventional engineering process, MBSE provides model-based approaches rather than document-based ones. Particularly, SysML is a general-purpose modeling language for MBSE, which supports the specification, analysis, design, verification and validation of a broad range of systems [31-33]. SysML defines nine kinds of diagram providing comprehensive description for system design, which can be used to cope with the limitation of formal models. The existing modeling propositions are investigated based on

formal language and MBSE method. Unfortunately, all of the existing contributions more or less have their limitations for the target of dealing the limitations of SCT and there is still gap in this research domain. Therefore, in this dissertation, it is of great interest and necessity to propose an integrated modeling framework based on SCT and MBSE.

1.2 Contributions

The main objective of the work is to propose a novel modeling framework for DES AC by combining Supervisory Control Theory with Model-Based System Engineering, in order to overcome the existing limitations of SCT. The main contributions of the work are as following:

- Based on the comparison between SCT-based process and MBSE process, the limitations of SCT are identified and validated. For the purpose for dealing with the limitations SCT, this study investigates and analyzes the SysML models which are necessary for AC from the perspective of MBSE.

- The MBSE modeling framework which integrate SysML models and SCT paradigm is proposed. Ten SysML diagrams are defined to present different aspects of the system description along with the modeling process. A case study is presented to validate the applicability and improvement of the proposed framework compared with typical one.

- The gap of formal model and SysML model are bridged. We investigate concepts and elements exclusively within SCT paradigm, such as controllable/uncontrollable event, plant/specification/supervisor by SysML elements. Besides, the methods of modeling and traceability verification which link formal models and SysML models are proposed.

1.3 Outline

This dissertation is structured as following:

- **Chapter 2** presents the state of the art of Supervisor Control Theory and Model-Based Systems Engineering Methods. The typical modeling process of SCT is compared with MBSE process and existing limitations and issues of SCT are discussed. Besides, the possible solutions are reviewed and discussed.

- **Chapter 3** introduces the global modeling process of the proposed framework for AC by integrating SCT and MBSE. The necessary models we need to describe a DES control system and the model types which should be used for modeling are analyzed based on the MBSE viewpoint. The details of the proposed framework such as sub-processes, models, activities, methods and input/output of each step are presented.

- **Chapter 4** presents the methods for dealing with the consistency problems between formal models and SysML models faced in the proposed framework. Aimed at formalization of plants and specifications respectively, different solutions are proposed and detailed in this chapter.

- **Chapter 5** presents a case study based on the proposed modeling framework. A global design process of controller for Customed Power Park (CPP) and the modeling results for each step will be detailed. Besides, some more situations will be discussed to verify the modeling scope of proposed framework.

A conclusion and global perspectives are commented at the end of this dissertation.

Chapter 2

Research Context and Problem Statement

2.1 Introduction

The Supervisory Control Theory (SCT) is one of the most important modeling theories for DES AC. Since the Supervisory Control Theory was introduced by Ramadge and Wonham in 1987 [2], the theory has developed prosperously in the academic field and a large number of contributions come out. The large number of application shows that SCT can be widely used in a variety of academic contexts.

However, despite the academic achievements of SCT over last several decades, there are few application cases in engineering practice. By reviewing related state of the art of SCT paradigm such as the concepts, the supervisor synthesis approach and typical modeling process, the existing limitations which lead to the gap between academic and engineering are recalled. On the other hand, from perspective of the engineering process, the investigation of the model-based system engineering is furthered to specify the limitations of typical modeling process in SCT and the interest and feasibility of integration of MBSE and SCT will be detailed.

In this chapter, based on the reviews on both SCT and MBSE and the comparison of these two domains, the issues and possible solutions will be discussed and the objectives of this study will be presented at the end of this chapter.

2.2 Supervisory Control Theory

In this section, the application, theoretical concepts, methods and modeling processes of SCT are reviewed. With regard to the challenge and limitation faced by SCT, existing issues within the SCT paradigm are discussed.

2.2.1 Goals of SCT

2.2.1.1 Automatic Control for Discrete Event System

System is one of those primitive concepts whose understanding can be defined in different ways according to different contexts. There are two common salient features among these definitions. First, a system consists of interacting "components", and second a system is associated with a "function" it is presumably intended to perform [34].

In the scientific domain of automatic control (AC), the model of system can be partially classify in a way shown in Figure 2.1 [34]. In Continuous-Variable Dynamic Systems (CVDS), the state continuously changes as time changes and the model is usually a differential equation on the variable of time. With every "clock tick" the state is expected to change, since continuous state variables continuously change with time. However, in event-driven systems, the state changes only at certain points in time through instantaneous transitions. In between event occurrences the state remains unaffected.

Definition 2.1 (Discrete Event System): is a discrete-state, event-driven system, that is, its state evolution depends entirely on the occurrence of asynchronous discrete events over time.

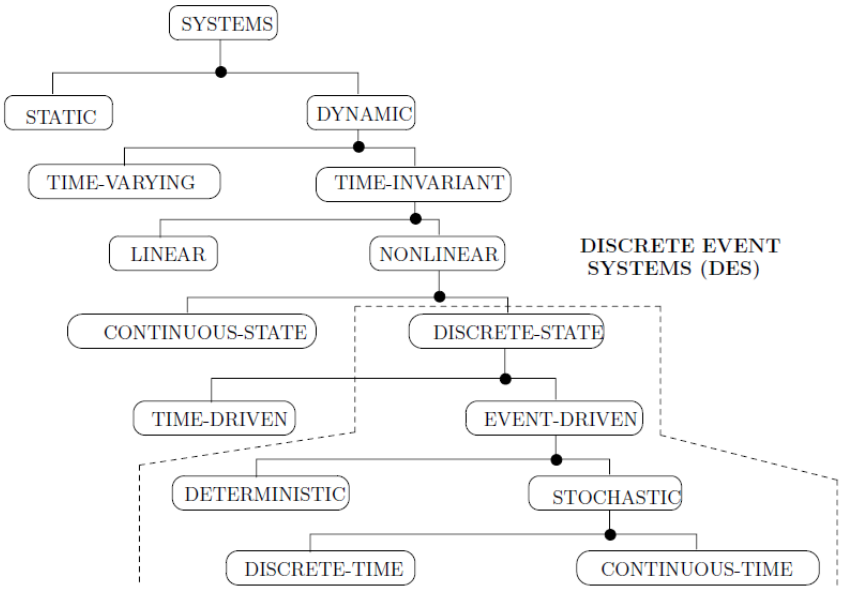


Figure 2.1 Partial classification of system [34]

Two aspects of DES are usually on focus at academic level. The first focus is on the formalization of DES. As the mathematic model of time varying dynamic system represented by differential equation and difference equation, the model which can accurately represent the system behavior is

of vital important. In DES, a number of modeling formalizations have been proposed such as automaton, Petri Nets, CTL, etc. On the other hand, another focus is on the methods of analysis, control, verification and simulations.

Supervisory Control Theory (SCT), firstly introduced by Ramadge and Wonham in 1987, is one of the most important academic theories and formal modeling paradigms for DES formalization, supervisor synthesis and verification. In CVDS, closed-loop (feedback) control is usually used to regulate the system performance based on the established target. Similarly, SCT takes advantage of "feedback loop" to achieve the real-time control on DES [34]. In the supervisory control paradigm, a supervisor continuously gets state information from the system and the environment, and feeds control information back to the system, in order to enforce some functional properties: either make invariant a given behavior which the system should always feature, or enforce a target configuration, which the system should eventually reach [35].

The controller and system to be controlled are formalized at the abstracted level within the SCT paradigm. The SCT provides algorithms for the synthesis of supervisory controllers from their specifications: a supervisor is computed from two distinct automata representing a given plant and the specification that describe the required controlled behavior of plant [25]. Therefore, supervisors operate synchronously with the plant to restrict the behavior of the plant to respect the specifications. Besides, recent studies have also proposed the SCT paradigm modeled by Petri Nets [15] [16] [36], which are out of the scope of this study however.

2.2.1.2 Application

Since the year when SCT was introduced by Ramadge and Wonham, the theory caught great academic interest and a large number of applications and solutions based on SCT emerge in endlessly over decades. The application cases which involve a wide range of domains prove the applicability of the theory and prominence in the AC community.

A. Manufacturing System

Manufacturing system is one of the application domains attracting earliest academic focus. The research team of Queiroz and Cury contributed a lot for SCT application in the domain of manufacturing system. In [22] [37], authors introduced SCT-based solution for coordinating equipment operation in a flexible manufacturing system, which usually consists of workstations, robots and buffers, which are modeled by plant. On the other hand, the producing processes are usually modeled by specifications. Local modular supervisors are converted systematically into a PLC application program and the solution allows one to use existing PLC hardware and software designed for the sequential control of subsystems.

The case studies show that SCT are quite applicable for AC of manufacturing system. Plants and specifications are used to model the behavior of operation units and manufacturing processes respectively. Methods for SCT implementation by PLC, which is widely used as concrete controller in manufacturing systems, make it prospective for SCT application in this domain.

B. Autonomous Robot

In [17], authors demonstrated an approach for formally synthesizing control software for a set of cooperating ground robots, which is based on modular supervisory controllers that are guaranteed to meet a given set of logical specifications. In the study, the behavior of each robot is modeled by plant and task precedence constraints are modeled by specification. The study explained how the supervisory level of control interacts with the robot-level algorithms that control each robot's heading and velocity. In [18], authors present a method for mobile robot navigation in industrial environments in which the open-loop behavior of the robot and the specifications are based on automata. A modular supervisor is built, which is the conjunction of two supervisors: the first one that enforces the robot to follow the path defined by a planner and the second one that guarantees the satisfaction of the specifications such as prevention of collisions and task and movement management. The modular supervisor is embedded in the mobile robot, whereas the planner runs in an external agent (plant), which makes the adaptation of the proposed navigation architecture to different environments easy. In [38], authors introduced an application of SCT on warehouse automation using mobile robots. The warehouse automation can be expected to provide a solution to the labor shortage problem so that it is a prospective and important application to meet the challenge of population decline. The supervisory control is used to realize navigation for assigned tasks. In the study, the supervisory control is implemented by the platform LEGO MINDSTORMS EV3 with an external laptop computer to realize a wireless communication. [39] presents a method for modeling and controlling autonomous mobile robots by SCT. In this study, through a sumo robot for competitions, are showed a systematic procedure for the modeling, design and implementation of the supervisor in a real controller, which provided a fast and efficient way to create strategies for robot attack and defense on the fly. The review shows that the robots in different domains can be applied by SCT. Plants and Specifications are usually used to model the behavior of robot and constraints for tasks respectively. The supervisory control can be implemented in different ways for robot: the industrial robots usually use PLC controller while the small scale mobile robots use microcontroller to realize supervisory control.

C. Electrical System

[23] introduces an application of a solution for the electric Vehicle Scheduling Problem (eVSP). A supervisory control is used to manage the power admission control of a group of local Plug-in Electric Vehicles (PEV). The admission control problem is formulated in the framework DES by creating the DES model of appliances and control specification. A supremal non-blocking

controller is designed so as to handle requests from PEVs while meeting the grid limits and fairly distributing the capacity among PEVs. Another case study of power park management in the application scene is power delivery networks, introduced by the same research team in [19]. A supervisory control is developed for high level control of the operation of a typical custom power park (CPP) using SCT. In this study, the behaviors of each electrical appliance are modeled by plants and the reaction strategy for different power qualities are modeled by specifications. Three modular supervisors are synthesized to realize the supervisory control for guarantee the high quality power to the loads. In [24], authors introduced an application of SCT in admission control of thermal appliances in the context of smart buildings. The scheduling of thermal devices operation is formulated in the framework of DESs, which allows for the modeling and design of admission control to be carried out in a systematic manner and ensuring the existence of the feasible scheduling prior to exploring control solutions. In [40], authors proposed an approach to manage the control responses to deploy in a High-Voltage Direct Current (HVDC) grid based on SCT. A decentralized grid control which focuses on local control and limits the number of non-critical events to be communicated is synthesized. These case studies in this domains show that the behavior of electrical components can be formalized as plants and power management strategies are modeled by specifications. Besides, in [19] [23] [24], the simulation studies are carried out in MATLAB and STATEFLOW. The simulation results validate the effectiveness and functionality of supervisory control.

D. Others

Apart from the aforementioned application domains, more case studies can be found in other domains. In [41], authors introduced a case study of development of control system of Multi Mover (theme park vehicle) based on SCT. The components on the vehicle are formalized by plants and three safety-related control strategies are modeled by specifications. The supervisor synthesized for the theme park vehicle has successfully been implemented and integrated in the existing resource-control platform. In [42], authors showed the applicability of SCT to the design of controllers for waterway locks. The controller is implemented by PLC and a code generation algorithm is implemented to generate structured text (ST) PLC code. The results achieved in this case study presents the applicability of the procedure for supervisor synthesis and automatic PLC code generation for industrialized systems. In [20], author proposed an application of SCT for Healthcare MRI scanner development. In this study, author applied SCT in two cases respectively, namely the patient support table and the patient communication system. Each case is modeled by a monolithic plant and requirement. In this study, the author focuses on the use of SCT to realize a formal development process instead of conventional process. The formal process shows its advantage in control code generation. In [43], authors used SCT to develop command sequences and fault recovery of a spacecraft propulsion subsystem. The supervisor controls the system in such a way that the design specifications are satisfied in both normal and faulty modes of

operation. In this study, authors extend the concept of supervisory control by “robust”. The study shows the use of robust supervisor is necessary in many aerospace systems when we try to solve the control and fault recovery problem.

2.2.1.3 Discussion

The growing complexity of the controllers, the demand for reduced development time, and the possible reuse of existing software modules result in the need for a formal approach in engineering practice [25]. The large number of case studies shows that SCT can be widely used in a variety of application domains and adopted by researchers thanks to its rigorous formal methods from formalization to implementation. According to the investigation result of applications, the major industrial ACs which need the help of formal approach of SCT usually have such features as follow:

Flexibility: In the field of engineering systems design, flexibility refers to designs that can adapt when external changes occur. Flexibility for an engineering system is the case with which the system can respond to uncertainty. The Flexible Manufacturing System (FMS), as introduced in the aforementioned cases in [22] [27] is a representative system with feature of flexibility. The covers the system's ability to be changed to produce new product types, and ability to change the order of operations executed on a part or ability to use multiple machines to perform the same operation on a part. The challenge of AC on flexible system structure or changeable tasks requires powerful approaches to guarantee that the designed AC can perform correctly and avoid such failures as blocking or unexpected system behavior. Apart from FMS, the feature of flexibility can also be found in other systems such as autonomous robot, whose trajectory are determined according to different control targets, and electrical system, which usually has to handle uncertain failures.

Complexity/large-scale: A complex or large-scaled system is a system which is generally composed of a couple of interconnected systems, possess different local dynamic behaviors, are equipped with multiple control loops and operate under different local environments and varying loads. Nowadays, industrial systems, such as FMS, power network, sustainable energy systems, transportation systems, and so forth, are becoming more complex, which have higher requirement for operation performance, productiveness, availability, reliability, and safety. Therefore, enormous challenges have great impacts on improving to development techniques in modeling, monitoring, and AC for complex industrial systems.

To face the challenge of AC on the systems above, formal approaches can be made the use of as excellent technique for modeling, development, synthesis, verification and validation of AC. In the applications, the SCT has been proven to be a unified modeling framework to handle the control

on different kinds of systems: the free behaviors of components are usually modeled by plant and the control targets or processes are modeled by specification. Different control architectures are synthesized for particular needs of each case study. On the other hand, the case studies show that supervisors can be implemented by concrete logic controllers or simulation. In a number of case studies, SCT are used to help to generate control code instead of conventional direct coding.

With the help of SCT, the development challenge can be handled faced in typical direct implementation process. For engineering practice, SCT-based formal approach has the advantages as follows:

- The formal approach provides rigorous mathematic modeling methods by which the modeling and computation results can be proven to be correct by designer;
- There are numbers of existing theoretical methods and algorithms proposed within the SCT paradigm which improve the reusability of the global modeling process;
- Based on the SCT, the verification of controller after synthesis and implementation is eliminated, as SCT can guarantee the properties of supervisory control. SCT increases safety, reduces the development cost and the cycle time and provides greater liberty for designers;
- Tool support is an important aspect for the application of formal approaches. In fact, a number of tools have been developed. TCT and Supremica are two main tools to realize the computation and verification of supervisory control. Other formal verification tools such as UPPAAL and NuSMV are usually used for formal verification and model checking.

In fact, the engineering can benefit a lot from formal approaches. Nonetheless, although SCT perform well in these case studies at scientific level, the case studies of SCT can hardly be found in engineering practice. Therefore, the study on the theoretical paradigm is furthered for the purpose of investing this issue of SCT.

2.2.2 Theoretical Concepts

The SCT has formed its systematic theoretical framework by scientific contributions. It includes s series of concepts, models and methods aimed at formalization and supervisor synthesis. This section presents an overview of the theoretical concepts within the original theory for supervisory control to briefly introduce the fundamental modeling principle of this paradigm. More details about SCT can be found in [34] or Appendix A.

2.2.2.1 Classification of Model

Models are expressed in some modeling language, and depending on the nature of the language, can be informal, semi-formal, and formal [44]:

- *Informal* models are expressed using natural language or loose diagrams, charts, tables, etc. They are genuinely ambiguous, heavily rely on human intuition, and no software tool can analyze them objectively;
- *Semi-formal* models are expressed in a modeling language that has a precisely-defined syntax, conveys some intuitive meaning, but has no formal (i.e., mathematical, self-contained, unambiguous) semantics. Examples of semi-formal specification languages are class diagrams, data flow diagrams, decision trees, entity relationship models, object models, pseudocode, state diagrams, etc.;
- *Formal* models are written in a language that has a precisely defined syntax and a formal semantics. Examples of formal specification languages are algebraic data types, synchronous languages, process calculi, automata, etc.

2.2.2.2 Formal Language and Automaton

The modeling paradigm of DES defines the behavior as a collection of sequences of events. The system is usually assumed as asynchronized which means only one event may occur at each moment. To model the behavior of a DES, the original SCT framework takes advantage of two formal models: formal language and finite-state automaton.

Definition 2.2 (Formal language): Each DES has an associated underlying event set Σ called an *alphabet*. A sequence of events from the alphabet is called a "word" or "string". An empty event string is denoted by ε. We denote the cardinality of an event set Σ as $|\Sigma|$. For a particular event set Σ, we denote its set of all possible finite strings of events by Σ^*. A *formal language* defined over an event set Σ is a subset of Σ^*.

Formal language, which is known as a kind of *regular languages*, specifies all admissible sequences of events that the DES is capable of "processing" or "generating", while bypassing the need for any additional structure. Actually, regular languages are seldom directly used to present the system and they are more suitable to prove properties. The reason is that regular languages are not intuitive way to model and present a repetitive behavior of system. Instead, the most usual modeling artifact is the finite state automaton [35].

Definition 2.3 (Finite-state automaton): A finite state automaton is defined as a 5-tuple:

$$G = (Q,\ \Sigma,\ \delta,\ q_0,\ Q_m)$$

Where,

Q is a finite set of states;

Σ is a finite set of events associated with G;

$\delta : Q \times \Sigma \to Q$ is the transition function;

q_0 is the initial state;

$Q_m \subseteq Q$ is a set of marked or accepting states.

Definition 2.4 (Active event function). Define active event function $\Gamma : Q \to 2^{\Sigma}$ as following:

$$\Gamma(q) = \{\sigma \mid \delta(q, \sigma) \text{ is defined}\}$$

Note that if there cannot be two transitions with the same event label out of a state, we call G a *deterministic finite-state automaton*. The event set Σ includes all events that appear as transition labels in the state transition diagram of automaton G. States are marked when it is desired to attach a special meaning to them. Marked states are also referred to as "accepting" states or "final" states. The transition function δ is usually a partial function. Thus each state can be associated with a set of events which are admissible at that particular state, i.e. for which $\Gamma(q)$ is defined.

The regular language and automaton are not two uncorrelated models. In fact, each regular language defined on a finite set of events can be generated by a deterministic finite state automaton. Therefore, there is equivalence between these two models [45].

2.2.2.3 Plant and Specification

In SCT, the model of behavior of the uncontrolled system is always called *plant*, which represents the formal abstraction of system and indicates the relation between an input signal and the output signal of a system without feedback, commonly determined by physical properties of the system. A complex system may have several modular models of plant based on the components or subsystems it consists of. SCT provides synchronous composition method to compose multi-plants to be a global monolithic plant. Whether to choose modular plants or a monolithic plant depends on the system complexity and control architecture in each particular case.

In order to constrain the free behavior of the system and make its behaviors respects the expectation, it is necessary to define *specifications.* In SCT, specification is also be modeling by automaton. Cassandra and Lafortune [34] introduce four typical kinds of specification shown as follow (examples for these specifications can be seen in Appendix A):

- Illegal states: specification identifies certain states of plant as illegal, then it suffices to delete these states from the plants;
- State splitting: a specification requires remembering how a particular state of plant was reached in order to determine what future behavior is admissible, then that state must be split into as many states as necessary;
- Event alternation: a specification requires that two events occur alternately;
- Illegal substring: a specification identifies as illegal all strings of $L(G)$ that contain substring $s_f = \sigma_1\sigma_2...\sigma_n \in \Sigma^*$.

In practice, the plants and specifications are usually modeled by people's expertise and directly formalized from the requirements. An issue of formalization exists: the use of an explicit model of the plant and specification may therefore be necessary to identify the correct behavior of the control realization, and to perform model analysis and control synthesis accurately [25]. Besides, there is no existing technique to indicate if the constructed models respect to the requirements, and all depend on the peoples' experience.

2.2.2.4 Supervisor Synthesis

The Supervisor Control Theory paradigm regards any system as composition of a plant to be controlled and a supervisor which ensures the control logic respect to the specifications. We usually introduce the supervisor synthesis started by the system consisting of one monolithic plant and one monolithic supervisor. The control feedback loop between the plant and supervisor is shown in Figure 2.2.

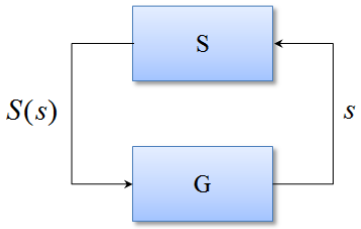


Figure 2.2 The feedback loop of supervisory control

Consider a DES modeled by a plant $G = (Q, \Sigma, \delta, q_0, Q_m)$ and assume that all the events in Σ generated by G are observed by supervisor. A supervisor, denoted by S, is adjoined to interact with G in a feedback manner. Here, let Σ be partitioned into two disjoint subsets

$$\Sigma = \Sigma_c \cup \Sigma_{uc}$$

With, Σ_c is the set of controllable events;

Σ_{uc} is the set of uncontrollable events.

The transition function of G can be controlled by S in the sense that the controllable events of G can be dynamically enabled or disabled by S. Formally, a supervisor S is a function from the language generated by G to the power set of Σ:

$$S : L(G) \to 2^{\Sigma}$$

For each $s \in L(G)$ generated so far by G (under the control of S):

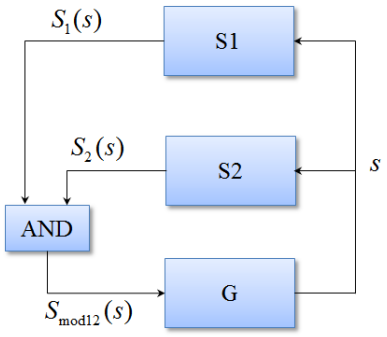
$$S(s) \cap \Gamma(\delta(x_0, s))$$

is the set of *enabled events* that G can execute at its current state $\delta(x_0, s)$. We call $S(s)$ the *control action* at s. In fact, the $S(s)$ is determined by the specifications which is purposed to constrain the behaviors of system.

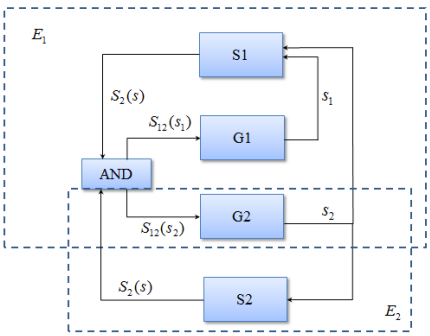
There are two basic properties of supervisor should be verified during synthesis: the controllability and nonblocking. The supervisor synthesis algorithm is the core of SCT, based on which the resulting supervisory control on plants with regards to the given specifications is proved to be controllable and nonblocking. SCT provides an iterative algorithm for supervisor synthesis and property verification (refer to Appendix A).

2.2.2.5 State Explosion Problem

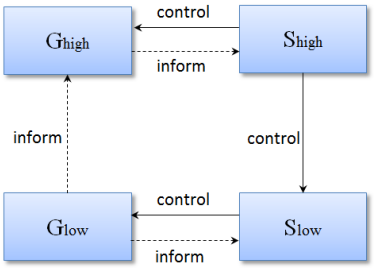
Generally, if the system is not complex, the centralized/monolithic control approach is enough to synthesis one unique supervisor S on plant G (Figure 2.2). Unfortunately, to model a large-scale system, the centralized approach cannot overcome the state-space explosion problem. The total number of states of a plant model increases quickly when the number of local components increases, due to the synchronous product which incurs Cartesian product [11].



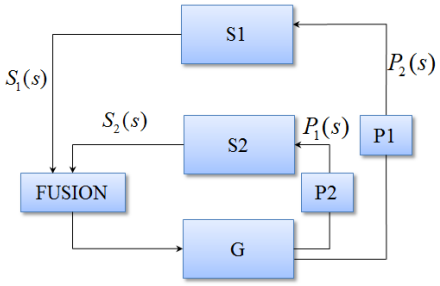
(a) Modular control



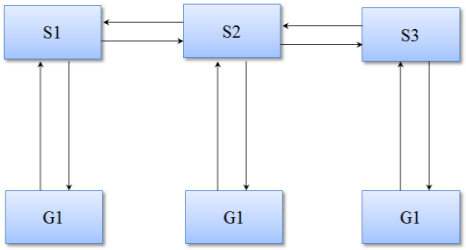
(b) Local modular control



(c) Hierarchical control



(d) Decentralized control



(e) Distributed control

Figure 2.3 Control architectures

A number of efforts focused on the methods to overcome this difficulty. One contribution proposes the concept of modularity (shown in Figure 2.3(a)) [8], which is extended to the concept of local modularity (shown in Figure 2.3(b)) [9]. In this control architecture, where the global supervisor is local modular, a global nonblocking of supervisory control must be verified.

In [46] [47], authors introduced a hierarchical method (shown in Figure 2.3(c)) that decomposes a system into two subsystems, and restricts the interaction of the subsystems by means of an interface. Thus, the complete system model never needs to be constructed, offering potentially significant savings in computational effort.

2.2.2.6 Distributed Control

In a system that have several “processing nodes” that are jointly controlling a given system, apart from the modular control, decentralized control (shown in Figure 2.3(d)) and distributed control (shown in Figure 2.3(e)) can also be used [48-50] to realize distributed control. A decentralized discrete-event system is a global plant modeled as a DES with two or more observed event streams and two or more inputs of enabled events. Each controller receives an observed event stream defined by either a projection or a mask, and inputs a subset of enabled events. In contrary, the distributed control usually has several local plants (agents) for large-scale DES. [51] studied the design of distributed control for large-scale discrete-event systems. Supervisor localization

approach is applied to allocate external supervisory control action to individual plant components as their internal control architectures. In [52], author proposed distributed supervisory control architecture for automated manufacturing system. This approach exploits modular models of the plant and logical equations in Boolean algebra for constraints modeling, which can avoids the combinatorial explosion of the state space.

2.2.3 SCT-Based Modeling Process: Principles and Limitations

2.2.3.1 General Process

[25] introduces the typical SCT-based development process by reviewing the state of the art of synthesis and implementation methods. The overall process, shown in Figure 2.4, involves three generic activities: formalization, synthesis (supervisor synthesis and control synthesis) and implementation as follows:

(1) Formalization

The formalization of the informal specification is a human core capability. Two kinds of formal models should be constructed. The models of specification formalize the desired behavior constraints to be fulfilled by the controller or the controlled process according to the requirements. Formalizing the required properties may also be a difficult task that causes many trials and errors and requires a dedicated expertise related both to the plant (to express the constraints related to the actuator and sensor operations and technology choices) and to behavioral model required for the subsequent synthesis procedure.

(2) Synthesis

In synthesis step, two sub-activities are necessary to perform: supervisor synthesis and control synthesis. The fundamental difference between a supervisor and a controller is that a supervisor forbids the occurrences of some controllable events in view of maintaining the plant behavior in legal states and sequences, while a controller acts by forcing the plant inputs in a way to achieve the desired goal [25]. Within the SCT paradigm, the synthesis approach consists in synthesizing a supervisor from a description of the plant and the specification. Such a supervisor aims at enforcing the specifications while offering a maximum behavioral flexibility. The controller can be regarded as the intermediate between formal supervisor and concrete controller. The controller is built based on the *control strategy*. Although there is not systematic study on the control strategy to transform supervisor to controller, a number of case studies have proposed their proper solutions. In [53], the authors proposed a trajectory choice approach in order to make the control behavior “deterministic”. The principle of the trajectory choice is to make sure the states in the

automaton are all accessible and co-accessible. In [54], the authors developed an algorithm for the generation of valid controllers which strengthened the coreachability property in order to guarantee that a marked state is eventually reached, irrespective of the plant's behavior. In [18], the authors proposed a planner in the navigation architecture under supervisory control. The planner, composed of several algorithms (including Dijkstra Algorithm), helps the modular supervisor to select a best choice of following path.

(3) Implementation

Regardless in academic context or industrial context, the supervisor/controller is usually implemented by *concrete controller* for validation, which is the target of this step. Based on the different objectives, the implementation can be classified by two kinds: concrete logic controller and simulation, which will be discussed in detail in the following part.

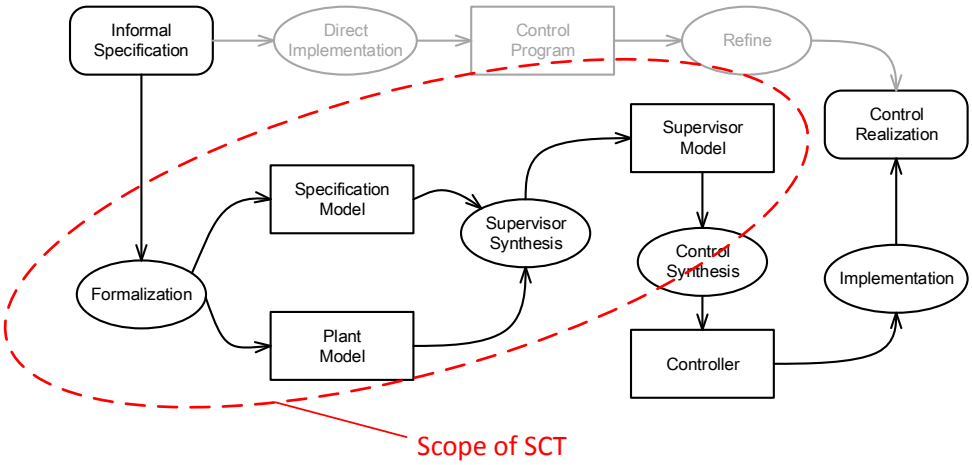


Figure 2.4 Activities and models involved in AC development process for DES

Remark *The concepts and terminologies in synthesis and implementation are important for the following study. Therefore, the taxonomy of relevant terminologies is presented in Table 2.1.*

Table 2.1 Taxonomy of terminologies

	Formal	**Logic**	**Physic**
Concept	Control architecture	Control strategy	Control implementation
Representation	Supervisor	Controller	Concrete controller

2.2.3.2 Controller Implementation

The case studies present that kinds of concrete logic controllers which can be used to realize supervisory control. As PLC is one of the primary executors of industrial automation, a lot of

study contributed in the SCT implementation by PLC control program, which are written in standardized languages, such as ladder diagram (LD), structured text (ST) or instruction list (IL), or SFC (Sequential Function Charts) (IEC 61131-3). Other case study [39] presents the supervisory control implemented by microcontroller from the Microchip PIC18F and dsPIC30F families. In [55], an application of supervisory control implemented the ATS, which is aimed at assisting RATP metro line management at the line control center to realize the traffic control and security control.

In addition, a part of contributions focus on performing control simulation based on SCT paradigm, which can be seen in case studies [19] [23] and [24]. The simulation is usually implemented by STATEFLOW toolkit in Simulink, which enables modeling of the controller in for of state transition based on signals from the workspace. It is also possible to send commands in each state to the workspace. One points should be taken into account when implementation by simulation. Physic characteristic of the controlled system components should be clarified when we want to validate the control performance by simulation. As shown in [19], the effects of control action are illustrated by the voltage change of three-phase AC. Therefore, besides the model of controller in simulation model, we also need the models of physic system.

The principle of concrete logic controller is shown in Figure 2.5. Controller receives input signals coming from sensors and send output signals to actuators, in accordance with control laws implemented into a user program. A controller cyclically performs three tasks: reading and storing the inputs, program execution, and outputs updating. The period of this task may be constant (periodic scan) or may vary (cyclic scan). For the outside viewer, and specifically the plant, the controller can be considered as a reactive system as the output signals change their state simultaneously in response to the input signals, given that the scan time is short with respect to the time constant of the plant.

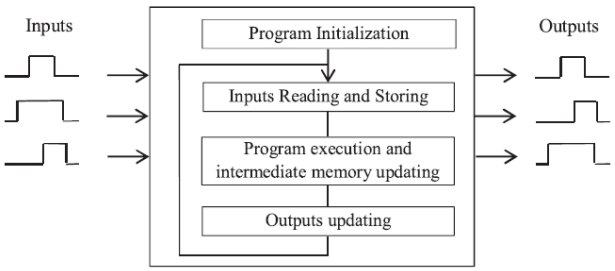


Figure 2.5 Basic Principle of Logic Controller

Due to some common characteristics such as input scan cycle, nearly all types of controller will encounter difficulties, which are systematically introduced in [56]. The study concluded five main concerns when moving from asynchronous, finite automata model of the supervisor, to the synchronous nature of, but not limited to, PLC control system. An interesting insight about the related phenomena is given in [57].

A. Inexact Synchronization

During the program execution a change in any controller input signal may occur and, this change will only be recognized at the beginning of the next scan cycle. The control reasoning is always performed on old frozen data. Therefore the communication between the controller and the plant is subject to delays due to periodic reading of the input signals. This inexact synchronization can be a problem when a change in a PLC input signal invalidates a control action (the choice made by the program, which corresponds to the generation of a controllable event).

B. Binary signals and events

The SCT deals with symbolic events that can occur asynchronously at any instant of time, whereas controller handles Boolean valued signals that are usually updated synchronously. *Avalanche Effect* occurs when a change on the value on a given controller input signal is registered as an event that makes the software jump over an arbitrary number of states within the same scan cycle.

C. Causality

SCT assumes that all events are spontaneously generated by the plant and that supervisors should only disable events generated by the plant. However, controllable events on practical applications are not spontaneously generated by the physical plant, but as responses to given control commands. Thus, for implementation purposes, "who generates what?" must be answered.

D. Simultaneity

Due to the cyclical nature of the control processing in which input signals readings are performed only at the beginning of each scan cycle, the occurrence of uncontrollable events from the plant is recognized by the controller once there are changes in the input signals values. Therefore, if in between successive scan cycles two or more signals change, they will all be recognized as simultaneous uncontrollable events regardless of their exact timing. As a result, the controller is unable to recognize the exact order of uncontrollable events that happen between scan cycles.

E. Choice

The supervisors obtained by the SCT are required to be “minimally restrictive”, which means that the supervisors might provide alternative paths for the plant to choose from. Often a supervisor presents more than one possible controllable event from a single state. Thus, before producing a signal-change in the controller outputs it may have to choose only one among them because according to [56], generating more than one controllable event in a scan cycle can be contradictory and catastrophic. [58] shows that when the choice problem prioritizes the execution of a controllable event over the other, the controlled behavior may be blocking, even though the originating supervisor is deadlock free.

In order to overcome the difficulties and narrow the gap between the supervisor and its implementation, different control implementation approaches have been developed to provide possible solutions. In [59], the authors propose a solution based on the reordering of controller program instructions or the introduction of supplementary memory elements in controller programs for the detection and elimination of the avalanche effect problem. [60] proposes a direct transformation method from supervisor to PLC program. The main problems of supervisor implementation on a PLC are pointed out and some solutions to alleviate the problems are proposed. In [58] [61], conditions and algorithms are provided to guarantee that every deterministic controller derived from the supervisor (i.e., whatever controllable event corresponding to a controller order at a given state is chosen/enabled, supposing that only one will be taken and the others will therefore be disabled) is nonblocking. In [21], solutions based on the reordering of PLC program instructions or the introduction of supplementary memory elements in PLC programs have been proposed for the detection and elimination of the avalanche effect problem. [37] proposes a three-level model of PLC implementation for local modular supervisory control. The proposition introduces the conceptual aspects of a systematic procedure to structure and to detail the program to be implemented in a programmable logical controller. The proposition can be used to the implementation of local modular control. In order to coordinate the control action of local controller, the three-level structure is deliberately design. [62] presents a nine step methodology, named DECON9, to implement SCT into PLCs in standardized, efficient and robust ways, closer to real size plants. The methods it allows the control logic to deal with many events at each scan cycle, which improves existing approaches that are constrained to only one event at a time and a nonblocking property is achieved due to the random selection of controllable events approach that solves the choice problem. Besides, there is no fear of an avalanche effect thanks to the use of auxiliary memories and uncontrollable events are prioritized. To sum up, the implementation of supervisory control based on PLC platform is systematically studied and the state-of-the-art methods are proven to be able to transform supervisor to PLC control program for different control architectures.

Table 2.2 A summary of Solutions for implementation problems

Problem	**Solution**
Inexact Synchronization	[37] [62]
Binary signals and events	[37] [59]
Causality	[21] [37] [58] [60][61] [62]
Simultaneity	[37] [60][62]
Choice	[21] [37] [58] [60][61] [62]

2.2.3.3 Problematic

Although a systematic theory is constructed for SCT according the aforementioned review, the paradigm has its limitations from the perspective of engineering context. As shown in Figure 2.4, the scope in SCT is limited the formalization and supervisor synthesis. Even if the controller implementation is considered as the extension of SCT, when SCT-based process is applied in engineering practice, several issues have to be faced:

Limitation 1: Formalization

The quality of the synthesis results highly depends on the relevance of the requirements proposed by the engineering inputs. The formalization of the informal requirements and free behaviors of the system to be controlled is one of the major difficulties facing the existing control synthesis approaches in engineering practice [25].

Firstly, systems' requirements are usually written in an informal narrative since it generally means a greater understanding among the various stakeholders. On the other hand, formal models such as automaton have unambiguous semantic, which means a sentence cannot be understood in different ways. Most of case studies, however, achieve their controller modeling and implementation by directly formalizing the requirement as the beginning. It is still difficult to link the formalization and informal narrative requirements. For example, a controllable event defined as uncontrollable event by mistake leads to problems for final results. In fact, the typical SCT-based modeling process pays little attention to the verification of consistency between informal requirements and formal specification in form of automaton. Besides, another limitation is lack of representation of the link between the informal requirement and formal specification. In engineering practice, the requirements must be traced and a number of tools such as requirement traceability matrix (RTM) or graphic models (SysML requirement diagram) are used in the development process [63] [64]. The requirement traceability representation for formal models within SCT paradigm still unsolved.

Secondly, the formalization of free behavior of the system is also questionable. The identification of plant from the system to be controlled is of important for engineering practice. However, few contributions can be found to provide systematic methods. For example, the buffer, as a component of physic system, is usually formalized by specification rather than plant. On the other hand, in [18], the behavior of mobile robot is formalized by several plants. A contradiction can be seen by comparing these two situations and a question can be put forward: how on earth formal plants can be linked with physic system to be controlled? Indeed, in a number of contributions, the plants are supposed to be formalized by identifying the signals from sensors and commands to actuators, in accordance with control laws implemented into a user program [20] [25] and [30]. However, there is no study can be found particular for this point. Secondly, similarly to

specifications, the verification of the correctness of plants is still unsolved, which is also important for the result of supervisor synthesis.

Indeed, the requirement traceability is recognized as a concern in an increasing number of standards and guidelines [65]. In engineering practice, the explicit trace relationship is of vital importance to guarantee the product to conform to what we need. Compared with SCT-based approach, the formalization problem which involves both plants and specifications leads to a lack of traceability between formal models and inputs. The reason of this issue is due to the lack of models within formal approach which can present the traceability.

Formalization and obtaining a precise formal model may be a difficult task, which are related both to the plant (to express the constraints related to the actuator and sensor operations and technology choices) and to the type of behavioral model required for the subsequent synthesis procedure [25]. The further study [66] has pointed out that human performance in problem solving in the context of SCT play an important role although a number of software packages supports the modeling and computation process. The result of the study shows that the visual appearance of the models is of such great importance to the subject - as they spent about a quarter of the time adjusting the appearance of the models. [67] indicated that when faced with a new problem, subjects frequently engaged in drawing a simple diagram of interactions between parts of the system which needed to be modeled.

Limitation 2: Structure Model

As the formal model can only represent the behavior of system to be studied, it is difficult to describe the structure aspect, which includes both physic and logic. Several issues arise aimed at this point. Firstly, the plant models which represent the free behavior of physic components are totally unable to contain any information on the system structure. In the case studies reviews aforementioned, the systems are simple and without any structure. However, the behavior of a physic component can be modeled by a number of plants (e.g. in [18], the behaviors of robots are formalized by several independent plants according to different modules). The gap between plants and physic components leads to the problem: the consistency between plants and physic components is unclear. Secondly, the supervisor/controller may also be structured. It is always the case that all kinds of supervisor are structured except from the monolithic one when the control architecture is modular or distributed. In Figure 2.3, a number of control architectures are presented by schema. Unfortunately, there is no model can be used to explicitly describe the details of the structure, which leads to misunderstanding or confusion for people who focus on the AC in the engineering practice. Similar, it is still the case for controller as the controller can also be structured according to different control strategies. In academic context, it is acceptable that the structure can be illustrated simply by schemas; however, it is unacceptable as engineering practice usually requires standardized forms of representation (such as SADT, SysML, BPMN and so on).

The problems above arouse little academic attention and few contributions can be found.

Limitation 3: Implementation Model

In SCT-based process, the controller is implemented based on the supervisory control architecture. Although a number of methods have been proposed to transform supervisor/ controller to concrete controller, there is still gap between both of them. The existing contributions focus on how to transform the supervisor to the software and hardware aspect of concrete controller is neglected. In fact, it leads to problems of controller implementation. For example, in the study [37], the local modular supervisors are implemented by PLC control programs in SFC. From the description of the study, the control programs are deployed in one monolithic concrete controller. However, modular control can also implemented by distributed concreted controller. Besides, the control connections are not clear as no model can represent the internal details of physic control system within the SCT paradigm. In fact, these two situations are due to ambiguity for hardware and software and lack of the implementation models. The existing supervisory control architecture and implementation methods are not able to specify the link between the models of supervisor/controller and models of concrete controller, which is unacceptable in the engineering context.

Limitation 4: Lack of engineering process

SCT paradigms provide the formal solutions for AC. However, there is no global framework to standardize the modeling process useful for engineering practice. In fact, for each modeling step in Figure 2.4 (p.18), a large number of different academic approaches for formalization, synthesis or implementation have been proposed. On the other hand, it seems always the case that most scientific contributions are classified into two parts: the supervisor synthesis and controller implementation. In [48-50], contributions focus on theoretical method for supervisor synthesis, while in [37] and [62], case studies that focus on how the supervisor can be implemented into control program can be seen. Supervisor synthesis and controller implementation seems to be two independent scientific domains. In contrast, engineers usually tend to take the advantage of development process in which a methodical series of steps that engineers follow to come up with a solution to a problem. Therefore, the lack of global modeling process of SCT leads to difficulty for engineering practice.

Several contributions can be found of the focus on this issue [41] develops an engineering process with supervisory control synthesis, shown in Figure 2.6. In this approach, advantage of this integration is that simulation and verification of supervisor control can be performed at different platforms such as MODELICA, UPPAAL or MATLAB/SIMULINK, which is based on CIF (Common Interchange Format) [68]. However, by further scan, the proposed process in the study makes little difference to the typical process in Figure 2.4 with the same inputs/outputs and the

problems mentioned in Limitation 1-3 still remains. For example, the structure of the plants, supervisor and controllers is still unclear. Besides, the proposition doesn't deal with the formalization problems.

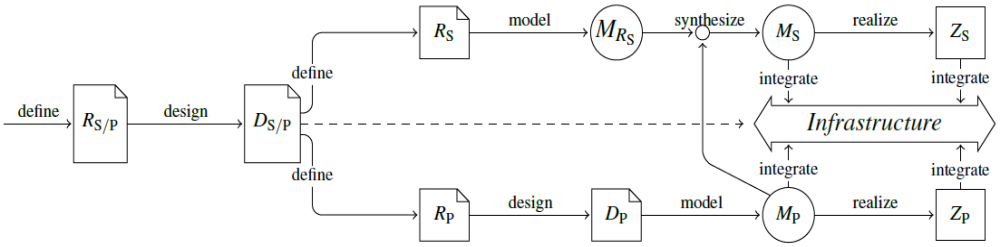


Figure 2.6 Engineering process with supervisory control synthesis [41]

[69] proposes a model-based systems engineering framework that enables supervisor synthesis of nondeterministic discrete event systems, and post-synthesis validation of functional and quantitative properties of the supervised system. In this study, a tool chain is proposed to realize different verifications (Supremica to MRMC for performance evaluation and to UPPAAL for Safety/liveness verification). Similar to the previous proposition, the limitations of SCT are still unsolved.

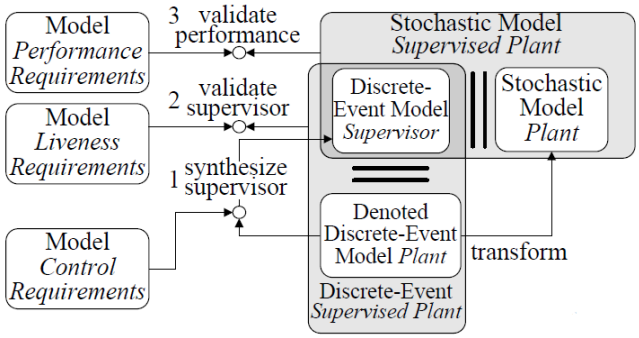


Figure 2.7 Integrated framework in proposition [69]

To sum up, engineering practice shows that standardized design processes are critical to improving design efficiency and implementations as a large number of technical standards and standardized methods are proposed in engineering domain [70]. In contrast, the state-of-the-art SCT-based process propositions are still within the scope of the process in Figure 2.4 (p.18) and show more or less their drawbacks aimed at engineering practice.

2.2.4 Summary

In the review of AC based on SCT, the state-of-the-art modeling approaches and framework are presented. Four limitations, which have to be faced so as to narrow the gap between the engineering practice and academic contributions, have been identified. By the discussion in this

section, it can be found that it is still a research gap in this field. The existing solutions are not satisfactory enough to deal with problems we've found. Therefore, in the following sections, the practical engineering methods and processes will be investigated for further the study on coping with the limitation of SCT.

2.3 Model-Based System Engineering

The SCT is a kind of model-based approach because the framework relies on formal modeling of system behavior. In engineering practice, Model-Based System Engineering (MBSE) is power methodology for system design. For the purpose of extending SCT as a global framework from requirement analysis to implementation to conform to the engineering practice, MBSE provides candidate solutions. The MBSE standards, methods and modeling languages make it possible to integrate relevant models, approaches, input/output and framework into a standardized development process. Questions should be answered: Is it possible to integrate SCT in a Model-Based System Engineering (MBSE) process to extend the usability of SCT in engineering? Compared with SCT-based approach, what are the advantages of MBSE? In order to answer these questions, the state of the art of MBSE will be reviewed in this section.

2.3.1 Overview

Model Based System Engineering (MBSE) Initiative was proposed within the International Council on Systems Engineering (INCOSE) SE 2020 Vision at the Albuquerque, January 2007 [71]. MBSE is a systems engineering methodology that focuses on creating and exploiting domain models as the primary means of information exchange between engineers, rather than on document-based information exchange. INCOSE SE Vision 2020 gave a definition of MBSE: Model-based systems engineering is the formalized application of modeling to support system requirements, design, analysis, trace, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases. Despite just ten years since being initiated, MBSE methods are widely applied in different engineering applications:

[29] introduced using MBSE and SysML to model a standard CubeSat and applying that model to an actual CubeSat mission, the Radio Aurora Explorer (RAX) mission, developed by INCOSE Space Systems Working Group (SSWG) Challenge Team. The SSWG demonstrated the ability to model behaviors, interface with commercial off-the-shelf (COTS) simulation tools, and carry out trade studies [72]. The team updates their project progress every year and the newest report can be found in [73]. [74] presented a project AUTOSAR which use EAST-ADL for the whole modeling process. The AUTOSAR development partnership was formed in July 2003 by BMW, Bosch, Continental, DaimlerChrysler, Siemens VDO and Volkswagen to develop and establish an open

industry standard for automotive E/E architecture. [75] presented a practical design approach for submarine subsystems using a model-based systems engineering methodology. The advantages of designing submarine subsystems reflects the objectives of MBSE, namely improved design consistency, precision, traceability, subsystem integration, and design evolution. In the shipbuilding industry, where traditional ship design practices persist, MBSE makes system modeling accessible to domain engineers. [76] introduced application of MARTE (Modeling and Analysis of Real Time and Embedded systems), the OMG standard for modeling real-time and embedded applications developed by UML profile. The framework is built over a multiplatform library that allows the execution of the same code in different operating systems, which provides the high-level interfaces with hardware, external configuration programs, and user interfaces, assuring at the same time hard real-time performances.

In the field of automatic control, case studies of MBSE application can also be found. In [77], a SysML-based approach is proposed to support the model-driven Manufacturing Automation Software design projects. The SysML is adapted to define the SysML-AT (SysML for automation), a specialized language profile that covers (non-)functional requirements, corresponding software applications and properties of proprietary hardware components and supports an automated software generation for run-time environments conforming to IEC 61131-3 for PLCs. In [78], authors develop an embedded control system for a car seat. The system requirement analyzed and classified and the structure of system is modeled. In the study, modeling tools such as DOORS and AutoFocus are used to support modeling different diagrams. In [79], an industrial Control System case study - heat treatment line is developed. The study identified three domains, control engineering domain, electronic - electric engineering domain and software engineering domain, to specify different views and components as well as connectors has been used in order to identify its role in each domain.

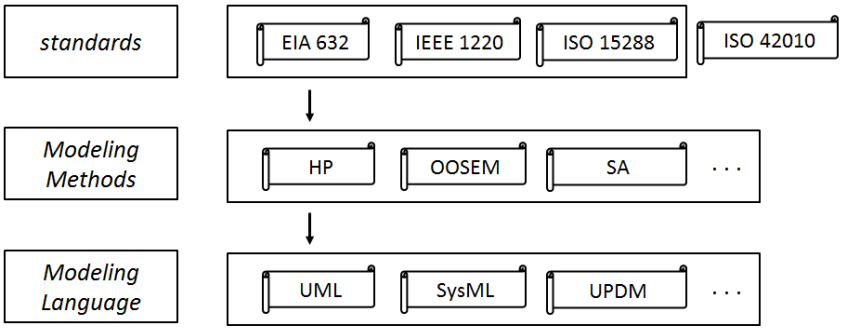


Figure 2.8 A partial systems engineering standards taxonomy [80]

With the development of system engineering in academic and industrial application over decades, MBSE have evolved a series of standard and method at different level within MBSE framework. Figure 2.8 shows a partial taxonomy of standards that includes systems engineering process

standards, methods and modeling standards, which will be reviewed in detail in the following sections.

2.3.2 Standard

A significant level of effort has been devoted in the past several years to the development, refinement, and ultimately, acceptance of processes for engineering systems or systems engineering processes [81]. The standards describing a SE process include EIA 632 [82], IEEE 1220 [83], ISO 15288 [84] and ISO 42010 [85], which address broad industry needs and reflect the fundamental tenets of systems engineering that provide a foundation for establishing a systems engineering approach.

- IEEE 1220: IEEE Standard for Application and Management of the Systems Engineering Process;
- EIA 632: Processes for Engineering a System;
- ISO 15288: Systems and software engineering - System life cycle processes;
- ISO 42010: Systems and software engineering - Architecture description.

Rather than presenting a single process or method that is applied to each phase of the acquisition process, EIA 632 identifies multiple processes that, when integrated together, provide the systems engineer with an overall methodology or process for developing a system The processes EIA 632 are grouped into the five categories: technical management, acquisition and supply, system design, product realization and technical evaluation. The ISO 15288 presents a series of processes - agreement processes, enterprise processes, project processes and technical processes - which are applied at different phases of system development and have unique purposes. These processes are organized in a hierarchy and are intended to be tailored to the specific application. ISO 15288 has a more extremely broad scope than EIA 632 as it applies to the full life cycle of systems, including concept, development, production, utilization, support and retirements of system and to acquisition and supply. IEEE 1220 differs from ISO 15288 in important ways with respect to both system structure and terminology. IEEE 1220 involves trade studies and assessments of requirements alternatives, functional alternatives, and design alternatives, which allow application of judgment and decision processes to determine best alternatives. Compare with EIA 632 and ISO 15288, standard IEEE 1220 described more at the task or application level while the other two standards focus on project or product-oriented management. The process standard define top-level criteria for system design and a number of MBSE methods deriving from the standard are proposed to serve as candidates for adoption and tailoring to an organization’s SE practices and procedures. The leading methods will be presented in Section 2.3.3.

Different from the previous three standards, ISO 42010 addresses the creation, analysis and sustainment of architectures of systems through the use of architecture descriptions. The standard introduces the concept of system's architecture which assists the understanding of the system's essence and key properties pertaining to its behavior, composition and evolution, which in turn affect concerns such as the feasibility, utility and maintainability of the system. ISO 42010 defines a series of architecture description (AD) elements such as concern, stakeholder, view, viewpoint and convention, etc. Architecture description is composed of a number of AD elements by which the engineering environment of a system can be described. In engineering practice, the AD elements are usually used in the development process. For example, in [86], authors describe a technique developed at JPL of applying SysML viewpoints and views to organize AD elements in the project and generate documents and reports. Therefore, a MBSE environment is envisioned with a central repository of model and Viewpoint information will be the key to integrating all the pieces needed to execute successfully in a model based project.

2.3.3 Methods

System development life cycle (SDLC) is a term used in systems engineering, information systems and software engineering to describe a process for developing a system. A system development life cycle is composed of a number of clearly defined and distinct work phases which are used by systems engineers and systems developers to plan for, design, build, test, and deliver a system. A number of SDLC models have been created, including waterfall, fountain, spiral, build and fix, rapid prototyping, incremental, synchronize, and stabilize [87]. The first known life cycle paradigm introduce in 1956 was waterfall approach, which is a relatively linear sequential design approach for certain areas of engineering design. The V-model (Figure 2.9) [26] is proposed to represent a development process that is considered as an extension of the waterfall model instead of moving down in a linear way. The process steps are bent upwards after the coding phase, to form the typical V shape.

The original V-model doesn't clarify the specific steps and models which should be used. Therefore, a number of MBSE methods and frameworks are proposed by the systems engineering community based on the V-model and MBSE standards. These methods provide the specific modeling steps and input/output within the frameworks.

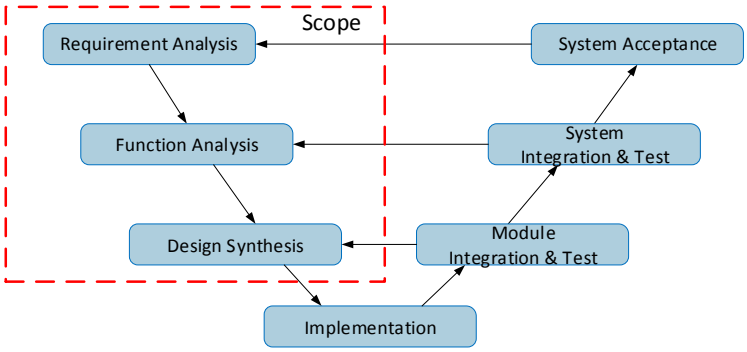


Figure 2.9 V-model

OOSEM (Objected-Oriented System Engineering Method) evolved from work in the mid 1990's at the Software Productivity Consortium in collaboration with Lockheed Martin Corporation [88]. It is a top-down, scenario-driven process that uses SysML to support the analysis, specification, design, and verification of systems. The process leverages object-oriented concepts and other modeling techniques to help architect flexible and extensible systems that can accommodate evolving technology and changing requirements. OOSEM is also intended to ease integration with object-oriented software development, hardware development, and test processes. As OOSEM is relative early method, it didn't define the modeling language and no toolkit supports it at all. Harmony System Engineering Process (HP) is a subset of a larger integrated systems and software development process known as Harmony® [89]. Key objectives of HP are identification and derivation of required system functions, identification of associated system modes and states, and allocation of the identified system functions and modes/states to a subsystem structure based on iterative workflow with incremental cycles through the phases requirements analysis, system functional analysis and design synthesis. HP defines the involved models by SysML and all modeling toolkits supporting SysML can be used. Vitech MBSE Methodology [90] consists of four main domains: source requirements (originating requirements, issues and decisions, risks), behavior (system behavior models, inputs/outputs, control/sequencing, and performance requirements), architecture (system architecture, components, interfaces, and allocated requirements), validation and verification (analysis, verification methods, test plans). Modeling in iterations, so called levels, helps to elaborate system specification. Domains (all except the validation and verification domain) and levels of the Vitech MBSE Methodology correspond to the pillars and abstraction layers of the MBSE Grid [91]. State Analysis (SA) [92] is a JPL-developed MBSE methodology over the last decade for architecting, designing and documenting complex control systems. The goal of this process is to make it easier for system engineers to precisely express design intent in a tool that actively helps to ensure consistency. The SA method defines an iterative process for state discovery and modeling, which allows the models to evolve as appropriate across the project lifecycle. Tool support for SA is provided by the State Database, which utilizes a Structured Query Language (SQL)-compliant relational database

management system (RDBMS) such as Oracle® with a front end user interface.

At academic level, MBSE frameworks are also proposed. In [93], author proposes a SPIRIT framework supporting model-based systems engineering (MBSE) tool-chain development of advanced cyber-physical systems (CPS) with emphasis on tool integration, process management, automated verification and validation. In this framework, the typical V-model is used and metamodel are used to represent development processes. The framework advocates using different tools (model language, program language and Simulink, etc.) for each step of the development process. In [94], authors proposed the Model-Driven Development Approach (MDDP) to address the issues by combining best-practices from MBSE and Lean Information Management. The framework is based on the recursive and iterative process of four layers: Requirements, Functional, Logical, and Physical. These contributions can be regards as an extended application of MBSE methods. In this proposition, a mix of standard UML, domain tools and specific MDDP artifacts are used, but it doesn't adopt SysML diagrams. In [95], Architecture Analysis and Design Integrated Approach (ARCADIA) has been developed by Thales for architecture centric and model driven engineering activities. ARCADIA introduces five engineering steps, which start from an operational analysis and end with a product breakdown analysis.

Although there are a number of methods proposed which share a similar development procedure of V-model whether at industrial level or academic level. To provide a detailed comparison of the SCT-based process and MBSE process, the input/output and global modeling phases are recalled in Figure 2.10.

(1) Input

In the process of SCT, the inputs are supposed to be the uncontrolled physic system and narrative control requirements. In the MBSE Process, the unique inputs are the stakeholder requirements. Compared with MBSE process, the SCT process can be regarded as starting from semi-black-box inputs as partial system has already been given as input while MBSE process start from total back-box.

(2) Output

Globally, in the state-of-the-art SCT process, the final output is the concrete controller. The MBSE process can produce both hardware and software results, which should be passed on to following process in life cycle such as test and validation. Although the outputs for both processes are similar, it is noticeable that the SCT process can just directly produce the "physic" final outputs while the outputs in MBSE can be presented by models.

By further study on the output of each modeling phase, only behavior models represented by formal plant, specification and supervisor are involved in SCT process, especially in the phases of

formalization and synthesis. The models involved in MBSE process depend on the modeling languages to be used. Taking SysML as example, a series of models are involved in different modeling steps. In the requirement analysis, system requirements and use cases are model by requirement diagrams and use case diagrams. In order to assure that all functional and associated performance requirements are covered by the use cases, respective traceability between requirement and use case is established. In the functional analysis step, behavior diagrams are used to represent the use case scenarios. There is no definite model to represent the function and each behavior diagram (i.e. activity diagram, state machine diagram and sequence diagram) can play a specific role in the elaboration of the use case behavior. In the design synthesis step, several models should be constructed. The architectural analysis model is to present trade study to determine the best means of achieving the capability of a particular function in a rational manner, which is usually in form of BDD. The black-box behavior models should be realized by white-box behavior models. The system architecture models and physic subsystem models represent the final results, which are modeled by BDD and IBD.

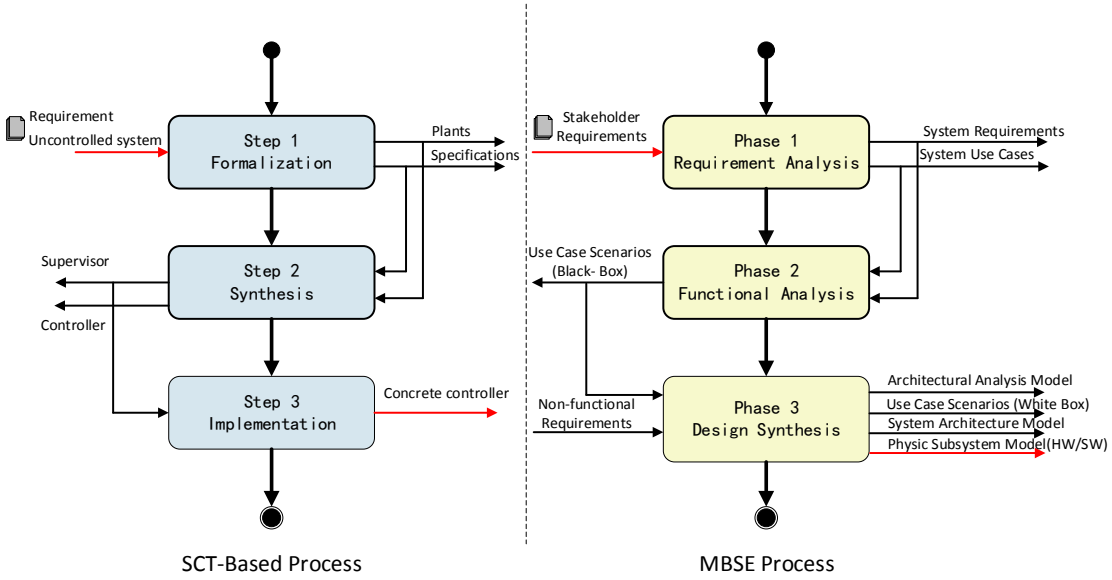


Figure 2.10 Processes comparison

Compared with SCT-based process, the advantages of MBSE can be identified as follows:

(1) In the requirement analysis, the stakeholders' needs are analyzed. Requirement models are constructed to present the results of analysis and traceability of the system requirements. In this phase, stakeholder requirements are translated into system requirements that define what the system must do and how well it must perform. The phase is important but lacked by AC development process. Compared with SCT-based process, the directly formalization leads to the difficulty to verify the consistency between formal models and inputs. In MBSE process, different models can be built to explicitly present the analysis, decomposition and traceability of requirements (e.g. requirement diagram, use case).

(2) In the design synthesis phase for the controller, a number of models are built to focus on the development of a physical architecture. In the SCT-based process, the implementation are realized by directly transform from supervisor/controller to physic controller. In contrast, MBSE provides a series of models to present the design process in details. For example, the concrete controller can be modeled in MBSE so that an explicit perspective for the final system can be built for verification and validation before implementing to real physic system.

By further comparing with SCT-based process, two drawbacks of MBSE can be identified for AC design:

(1) The first drawback of MBSE is that it doesn't attempt to provide any formal models and methods for functional analysis. Usually, the models to present the system function are semi-formal such as activity diagram or sequence diagram. When consideration applying MBSE into AC design, the function models based on the informal/semi-formal models cannot be regarded as better solution than SCT-based process, as the latter provides rigorous mathematic methods in the formalization step. In fact, the problem has caught a lot of attention in some other domains. [96-98] introduced an integration of Method-B with MBSE and an integration of PN/HCPN with MBSE is proposed in [99-102]. In these propositions, the formal methods are used for verification for the purpose of making up for the drawback of lack of formal methods. However, no proposition of integration with formal methods for AC can be found.

(2) Secondly, within the whole process, MBSE doesn't provide any methods and profiles particular for AC design. For example, in design synthesis phase, a general architectural analysis based on trade study is defined by MBSE. The objective of a trade study is to determine the best means of achieving the capability of a particular function in a rational manner. Assessment criteria are weighted according to their relative importance to the overall solution. However, it is still unclear how it works for the evaluation of AC. On the contrary, the supervisor synthesis and verification method within SCT paradigm provides an excellent solution on this point.

To conclude, MBSE perform better in requirement analysis as methods and models are provided for systematic analysis, decomposition and traceability of requirements while is lacked by SCT. On the other hand, SCT perform better than MBSE for function analysis due to the formal approach. For design and synthesis, both SCT and MBSE have their proper advantages. It seems the situation where SCT provides a number of concepts, architectures and methods for AC, which cannot be fully described due to the limitation of modeling scope of formal models, while MBSE can provide a variety of models but lack of methods for the context for AC. Therefore, SCT and MBSE can be complementary for each other for AC attempting and it is a prospective solution with lots of interest to integrate SCT and MBSE to provide both formal methods and semi-formal methods.

2.3.4 Modeling Language

For further study on the models, the modeling languages of MBSE are recalled. A variety of modeling languages are proposed by OMG and ISO for realizing the MBSE methods. Structured analysis and design technique (SADT)/IDEF0 [103] was first published in 1981, which is relatively earlier proposed modeling language in the history of MBSE. SADT/IDEF0 offers a functional modeling language for the analysis, development, reengineering, and integration of information systems, business processes, or software engineering analysis. A newer modeling language proposed by OMG is Business Process Model and Notation (BPMN) [104]. BPMN provides businesses with the capability of understanding their internal business procedures in a graphical notation and will give organizations the ability to communicate these procedures in a standard manner. BPMN is based on a flowcharting technique similar to activity diagrams, which makes it applicable more for process-oriented design (e.g. process miner [105]). Due to the lack of diversity in model types, from an engineering point of view, BPMN is a modeling language to support business process management and represent complex process semantics, which is similar to activity diagrams in UML.

Specialized modeling languages aimed at particular domains have also been published. For example, EAST-ADL (Electronics Architecture and Software Technology-Architecture Description Language) is an architecture description language specially targeting automotive systems and developed in several European research projects such as EAST-EEA, ATESST I, ATESST II, etc. [106]. It is designed to complement AUTOSAR (AUTomotive Open System ARchitecture) with descriptions at higher level of abstractions. Aspects covered by EAST-ADL include vehicle features, functions, requirements, variability, software components, hardware components and communication.

By focusing on the domain of engineering application, two leading-edge general modeling languages proposed by OMG catch more widespread attention: UML (Unified Modeling Language) and SysML (SYStems Modeling Language). Compared with other modeling language, UML and SysML have their advantage in terms of model diversity to make them more efficient for system design and to be made use of in a number of MBSE methods (e.g. Harmony-SE, OOSEM, etc.) as different model types are required in different modeling steps.

2.3.4.1 UML

UML is a modeling language used to specify, visualize, construct, and document aspects of the system-development process [107]. UML has many types of diagrams, which are divided into two categories [108]. One category represents structural information, including class diagram,

component diagram etc., and the other represents general types of behavior, including a few that represent different aspects of interactions. These diagrams can be categorized hierarchically as shown Figure 2.11.

Structure diagrams describe static view that emphasizes the static structure of the system using objects, attributes, operations and relationships. Since structure diagrams represent the structure, they are used extensively in documenting the software architecture of software systems. Behavior diagrams describe the dynamic views what must happen in the system being modeled. Since behavior diagrams illustrate the behavior of a system, they are used extensively to describe the functionality of software systems. Interaction diagrams, a subset of behavior diagrams, emphasize the flow of control and data among the things in the system being modeled.

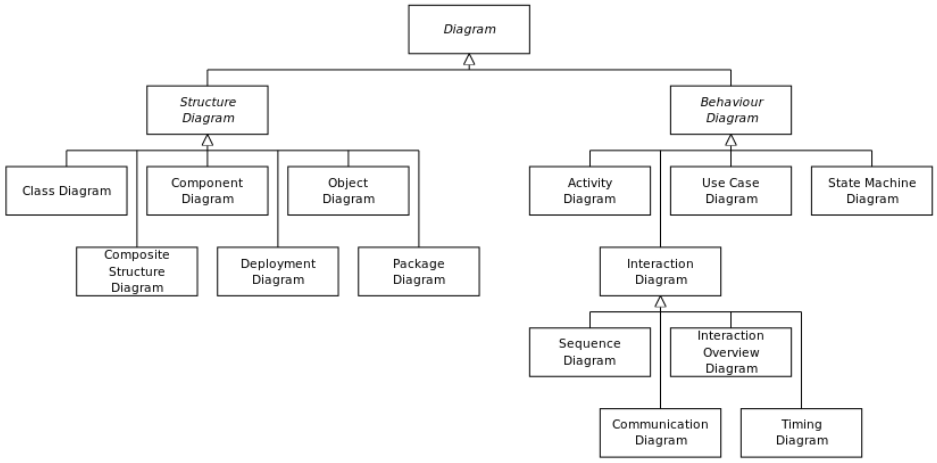


Figure 2.11 UML diagram taxonomy

UML focuses on system modeling in the field of software engineering. With the help of UML, engineer can develop the system by graphic models and have more efficient development process during the whole developing life cycle. For example, when engineering define a class for the system, it becomes an integral part of current design simultaneously. All contents related to the model will be automatically modified when we try to modify the class in the next iteration.

2.3.4.2 SysML

SysML is one of the graphic modeling languages for systems engineering applications, first proposed by Object Management Group (OMG) together with the International Council on Systems Engineering (INCOSE) in 2001 and adopted as a standard in May 2006 [29]. SysML is an extension of the subset of Unified Modeling Language (UML) with nine kinds of diagrams, by which SysML can represent systems and each component, as well as their behaviors and structures.

SysML includes nine diagrams as shown in the diagram taxonomy in Figure 2.12. Hereinto, the package diagram, activity diagram, state machine diagram, sequence diagram and use case diagram are derived from UML. Requirement diagram, block definition diagram, internal block diagram and parametric diagram are newly defined in SysML [80]:

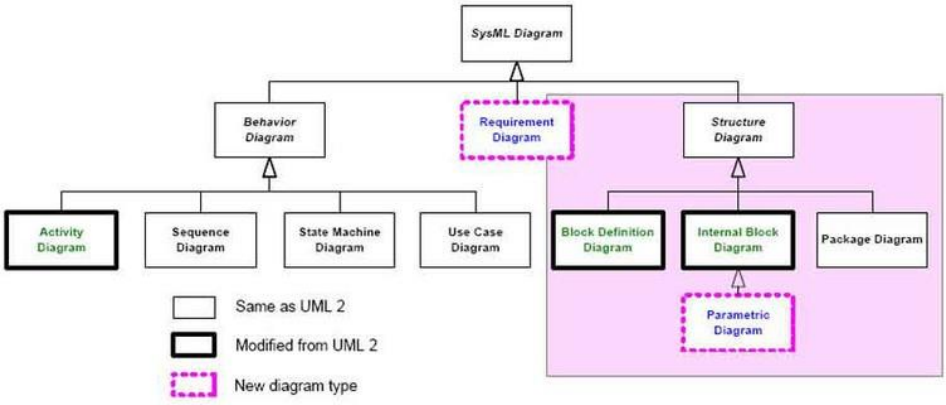


Figure 2.12 SysML diagram taxonomy

■ *Requirement diagram(REQ)* represents text-based requirements and their relationship with other requirements, design elements, and test cases to support requirements traceability;

■ *Block definition diagram (BDD)* represents structural elements called blocks, and their composition and classification;

■ *Internal block diagram (IBD)* represents interconnection and interfaces between the parts of a block;

■ *Parametric diagram (PAR)* represents constraints on property values used to support engineering analysis.

2.3.4.3 Comparison

SysML [109] has the same advantages as UML for system modeling. Both of them provide a standardized and unified modeling language by which system design can be visualized and modeled. Besides, both SysML and UML provide stereotype and profile to extend the applicability. On the other hand, SysML offers systems engineers the following advantages over UML for specifying systems and systems-of-systems:

- UML does not support the needs of engineers designing from the broader systems-based perspective. SysML expresses systems engineering semantics (interpretations of notations) better than UML [110]. It reduces software bias in SysML and adds two new diagram types

for requirements management and performance analysis: requirement diagrams and parametric diagrams, respectively; SysML also includes relationships to other artifacts such as test cases or design blocks, for traceability purposes. For example, the requirement diagram is the primary medium in SysML for conveying traceability among requirements as well as traceability from requirements to structures and behaviors in the system model.

- UML is usually used for modeling object-oriented software system while SysML mainly focuses on system modeling for hardware and software integration. SysML can describe the physic system with block definition diagram and the interconnection between the components with internal block diagram;

- SysML model management constructs support the specification of models, views, and viewpoints that are architecturally aligned with ISO 42010 [111].

Compare with formal models of SCT paradigm, SysML provides a variety of semi-formal models, which can benefit for the AC and extend the modeling scope of SCT. From this point of view, the limitations of SCT can be handled:

(1) Requirement

For MBSE, the requirement analysis is always at the very beginning stage of whole system life cycle. The original stakeholder requirements are usually written by using natural language, occasionally with the help of domain specific models or even informal models not related to any method or language [112]. As indicated in limitation of formalization, the issue within the framework SCT is that the automaton model of plant and specification are always modeled by directly interpretation of inputs. The SysML requirement diagram can be a solution for dealing with specification formalization in SCT. The natural language narrative requirement can be modeled block of requirement, by which the management and organization of requirements can be performed. In [113], Model-driven approach to Requirement Engineering (RE) based on SysML requirements models, which shown that modeling requirements through diagrams can be useful to explicitly represent the various ways that requirements can be related to each other. As a result, the original requirements can be analyzed and decomposed to a series of sub-requirements which can be the traced by formal specifications. The solution is similar to the proposition in [114], in which the particular requirement graph is constructed, while SysML can provide its alternative.

Relationship for requirement analysis and decomposition are defined by SysML. Requirement diagram includes requirements relationships for derivation, satisfaction, verification, refinement, and trace that support a robust capability for relating requirements to one another and to other model elements [80]. Therefore, requirements, block, test case or other elements can be linked, which is of importance in MBSE process. The requirement can be analyzed, decomposed and traced based on these relationships. These mechanisms can be used deal with the formalization

issues in development process of AC. With the help of the relationships, the requirements can be decomposed and therefore traced by the formal specifications.

(2) Structure models

From the perspective of MBSE, the structure definition of target system is indispensable for the global design process. The structure models can be well realized by SysML structure diagram, including BDD and IBD. Compared with formal model, BDD and IBD provide explicit graphic-based representation of system structure, which automaton is not able to achieve. Based on blocks and relationships defined in BDDs, types of physic entities (i.e. system, system component part and hierarchical structure) in control system can be presented. The inner elements of a classifier (parts, ports, and connectors) the physic control connections can be described. The large number of case studies shows that using SysML structure diagrams is a possible solution as a supplement for system structure description [115-117]. On the other hand, the structure of formal models can also be presented. The block in BDD is model element which is not just restricted within the scope of the physic entities, but an extensive representation including concepts, definition, or even dynamic models (e.g. activity). Therefore, the block can be used to represent the formal concepts such as plant, specification or supervisor so that the structure of these formal models can be presented by SysML structure diagram.

A number of model elements of relationship are defined in SysML to visualize the link between models. In BDD, the relationship composition, aggregation, generalization and association are usually used to present the relationship or structure, by which complex or structured system can also be modeled. Considering the AC, the structure of system to be controlled can be well models even if the system is hierarchical, which is out of the scope of the formal models used within SCT paradigm.

(3) Implementation

The controller implementation and deployment models are also be lacked in SCT-based process. As mentioned before, due to the control system is a HW/SW system, the supervisor constitution based on different control architecture doesn’t directly means the structure of concrete controller. To be more specific, engineers are willing to have not only the formal models which represent the control logic of program, but also the connection between physic controller and controlled system. For example, the I/O connection between different components should be described in the model. Based on SysML BDD and IBD, controller implementation and deployment models can be realized. The supervisor architecture can be implemented to concrete controller and programs defined and described by BDD. In addition, the control deployment can be defined in IBD, by which the connection between controller and components of control architecture, even if complex controller such as modular control, can be presented.

2.3.5 State of the Art

In order to further the study of methods by which SysML diagram can be proven to be able to handle the issues of AC, the state-of-the-art contributions of system modeling by SysML are investigated.

In [118], the authors model handling system which consists of a number of components such as gripper, pallet and is controlled by PLC. In this work, BDD is used to present the structure the entire controlled system. An IBD is modeled to detail the control connections between PLC controller and OPC server. The component behaviors under control are modeled by several state machine diagrams. The study presents a demonstration of the implementation aspect of the controlled system while other aspects in the development such as requirement analysis cannot be found.

In [119], the author proposed a SysML-based approach applied to a Flight Control System (FCS) for design and validation. In this work, A BDD is used to model the context of FCS and its interactions with different external elements. Secondly, another BDD is constructed to show the system composition along with an IBD to detail the interactions among the components. For the function aspect, an activity diagram is modeled to show how the different external input flows are progressively transformed to provide the required output flows. Compared with previous contribution, this work provides more engineering perspectives and models them by different SysML diagram. It is shown that the BDD can not only used to present the physic structure, but also the logic structure.

In [32], the authors present a modeling application of Manufacturing Execution System (MES) under PLC control. Similarly, the BDDs are used to describe the context and system structure. Furthermore, three behavior diagrams are used to model the function aspect of system. State machine diagrams are used to describe the behaviors of sub-systems under control and use case diagram depicts the high-level performance analysis. Sequence diagrams are used to show the scenarios of the interaction among components. Stereotypes are used in this work to present the semantics classification of the system and components.

In [89], the HP method is applied to a case study of security system design process. Apart from the SysML models which describe structure and behavior, more models are built for requirement analysis and traceability. A requirement diagram is used in this proposition to present the decomposition of the system requirements. Besides, other requirements are used to show the traceability of stakeholder requirements, system requirements and use cases. Relationships are used to represent the different semantics of traceability such as trace, satisfy, etc.

In fact, large numbers of case studies can be found but unable to be exhausted here as the length

limitation of this study. Whereas, by comparing the referred case studies with SCT-based approach, the following conclusions can be obtained:

(1) The problematic of the limitations of SCT pointed out in Section 2.2.3.3 can be validated. It can be seen in the contributions that the structure models and implementation models are indispensible within the modeling process. Besides, requirement diagrams are usually used in MBSE to present the analysis, decomposition and traceability of informal requirements. All these models are lacked by SCT paradigm.

(2) The contributions provide alternative solutions for dealing with the problematic. Different kinds of SysML diagram are introduced for modeling different aspects of studied system.

To sum up, for reflecting the limitations of SCT put forward in Section 2.2.3.3, Table 2.3 presents a collection of the possible solutions based on discussion in the section.

Table 2.3 A summary of possible solutions

Issue	**System Term**	**SysML Concept**	
		SysML Element	**SysML Diagram**
Limitation 1: Formalization	Mission, Requirement	Requirement	Requirement Diagram (REQ)
Limitation 2: Structure	Concept, Definition, model, structure	Block, Element	Block Definition Diagram(BDD) Internal Block diagram (IBD)
Limitation 3: Implementation	Component, Connection	Block, Part, Port	Block Definition Diagram(BDD) Internal Block diagram (IBD)
Limitation 4: Global Process	Integrated process of MBSE and SCT		

2.3.6 Summary

MBSE is proved to have been widely applied in engineering practice. Application cases also show the modeling capabilities in different domains. Besides, a number of methods, modeling language and standards based on MBSE are developed. In this section, the advantages and drawback of MBSE are analyzed compared with SCT-based modeling process. The interest of integrating SCT and MBSE for AC can be concluded based on the results of discussion. Secondly, SysML is proved to be a solution to extend the modeling scopes of SCT as SysML provides a variety of important model types for AC especially for structures, implementation and requirements, which is lacked by SCT.

2.4 Challenges of Proposed Integrated Approach for AC

The aforementioned discussion shows the interest and possibility to integrate SCT and MBSE for AC. In order to propose a corresponding solution, some local challenges and issues should be faced:

2.4.1 Formal Models and Semi-Formal Models

The first challenge to be faced is the integration of formal models of SCT and semi-formal models of MBSE. The challenge, in fact, involves not only the models themselves, but also a series of methods which bridge the formal models and semi-formal/informal models. Several key points should be taken into account:

(1) Input treatment

The formalization of requirements is always one of the focuses for SCT-based process of AC. Although is the semi-formal models such as SysML diagram can help to describe the analysis and decomposition and modeling of informal specification, an approach to bridge textual-based requirements and formal specifications is needed.

For requirement formalization, a process, called Natural Language Processing (NLP), of automatic translation of requirements expressed in natural language into models was developed by [120-122]. Despite the interest of this work, they are only applicable to requirements that are not written in free natural language. In fact, the processed requirements must be very structured and regular, use precise language as well as a well-defined vocabulary. In [114] [123], authors propose a methodology based on CTL* to refine the natural language description. The refinement process is supported by a formal graph structure called the pseudo-requirement graph or in SysML diagram. The proposed method decomposed the high-level requirement according to CTL* logic so that it bridges the gap between formal specification and informal requirement. Several mechanisms have been implemented in order to ensure the traceability for the refinement process, which is a crucial feature when engineering critical systems. However, it is still not clear that if the approach can be used to link the informal specification to automaton-based specification. Besides, the particular semantics of requirements for the context of AC need also be discussed. The use of template is effectively recommended by the best practice guidelines of requirements engineering [124]. However, the main problem of these templates is the rigidity they impose on the requirements writers, because of the fixed forms to respect. This reduces the expressiveness accordingly. In fact, the specifications of SCT should formalize a variety of requirements which have tremendous difference according the application, even if the systems studied are the same.

Contrary to specification, the solution for plant can be the use of predefined DES units by engineers which lead to a much easier application of supervisory control, as the behaviors of component are usually definite. In [125], the authors describe an approach where the controlled behavior of a discrete-event system is designed using a set of very simple specifications. Each specification is built from a prototype structure, a template, and exercises control over a single aspect of the system, such as the operation of a gripper. All specifications are executed in parallel and thus, simultaneously, provide control for the whole system. However, the proposition is too simple to express more complex requirements and cannot show its universality. In [67] [126], a method for the design of DES control is proposed which allows for the creation of high-level conceptual designs by using encapsulated low-level elements, where typical behaviors for both DES modules and specifications are represented in an abstract way. The parameterizing templates which consist of varying the number of components achieve a rapid modeling and integrate seamlessly with the supervisory control framework. In [127], authors present the modeling and parameterization features supported by Supremica, denoted by *module* which represents a reusable collection of events and associated automata. The propositions above show that the parameterization modeling method can be used to solve the formalization problem of SCT. In short, the effort to reduce the effects of human performance for modeling is another point which should be taken into consideration for engineering practice.

(2) Formal approaches

Another issue is methods to coordinate the SysML models and formal models. Some references focus on the transformation between automaton and SysML behavior diagram such as sequence diagram and activity diagram. In [128] present a technique for formal analysis and verification of UML sequence diagrams. In this approach, a method of transforming sequence diagram to an equivalence automaton is proposed. This approach is applied in [129] to improve the test automation in software development process. By model checked with projection temporal logic, the correctness of automaton model is verified. In [130] author proposes algebraic graph transformation rule for transforming from sequence diagrams to state machines. The specification of the transformation rules is based on the concrete syntax of sequence diagrams and state machines. [131] introduces an approach developed for transformation of simple sequence diagram by defining grammar rules. Formal specification of the procedure is described using Z notation by capturing hidden semantics under the diagrams and therefore the model can be analyzed and validated using Z/Eves tool. [132] presents a formalization using the automata-based technique to detect the deadlock in the activity diagram. The transformation from the activity diagram to the process automata is provided in the study. However, the propositions are general without the taking the context of SCT into consideration. In fact, automaton in SCT has its particular semantic and further study should be done to verify the feasibility. For example, the specification in SCT usually has multi-path loops which may lead to the difficulty the using sequence diagram to handle

this situation.

For further study, we not only focus on method based on SCT, but investigate approaches integrating other formal model in MBSE method. Some contributions focus on the formal approaches based on B. In [96], the authors propose an approach to formalize the UML state chart by B specification. The static model of state chart diagram is built by extracting the meta-class as the static aspects and every meta-class of state chart is formalized as an associated abstract machine. The proposition transforms independent elements of state chart to B and classifies the state chart to simple state diagram, sequential composite state diagram and concurrent composite state diagram. In [97], the methodology of describing the software specification combining SysML with the B method is proposed. The abstract machines of the B method are translated by integrating different SysML diagrams such as BDD, use case diagram and sequence diagram. In [98], the class diagram of the flight control system is presented and each class operation is mapped to a B abstract machine. The B method is adopted to translate the state charts into B specification of flight control software.

Some other contributions focus on the formal approaches based on Petri Nets. [99] introduces a component-based development process to deal with the complexity of the development of embedded software systems. The reuse of colored Petri Nets (CPN) and an associated systematic process promotes a greater confidence in the models reducing the time and errors to develop complex embedded software systems. However, the structure models are not represented by SysML or UML. In [101], the authors propose a method to precisely design communication subsystem and specify its parameters using three kinds of diagrams from the SysML. The simulation model, based on the formalism of TCPNs, is generated automatically from graphical diagrams and used to verify whether assumed parameters of the designed software can be met in a specific environment. In [102], the authors proposed to formalize the behavioral aspect of SysML IBDs by HCPN. Two special characteristics related to these diagrams are focused on. For the block nesting feature a suitable target structure is proposed based on HCPN substitute transitions. The latter integrate appropriate in/out interfaces to handle in an efficient way the nesting.

To sum up, the formal approaches which are used for formal interpretation and verification has been proved to be able to be integrated by MBSE models. However, the state-of-the-art approaches have their limitations. Firstly, no propositions can be found which conform to particular context of SCT paradigm. For example, the link between system structure and formal models are still unclear. Secondly, the existing contributions focus on using formal approaches for computation and verification. Few propositions pay attention to the representation of the formal concepts by MBSE models, which is one of the focuses in this study. The limitations of formal models require the need of using MBSE models to explicitly describe the formal models of SCT such as control architecture and therefore the traceability between formal models and physic models can also be presented.

2.4.2 Modeling Process for AC

From perspective of system engineering level, there are a number of methods in which formal models and MBSE models are combined or formal model are integrated in MBSE method. [41] and [133-135] proposed the similar modeling approach combining SCT and model-based process. The approach integrates SCT methods into the modeling process in order to structure the process of supervisory control synthesis. However, the approaches don't provide the methods for global process of AC for requirement analysis to implementation. In fact, the approaches are just trying to use SCT model to replace the documents during the design process, which cannot be regarded as an integrated modeling process.

Collaborative approach

Some propositions focus on the collaborative approaches for model coordination which is necessary in order to bring together all the heterogeneous established models. In [136] [137], the model federation approaches are proposed. To face the different models involved in the system design, three interoperable spaces are defined: the information space is a collection of databases, files and tools in which information is stored; the conceptual space stores the federation of models implemented from models of the information space; the design space is dedicated to the management of the two previous spaces and the tools necessary for their synchronization. The proposition, however, is towards integration of a number of tools and requires an information system and effective database.

Metamodel approach

Metamodel, or a model of the model, is a special kind of model that specifies the abstract syntax of a modeling language [138]. Model-driven architecture is one of the currently most active branches of MBSE proposed by OMG (shown in Figure 2.13). The metamodel can be regarded as high-level abstract and a model always conforms to a unique metamodel. In fact, the MBSE modeling language such as UML and SysML are typical metamodels proposed by OMG. Metamodel approach is usually used in systems engineering for the model analysis and construction and model transformation.

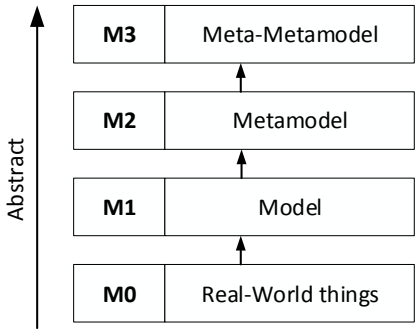


Figure 2.13 Model-driven architecture

In [139], a synchronization approach based on metamodel is proposed to map viewpoints and models in two different disciplines. This proposition focuses in particular on the interactions of the construction activities between system architecture design and dependability analysis. The needs of synchronization are analyzed and then these synchronizations are implemented by means of meta-model. In this proposition, the process in each discipline has its proper models.

In [140], the authors propose a metamodel based approach to integrate models by Discrete EVent systems Specification (DEVS), Object-Z and UML to construct a new model kind to represent complex system. In this study, both syntax and semantics of models integrated for proposed model kind and therefore the advantages of the three models can be made use of simultaneously. However, one limitation of this proposition is that the standardized models are modified which is more applicable at academic level but may not be able to have strong usability for engineering practice.

ATL is a frequently-used model translation language designating the meta-models and specific grammar, by which different model can be transformed based on metamodel which they conform to. For example, in [141], the authors propose a method based on meta-model for the translation from UML into Event-B to realize formal verification. In the work, use case diagram, activity diagram and collaboration diagram are interpreted by Event-B.

In fact, when proposing a novel approach by using MBSE modeling language, the metamodel will be usually taken into account. However, the existing propositions referred above have common drawbacks when considering their feasibility for AC: The particular context of AC is not detailed in these propositions as they are too general. Although the original SysML/UML diagrams has their rigorous metamodel, the diagrams which describe AC should be tailored to present particular semantics rather than using the original ones.

Viewpoint-based approach

The viewpoint-based approach catches wide attention and some useful relevant works can be found in some contributions. [142] outlines a viewpoint-oriented systems engineering (VOSE)

framework which supports the use of multiple perspectives in system development, which uses "viewpoints" to partition the system specification, the development method and the formal representations used to express the system specifications. It shows that the framework can handle the multiple perspectives problem. In [143], proposes a generic method for defining viewpoints in SysML by generalizing the concepts used in defining viewpoints in Reference Model of Open Distributed Processing (RM-ODP). By using this method, viewpoints to be used in different problem domains can be defined in a systematic way, and reuse and/or sharing of viewpoints across different problem domains can also be facilitated. In [86], authors describe a technique developed at JPL of applying SysML viewpoints and views in order to generate documents and reports by XML. The technique shows that a variety of views that can be generated, and provide some insight into how collaboration and integration is enabled. In [144], authors introduced an application of viewpoints to be able to deal with the evolving scope and requirements on mechatronic products, by which stakeholder and models of development can be integrated. These contributions show that the viewpoint-based approaches can provide good solutions for the system development where we need different perspectives and concerns. The architecture elements can be well organized under viewpoints. Therefore, taking this as the breakthrough point, we study on the possibility of redefining the modeling architecture for controller development. For example, to propose a SysML diagram for the controlled design, the focus and concern can be specified based on the corresponding viewpoint.

2.4.3 Integrating concepts

In the context of AC, the semantic of models have to be taken into consideration. The concepts and elements between SysML and SCT should be linked. For example, the following issues must be deal with: (1) what is the link between SysML models and plant, specification and supervisor models in SCT? (2) What the controllable event and uncontrollable event can be represented by SysML elements? (3) How we can visualize the concepts of synchronization, formalization and supervisor synthesis? These issues are important for to bridge the gap between formalization and engineering whereas the answers are far from clarity in the state-of-the-art contributions. Secondly, the standardized MBSE processes guidance for the models to be built. However, the process should be tailored to conform to the particular context of AC. Besides, the model elements in each diagram should also be specified. As discussed before, it is necessary to clarify the semantic of the diagram rather than using the original metamodel for the diagram provided by SysML, which just specifies the syntax. Due to the target of high reusability, the metamodel of each proposed SysML diagram should specify the necessary model elements and relationships between these elements. For example, when we propose a BDD to describe the control architecture, the models for supervisors, corresponding plants and specifications and their relationships must be detailed by metamodel. On the other hand, the modeling method for each model in the framework should be

defined, which is not be provided by the standardized MBSE process. Usually, the methods can be, but not limited to, presented by metamodel. In SCT, the supervisor synthesis can be regarded as methods for modeling supervisor. However, when SCT is extended by SysML models, no state-of-the-art modeling method can be found for this context. Therefore, it is another challenge which should be faced in this study.

2.5 Conclusion

In this chapter, the SCT paradigm for AC is reviewed. Firstly, the concepts, models and methods are recalled by covering following aspects: the formal model, the supervisor synthesis method, control architecture, implementation methods. Based on the investigation of typical SCT-based modeling process, four limitations of SCT are identified which lead to the gap between SCT paradigm and engineering practice: (1) Limitation 1: lack of interpretation of the link between informal requirements and formal specifications, and between free behaviors of physic system and formal plants in typical development process; (2) Limitation 2: lack of structure models (3) Limitation 3: lack of implementation models, and (4) Limitation 4: lack of global framework to standardize the modeling process of AC for engineering practice.

MBSE is introduced for studying the solutions to deal with the issues of SCT from the perspective of engineering. MBSE standards, modeling methods and modeling language are reviewed. The investigation and its conclusions show that MBSE methods are able to make up for the shortcomings of SCT and solution can be proposed based on integration of SCT and MBSE. For the purpose of narrowing the gap between SCT and engineering practice, the solutions for integration SCT paradigm and MBSE are further studied. Several challenges have to be faced of integration of SCT and MBSE are put forward, which shows the necessity of proposing a novel modeling framework in this study.

Chapter 3

Proposed Modeling Framework

3.1 Introduction

According to the aforementioned analysis, the modeling scope of formal models and lack of global modeling process within the SCT paradigm leads to the application difficulty of this theory in engineering practice for AC and four limitations of SCT paradigm are concluded from the perspective of engineering context:

(1) Limitation 1: There is a lack of interpretation of the link between informal requirements and formal specifications, and between free behaviors of physic system and formal plants in typical development process.

(2) Limitation 2: It has been proven that the structure models are indispensable for engineering modeling process, which are also lacked by SCT paradigm. The structure models include not only the structure of physic system but also formal/logic concepts.

(3) Limitation 3: The models which represent the implementation of the concrete controlled are lacked by SCT. The existing supervisory control architecture and implementation methods are not able to specify the link between the models of supervisor/controller and models of concrete controller, which is unacceptable in the engineering context.

(4) Limitation 4: there is no global framework to standardize the modeling process of AC for engineering practice. The necessary models, the modeling activities along with the traceability between models are still unclear.

The investigation on MBSE shows that it is a feasible and prospective solution to deal with the limitations of SCT, as MBSE provides a global modeling process and corresponding modeling languages. SysML is proved to be an excellent alternative modeling language, which can be used to extend the modeling scope for SCT-based AC development process. However, the further

comparison of SCT and MBSE shows that there are still several challenges to be faced to integrate both for proposing a novel framework for AC.

Based on the discussion above, a novel modeling framework is proposed in this section. To realize the objectives for bridging the gap between SCT and engineering practice and dealing with the limitations of SCT, as well as facing the challenges of integrating SCT and MBSE, the responses to the following questions should be covered in the proposition: (1) What kinds of models should be defined in the proposed framework? (2) What's the semantic of elements to be defined in the models particular for the context of AC? (3) How can SCT models and methods be integrated in the MBSE process? (4) What is the global MBSE modeling process for AC?

In this chapter, the solutions, which respond to the questions above along with the limitations of SCT, will be presented. In Section 3.2, the solution of the architecture which integrated SCT and MBSE will be discussed aimed at issues raised within the SCT paradigm. In Section 3.3, the global modeling process of the proposed framework for AC will be introduced in detail, including the sub-processes, models, activities, methods and input/output of each step.

3.2 Integrated Architecture

3.2.1 Objective

SysML provides nine model types and SCT is usually modeled in a unified way by automaton. To integrate models from both frameworks (shown in Figure 3.1) to form an appropriate combination of a variety of models, two points should be should be considered:

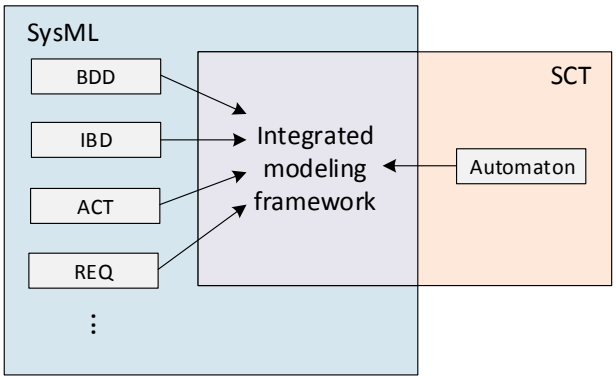


Figure 3.1 Model integration

(1) Model scope: the nature of model is representing a perspective for system. For an MBSE project, the team must determine the modeling scope, which means the following questions should be answered [145]: which parts of system should be modeled? Which behaviors should be

modeled? Which details are necessary in the models and which are neglected? Therefore, the models particular for the proposed framework of SCT-based AC must be determined.

(2) Model type: each model is governed by a model type which establishes the notations, conventions and rules for models of that type. As a concrete carrier of model, it is necessary to designate appropriate a model type for the corresponding model. The selection of the nine model types provided by SysML is a challenge.

The solution to respond to both aforementioned challenges is to perform a model necessity analysis based on MBSE viewpoint, by which the necessary model can be determined particular for integrating SCT in MBSE for AC.

3.2.2 Architecture: Viewpoint, View and Model

Standard ISO 42010 [85] gives the definition of view, viewpoint and model:

Definition 3.1 (Architecture, ISO 42010 [85])): fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution.

Definition 3.2 (Architecture viewpoint, ISO 42010 [85]): work product establishing the conventions for the construction, interpretation and use of architecture views to frame specific system concerns.

Definition 3.3 (Architecture view, ISO 42010 [85]): work product expressing the architecture of a system from the perspective of specific system concerns.

Definition 3.4 (Architecture model, ISO 42010 [85]): An architecture view is composed of one or more architecture models. An architecture model uses modeling conventions appropriate to the concerns to be addressed. These conventions are specified by the model kind governing that model.

Definition 3.5 (Concern, ISO 42010 [85]): interest in a system relevant to one or more of its stakeholders. A concern pertains to any influence on a system in its environment, including developmental, technological, business, operational, organizational, political, economic, legal, regulatory, ecological and social influences.

The concept of architecture description defined by the standard ISO 42010 is shown in Figure 3.2. Viewpoints provide a framework for capturing reusable architectural knowledge that can be used to guide the creation of a particular type of partial architectural elements. A view of a system is a representation of the system from the perspective of a viewpoint. In other word, a view expresses

what is seen from the position of viewpoint. However, view is an abstract concept and need concrete representation. Views are realized by models of any sort or type such as SysML models, ontologies, structured data from a database, etc.

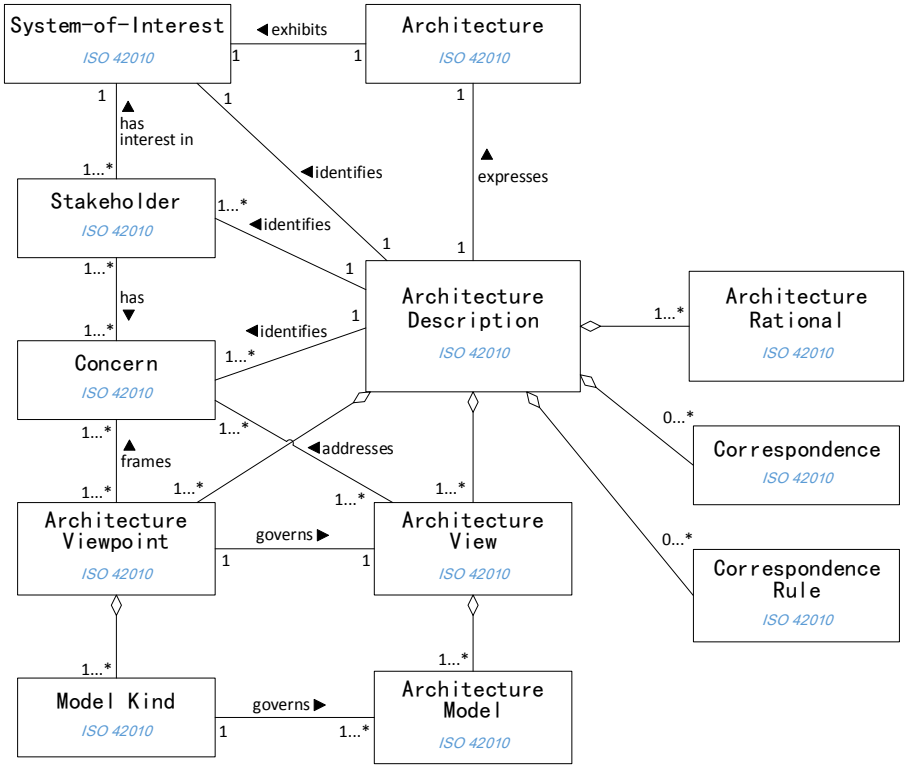


Figure 3.2 Concept Model of ISO 42010 [85]

3.2.3 Viewpoints for AC

The concept of viewpoint helps to analyze the needs of models to describe the architecture of the system to be studied. In the typical SCT-based modeling approach, three kinds of formal models are built (plants, specifications and supervisors) as we just concern about the behaviors of studied system. By taking into consideration of the concerns corresponding to the limitations (Limitation 1-3), eight novel viewpoints and are proposed (Table 3.1). The concerns of limitations 1 is represented by two corresponding viewpoints which focus on the formalization and traceability of requirements. In fact, the concern on formalization of plants is partially related to the structure of physic system and therefore we classify the concern and corresponding viewpoint for limitation 2. Apart from Viewpoint: Plant Structure, three other viewpoints whose concerns focus on the structures of formal/logic concepts of supervisor and controller are proposed for limitation 2. Furthermore, two aspects of concerns on implementation are detailed by two viewpoints which focus on the implementation and connection of the final controlled system.

Table 3.1 Proposed viewpoints and concerns for AC

Solution for	Viewpoint	Concern
Limitation 1: formalization	**Viewpoint: Informal Requirement**	*The input needs in form of informal description should be should be analyzed and decomposed for the traceability by formal specification.*
	Viewpoint: Requirement Traceability	*The traceability between informal requirements and formal specification and plant should be explicitly presented.*
Limitation 2: Structure	**Viewpoint: Plant Structure**	*The link between plants and physic components in the uncontrolled system should be explicitly presented to show the identification of plants*
	Viewpoint: Control Architecture	*The control architecture should be defined and presented explicitly.*
	Viewpoint: Architecture Structure	*The details of control architecture should be specified including structure and link of formal elements*
	Viewpoint: Control Strategy	*The structure of controller should be presented.*
Limitation 3: Implementation	**Viewpoint: Controller Implementation**	*The concrete controller structure should be defined and clarified, including hardware and software.*
	Viewpoint: Control Connection	*The viewpoint is on the controlled system. It is necessary to show how connection of component, sub-system and controller based on the output of AC.*

According to ISO 42010, models should be defined for reflecting each viewpoint. Figure 3.3 show the Viewpoints and models for the architecture description of AC by extending the modeling scope of SCT. Three basic viewpoints and corresponding models, which have already defined in the typical modeling process, are plant, specification and supervisor respectively. In fact, there are three viewpoints which are concerned about within SCT paradigm but no models can be used to reflect them: Viewpoint Control Architecture, Viewpoint Control Strategy and Viewpoint Controller Implementation. In the Figure 3.3, the viewpoints and models in white block are the elements not defined within the paradigm of SCT. The new viewpoints are proposed aimed at solving the corresponding SCT limitations. It should be pointed out that the model type (marked by "??" in the figure) related to the proposed viewpoints are still undetermined as automaton cannot be used to present the models should be realized by appropriate SysML diagrams.

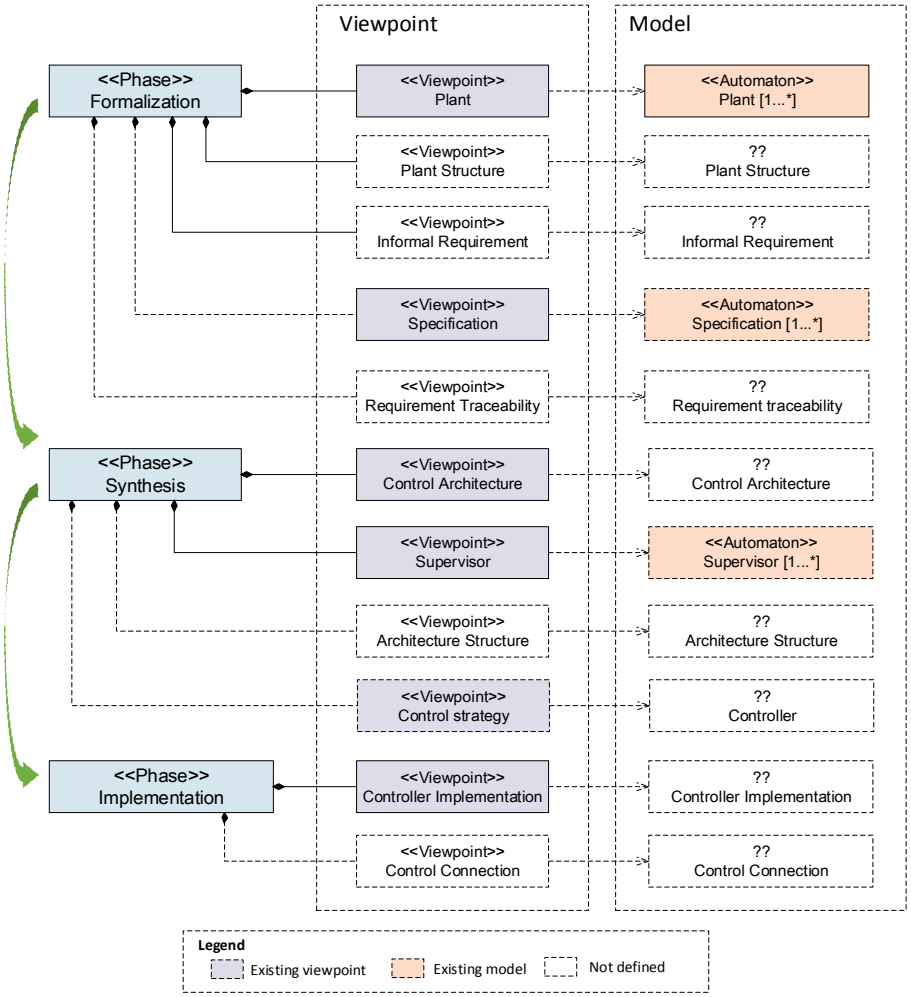


Figure 3.3 Viewpoints and models for the adapted SCT-based architecture

3.2.4 Integration in MBSE Process

The typical MBSE process should also be adapted particularly for the context of AC. Compared with the typical MBSE process, some special differences should be taken into account:

(1) The input needs of typical MBSE process are usually the stakeholder requirements. However, for AC, the system to be controlled should be given from the stakeholders, as the objective is to develop a controller based on the given system. In fact, the uncontrolled system, which should be formalized as plants, can be regarded as a part of input needs which is in form of concrete system rather than narrative requirements. On the other hand, the given uncontrolled system is not an external system or actors as the whole controlled system is the final output of the modeling process.

(2) The system function requirements in general MBSE process describe the system-level behavior while the models in the context of AC can be decomposed to the components level. Secondly, as the uncontrolled system is given as input, the behavior of each component in the system can be

determined. In fact, the step is equivalent to the modeling of plant in formal process. Therefore, the formal model can be directly used for represent the behavior of physic components.

Focusing on the input processing of uncontrolled system, two complementary viewpoints and concerns should be taken into account (shown in Table 3.2). The complementary viewpoints focus on the aspects of physic system analysis. As indicated above, the uncontrolled system and the context should be a part of the inputs of the design process, which is important prerequisite for plant formalization.

Table 3.2 Complementary viewpoints and concerns for MBSE

Viewpoint	**Concern**
Viewpoint: Context	*The global environment of system-of-interest for AC should be identified.*
Viewpoint: System Structure	*The uncontrolled physic system and components should be analyzed to specify the structure of the system-of-interest.*

According to the review in Chapter 2, the typical MBSE process can be divided into three phase: requirement analysis, function analysis and design synthesis. To integrate all the viewpoints and models for AC in MBSE process, we classify the viewpoints into three categories accordingly, which are presented in Figure 3.4. All the viewpoints which focus on the analyzing inputs are allocated in the requirement analysis phase, which are Viewpoint Context, Viewpoint System Structure and Viewpoint Informal Requirement. In this stage, the main purpose is to process and extract the necessary information from the inputs for formalization of both plant and specification. Secondly, the viewpoints related to formalization are allocated to the function analysis as they concentrate on the modeling of behavior, which conform to the origin target of this phase in MBSE. Besides, the link and trace between formal models and inputs are constructed. Four viewpoints are allocated into phase: Viewpoint Plant Structure, Viewpoint Plant, Viewpoint Specification and Viewpoint Specification verification. For the design synthesis phase, the rest of viewpoints are allocated in it. In this phase, the main objective is to perform supervisor/controller synthesis and implementation based on the given plants and specifications provided by Phase 1 and Phase 2. Compared with typical MBSE process, there are several differences in the proposed framework:

(1) As the global inputs of the modeling process are not only stakeholders ‘requirements but also the uncontrolled system, system structures should be analyzed in the requirement phase.

(2) The input is semi-black-box as the uncontrolled system is given and therefore, the behavior of each component in the system can be identified. Usually, in the typical MBSE process, the constitution of the system to be studied is determined in the design synthesis phase. The behavior

of system is identified by case as the system is black-box at the beginning of the modeling process.

(3) As formal approach are integrated in the modeling process, the SCT supervisor synthesis and property verification method are used as the architectural analysis instead of evaluation approach.

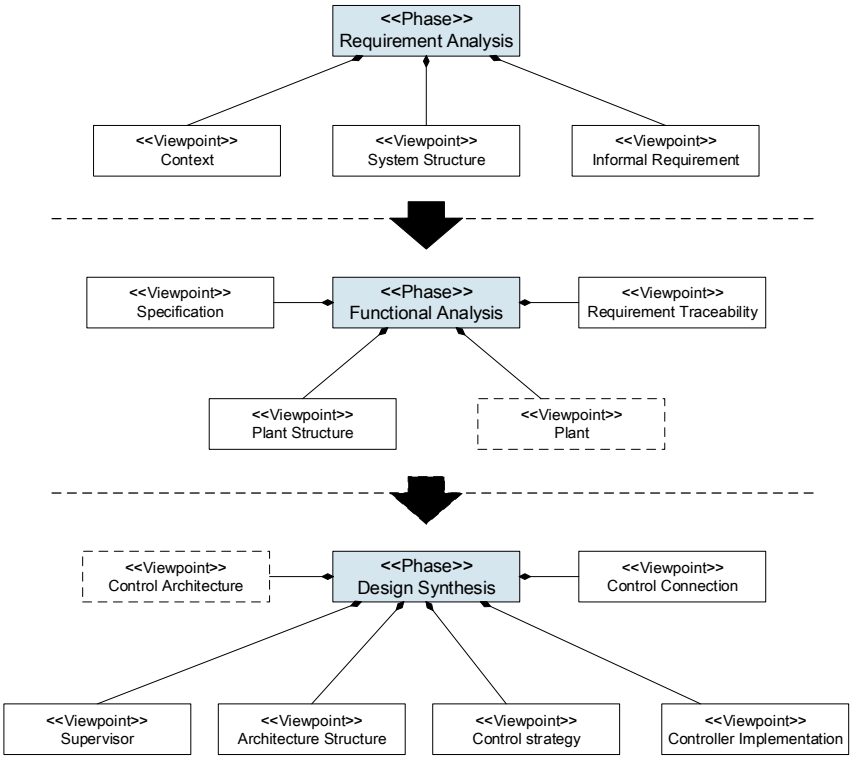


Figure 3.4 Viewpoints allocated to the MBSE process

3.2.5 Model Definitions

Model should be defined to realize the view which is governed by the corresponding viewpoint (see Figure 3.2). The formal models of plant, specification and supervisor can be directly be used as behavior models instead of SysML behavior diagram (sequence diagram, activity diagram and state machine) as the formal models are rigorous in both syntax and semantics. On the other hand, SysML diagrams must be used to realize other kinds of models such as structure, requirement and traceability. Based on the discussion on the model type in Table 2.3 (p. 40), models for all the viewpoints can be determined. Apart from the formal models, 10 SysML diagrams are defined to be allocated to the viewpoints within the proposed framework. To compared with the formal models in typical modeling process, the models can logically be grouped by three divisions centered by plant (in red), specification (in black) and supervisor (in blue) respectively.

The models of specifications are extended by two requirement diagrams which reflect the Viewpoints Informal Requirement and Viewpoint Requirement traceability. The responsibilities of

the diagrams are defined as follows:

Model 1: REQ: System Requirement

In the requirement diagram System Requirement, the informal narrative description can be analyzed and decomposed explicitly by SysML requirements and therefore, ambiguous high-level requirements can be specified their descriptions so that specification can well refine the internal nature. Secondly, the structured requirements in the diagram make it possible for traceability by formal specifications. Therefore, the diagram is of vital important for the link between the stakeholders' need and specification.

Model 2: REQ: Requirement Traceability

When the traceability is verified, the link between formal specification and requirements along with plants can be presented explicitly by the requirement diagram Requirement Traceability. There are two objectives of the link: (1) the presentation the traceability between requirements and formal models and (2) the prerequisite for supervisor synthesis.

The models of plants are extended by three BDDs whose responsibilities are defined as follows:

Model 3: BDD: Global Environment

By respecting to Viewpoint Context, the BDD is built to specify the global environment of the system-of-interest. Usually the targeted system interacts with the external system or participants, the system boundary must be identified in the diagram. For example, in [37], an inlet conveyor is formalized by plants. Therefore, the boundary to system to be studied is of importance for plant identification.

Model 4: BDD: System Structure

At system-of-interest level, a BDD to present decomposition of the system structure must be built. The plant candidates are identified and explicitly linked to the conceptual controller (i.e. controller to be developed). In fact, even if within the scope of system-of-interest, not all the physic components should be formalized by plants (e.g. the buffer in some cases), which is important for the following plant formalization.

Model 5: BDD: Plant Structure

For the plant candidates of physic components, the final plants should be determined. In this diagram, the plant is defined by a block and will be specified in the following steps. As fact that there may not be a one-to-one relationship between physic component and plant, a BDD is able to explicitly represent the relationships. The construction of the diagram is based on the identification

results in Model 4 and is the foundation of formalization of plants.

The models of supervisors are extended by three BDDs and two IBDs:

Model 6: BDD: Control Architecture

To reflect to the Viewpoint control architecture, a corresponding BDD must be built. When the architecture is determined, the structure of the involved global/local plants, specification and global/local plants should be defined in the diagram. In this diagram, the semantic of two formal concepts should be presented: the synchronizations of local plants and supervisor synthesis.

Model 7: IBD: Architecture Structure

The purpose of the IBD is to detail the defined control architecture in Model 6, in which the plants and supervisors are linked. The idea of the diagram is inspired by the proposition in [20], in which a diagram is built to link the supervisors and plants by the shared events. SysML IBD is an alternative to standardize the representation instead of the proposed diagrams. In the IBD, it is necessary to explicitly to present how the global/local plants and global/local supervisors can be linked by formal events. The semantic of port must be defined for this purpose.

Model 8: BDD: Control Strategy

As discussed in Chapter 2, the supervisor cannot be directly transformed to implantation. The BDD Control Strategy, which reflects to the corresponding viewpoint, specifies the definition of controller. The construction of the BDD is based on the supervisors and the strategies to be taken. Based on the diagram, an explicit structure of controller can be described and provides the necessary models for implementation (Model 9).

Model 9: BDD: Control Implementation

With regard to the Viewpoint Control Implementation, a BDD is built to present the implementation architecture. The diagram focuses on the definition of concrete controller at physic level, by transforming controller according to the corresponding implementation methods. Two physic elements should be designated: hardware and software, and the link between hardware and software should be present explicitly.

Model 10: IBD: Control Connection

To reflect to the Viewpoint Control Connection, the IBD presents the link details between concrete controller and physic uncontrolled systems. Physic ports for command and signal care defined and control connection is presented in the diagram. Compared with model 7, this diagram focuses on the physic level rather than the formal level. In fact, the difference between supervisor and

concrete controller can be explicitly discovered by comparison of these two models.

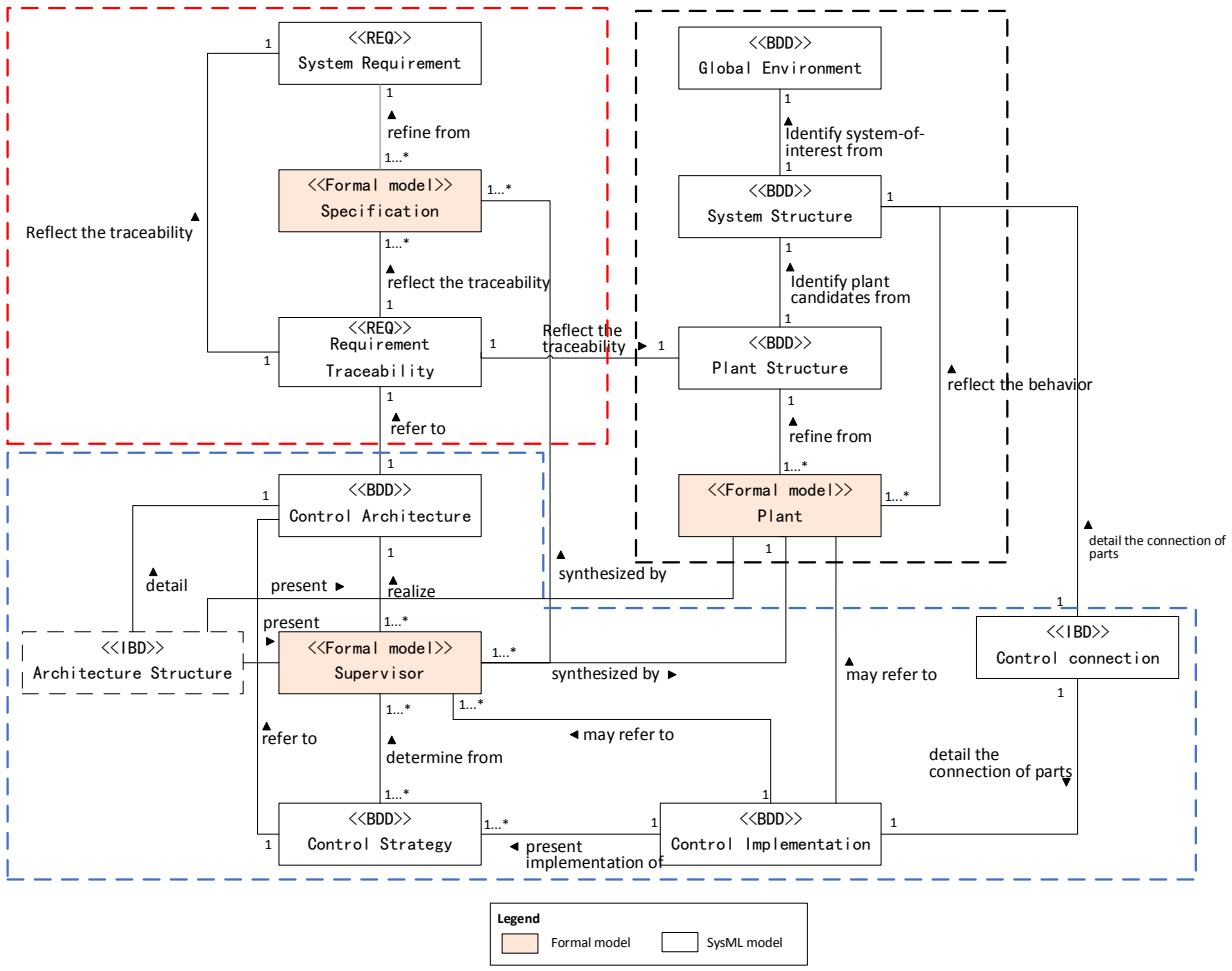


Figure 3.5 Definition and links of models

The relationships of models are presented in Figure 3.5. The modeling of plants depends on three BDDs which represent the system structure. The BDDs describe the uncontrolled system from context level to system level and component level. The decomposition of the system and components of interest are identified, based on which the link between the behavior models and structure models and internal link among physic system can be traced. Two requirements diagrams are modeled for the purpose of link formal specification and plant. The formalization of specification is based on the REQ: System Requirements which detail the analysis and relationship of the narrative requirements from stakeholders. Therefore, the requirements can be traced by formal specifications. Another requirement diagram explicitly presents the traceability between requirement and formal models including both plants and specifications. As formal approach for supervisor synthesis requires corresponding plants and specifications, the requirement diagram also provide an explicit relationship of formal models which is indispensable for control architecture. Focusing on supervisor synthesis and controller implementation, BDDs and IBDs are modeled to present the AC at formal/logic/physic level respectively. For the model of each level,

the models conform to the particular semantics. Besides, the traceability between formal models or formal models to other models is presented in the diagram.

3.2.6 Summary

In this section, the architecture of integrated framework based on the viewpoints for AC is preliminarily analyzed. With regard to the limitation of modeling scope of formal models (limitation 1-3(see Section 2.2.3.3)), 13 viewpoints are identified and allocated into the phases in MBSE modeling process. For each viewpoint without defined by formal model, a SysML diagram is defined. As a result, 10 SysML diagrams are proposed. However, the SysML models are just preliminarily defined and the details and particular semantics of each SysML model are still unclear in this section. Secondly, as the limitation 4 (see Section 2.2.3.3) requires a global modeling process, it is still necessary to integrate all the formal models, SysML models and modeling methods into a proposed modeling process.

3.3 SysML Profile for AC

3.3.1 Objective

Taking into consideration of the modeling context for AC, the proposed models must be defined based on the particular characteristics and semantics. For example, as part of the SysML diagrams involve the presentation of formal models (e.g. Model2, Model 5, Model 6, Model 7 and Model 8), one problem should be dealt with that how SysML models can be defined to represent the formal models. Secondly, modeling elements which represent the physic aspect or formal aspect of the studied system should also be clearly identified. For this purpose, the classification of the model elements for AC should be explicitly presented in the diagram. Thirdly, the formal concepts such as supervisor synthesis, plant formalization and synchronization should be described by SysML modeling elements in the diagram.

Fortunately, SysML supports a generic extension mechanism for customizing SysML models for particular domains (e.g., aerospace, healthcare, financial) and platforms (J2EE, .NET). By defining SysML profile particular for AC, the standard semantics of SysML model elements can be extended and enabled to be more accurate, which is important for the proposed SysML diagrams. SysML profiles are usually defined using stereotypes, tag definitions, and constraints which are applied to specific model elements, like classes, attributes, operations, and activities. Therefore, in this proposition, the model elements conforming to the particular context of AC are proposed in order to attach specialized semantics for the models in the SysML diagram.

3.3.2 Block Extension

In SysML, the block is extended by default from concept of class in UML. As several BDDs are defined in the proposed framework, which involve models and concepts of both physic and formal, the standard block are extended by generalization to six stereotypes which must be identified for AC (shown in Figure 3.6).

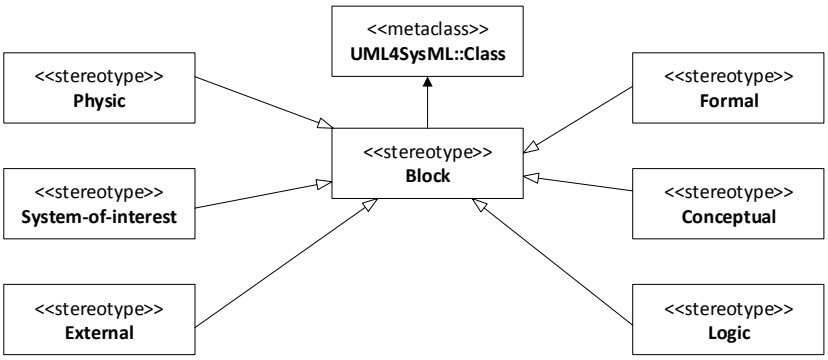


Figure 3.6 Extension of block

<<Physic>>: the block stereotyped by <<Physic>> indicates that the model is identified as physic system or component. For example, the component to be controlled or the concrete controller should be stereotyped by <<Physic>>.

<<System-of-interest>>: the block stereotyped by <<System-of-interest >> is model of the system to be centralized for AC, which consists of the uncontrolled system and controller. Usually, only one block should be stereotyped by << System-of-interest >> as we concentrate on one system to be studied at a time.

<<External>>: the block stereotyped by <<External>> represents the system which is out of the boundary of system-of-interest but within the scope of the global context. The << External>> block can be either physic systems or human participants.

<<Conceptual>>: the block stereotyped by <<Conceptual>> is uniquely used to represents the controller to be developed. At the very beginning stage of the global modeling process, the controller is still undetermined. Therefore, in order to present the controller to be developed in the diagram, the block of controller is labeled by this stereotype.

<<Formal>>: the block stereotyped by <<Formal>> is used to represent the formal models in the diagram. Each formal model is represented by a SysML block stereotyped by <<Formal>> so as to be distinguished from block for physic system.

<<Logic>>: the controller at logic level is represented by the block stereotyped by <<Logic>>.

Therefore, the supervisor/controller/concrete controller can be directly identified by the corresponding stereotype.

In view of controller are usually composed of hardware and software, the stereotype <<Physic>> is further generalized by <<Hardware>> and <<Software>> (shown in Figure 3.7). The block which is stereotyped by <<Hardware>> represents designated device of the concrete controller such as PLC, microcontroller and so forth. On the other hand, the block which is stereotyped by <<Software>> represents the control programs executed by the designated concrete controller.

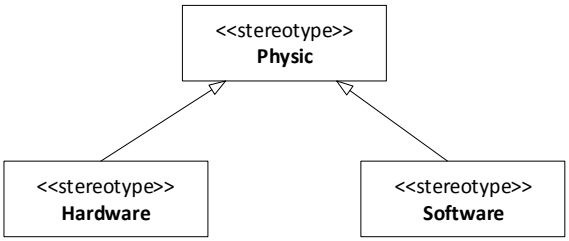


Figure 3.7 Extension of stereotype <<Physic>>

3.3.3 Block for Formal Model

As formal model is out of the scope of SysML, it is difficult to find a corresponding modeling element in SysML. Therefore, it is necessary to define the blocks which represent the corresponding formal models. Stereotyped by <<Formal>>, the SysML block can preliminarily extend its semantics to represent the formal models (plant, specification or supervisor). However, two points should be taken into account to detail the block of formal model, which meet the needs of proposed SysML diagrams:

(1) To present the link between physic systems and plants in the diagram BDD: Plant Structure, the formal events should be explicitly presented in the block of formal models.

(2) In the diagram IBD: Architecture Structure, we intend to link between plants and supervisors, which can be constructed through formal events.

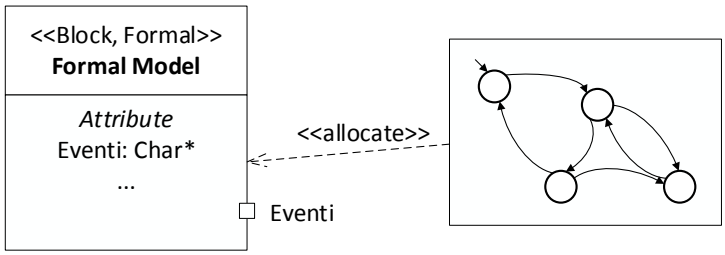


Figure 3.8 Block stereotyped by <<Formal>>

By integrating the two considerations above, the block for formal models can be defined as shown

in Figure 3.8. For each block of formal model, a corresponding automaton is allocated to the block. In fact, the SysML blocks can be regarded as front end of the formal models. All the formal events of the automaton are listed in the attribute of the block. Furthermore, a port is defined for each formal event by using the same name for simplicity. It should be pointed out that although the defined unified block is applied for all kinds of formal models, it is not necessary to display all elements of the blocks in each diagram.

3.3.4 Relationship Extension

There are three main formal concepts related to activities are within the paradigm of SCT: formalize, synchronize and synthesize. In order to present the corresponding semantics in SysML, relationship of dependency and composition can be an alternative solution. A dependency relationship is a directed relationship in which one element, the client, uses or depends on another element, the supplier [109]. The dependency relationship is extended by default in SysML to different kinds of relationship stereotypes such as *create*, *trace*, *refine*, *derive*, *use* and so on. However, the default dependency extensions are not able to explicitly represent the semantics of formalization and synthesis. Therefore, two new extensions of dependency, which are related to these two concepts, are proposed in this SysML profile for AC (shown in Figure 3.9).

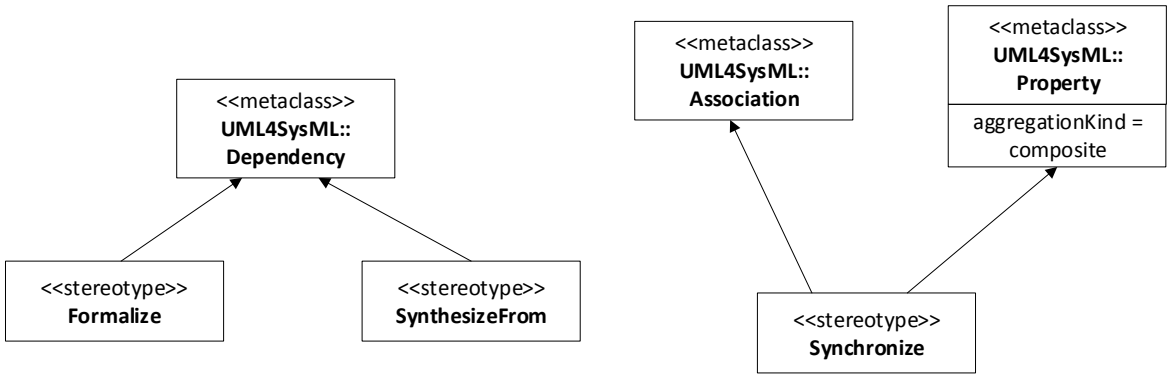


Figure 3.9 Relationship extension

<<Formalize>>: As formal plant and specification are two models which should be formalized from the given input, two situations can make the use of <<Formalize>> relationship in the SysML diagrams. Firstly, the <<Formalize>> can be used to link the blocks of physic components and block of plants to present the semantics of formalization. Furthermore, the link between requirements and blocks of specifications can also linked by the relationship.

<<SynthesizeFrom>>: The computation results of supervisor synthesis depend on the models of plants and specifications and therefore the relationship dependency can be extended to link the blocks of formal models. The necessity of using relationship <<SynthesizeFrom>> is that Model 6 has to describe the control architecture in which the semantics of supervisor synthesis based on the

plants, specification and supervisors should be explicitly presented in the diagram. To this end, the relationship <<SynthesizeFrom>> provides a direct connection from blocks of supervisor to the corresponding plants and specifications.

To explicit represent the semantics of concept plant synchronization in the diagram, the relationship composition is extended. The relationship composition in SysML is defined by a combination of <<Association>> and <<Property>> with aggregationKind = composite (shown in Figure 3.9). Therefore, the stereotype of <<Synchronize>> should be extended from both <<Association>> and <<Property>>.

<<Synchronize>>: The semantics of this stereotype is the presentation of synchronizing local plants to global plants. As the relationship is extended from composition, the local plants therefore are parts of global plants. In this way, the hierarchical structure of plants can be explicitly described in the diagram IBD: Architecture Structure, in which the local plants can be directly put inside the global plants.

3.3.5 Summary

In this section, a SysML profile for AC is defined in order to prepare a framework in which the semantics of formal models and physic models can be identified explicitly and the formal concepts can be describe by SysML modeling elements. The profile is important for the proposed modeling process by which the SysML diagrams can be well defined in the following section.

3.4 Global Modeling Process

3.4.1 Overview

A global modeling process is proposed to integrate the adapted architecture for AC (shown in Figure 3.10), including formal models, proposed SysML models and a series of modeling, synthesis and verification methods. By taking into consideration of the engineering practice of AC, the global inputs and outputs in this study are assumed as follows:

Global input:

(1) Stakeholders' need: in MBSE, the Stakeholders' needs usually involve a large number of requirements at different aspects including functional requirement and nonfunctional requirements. In order to simplify the requirements and focus on the context of AC, this input is assumed to only specify the needs for control specifications.

(2) Context: the context is assumed to be the global environment of the system to be studied including external systems and participants.

(3) Uncontrolled system: the structure and physic components of uncontrolled system must be provided in details including the functions and operations.

Global output:

(1) Concrete controller: The concrete controller should be designated by one kind of controller (e.g. PLC, microcontroller…). In this study, the BDD: Control Implementation is used to represent this global output, which is advocated by MBSE.

(2) Control program: the control program must be generated by the appropriate implementation method and be able to execute on the defined concrete controller.

(3) Controlled system: the output requires a whole system including controller and controlled system. Similar, it is assumed to be presented by the model IBD: Control Connection.

Activities:

15 activities are defined in the global process. For each model defined in the previous section, an activity is defined for modeling (13 activities in total including activities **AR1.1**, **AR2.1**, **AR2.2**, **AF1.1**, **AF2.1**, **AF2.2**, **AF3** and **AD1-AD6**). Besides, two additional activities are defined for the purpose for pre-processing stakeholder's needs (Activity **AR1.1**) and in charge of consistency verification between requirements and formal specifications (Activity **AF1.2**) respectively.

As the proposed process involves the formal approaches, which is supposed to be commanded the engineers, a hypothesis is proposed in this study:

Hypothesis 3.1 (Expertise): At least one engineer in the development team has a good command of formal approaches and SCT paradigm. There is no difficulty to communicate between members of the team in order to reach a consensus during the modeling process.

Although the process is divided into the three main phases of MBSE, it is more readily comprehensible that we divided the process into three sections logically, which involve in formalization of plant, specification and supervisor synthesis respectively. Section A and Section B are parallel sub-processes which focus on formalization of plant and specification respectively. The objectives of each section are designated as follows:

(1) Section A

The objective of sub-process Section focuses on the formalization of plants. Therefore, the uncontrolled system and the context should be analyzed. As the fact that the system structure plays

an important role for the modeling results of plant, it is necessary to pay much attention to the link between formal models and physic components in this section.

(2) Section B

The objective of sub-process Section B is to analyze the stakeholders 'need for the control specifications, decompose the requirements, formalize and verify the specifications. Aimed at these purposes, the Section B is composed of activities which cover following aspects: (a) the preprocessing of the information in stakeholders' needs, (b) the analysis and decomposition of narrative requirements, (c) the formalization of requirements, (d) the verification of consistency between requirements and formal specifications and (e) the traceability of requirements and formal models.

(3) Section C

The Section C overlaps the MBSE phase of design synthesis. In this section, the thread of development process is rigorously in the light of the order of supervisor synthesis, controller synthesis and controller implementation. In fact, the following two sub-phases can be regarded as architectural analysis and the controller implementation can be regarded as architectural design. All the global outputs should be produced in the section.

The details of the modeling processes are presented in the following sections.

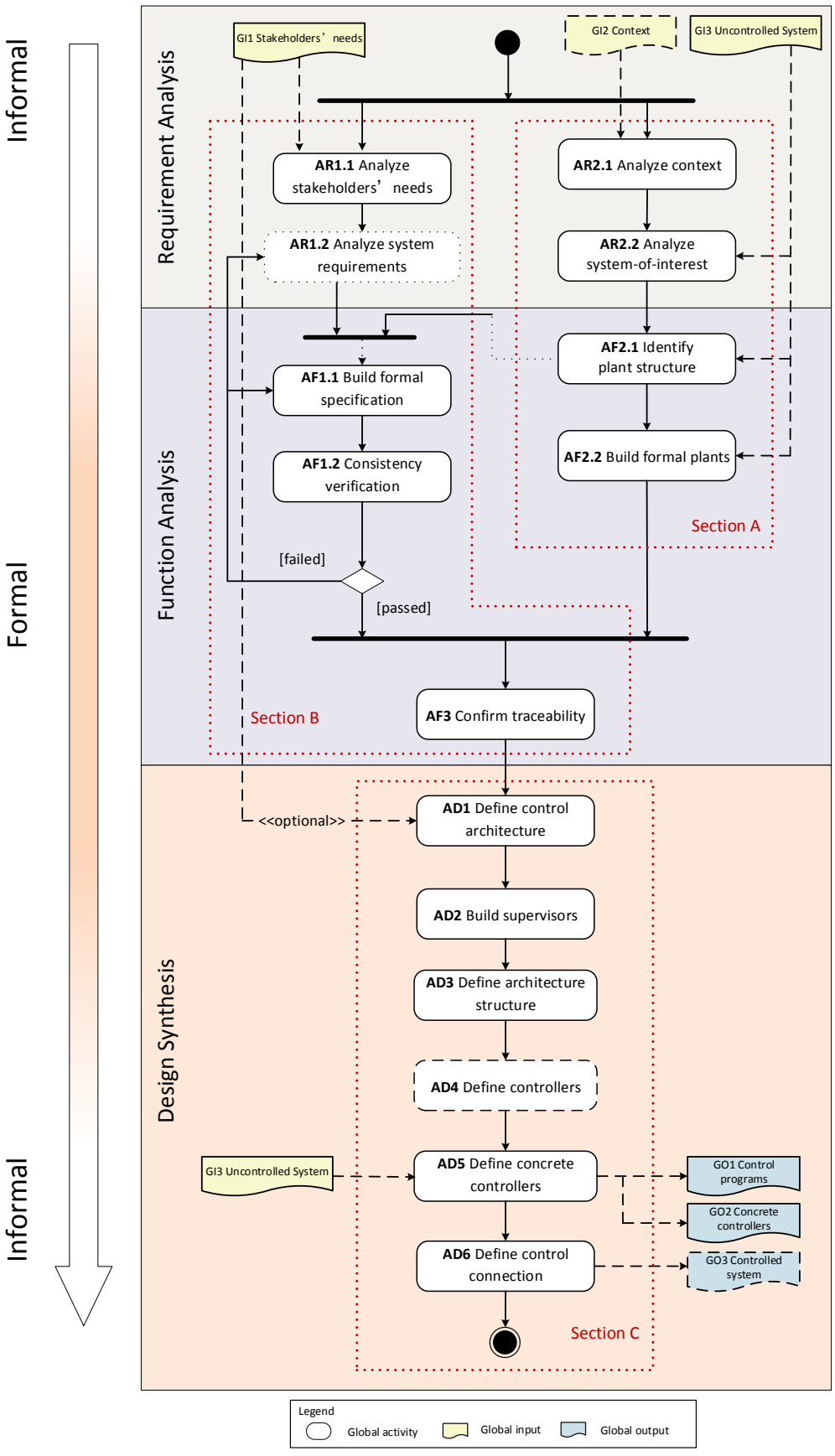


Figure 3.10 Global modeling process

3.4.2 Sub-process: Section A

According to the objective of process Section A, four activities are included in the section: **AR2.1**, **AR2.2**, **AF2.1** and **AF2.2**, which focus on analyzing the structure aspects of the input and construct the physic structure models and behavior models (shown in Figure 3.11). Four outputs will be produced in this section: three SysML BDDs and formal plants. The details of each activity in Section A will be given in the follow parts.

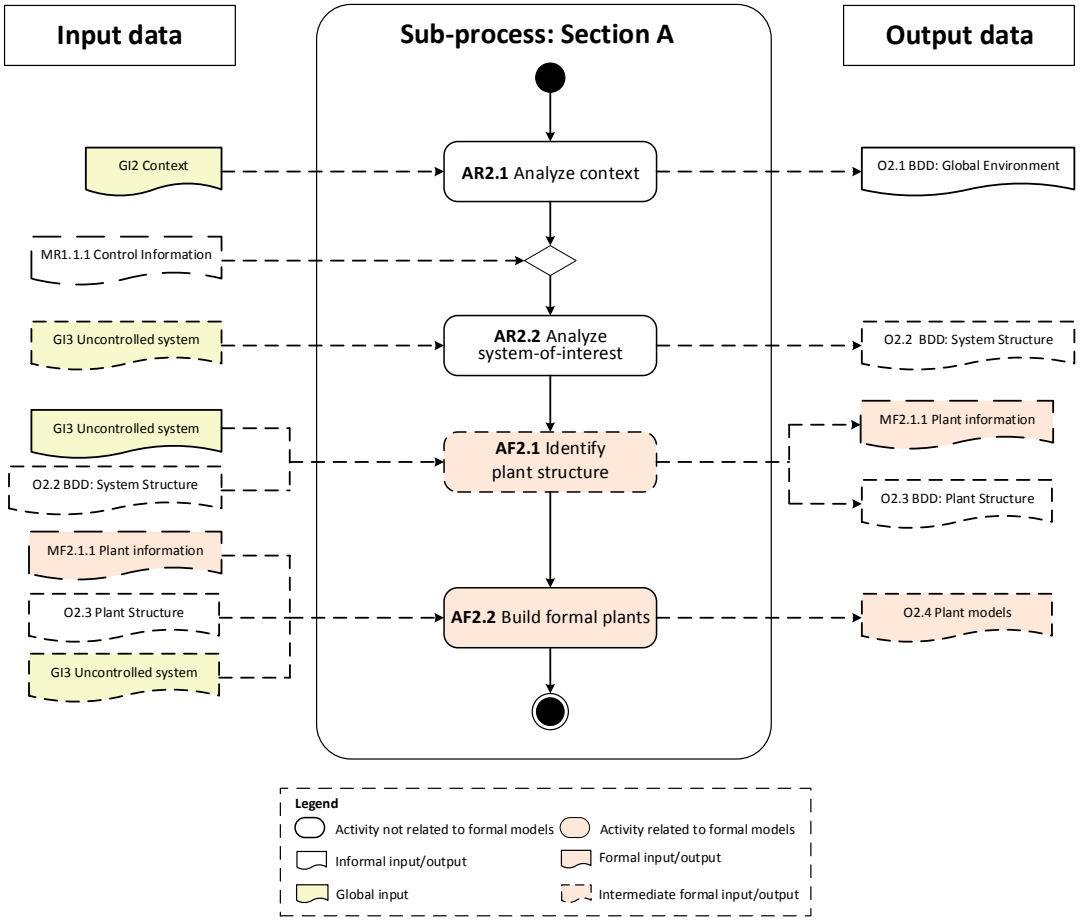


Figure 3.11 Workflow of sub-process of Section A

3.4.2.1 Analyze Context (AR2.1)

The objective of the activity **AR2.1** is to analyze the global environment of the whole system to be studied and build the model of global environment. The context identifies the system boundary and all system actors-humans and external systems which play important roles in the global environment. Four activities are composed of **AR2.1** (shown in Figure 3.12):

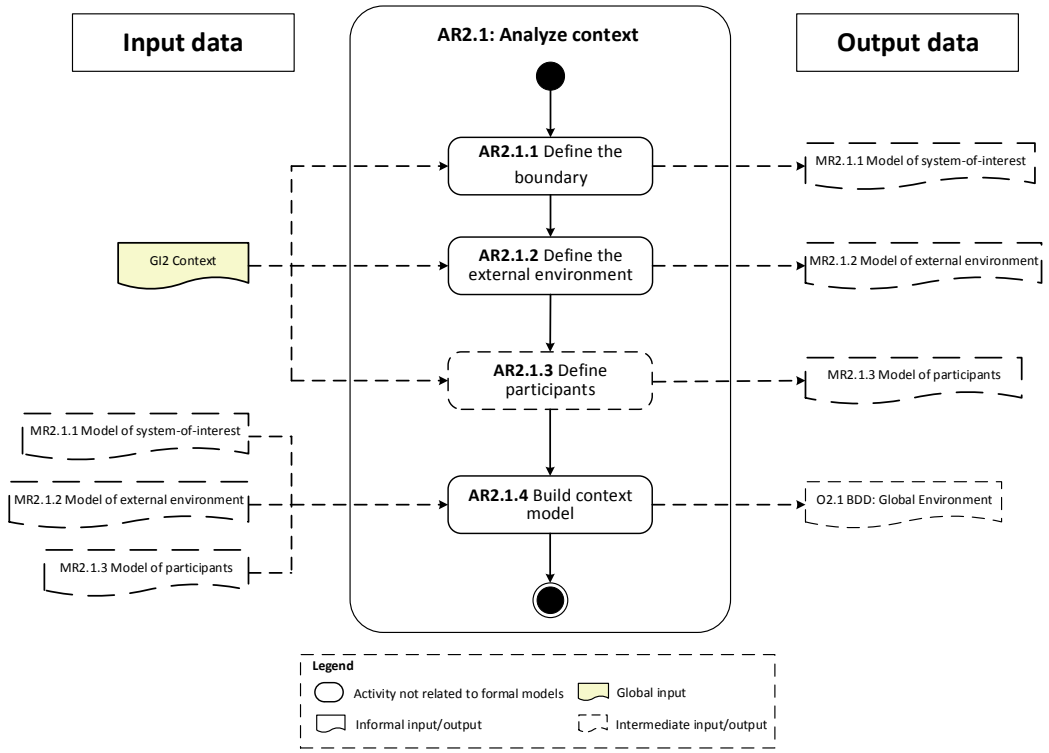


Figure 3.12 Workflow of activity **AR2.1**

- **AR2.1.1** (Define the boundary): This sub-activity focuses on the definition of boundary of system-of-interest. The definition of system-of-interest varies in different engineering context [146]. In consideration of the modeling scope, the model of system-of-interest should be limited to the focus on AC. Therefore, the result of system-of-interest is compulsively decomposed by two physic parts: uncontrolled system and controller. In this step, the internal details of system-of-interest won't be studied due to the objective of system boundary identification.

- **AR2.1.2** (Define the external environment): When the system boundary is determined, the external environment should also be identified. In practice, the system-of-interest is usually a part of the global environment. For example, a manufacturing system interacts with external environment by inlet and outlet to realize the exchange of raw materials and products [147]. The external environment is optional as the system-of-interest can also be independent from any environment or not within the scope of the study.

- **AR2.1.3** (Define participants): Actors-humans are another important factor in the global environment. Although we focus on the design of automatic control system, human operation can also be involved in the system (e.g. emergency stop). Therefore, the involved human operations should be identified in the models.

- **AR2.1.4** (Build context model): The step is to integrate all the models obtained by the previous three steps and present them in a BDD. The models are built at the beginning stage

of the global process and therefore no detail should be presented in the diagram.

The output of **AR2.1** is BDD Global Environment, whose metamodel is defined in Figure 3.13. The block context represents the model of development itself at a global level. The context consists of three elements: system-of-interest, external environment and participants. When applied into real case study, these elements should be replaced by the corresponding systems. For the scope of AC, the external environment and participant just represent the element which are involved in control or affected by the control action.

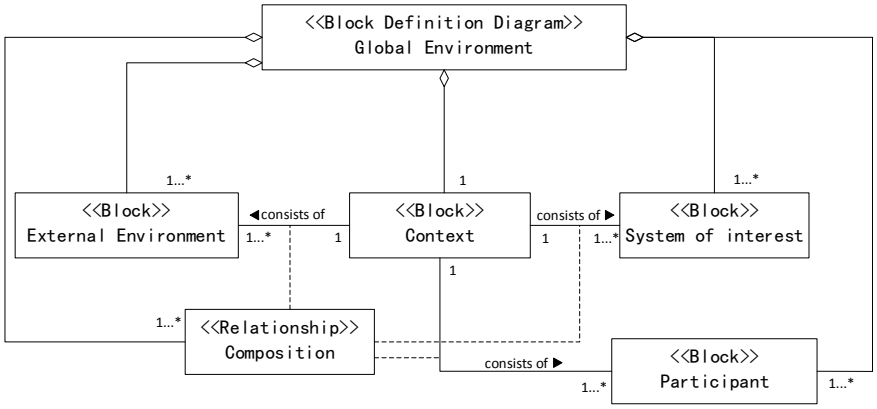


Figure 3.13 Metamodel of BDD: Global Environment

The decomposition of system-of-interest by two parts (shown in Figure 3.14) just represents the conceptual definition (stereotyped by <<conceptual>>) and the model detail should be clarified in following steps.

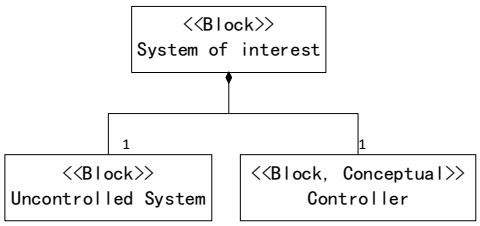


Figure 3.14 Definition of system-of-interest

3.4.2.2 Analyze System-of-Interest (AR2.2)

The objective of the activity **AR2.2** is to detail the structure of system-of-interest, as it is the targeted system of the AC and the plant models largely depend on this step. The controller is still conceptual at this stage and therefore the focus lies in identifying the structure of uncontrolled system.

Definition 3.6 (Uncontrolled system): the constitution of the all physic parts defined in the input

GI3 and within the scope of system-of-interest.

The physic components mentioned in the Definition 3.6 include the components to be controlled by controller and components out of control. Although parts of this work are realized in Section B, it is also necessary to present the control relationship in the model of system-of-interest. Three sub-activities are composed of **AR2.2** (shown in Figure 3.15):

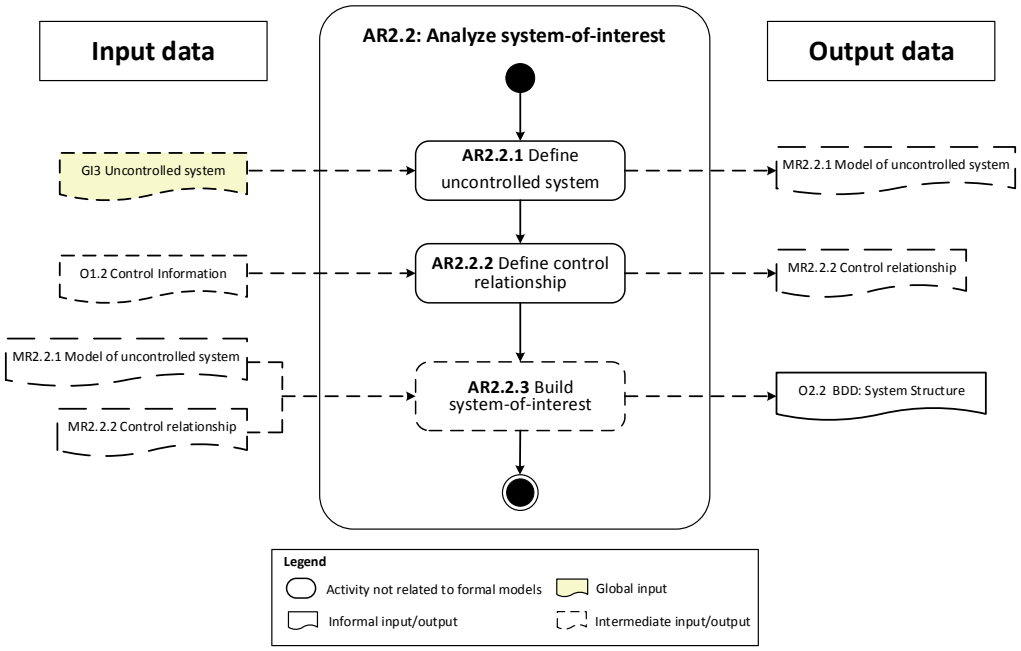


Figure 3.15 Workflow of sub-process of activity **AR2.2**

- **AR2.2.1** (Define uncontrolled system): In this activity, the physic uncontrolled system should be analyzed from the global input GI3, including the involved components and structure. As the uncontrolled system is sometime hierarchical and can be decomposed into sub-systems, the relationship between components in different levels should be presented.

- **AR2.2.2** (Define control relationship): Each component in the uncontrolled system is initially regarded as plant candidate. In this activity, the link of control between components to be controlled and conceptual controller model are explicitly presented.

- **AR2.2.3** (Build system-of-interest): The activity is to integrate all the models obtained by activities **AR2.2.1** and **AR2.2.2**. The SysML BDD is built to explicitly present the models and structure of the system-of-interest.

The BDD: System Structure globally describes the system-of-interest which is presented by a block <<System-of-Interest>>. As the system structure model is usually constructed at the beginning of modeling process, the conceptual controller is just preliminarily decomposed by hardware and software, which are also conceptual models. The semantics should be presented in

the diagram as follows:

(1) Component: The components are modeled by block in BDD. To present this structure, the relationship <<Composition>> is used to connect components and present structure. It should be pointed out that the composition should be explicitly titled by name of part.

(2) Control relationship: To present the control relationship, an association <<control>> is used to connect involved component and controller.

Remark *The difference of block and part in BDD is similar to that of class and instance in UML. In physic system, there can be several parts which share the same "type". For example, a car has four wheels in the same type. In BDD, a block <<wheel >> can be defined and four composition relationships with four different names represent the parts. Therefore, when we want specify the internal details of the car in IBD, four different wheels can be presented. Indeed, a part should have a direct link to physic component rather than the block. In this study, we suppose that the components (parts) defined by the same block have the same control attributes, in other words, both of them are controlled or not at the same time. Therefore, the <<control>> relationship is directly linked to blocks.*

The metamodel of BDD: system structure is shown in Figure 3.16.

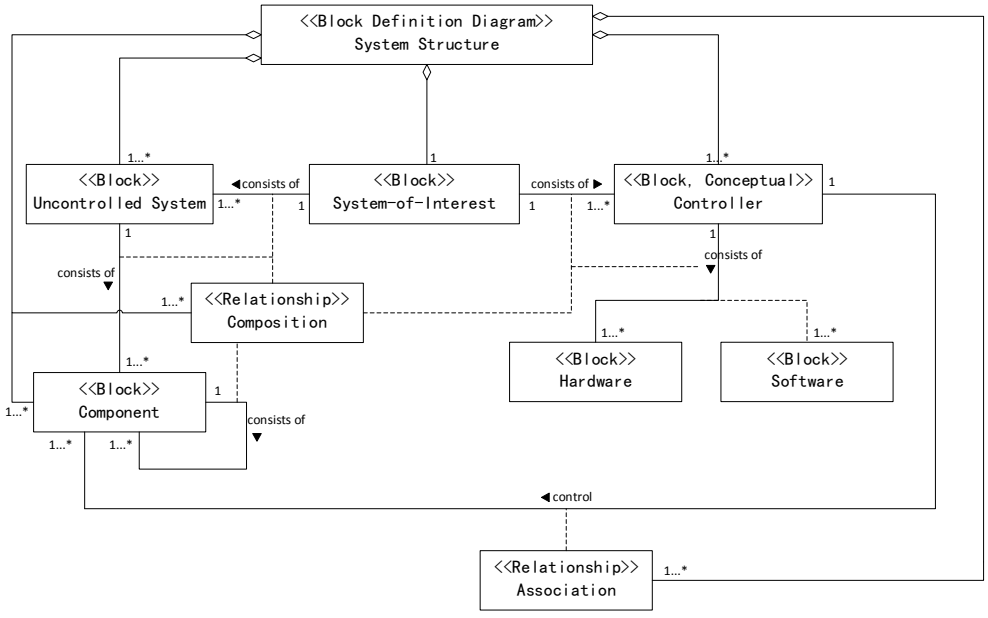


Figure 3.16 Metamodel of BDD: System Structure

3.4.2.3 Identify Plant Structure (AF2.1)

The objective of the activity **AF2.1** is to preliminarily define the link of the physic components and formal plants. Two outputs are produced in this activity: MF2.1.1 plant information, which

presents the element link of physic components and formal plants, and O2.3 BDD: plant structure which presents the link between physic components and plants. Four sub-activities are composed of **AF2.1** (shown in Figure 3.17):

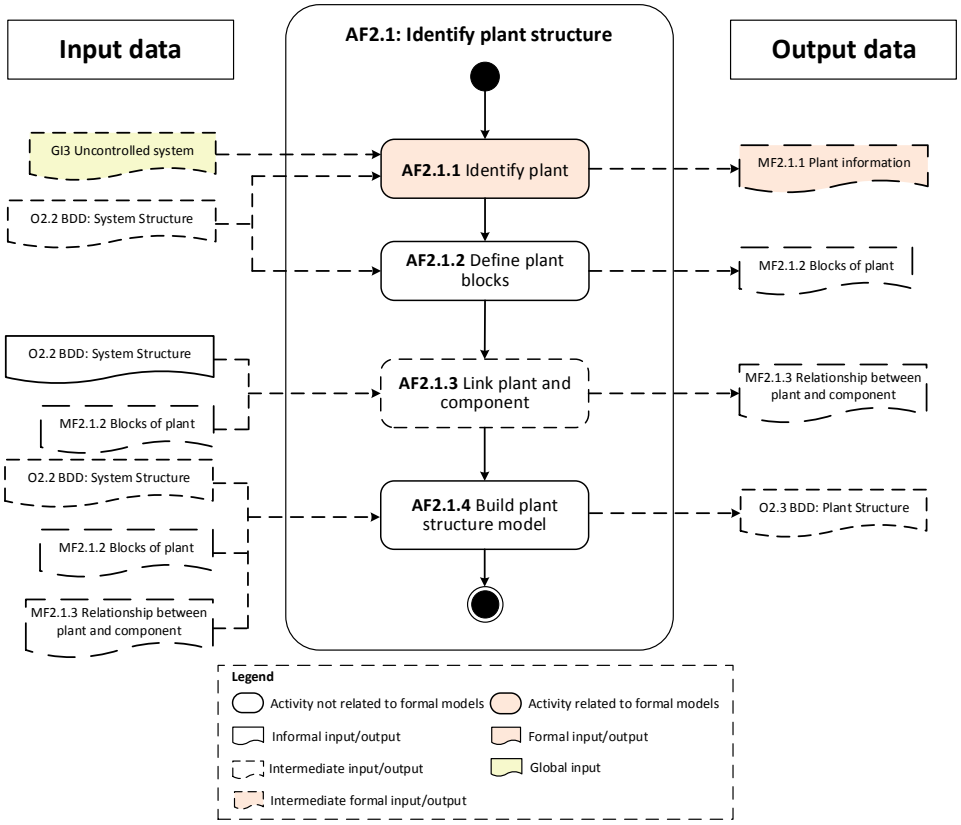


Figure 3.17 Workflow of sub-process of activity **AF2.1**

- **AF2.1.1** (Identify plant): In this activity, the corresponding model elements between physic components and formal plants are identified. Attributes can be identified by input GI3 and O2.2: the actuators/sensors of each part and corresponding operations/signals. Each operation of actuators and sensors can be formalized by corresponding controllable/uncontrollable event. The example of MF2.1.1 plant information list is presented in Table 3.3.

Table 3.3 Example plant information list

Component	**Type**	**Description**	**Event Notation**	**Event Type**
part: block	actuator/sensor	*description*	event name	Controllable/Uncontrollable
…	…	…	…	…

- **AF2.1.2** (Define plant blocks): The plant is automaton-based formal model and cannot be directly presented in SysML. To this end, block is defined to represent the plant model in the diagram. In fact, all the parts should carefully analyze in order to determine the plant models related to the corresponding physic components. The method of plant identified will be

introduced in detail in the following chapter.

- **AF2.1.3** (Link plant and component): In this activity, the link between parts and corresponding plant block should be confirmed. In order to construct the correct link, it is necessary to make sure that all the parts have their corresponding plant block by referring to O2.2.

- **AF2.1.4** (Build plant structure model): In this activity, a BDD should be built by integrating the outputs of **AF2.1.2** and **AF2.1.3** to provide a graphic description between plants and parts.

The BDD: Plant Structure is built to present a graphic link between formal plants and physic components. As plant are formal models represents the behavior of components, the structure of plant depends on the structure of components. To build the link between plants and components (parts), the semantics of the elements in the diagram should be identified as follows:

(1) For the parts representing the components, the operations/signals are presented. As link the controllable event and uncontrollable event to operation of actuator and signal of sensor of the physic component are built based on the identification.

(2) Each components interact with controller must have a plant model to represent the behavior. However, a plant is formal model which cannot be directly presented in the diagram. A block named by each plant and stereotyped by <<formal>> is defined to represent the plant models in the diagram. Therefore, the block plant should be allocated to the corresponding physic components, which is linked by the relationship <<Formalize>>.

To sum up, the metamodel of BDD: Plant Structure can be described in Figure 3.18.

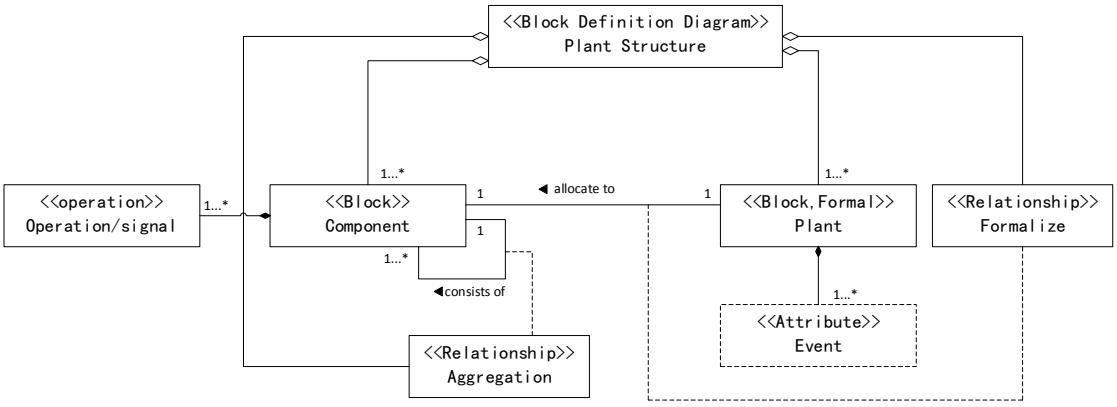


Figure 3.18 Metamodel of BDD: Plant Structure

3.4.2.4 Build Formal Plants (AF2.2)

The activity **AF2.2** is to build the automata models for all blocks of plant defined in **AF2.1**. This step is similar to formalization of plant in the typical modeling process. However, to link the formal models and SysML models, the activity can be decomposed to a loop of three sub-activities (Figure 3.19):

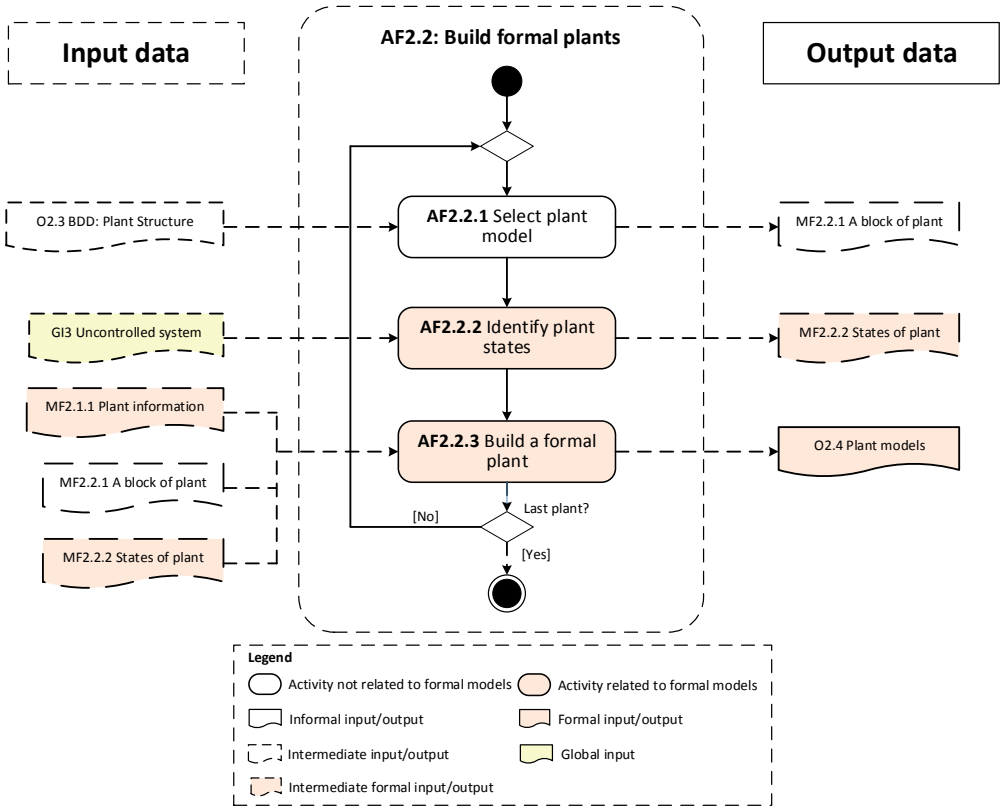


Figure 3.19 Workflow of sub-process of activity **AF2.2**

- **AF2.2.1** (Select plant): After checking that there are still plants to be modeled, the next plant block is selected for the diagram O2.3.

- **AF2.2.2** (Identify plant states): In this step, the operating states of the physic component related to the plant are identified. The semantic of physic states are identified and formalized to the states of automaton.

- **AF2.2.3** (Build formal a plant): An automaton is constructed for the selected plant in this step. For correctly using the notation of events and states in the automaton and analyze the behavior of the physic component, MF2.1.1 and MF2.2.2 should be referred to.

3.4.3 Sub-process: Section B

The process of Section B is a little more complex than Section A, as it includes a loop to realize the requirement verification process. Section B consists of five main activities: **AR1.1**: Analyze stakeholders' needs, **AR1.2**: Analyze system requirements, **AF1.1**: Build formal specifications, **AF1.2** Consistency verification and **AF3** Confirm Traceability (shown in Figure 3.20). Three outputs are produced in the section including three SysML diagrams and specification models.

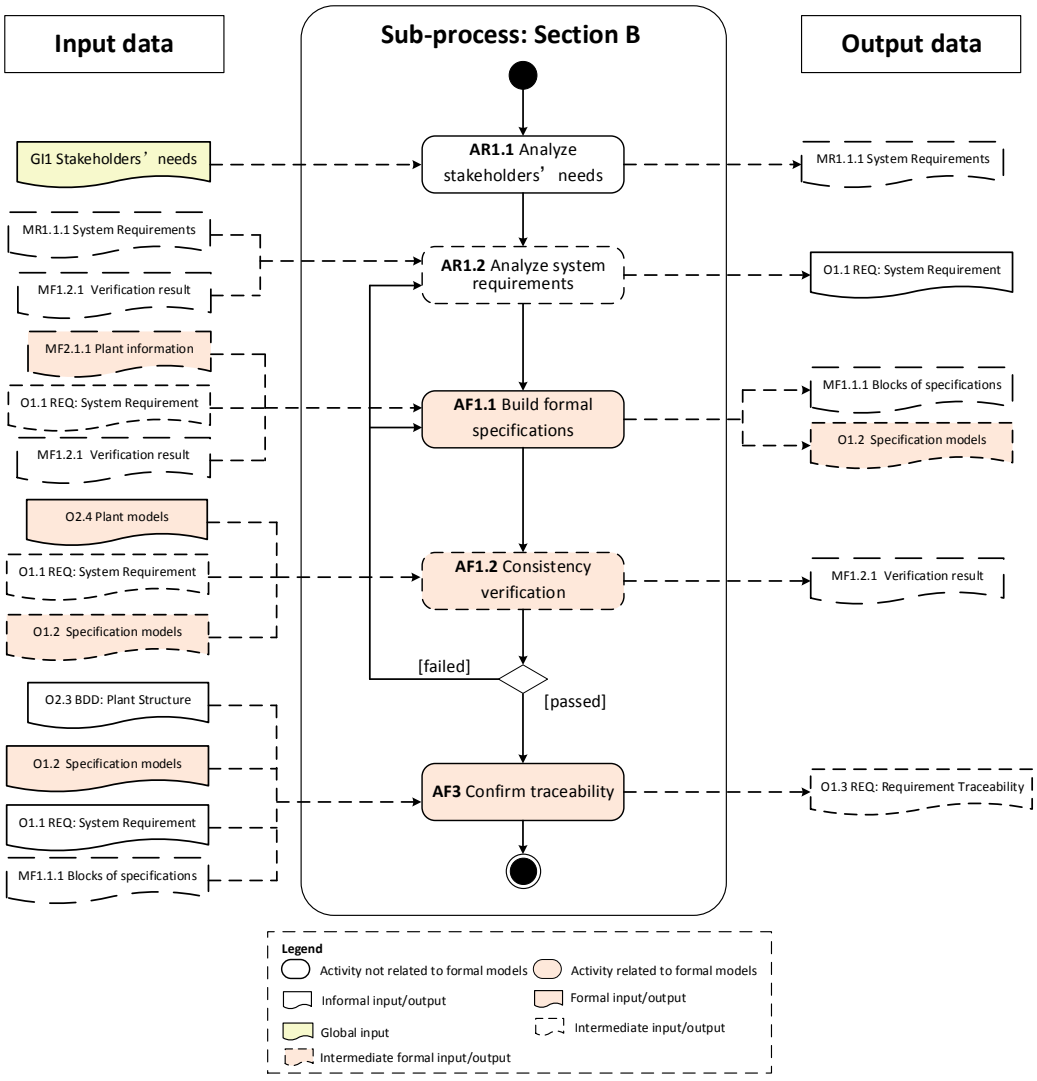


Figure 3.20 Workflow of sub-process of Section B

3.4.3.1 Analyze Stakeholders' Needs (AR1.1)

The activity **AR1.1** focuses on preprocessing the global input of Stakeholders' needs. In this step, the system requirements which describe the control target and control specifications are initially analyzed and decomposed at high-level.

In SysML, requirement is allowed to be decomposed to sub-requirements for requirement analysis. The topmost level requirements usually specify main targets and the low level requirement refined the upper level requirement by detailed description. One problem to be faced for the AC is that to what extent the refinements should be refined is unclear. It leads to the difficult to make the trace for formal specification.

The requirements are usually classified to functional and non-functional [148]. A functional requirement defines a function of a system or its component, where a function is described as a specification of behavior between outputs and inputs [149], while non-functional requirement, often called “quality attributes”, specifies criteria that can be used to judge the operation of a system, rather than specific behaviors (e.g. maintainability, Operability, Durability, Extensibility, etc.). For the purpose of focusing on the AC, we limit the modeling scope at functional requirement level as formal models can be used to identify non-functional requirements. Therefore, we concentrate on figuring out the analyze method of the functional requirement to decompose them at an appropriate refinement level. Two level of requirement should be determined in the output of this step:

Level 1: Mission:

The topmost level of requirement defines the target of the control mission. At this level, one or several top requirements whose descriptions specify what the global goals of the system should be achieved under control. The description of the mission requirement should be concise by using several key words and no function details should be presented.

Level 2: Function:

The requirements defined at function level are designated to specify the functional of controlled system to achieve the global target of mission requirements. The descriptions can be either logic or quantitative so long as the mission can be well explained. Besides, the decomposition of requirement at this level can be hierarchy.

3.4.3.2 Analyze System Requirements (AR1.2)

The objective of the activity **AR1.2** is to analyze and decompose the high-level narrative requirements into low-level requirements for the formalization and traceability in the following activities. This activity is composed of three steps (Figure 3.21):

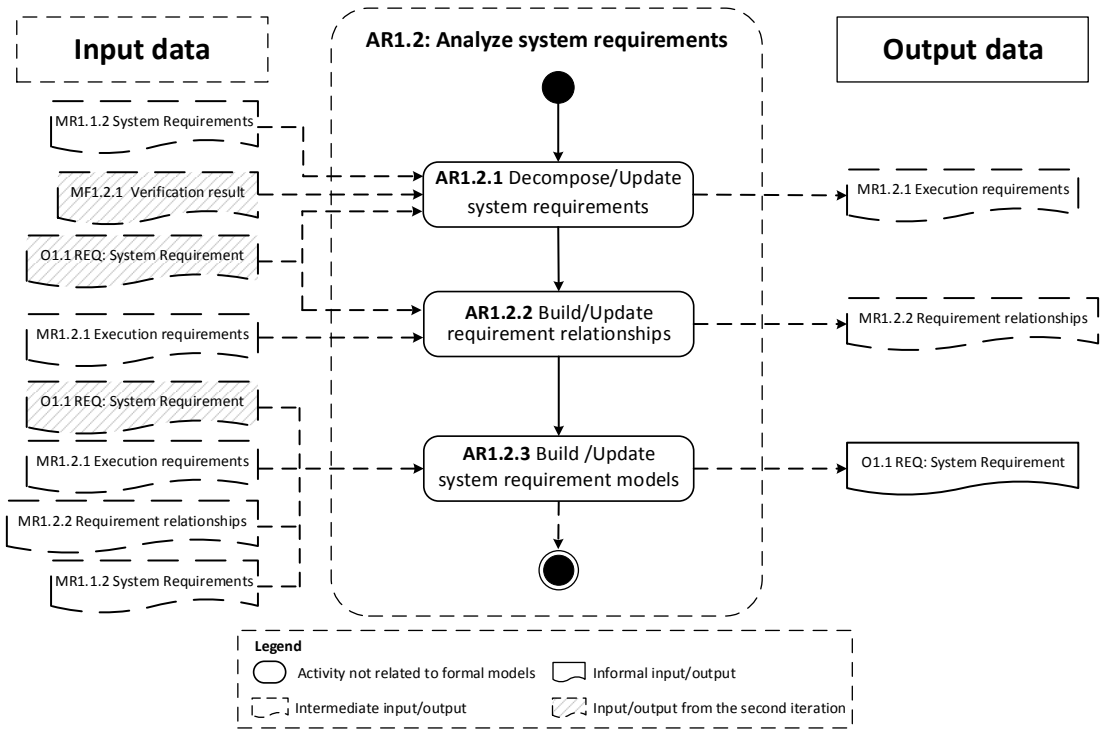


Figure 3.21 Workflow of sub-process of activity **AR1.2**

- **AR1.2.1** (Decompose/Update system requirements): Two situations should be defined for this step: (1) when this step is performed by the first time, we can directly decomposed the requirements provided by **AR1.1**. (2) when this step is entered from the second iteration due to the fault of consistency verification, the requirements should be modified according to the verification results. The high-level requirements should be decomposed to the *E-requirements* (Execution requirements) in this step.

- **AR1.2.2** (Build/Update requirement relationships): In order to explicitly present the relationships among requirements, all relationships should be identified or updated. The relationships can be defined by standardized types provided by SysML (e.g. <<Derive>>, <<Contain>>…).

- **AR1.2.3** (Build/Update system requirement models): A requirement diagram should be built to present the system requirement decomposition and their relationships. The requirement diagram integrates the results in **AR1.2.1** and **AR1.2.2**. For the second time to be performed, the diagram should be updated rather than rebuilt.

The third level of requirements is defined in this step:

Level 3: Execution:

At execution level, requirements are specified for the realization of function requirement at

component level. Aimed at traceability for formal specification, the execution requirement is defined with the description in which the nature language narrative should be interpreted to formal specification. For the purpose of making sure that all high-level requirements will be finally decomposed into execution requirements so that the traceability between execution requirements and automaton specifications can be constructed, the execution requirements are set the bottommost models for the requirement analysis. Hereof, such execution requirement is denoted by *E-requirement* and its text description is denoted by *E-specification*.

The benefit of E-requirement is to make the requirement traceable. When the requirement traceability is checked, the requirement traceability diagram can be built to represent it. Each root E-requirement can be mapped by a formal specification (in activity **AF1.1**), which bridges the gap between SysML model and formal model. In order to realize this objective, special narrative descriptions are defined for the E-requirements. The precise definition of E-requirement, the relationships between requirements, the method of requirement decomposition and the semantics of the diagram will be detailed in Chapter 4.

3.4.3.3 Build Formal Specifications (AF1.1)

The objective of the activity **AF1.1** is to build the formal models according to the narrative requirements. Similar to activity **AR1.2**, the verification result is also involved. The sub-process of **AF1.1** is shown in Figure 3.22.

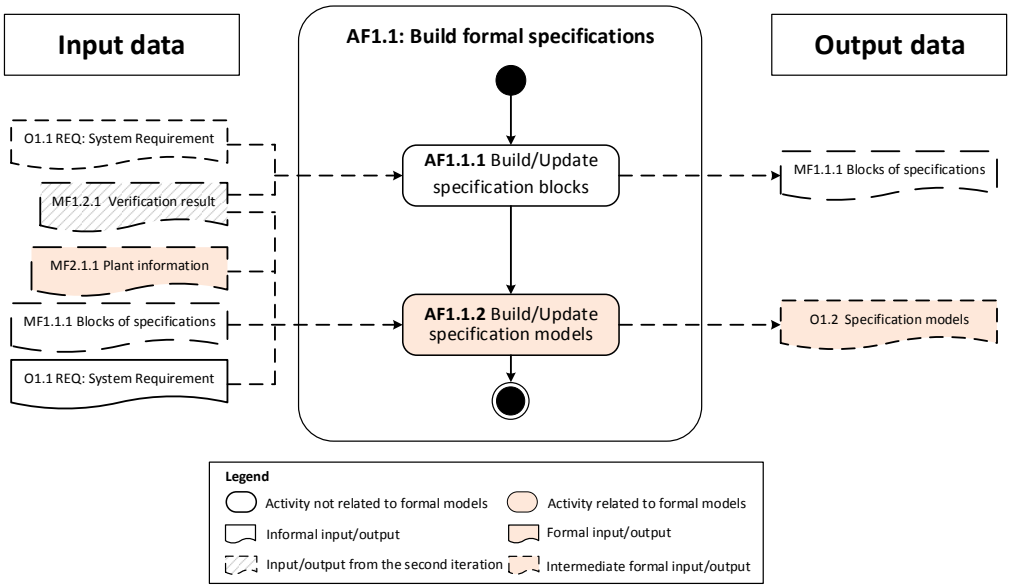


Figure 3.22 Workflow of sub-process of activity **AF1.1**

- **AF1.1.1** (Build/Update specification blocks): The blocks of specifications should be built for each root E-requirements. From the second iteration, two situations should be handled: (1) when the new iteration is directly from **AF1.1**, the verification result is read and the block of

specification to be modified should be determined; (2) when the new iteration is from **AR1.2**, the modification should be determined from the results of **AR1.2**.

- **AF1.1.2** (Build/Update specification models): The formal specifications are built in this activity according to MF1.2.1, MF2.1.1, MF1.1.1 and O1.1. From the second iteration, only a part of specifications should be undated according to the **AF1.1.1**.

3.4.3.4 Consistency Verification (AF1.2)

The objective of the activity **AF1.2** is to verify the consistency between requirements and formal specifications. This step is of vital importance as it bridges semi-formal and formal models. The activity is not isolated, but has a strong connection with activity **AR 1.2** an **AF1.1** (shown in Figure 3.20). The outputs of this activity provide the verification results which indicate explicitly the fault information of verification, according to which the problem requirements or specifications are modified. The proposed solution of verification and process details of **AF1.2** will be detailed in the next chapter.

3.4.3.5 Confirm Traceability (AF3)

When the verification in **AF1.2** passed, the process will be moved to this activity. As the last step before design synthesis, the objective of the activity **AF3** is to integrate the requirements, formal plants and specifications and present explicit in a diagram the links among these models as the foundation of supervisor synthesis. The activity is composed of four steps:

- **AF3.1** (Link specifications and requirements): In this activity, the block of specification and requirements should be linked. This step is just to build the relationship between SysML models to be prepared for the traceability model according to the link between requirements and formal specifications.

- **AF3.2** (Link plants and requirements): The link between blocks of plant and requirements are constructed. The link can be realized by analyzing the events in formal specifications which can be mapped to the corresponding plant models.

- **AF3.3** (Build traceability model): To integrate the outputs of activity **AF1.1.1**, **AF3.1**, **AF3.2** and BDD: Plant Structure and, a BDD: Requirement Traceability is built to present the relationships among these models.

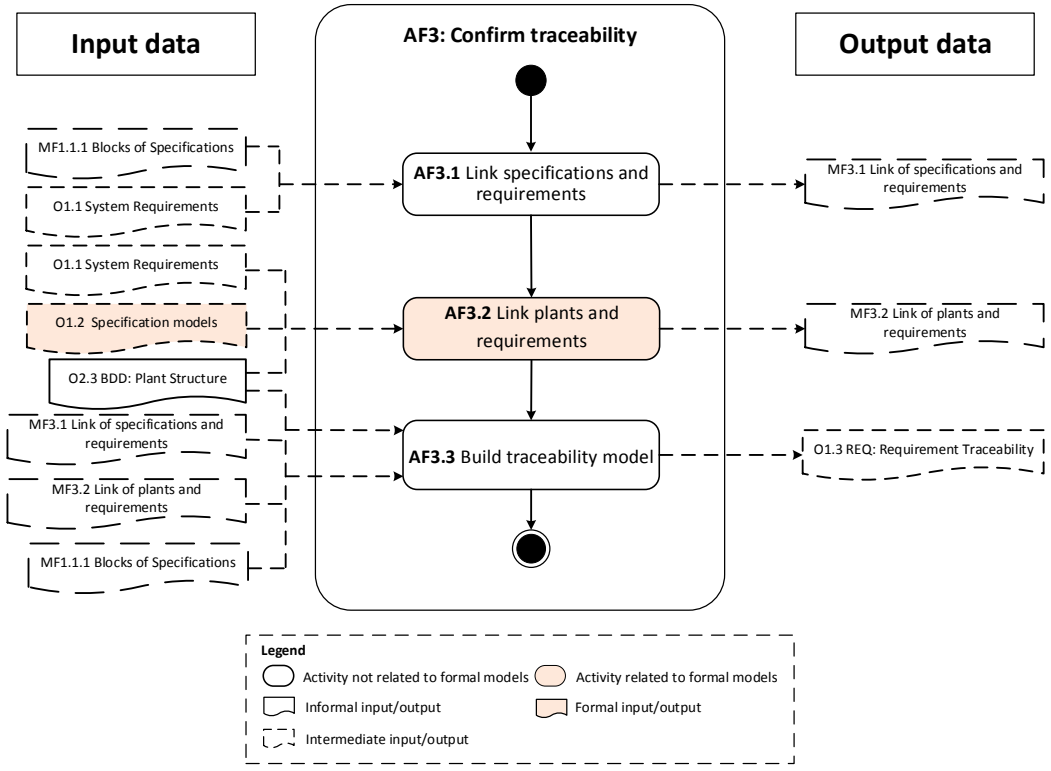


Figure 3.23 Workflow of sub-process of activity **AF3**

Figure 3.24 shows the metamodel of REQ: Requirement Traceability. SysML supports relationships between requirement and other models. In order to present the link between requirements and formal models, the semantic of the relationships are defined as follows:

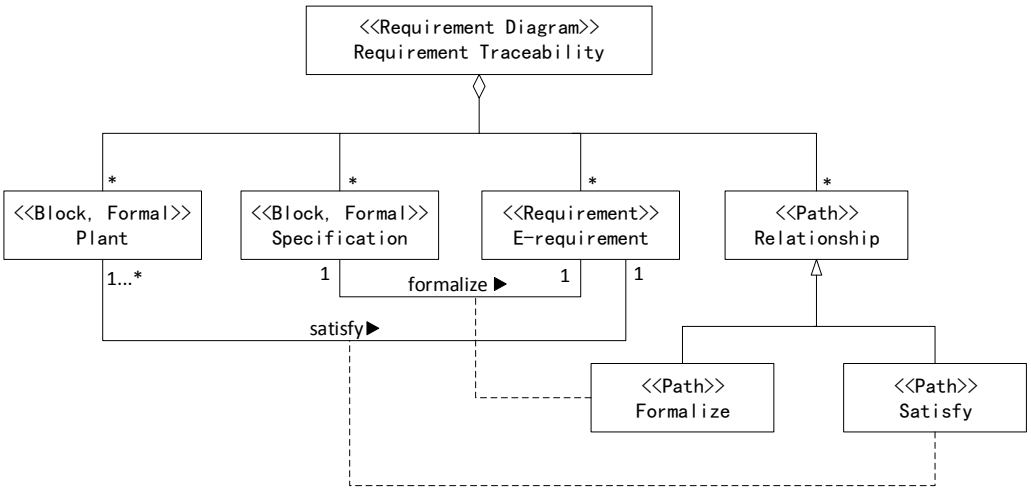


Figure 3.24 Metamodel of REQ: Requirement Traceability

(1) To represent the link between E-requirements and specification blocks, the relationship <<formalize>> is used. For each root E-requirement, a corresponding block of specification is linked. The semantics of the <<formalize>> in this diagram present that the formal specification is

consistent with the E-requirement.

(2) The relationship between plant and E-requirement is presented by <<satisfy>> as the plant is the representation the behavior of physic component. The semantics of the <<satisfy>> in this diagram is that the behavior represented by formal plant is under constraint by the E-requirement.

3.4.4 Sub-process: Section C

The process of Section C focuses on the supervisor/controller synthesis and implementation based on the models of Section A and Section B and methods within SCT paradigm. Section C consists of five main activities: **AD1.1**: Define control architecture, **AD2**: Build supervisors, **AD3**: Define architecture structure, **AD4**: Define controllers, **AD5**: Define concrete controllers and **AD6**: Define control connection (shown in Figure 3.25). Seven outputs are produced in the section including models for supervisors, controllers and controller implementation. Among them, the controller implementation is the global output of the modeling process, involving GO1 Control program, GO2 Concrete controller and GO3 Controlled system.

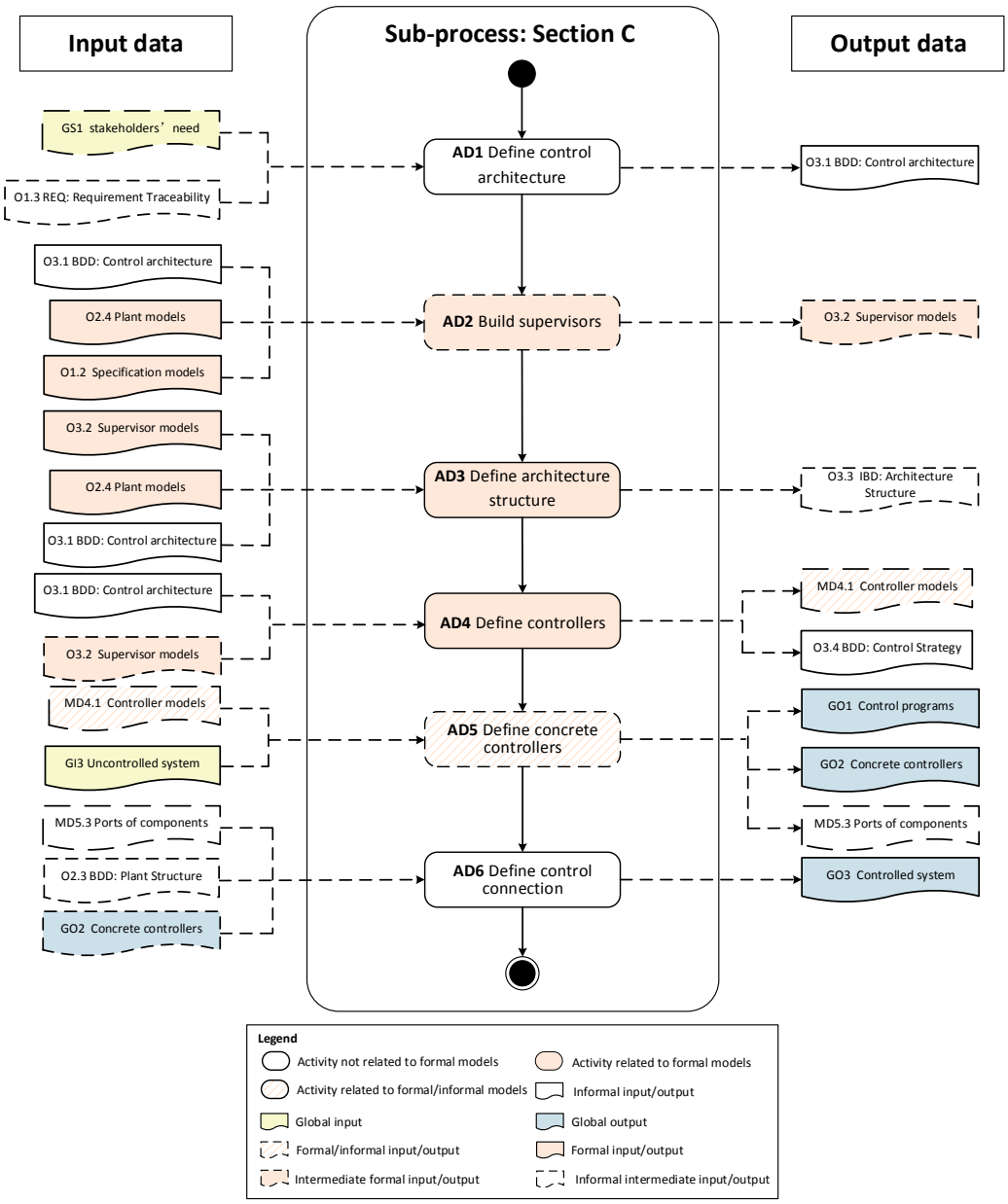


Figure 3.25 Workflow of sub-process of Section C

3.4.4.1 Define Control Architecture (AD1)

The objective of the activity **AD1** is to determine and present the control architecture. In SCT, we just have the supervisor models but the structure of supervisors cannot be presented. In this activity, a series of steps should be performed before producing the model of control architecture. To make the design conform to the MBSE process, an optional trade study is integrated in the activity. The activity **AD1** can be decomposed into several step (shown in Figure 3.26):

- **(Optional) AD1.1** (Analyze stakeholders' needs): The objective of the sub-activity to make sure whether the control architecture has been predefined by stakeholders, which should be

either on the structure of the supervisor or on the specifications on the concrete controller. For example, if there are specifications in the stakeholders' needs which predefines that the implementation should be realized by several PLCs, the control architecture should be modular, decentralized or distributed rather than monolithic. This step is optional as it may be no predefinition from the stakeholders' needs at all. Therefore, the output of **AD1.1** should be one or several control architecture candidates.

- **AD1.2** (Build architecture models): The work in this activity focus on the modeling for each control architecture candidates. A BDD should be built for each candidate where the supervisors are defined based on the specifications and plants. Therefore, the model O3.1 should be referred to.

- **(Optional) AD1.3** (Determine a solution): The activity is also optional as it is applicable when there are several candidates to be compared. A trade study should be performed to determine the best or most feasible candidate as the final solution of the control architecture. To this end, a unique BDD will be the final output of activity **AD1**.

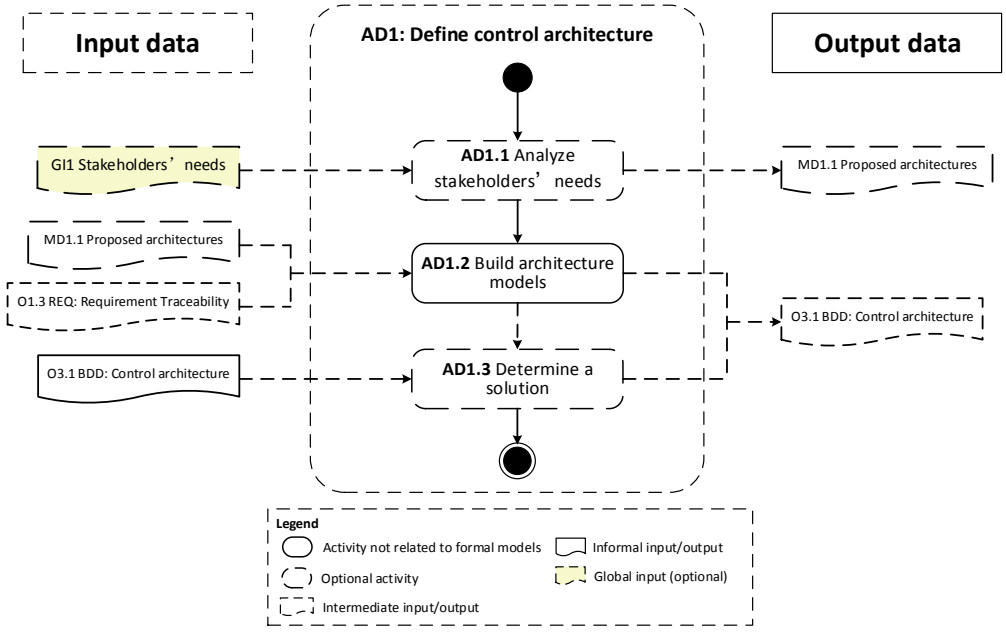


Figure 3.26 Workflow of sub-process of activity **AD1**

The BDD of Control Architecture (see metamodel in Figure 3.27) describes the structure of supervisors along with the relationships with corresponding plants and specifications. In this diagram, all the involved blocks represent the formal models including plants, specifications and supervisors. The structure of the supervisor should be explicitly presented for the designated control architecture. To this end, two kinds of relationship used in the diagram, whose semantics are defined as follows:

(1) The relationship between global/local supervisors and global/local plants are presented by <<

Synchronize>>. As the diagram O3.1 just provides the local plants, the global plants should usually be synthesized by several local plants. Therefore, the semantics of <<Synchronize>> is the global plants synchronized by corresponding local plants.

(2) The relationship between supervisor and corresponding plants and specifications are presented by <<SynthesizeFrom>> to present that the supervisor is synthesized by corresponding plants and specifications.

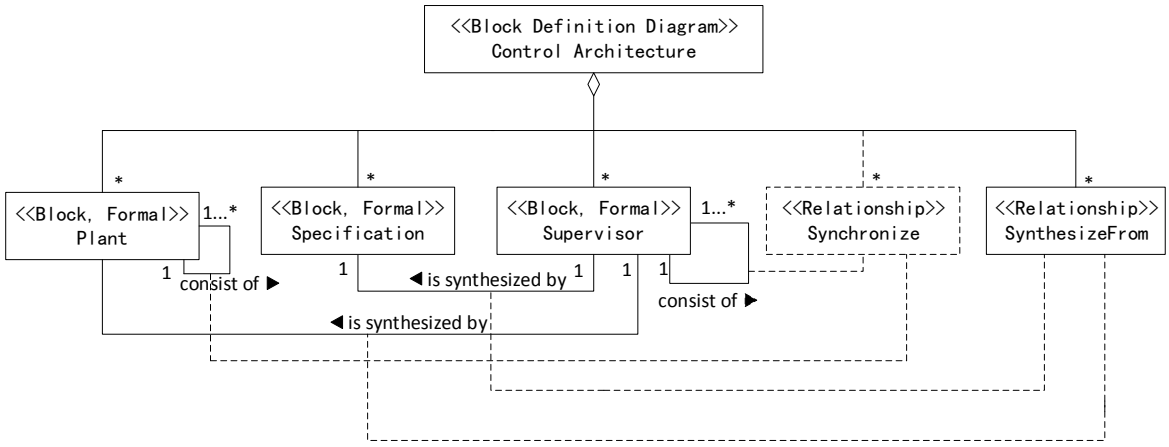


Figure 3.27 Metamodel of BDD: Control Architecture

3.4.4.2 Build Supervisors (AD2)

The objective of the activity **AD2** is to build the supervisor models based on the defined control architecture in **AD1**. This step is similar to the supervisor synthesis step in typical SCT process. To integrated supervisor synthesis in the MBSE process, the activity **AD2** can be decomposed as follows (shown in Figure 3.28):

- **AD2.1** (Confirm architecture): The activity is to do the preparation for supervisor synthesis. In this step, the plants and specifications involved in the computation of each global/local supervisor are regrouped.

- **AD2.2** (Synchronize global/local plants): For each global/local supervisor to be synthesized, the involved plants should be synchronized to global/local plants before computation.

- **AD2.3** (Synthesis supervisors): Each global/local supervisor with regard to the specifications in O1.2 is computed according to the corresponding SCT supervisor synthesis algorithm (Appendix A).

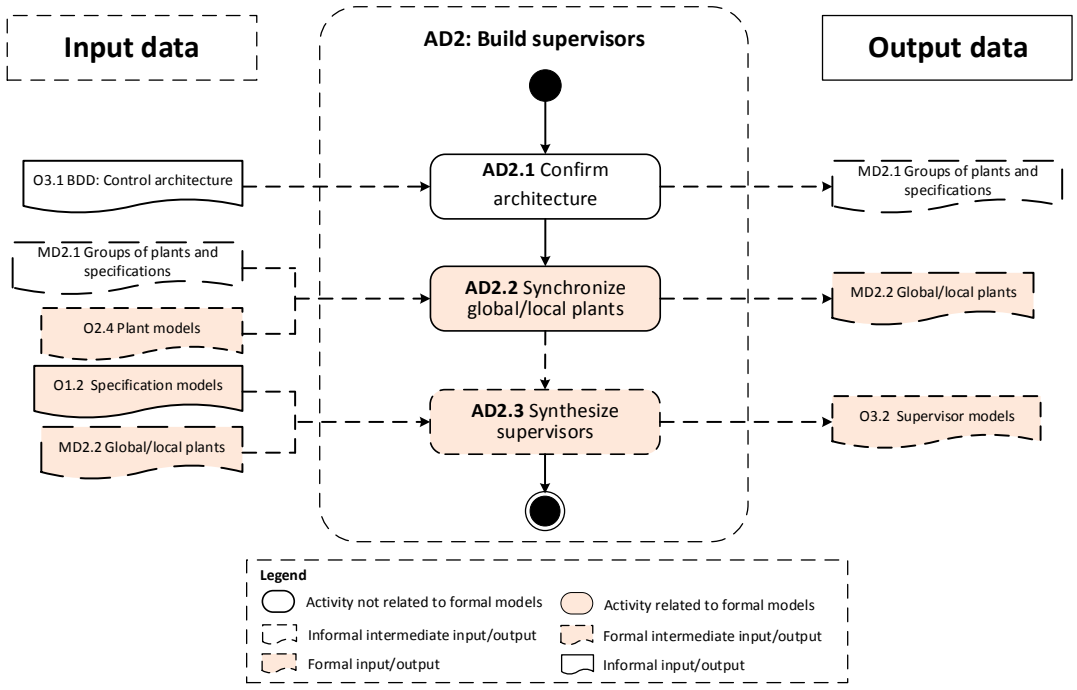


Figure 3.28 Workflow of sub-process of activity **AD2**

3.4.4.3 Define Architecture Structure (AD3)

IBD: Architecture Structure is built to detail the structure of control architecture, as a visualization of the architecture presented in Figure 2.2 and Figure 2.3. For building the diagram, the activity **AD3** can be decomposed as follows (shown in Figure 3.29):

- **AD3.1** (Determine supervisor and plant): The step is to identify the structure of plants and supervisors. According to different control architecture, the plants and supervisor may be structured. As defined in **AD1**, the semantics of which a global plant which synthesized by several local plants is presented by <<Synchronize>>. The IBD must present this structure by directly putting local plants in global plant.

- **AD3.2** (Define ports): For constructing the link between formal models, the semantics of ports should be defined as event in each model. All events in each formal model including plants and supervisors are identified and a corresponding port of the same event names is defined.

- **AD3.3** (Build architecture structure model): To integrate the results in **AD3.1** and **AD3.2**, the IBD can be constructed. As all the ports are named by the corresponding events, a direct connection can be built for all the ports with the same names.

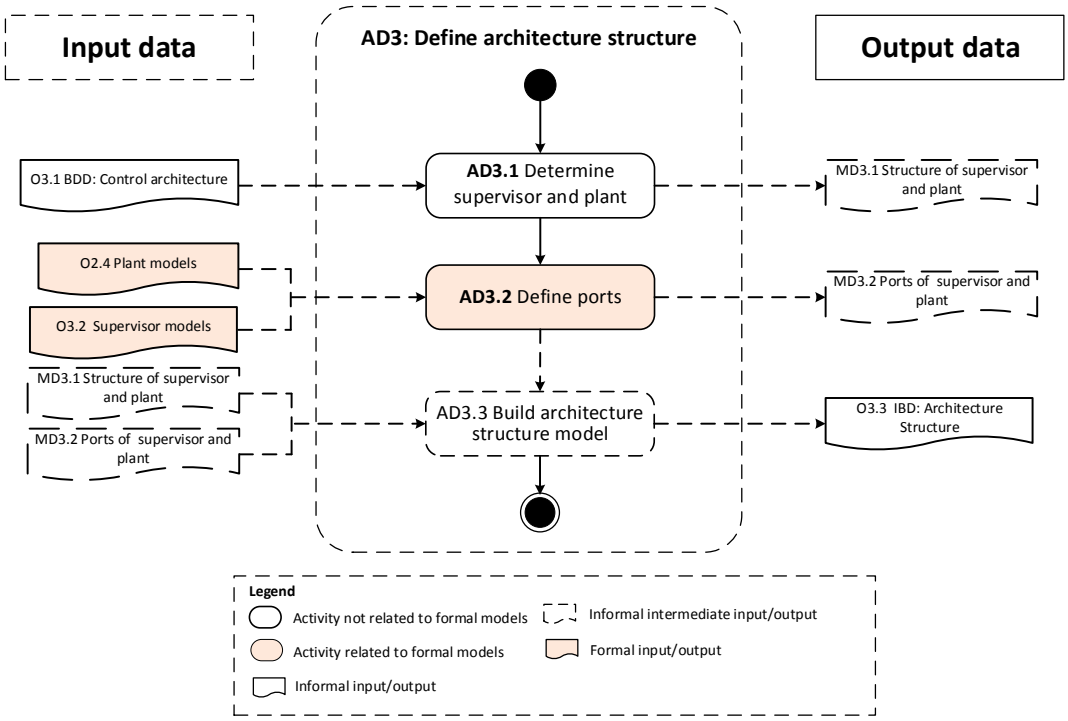


Figure 3.29 Workflow of sub-process of activity **AD3**

The metamodel of IBD: Architecture Structure is shown in Figure 3.30. In the diagram, formal models are represented by parts. Two kinds of connection which represent the formal semantics should be constructed:

(1) The connection between local plants and global plants: the semantics of this connection represent that the global event set includes the event set of local plants;

(2) The connection between global plants and global/local supervisors: the semantics of this connection represent that event sets which is observed and controlled by the global/local supervisors.

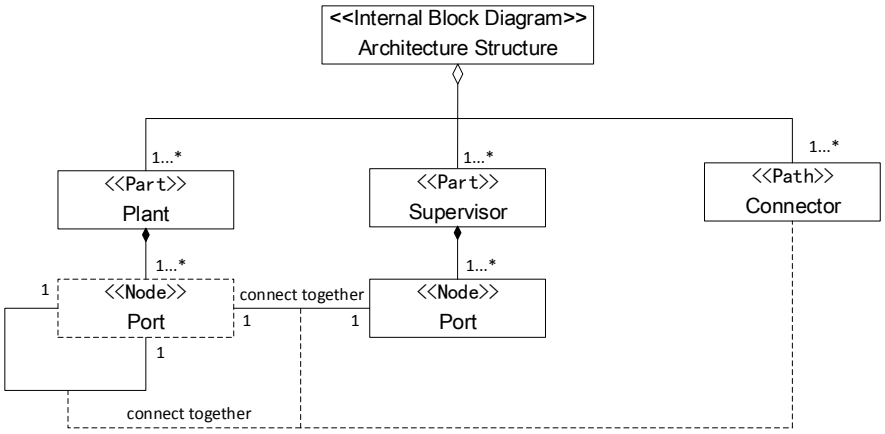


Figure 3.30 Metamodel of IBD: Architecture Structure

3.4.4.4 Define Controllers (AD4)

The objective of the activity **AD4** is to transform the supervisors to controllers based on the control strategy. As the intermediate models bridging formal supervisor to controller implementation, the controller should explicitly present the nature of the designated control strategy. As there is no standardized methods to build the controller models within SCT paradigm, the form of controller is undetermined, which can be either automaton or in other form. In this study, the controller model is defined as formal or informal model (shown in Figure 3.31). The activity **AD4** is composed of three sub-activities:

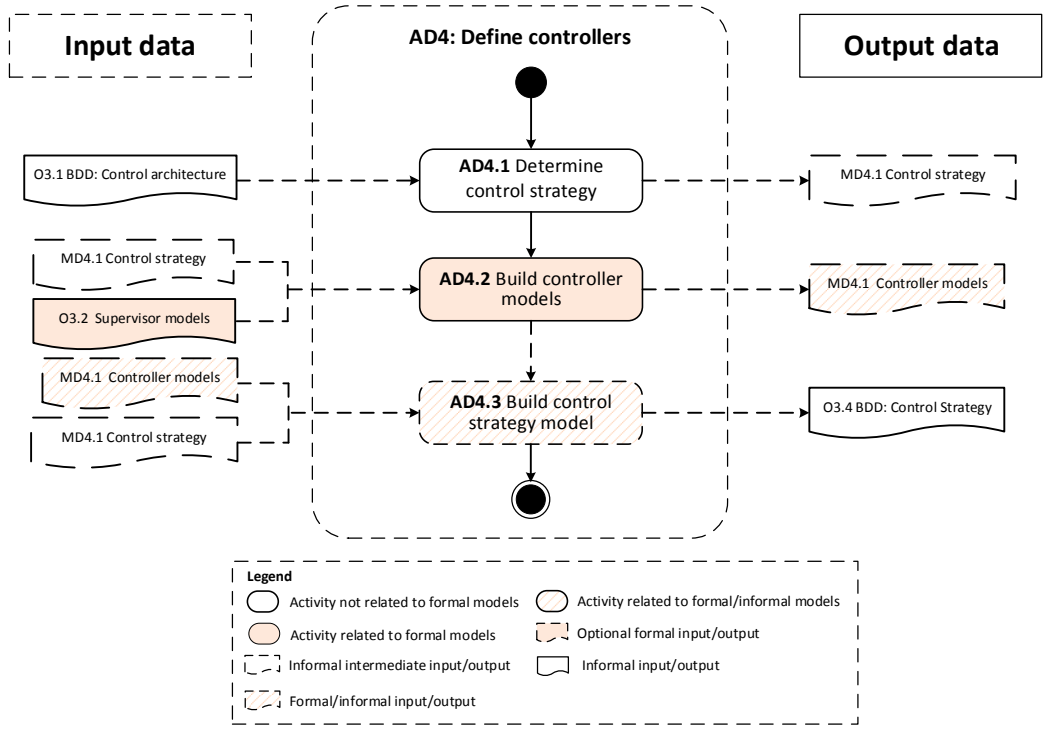


Figure 3.31 Workflow of sub-process of activity **AD4**

- **AD4.1** (Determine control strategy): Although there is not systematic study on this point, a number of case studies have proposed their solutions (see Chapter 2, Section 2.2.3.1). In this step, one of control strategies should be determined we will apply to define the controllers according to situation of the particular case study.

- **AD4.2** (Build controller models): The controller models are representation for the designated control strategy. The form of this models vary as some of the can be represented by automaton trimmed from the supervisors, while others can be an algorithm, a module or even a series of narrative specifications. Different blocks can be built to represent the control strategy and therefore can be presented in SysML diagram. For example, in [53], the authors proposed a trajectory choice approach in order to make the control behavior "deterministic".

The controller can still be modeled by automaton and the corresponding block related to the controller stereotyped by <<controller>> is built. In [54], the authors developed an algorithm for the generation of valid controllers. To model this situation, the algorithm should be in form of narrative description and related to the block present it.

- **AD4.3** (Build control strategy models): This activity integrates the all the necessary elements constituting the controller. A BDD: Control Strategy is built to graphically present the composition of controller. As we have defined a series of blocks which represent the controllers, the controller can be explicitly in the diagram. The relationships can also be used to represent the link between relevant model elements.

3.4.4.5 Define Concrete Controllers (AD5)

The objective of the activity **AD5** is to construct the concrete controllers and generate control programs based on the controllers of activity **AD4**. Two global outputs can be produced in this activity: GO1 control programs and GO2 Concrete controllers. The activity can be decomposed as follows (Figure 3.32):

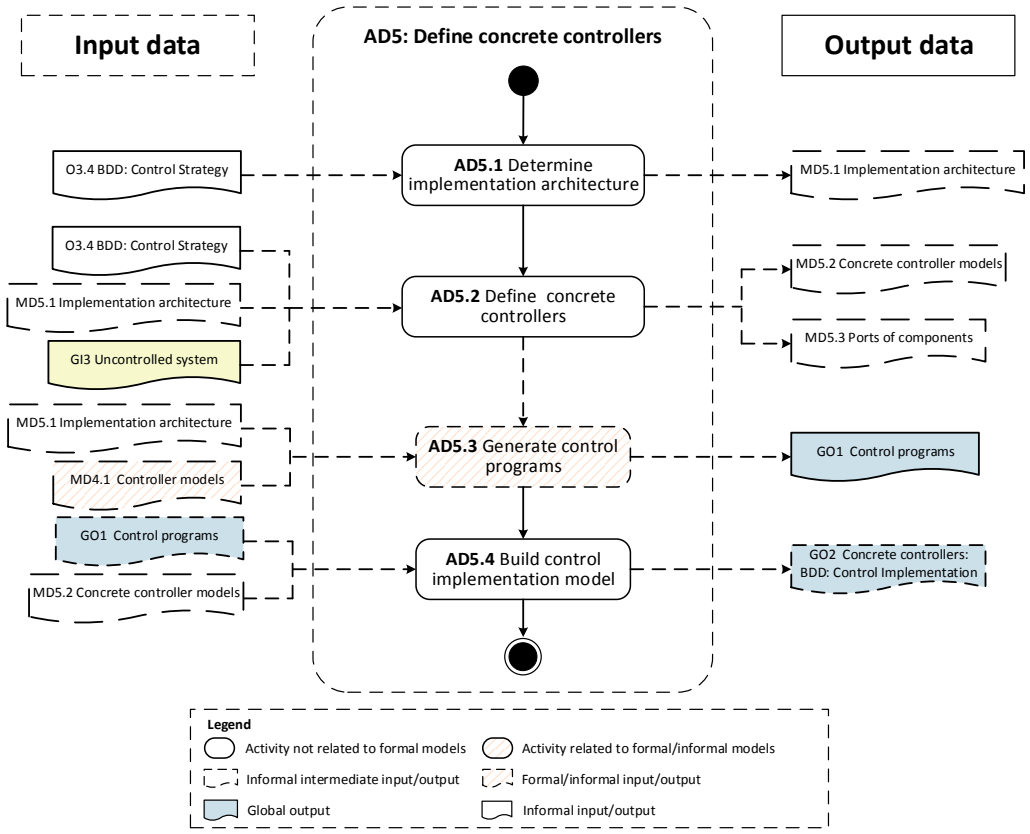


Figure 3.32 Workflow of sub-process of activity **AD5**

- **AD5.1** (Determine implementation architecture): As the fact that the structure of

supervisor/controller doesn't mean the structure of concrete controller, as control programs can be executed in one or several concrete controllers. The activity bridges the gap between supervisor/controller to concrete controllers. Due to the fact that there is no standardized implementation method from supervisor/controller to concrete controller, it is necessary to determine the implementation architecture (e.g. monolithic controller, distributed controller). In this step, we don't consider the details of the concrete controller, but determine a concept of which kind of architecture should be applied.

- **AD5.2** (Define concrete controllers): In this step, the details of architecture in **AD5.1** should be specified. Elements such as the concrete controller type (PLC, micro-controller…), the number of concrete controllers, communication ports, the structure and deployment should be defined. In order to determine the ports of concrete controller, the ports of plants should also be identified, which is based on the physic components by analyzing global input GI3.

- **AD5.3** (Generate control programs): In this activity, we focus on the realization of the controller by the software part of concrete controllers. The control programs should be executable in the designated concrete controllers. Besides, it is necessary to make sure that the control programs are generated for all global/local controllers. An appropriate program implementation method should be applied in this step (implementation methods can be seen in Table 2.2).

- **AD5.4** (Build control implementation models): A SysML BDD is built to graphically present the models of concrete controller by integrating the modeling elements in **AD5.2** and **AD5.3**. In the diagram, all concrete controllers and control programs should be presented.

In the model of system structure (Model 4), we just predefine concrete controller by decomposition to hardware and software as it is impossible to detail the detailed controller structure at the very beginning. Therefore, the purpose of BDD Controller Implementation is to make precise the controller after the supervisor/controller synthesis and programming. The metamodel of BDD control implementation is shown in Figure 3.33. The semantics of the elements in the diagram are defined as follows:

(1) The block hardware represents the concrete local controller and software is implemented by control program, which is represented by the block programs. The hardware should be presented by the designated physic controllers for implementation.

(2) To specify the execution relationship between concrete controllers and control programs, <<Allocation>> [145] is used to represent the cross-association between hardware and software.

(3) The ports for each concrete controller represent the control input/output to be connected to the components, which are indispensable elements in the following activity. The ports are defined by

referring to the control programs and physic components.

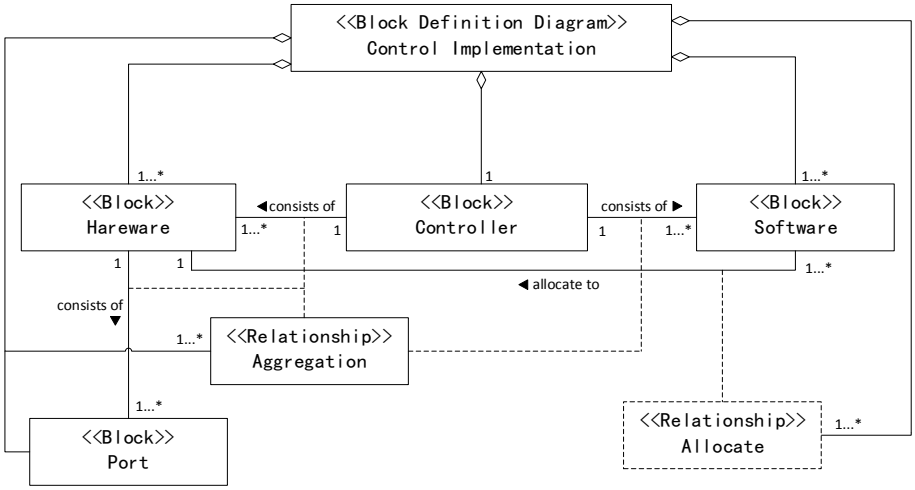


Figure 3.33 Metamodel of Control Implementation

3.4.4.6 Define Control Connection (AD6)

The objective of the activity **AD6** is to explicitly present the control connection between concrete controllers and physic components in order to clarify the physic control structure of the global controlled system. Three activities constitute activity **AD6** (Figure 3.34):

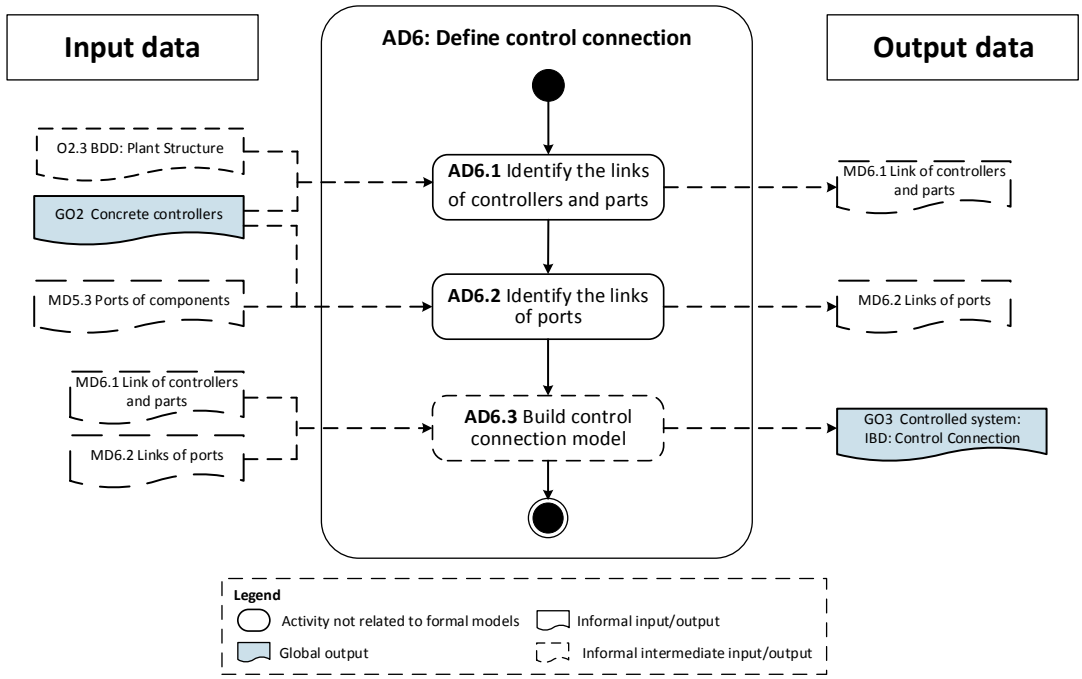


Figure 3.34 Workflow of sub-process of activity **AD6**

- **AD6.1** (Identify the links of controllers and parts): As the concrete controller can be

structured, it should firstly determine the connection at component level. In other words, the question “Which concrete controller controls which component?” should be answered. The identification is based on the diagram of plant structure and control implementation, which specify the control information.

- **AD6.2** (Identify the links of ports): When the links of controllers and parts are identified, the connections at signal level should be determined. The ports of physic components have already been identified and the ports of concrete controller are defined in activity **AD5.2**. Based on the results, the corresponding ports can be linked.

- **AD6.3** (Build control connection model): This last global output is produced in the activity and an IBD is built to present the control connections of the whole controlled system.

The metamodel of IBD: Control Connection is shown in Figure 3.35. Two physic elements are presented in the diagram: physic components and concrete controller. The connections of ports between components and controller should be linked. For convenience, the names of ports are defined to be related to the corresponding variable in control the programs. The type of the port is usually Boolean within the context of engineering.

In order to differentiate the command connection and signal connection, the port should follows the principle: for each command connection, the port of components should be input and port of concrete controller should be output. For each signal connection, the port of components should be output and port of concrete controller should be input. Besides, the connection can be colored to make distinction of connection type.

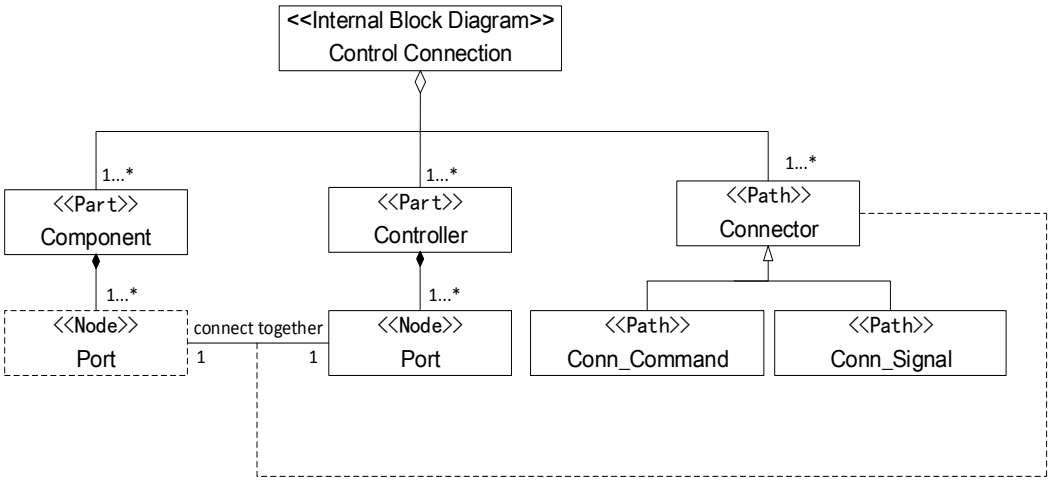


Figure 3.35 Metamodel of IBD: Control Connection

3.5 Conclusion

In this chapter, we proposed the integrated modeling framework aimed dealing with the problems and challenges mentioned in Chapter 2. Based on integration with SysML diagram, the modeling scope of SCT is extended. In order to determine the models which should be used to achieve a complete modeling for control system, we take the advantage of concept viewpoint to systematically clarify the necessary models for the target of AC. 10 SysML diagrams are defined as complementary models out of the scope of formal models so as to deal with the limitation of SCT (limitation 1-3). The proposed global modeling process (limitation 4) is introduced which involves all the models reflecting the 13 viewpoints. The elements involved in each models and the activities for each modeling step are detailed. To sum up, a collection of proposed solutions corresponding to the issues is presented in Table 3.4.

Table 3.4 A summary of proposed solutions

Issue	Proposed Solution
Limitation 1: Formalization	REQ: System Requirement REQ: Requirement Traceability
Limitation 2: Structure	BDD: Plant Structure BDD: Global Environment BDD: System Structure BDD: Control Architecture IBD: Architecture Structure BDD: Control Strategy
Limitation 3: Implementation	BDD: Control Implementation IBD: Control Connection
Limitation 4: Global Process	Proposed integrated modeling process

Although a number of SysML models and formal models are defined, the consistencies between them are not clarified. This issue, as initially discussed in Chapter 2 (Section 2.4.2), is involved in the entire modeling Section A and Section B, which focus on the formalization of plants and specifications respectively. The solutions for the issue will be detailed in the following chapter.

Chapter 4

Consistency of Formal Models and SysML Models

4.1 Introduction

In the previous chapter, the proposed modeling process for AC are presented, which integrates both formal models and SysML models. The activities and corresponding inputs and outputs of the global process are detailed. However, there is still one last key issue to be addressed: the link between formal models and SysML models. This issue has initially been discussed in Chapter 2 (Section 2.4.2) in which formalization problem has to be faced for plant and specification. In the proposed modeling process, the activities in both sub-process Section A and Section B are involved in the formalization problem. In this chapter, the following issues and solutions related to formalization will be presented in detail:

(1) The consistency between formal plants and physic components: in this part, the links between physic and formal elements will be defined, based on which the identification of plant models will be introduced. Secondly, in order to fully eliminate the inconsistency risk between plant and real behavior, a template-based approach will be proposed based on the SysML models.

(2) The consistency and traceability method between formal specifications and requirements: in this part, in order to make the best of SysML requirement diagram for requirement analysis and formalization, the traceability verification method will be proposed.

4.2 Consistency of Plants and Physic System

The identification and formalization of plant is one of the difficult steps, which deserves much care within the global modeling process. The supervisor synthesis procedures require the use of plant models that are consistent with the behaviors of the components. If these aspects are not expressed in the plant model, the synthesized controller may not function as expected [150]. In this section, we focus on two issues of the formalization of plant: (1) what kinds of physic components

should be identified as plant from the uncontrolled system? The question is closely related to the modeling activity **AF2.1.1** (Identify plant); and (2) if there is any solution to keep the plant models strictly reflecting the behavior of target physic components? The solutions for responding to these two problems will be detailed in this section.

4.2.1 Concepts

4.2.1.1 Hypothesis: Semantics of Event in Physic System

The formal plant in SCT defines a partition of event set into two disjoint subsets: controllable event set Σ_c and controllable event set Σ_{uc}. In the majority of contributions, the uncontrollable events and controllable events are usually supposed to be related to the signals from sensors and commands to actuators, in accordance with control laws implemented into a user program [20] [25] [30] and [150-152]. A schematic of link between physic system and formal models in typical case studies can be seen in Figure 4.1. The behavior of uncontrolled physic system is formalized by identified actuators and sensors. However, the details of the link are not specified in the existing contributions. In fact, the problem lead to difficulty to decompose and formalize the physic uncontrolled system and the trace between physic component and formal plant is still the ambiguous. For this purpose, we propose our solution in this study. The semantic link of formal event and SysML models can be defined as follows:

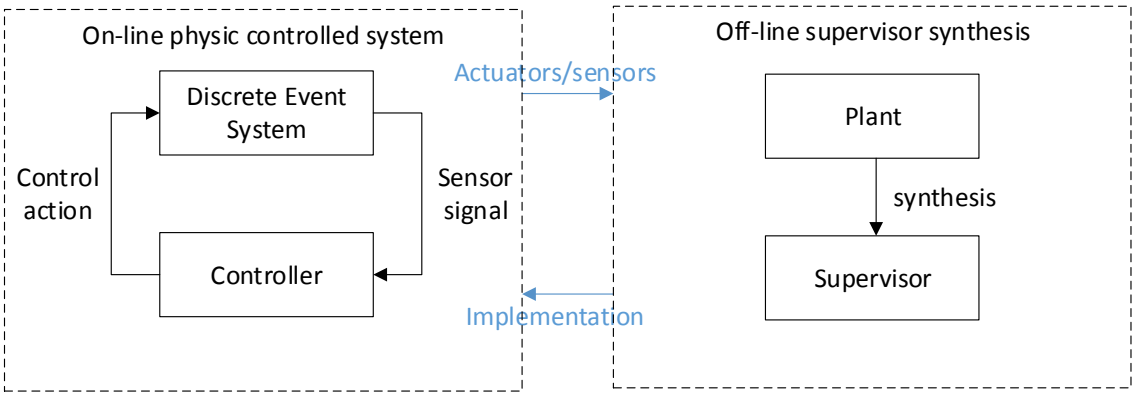


Figure 4.1 Schematic of link between physic system and formal models

Controllable event: A controllable event can be linked to an actuator of a component. The actuator is the execution part of the component. The semantics of a controllable event is the representation of an operation of the actuator triggered by the command signal from the controller.

Uncontrollable event: An uncontrollable event can be linked to a sensor of component. The semantic of an uncontrollable event is the representation of a signal of some change happening in the system detected by the sensor and sent to controller.

To sum up, the formal events have a bijective mapping to command/signal. However, the mapping between formal event and actuator/sensor is injective. In other words, the actuator/sensor can receive/send several commands/signals and each commands/signals can be formalized to the corresponding event. An example of the link between events and model elements are shown in Figure 4.2. The actuator can be formalized by two controllable events *CoEvt1* and *CoEvt2*, which usually represent the operations performed by the actuator. On the other hand, Sensor 1 is formalized by three controllable events (*UcEvt1*, *UcEvt2* and *UcEvt3*) to represent the three kinds of change detections.

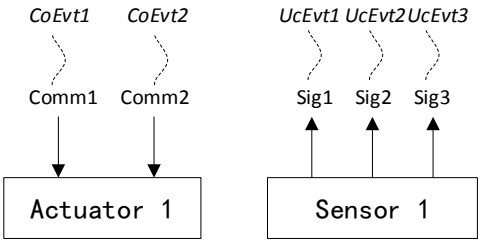


Figure 4.2 Example of link between events and model elements

Remark *When taking the situation into consideration where actors-humans are regarded as parts of the system-of-interest, the situation is more complex. To realize the actors-humans interacting with "machine", it is not enough to just identify the behaviors of human-beings. In fact, the interaction between human-beings and physic system must be performed with the help of Interaction components. For example, the event of the arrival of somebody should detected by sensor such as camera. Besides, if the system wants to order somebody to do the operations (like command), some indicator such as light or ring should be used. Therefore, it is still the case that the physic systems should be identified.*

4.2.1.2 From Physic Component to Plant

It is always the case that the plant models can contain both controllable events and uncontrollable event. In other words, the physic component formalized by the plant can consist of both actuators and sensors. Therefore, we define that the plant can be formalized for a physic component when at least one operations of actuator or one signal of sensor identified to interact with controller. To this end, in order to link the physic components to formal plants, all the operations of actuators and signal of sensors should be identified.

Figure 4.3 presents the element identification of a physic component, which is composed of *m* actuators and *n* sensors. For each actuator and sensor of the component, the corresponding operations and detection signals are identified respectively. The identification is related to the activity **AF2.1.1** (Identify plant), by which the physic components and formal plants can be linked

initially.

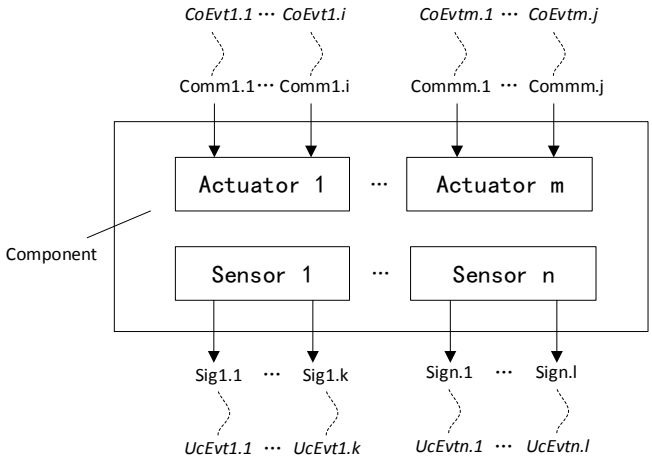


Figure 4.3 Composition identification of physic component

Remark *For example, buffer is usually a contradiction in the application. As a physic component of uncontrolled system, buffer is usually not formalized by plant, sometimes even by specification (e.g. [153]). In fact, the buffer in this study cannot be identified any operations of actuator and signals of sensor which can be commanded/detected by controller. Therefore, in the formal models, the buffer should be formalized as a plant.*

4.2.1.3 Orthogonal Sub-system

The relationship between actuators and sensors of the component are important for plant identification. In [20], the authors link the actuators and sensors by formalizing the relationship between them by synchronizing one or several automata, so that the behavior of component can be formalized by a global plant. The difference of the relationship and specification is the former is the physic nature of components to be identified and the latter is behavior constraint for the control target. However, there is problem where a physic component may not be formalized by only one plant. To extend this proposition, suppose a situation in which the actuators and sensors can be divided by several independent functional groups (Figure 4.4). Suppose a global component $\boldsymbol{C}$ and two corresponding plant models $G_1 = (Q_1,\ \Sigma_{c1} \cup \Sigma_{uc1},\ \delta_1,\ q_{01},\ Q_{m1})$ and $G_2 = (Q_1,\ \Sigma_{c1} \cup \Sigma_{uc1},\ \delta_1,\ q_{01},\ Q_{m1})$ represent the behavior of sub-system $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ respectively. If G_1 and G_2 are asynchronized, $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ are called *orthogonal sub-system.* In fact, there is possibility that a component $\boldsymbol{C}$ can be divided by several sub-systems. Therefore, it can be seen that the plants and physic components doesn't have a definite one-to-one relationship and the actuators and sensors should be carefully identified to make sure the formal plants built can reflect the behaviors of the components. The concept orthogonal specifies the independent relationship between the sub-systems/sub-components, which means that sub-components can perform concurrently. One

concept difference should be taken into consideration. Within SCT paradigm, the synchronization of two plants is similar to the behavior composition of two components. However, they are different concepts as the plant synchronization is a logic composition particular for supervisor synthesis while the behavior composition of components is concrete and physic.

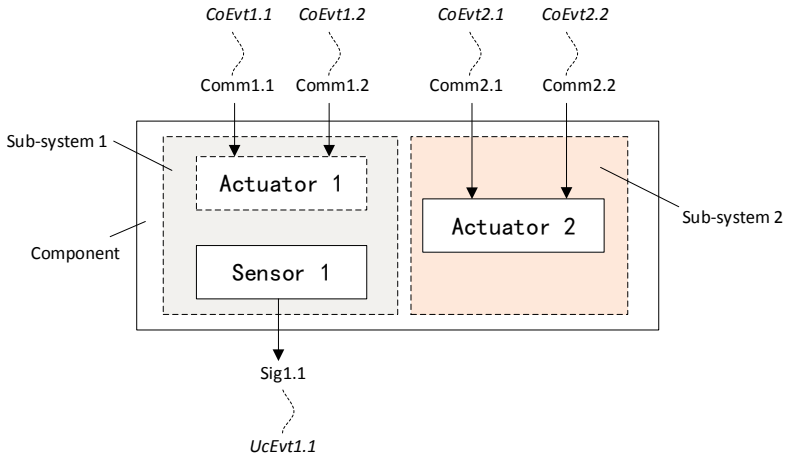


Figure 4.4 Example of orthogonal sub-system

To give a formal definition, suppose a component $\boldsymbol{C} = \{A_1, A_2, ..., A_m, S_1, S_2, ..., S_n, m, n \in \mathbb{N}\}$, where A_i and S_j are the actuators and sensors which constitute the component $\boldsymbol{C}$. Denote the set of operations of A_i that can be commanded by $\boldsymbol{Op_i}$ and the corresponding behavior model is defined by a plant $G_A^i = (Q^i, \Sigma_c^i, \delta^i, q_0^i, Q_m^i)$, then there is a bijective function $\boldsymbol{f}_A : \boldsymbol{Op_i} \to \Sigma_c^i$ which represents the mapping between operations and formal controllable events. Similarly, denote the set of signals of S_1 that can be detected by $\boldsymbol{Sig_i}$ and the corresponding behavior model is defined by a plant $G_S^j = (Q^j, \Sigma_{uc}^j, \delta^j, q_0^j, Q_m^j)$, and a bijective function $\boldsymbol{f}_c : \boldsymbol{Sig_i} \to \Sigma_{uc}^i$ which represents the mapping between signals and formal uncontrollable events. Denote an automaton by $R^k = (Q^k, \Sigma^k, \delta^k, q_0^k, Q_m^k)$ to represent the relationship which can be identified between the actuators and sensors to constrain their behaviors. For every relationship R^k, we have

$$f_r : \Sigma^k \to 2^{\boldsymbol{A} \cup \boldsymbol{S}}$$

The function represents that for each given R^k, a set of actuators or/and sensors can be mapped, which is based on the bijective function $\boldsymbol{f}_A$ and $\boldsymbol{f}_c$. Therefore, a plant can be identified by synchronization of $G_k = \boldsymbol{A} \,||\, \boldsymbol{S} \,||\, R^k$. In order to identify all the plants for orthogonal sub-systems of component, the relationship should be grouped. Denote $Group(R) = \{R^k, k = 1, 2, ..., n\}$ are group of relationship, where $\forall R^i \in Group(R), \exists R^j \in Group(R) \to \Sigma^i \cap \Sigma^j \neq \varnothing$. Denote that all the actuators and sensors involved the relationships in $Group(R)$ by $\boldsymbol{A}_{Group(R)} \cup \boldsymbol{S}_{Group(R)} = \cup \Sigma^k \to 2^{\boldsymbol{A} \cup \boldsymbol{S}}$. Therefore, a plant can be built by synchronization of $G = \boldsymbol{A}_{Group(R)} \,||\, \boldsymbol{S}_{Group(R)} \,||\, Group(R)$. To integrate the process, an algorithm for plant identification is presented in Algorithm 4.1.

Algorithm 4.1: Plant Identification

Input: Component $\boldsymbol{C} = \{A_1, A_2, ..., A_m, S_1, S_2, ..., S_n, m, n \in \mathbb{N}\}$

Output: a set of plant $\{G^i = (Q^i,\ \Sigma_c^i,\ \delta^i,\ q_0^i,\ Q_m^i) \mid i = 1,2,...,n\}$

Begin

If $m, n = 0$

return no plant identified;

End if

Identified operations/signals for each A_m and S_n;

Built formal models G_A^m and G_S^n for each A_m and S_n;

Identified relationship and build formal model R^k for each relationship;

Group R^k to $Group_i(R), i = 1,2,..,n$;

Compute $G_i = \boldsymbol{A}_{Group_i(R)} \,||\, \boldsymbol{S}_{Group_i(R)} \,||\, Group_i(R)$;

return G_i;

End

4.2.2 Identification Process

To realize the proposition above, the activity **AF2.1.1** should be detailed. Three sub-activities are involved in activity **AF2.1.1**:

- **AF2.1.1.1** (Identify actuator/sensor): This activity is to analyze all the components to be controlled based on the global input and BDD: System Structure, which specifies the plant candidates. This step focuses on the identification of actuators and sensors of each component to be controlled.

- **AF2.1.1.2** (Link to formal events): The operations/detections of each actuator/sensor are formalized. The plant information list which presents the results of **AF2.1.1.1** and **AF2.1.1.2** are built in this step.

- **AF2.1.1.3** (Identify orthogonal sub-system): Based on the results of previous two activities and analysis of the uncontrolled system, the orthogonal sub-systems for each component/sub-system are identified by grouping the actuators and sensors in the components. To this end, all the plant candidates are determined.

Therefore, in the following activity **AF2.1.3** (Link plant and component) and **AF2.1.4** (Build plant structure model), the BDD: Plant Structure is built to present the link between components and plants. In the diagram, the operations of components and events of plant should be explicitly presented to show the traceability.

Example 4.1 *Formalization of a robot with two degrees of freedom*

Suppose there is a robot with two degrees of freedom to be formalized in the global uncontrolled system. The robot can move along X axis and Y axis independently and two positions can be reached for each axis.

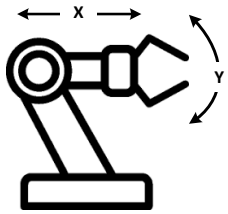


Figure 4.5 Robot with two degrees of freedom

The result of identification and formalization of robot is shown in Table 4.1.

Table 4.1 List of plant information

Component	Type	Description	Event Notation	Event Type
R1: Robot	Actuator1	Move to X+ position	pX+	Controllable
		Move to X- position	pX-	Controllable
	Actuator2	Move to Y+ position	pY+	Controllable
		Move to Y- position	pY+	Controllable

*In can be concluded from the system description that Actuator 1 and Actuator2 constitute two orthogonal sub-systems, and therefore, the robot should be modeled by two plants. In the BDD Plant Structure (built in activity **AF2.1.4**), two blocks of plants are built and allocated to part r1. Plant A and Plant B present the behaviors of the two orthogonal sub-systems. Four operations are identified and presented in the part r1 and for each plant, the events are also presented.*

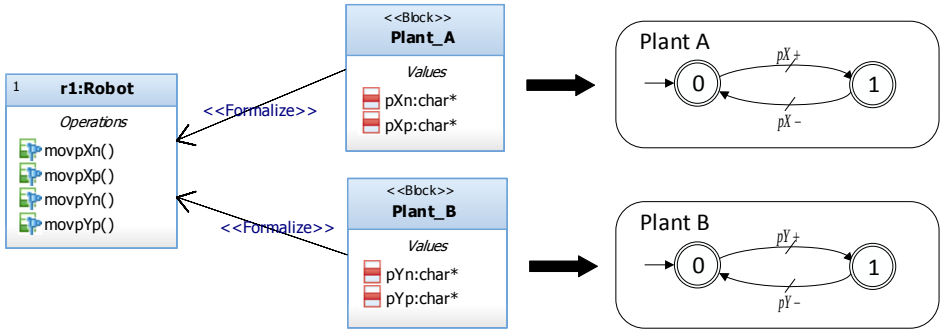


Figure 4.6 Plants of robot R1

4.2.3 Template-Based Approach

4.2.3.1 Objective

Although the identification process indeed improves the formalization of plant, it still cannot thoroughly guarantee the correctness of the model of plant. Unlike specification formalization, which can be traced by narrative requirements, plants must be formalized directly by analyzing the behavior of the given uncontrolled physic components. The modeling expertise still determines the quality of models in engineering practice. As discussed in Chapter 2 (Section 2.4.1), the parameterized template can be a solution to enhance the reusability of the modeling process. With the help of SysML BDD diagram, the parameterization method can be further improved. In practice, the components of the system to be controlled are what is "selected" rather than what is "designed", which conforms to the trend of COTS (Commercial Off-The-Shelf). The properties of each component can be identified in advance. The components of the system to be controlled can usually be standardized and the behaviors can be parameterized before modeling. For each specific system (e.g. manufacturing system, electrical system, etc.) to be controlled, a model library can be constructed based on the templates which represent typical behaviors of parameterized components. The library can be realized by the parameterized block in SysML. In other words, when the BDD System Structure is built, the behavior models of components are automatically built at the same time. Therefore, we can benefit from this approach:

(1) As the plant models are automatically built, the time-consuming problem of formalization can be solved. Compare with the existing parameterization methods, a combination with SysML BDD eliminate the modeling of abstracted parameterized models;

(2) The template-based approach guarantees that the plant models can strictly reflect the behaviors of physic components, regardless the human performance. The library constructed in advance contains a number of predefined parameterized behavior models for concrete components, which are made by expertise.

4.2.3.2 Principle

To realize the proposed approach, a three-level model template for each component is proposed. A collection of templates constitutes the library. The principle of the template is that it parameterizes the typical behavior of the corresponding physic component and is instantiated when the component is given. The three-level model template consists of a SysML model, an interface and a plant model (Figure 4.7). The SysML block template is at front end directly used by the designer, which encapsulates the plant model. Between the two levels, the interface should be constructed to

transform the SysML template into SCT model. When modeling a system to be controlled, the user models each component by SysML template and organizes them in BDD. Then this front-end model will be transformed to the SCT model and computer automatically. The template in this study is defined as follows:

Definition 4.1 (Template). A template of component T_C is defined by a three-tuple as follows:

$$T_C = (B_C, t_C, P_C)$$

Where, B_C is the parameterized template block;

$t_C : B_C \rightarrow P_C$ is transformation method of each template;

$P_C = (Q_C, \Sigma_C, \delta_C, q_{0C}, Q_{mC})$ is the plant model of component.

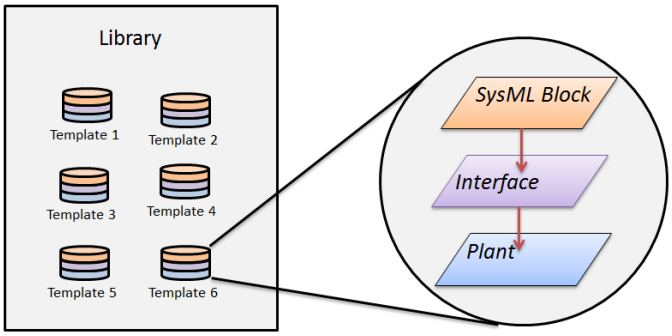


Figure 4.7 Schematic of the three-level template and library

Definition 4.2 (Parameterized Template Block). A parameterized template block B_C is defined by a three-tuple as follows:

$$B_C = (BName_C, part_C, PAR_C)$$

Where, $BName_C$ is the block name;

$part_C$ is part name of component;

PAR_C is a set of additional parameters.

The block name can be used to classify the template of block, based on which we can determine the corresponding template in the library. In practice, the block name should be predefined in the library and selected by the designer for a specific design case. Secondly, part name is another parameter as the input of transformation. For each component which shares the same block, part name is the unique identifier to differentiate the component and becomes the subscript of the elements in the corresponding behavior model. Therefore, the plants of components can make sure to be asynchronous, which satisfies the following assumption [67]:

$$\forall i \neq j, G_i, G_j \in Module : \Sigma_i \cap \Sigma_j = \varnothing$$

Thirdly, a number of additional parameters are defined for each block template to realize the

parameterization modeling. The parameter can be statistical quantities or part names of other components. For example, in a manufacturing system, two workstations which have the same block name <<Machine>> are supposed to have different parameters. Workstation 1 has two parameters A and B which represents that it process part type A and part type B while Workstation 2 processes only part type C. When transforming to plant models, the states, events and transitions in automaton vary according to different parameters.

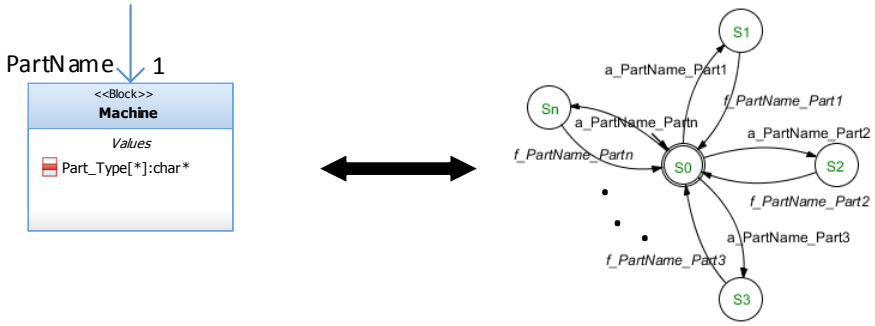


Figure 4.8 Example of parameterized component

An example of parameterized block is shown in Figure 4.8 (one the left). The parameterized block in the figure represents a workstation in the manufacturing system. The block name <<Machine>> mapped to the template in the library can be realized by the parameterized automaton on the right. The automaton represents the typical behavior of a machine, where two kinds of events are involved: the starting of processing a part and the end of processing a part. The automaton can be determined when the corresponding parameter in the block is given: (1) The block name determines the template to be used in the library; (2) The number of part type to be processed determines the number of state of automaton; (3) The part name of component and parts to be process by the machine determined the events and transitions of the automaton.

In order to deal with the situation of structured system, a special stereotype <<System>> or <<Subsystem>> is provided in the library to represent the system/sub-system. The block labeled by this stereotype won't be transformed to any plant, but records the components which are composed of this system/sub-system and then the components in the system/sub-system are added to the list for transformation. Besides, the template can also be able to deal with the orthogonal sub-system of a component, which can automatically create the corresponding set of plants

The transformation method t_C of the template defines a series of operations, which are automatically performed by template itself, that map the parameters in B_C to the elements which constitute plant P_C. Taking the example in Figure 4.8 once more, the transformation method of machine $t_{machine}$ is defined as follows:

$t_{machine}$: *Automaton Type = plant*;

$X = \{0,1,2,...,k\}$, *k is the number of part type*;

$$\Sigma = \Sigma_c : \{a\_PartName\_Partn\} \cup \Sigma_{uc} : \{f\_PartName\_Partn\},\ n = 1,2,...,k\ ;$$

$$\delta = \{(0, p, n) \mid p = a\_PartName\_partn\} \cup \{(n, q, 0) \mid q = f\_PartName\_partn\},\ n = 1,2,...,k;$$

$$x_0 = \{0\},\ X_m = \{0\}.$$

4.2.3.3 Implementation

Object Management Group (OMG) provides a standard for exchanging metadata information XML Metadata Interchange (XMI) for SysML model [154], which is commonly used as the medium by which models are passed from modeling tools to software generation tools as part of model-driven engineering. Therefore, the BDD System Structure diagram can be interpreted by XMI file and transformed to a XML file recording automaton models. The architecture of the implementation of transformation method is presented in Figure 4.9.

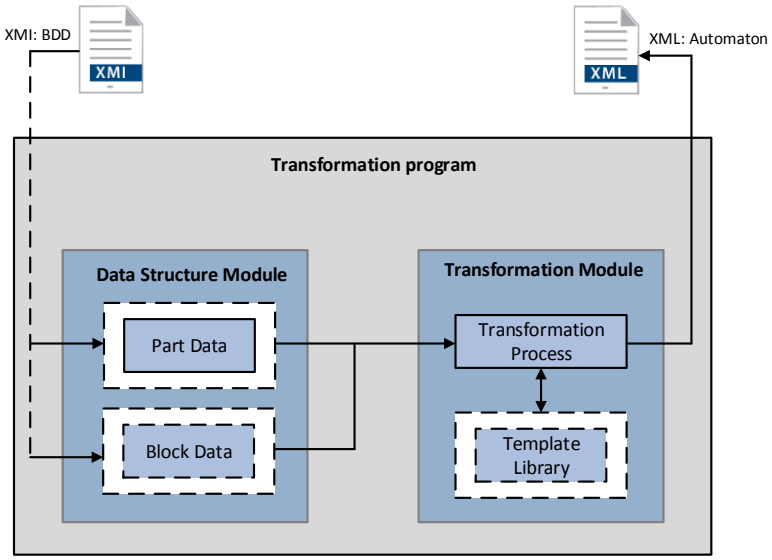


Figure 4.9 Program architecture

The transformation is composed of two modules: data structure module and transformation module. The responsibility of the data structure module is to parse the input XMI file, structure and store the required data for transformation. The data structure can be further decomposed into part data and block data. Both of them are highly abstracted data structure, which can be regarded as unified data models applicable for the AC of all system types. Two kinds of information are recorded for each block: the blocks and their corresponding identification number, which is used to match the corresponding template in the library. Three kinds of information are recorded in each: the part name, the block and parameters. This information is important for template matching and creating corresponding plants.

The transformation module is in charge of processing the data and constructing the corresponding

automata based on the template library (presented in Figure 4.10). For each part recorded in the process, a template is matched by block information. When a template can be matched in the library, the parameters in the part should be written in the template to instantiate a automaton for the plant model. The process ends when all parts are processed.

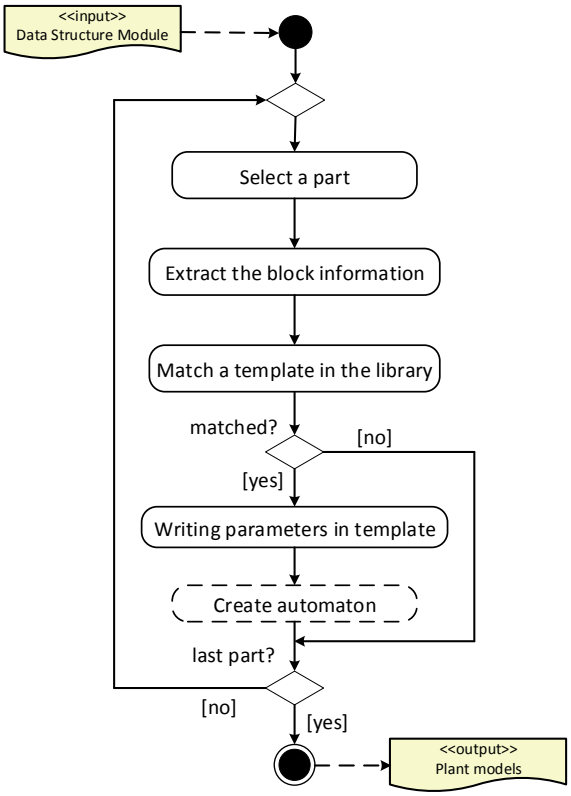


Figure 4.10 Transformation process

Due to the template-based approach, the modeling process of Section A can be simplified. In the adapted process Section A', the former activity **AF2.1** (Identify plant structure) is no long necessary and the former activity **AF2.2** (Build formal plants) can be directly performed (new number **AF2.1'**) as the identification of plant information can be replaced by the predefined templates. When finishing the modeling of **AR2.2**, a library is imported to transform the component defined in the BDD: System Structure to the plants models.

4.2.4 Summary

The identification of actuators and sensors of physic systems is important to determine the plant models as the components can be structured and formalized by several plants. The proposed method can be used to decompose the system behaviors and the corresponding plant models can be built. Besides, an adapted modeling process based on template-based approach is proposed to guarantee that the results of plants can correctly reflect the behaviors of physic components and the modeling process can be improved.

4.3 Consistency of Formal Specifications and Requirements

As is concluded by the contribution review, a gap exists between the informal requirements described by natural language and specification represented by formal model and it is necessary to construct the consistency of requirements and formal specifications. With the help of SysML requirement diagram, the informal narrative requirements can be analyzed and decomposed by semi-formal SysML models. Three levels of the requirement decomposition are defined in the Chapter 3. In this section, the main objective is to present in detail the E-specification and E-requirement, based on which the method for requirement traceability by formal specifications is proposed.

4.3.1 E-specification

A formal specification, constructed based on automaton with the elements such as states, transitions, must be verified its consistency with E-requirements. The narrative requirement description, however, can be expressed in different ways. For example, a requirement is supposed to have a quantitative specification, which cannot be verified on the consistency with formal specification because the latter can just represent the logic behavior or function. Besides, the narrative description precision should also be taken into consideration. In fact, it is possible to translate all transition in the automaton by natural language, but this work is meaningless. Due to the diversity of specification expression of nature language, it is necessary to define a standardized specification for the semantic consistency of automaton-based specification.

To deal with problem of the format of textual description, a relevant solution STAMP (Systems Theoretic Accident Modeling and Processes) model is introduced in [155] [156]. According to STAMP model, requirements is specified using a constrained structure natural language by format as *Requirement = Controller + [Constraint] + Controlled Process*. Another proposition called CARE (Concern-Aware Requirements Engineering) introduced in [157], provide a format of requirement as *Requirement = Subject + Verb + Target + [Way]*. In both solutions, either *Controller* or *Subject* represents the agent (or resource) who executes the behavior or change the state. However, in order to taken into account the context of AC and the link with formal specifications, the description of requirement should contain:

- One or several involved subjects: subjects represent the target physic component or sub-system, which determines what component or system should perform the designated function; the subject must be within the scope of plants.

- Event: the event described in the text should be high-level abstraction as the event in formal model rather than the component-oriented one. Therefore, the event can be

directly mapped to the counterparts in formal specification.

- Function: function indicate the behavior constraint on free behavior of subjects to realize the designated functional objectives.

- Condition: the condition represents the restrictions to which the function of the subject should respect. The description can be the function of another subject.

Definition 4.3 (E-specification). A textual execution specification ***ESpec*** is defined as follows:

$$\boldsymbol{ESpec} = Sub + Evt + Fun + [Cond]$$

Where, *Sub* are corresponding subjects;
Evt are the events which happen under specification;
Fun represents the actions or logic functions which subjects perform.
Cond represents the condition, whose structure is $Conj + Sub + Evt + Fun$.

The condition is written by starting with the conjunction such as “after”, “when” “Once” or “if”. Each E-specification should describe the definite logic function, which articulately expresses a unique objective of the behavior constraint. Secondly, in order to keep consistency between E-requirement and formal specification, the E-specification should be identified by formalism, which will be of vital importance for traceability verification in the following activity.

4.3.2 E-Requirement

The informal narrative requirements can be modeled by SysML requirement models. By requirement model, we can perform the decomposition and explicitly describe the relationships between requirements. Therefore, we need to define an E-requirement model which contains E-specification.

Definition 4.4 (E-Requirement). An E-Requirement *EReq* is a requirement whose textual specification is ***ESpec*** . The E-Requirement can be defined 4-tuple,

$$EReq = (Name, ID, ESpec, Re)$$

Where, *Name* denotes the name of requirement;
ID denotes a unique identifier of requirement;
*E S p e*denotes the P-specification;
Re denotes the relationship with other requirement.

Figure 4.11 shows the complete definition of E-requirement by Metamodel. E-requirement is generalized from functional requirement, which specifies an operation or behavior that a system,

or part of a system, must perform. Like a common requirement, an E-requirement uses a requirement name to briefly introduce the subject and ID for identification.

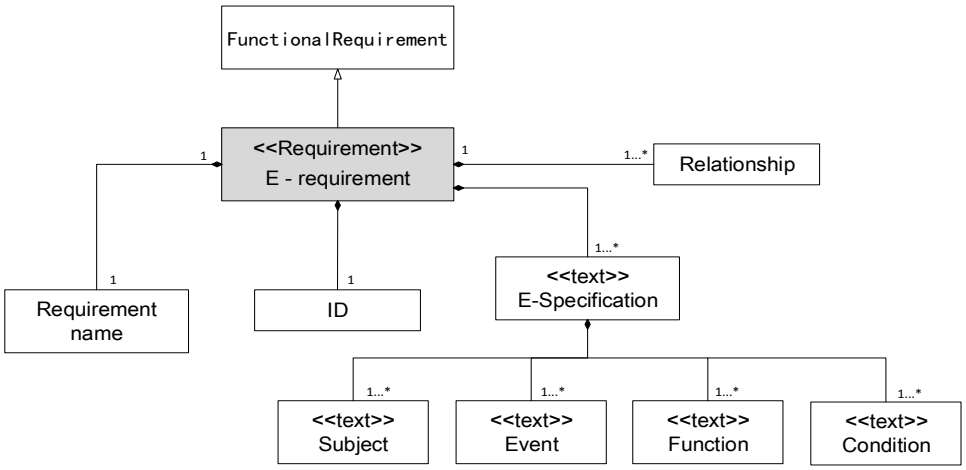


Figure 4.11 Metamodel of E-Requirement

EReq is nothing but a special requirement which contain P-specification. *EReq* helps to construct the link between automaton specification model and requirement in SysML. Each description of E-requirement can contain several E-specifications.

4.3.3 Requirement Definition Diagram

A requirement diagram should be built to present the system requirement definition and decomposition (in activity **AR1.2**). The metamodel of system requirement diagram is shown in Figure 4.12. In this diagram, the system requirements of three levels and the relationship between requirements are described. All high-level mission requirements are decomposed to leaf sub-requirements which are defined as E-requirements in order to make the consistency with formal specifications.

Three kinds of requirement relationship are used in the diagram:

- *Derive*: The relationship (<<Derive>> or<<DeriveReqt>>) usually consists in linking requirements of different levels (e.g. requirements from system to sub-system). The requirement between different levels (mission, function, execution) can be linked by *derive* in the diagram. Therefore, in the requirement diagram, the relationships of requirements between different levels (i.e. mission, function and execution) should be presented by <<Derive>>.

- *Contain*: A requirement may be decomposed into one or more sub-Requirements, for example when the requirement is not atomic in nature and it is desired to decompose it into a number of related atomic sub-Requirements. Therefore, the relation is used to link the requirement

within the same level in the diagram.

- *Refine*: The refine requirement relationship can be used to describe how a model element, or set of elements, can be used to further refine a requirement. The refinement of a requirement is to describe at more precise level. In order to verify the traceability, the E-requirement must be refined so as to be linked to formal specifications.

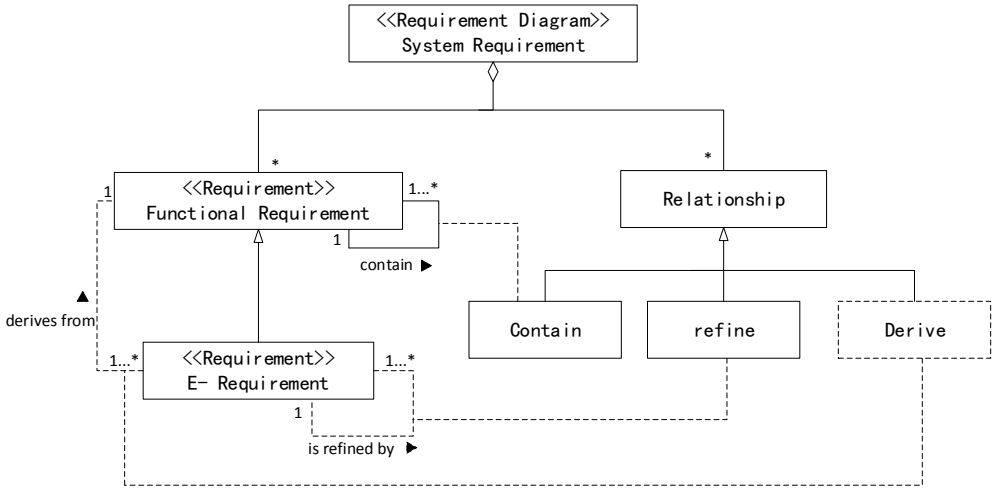


Figure 4.12 Metamodel of requirement diagram

4.3.4 Requirement Traceability

4.3.4.1 Principle

With the help of SysML requirement models, informal requirement descriptions can be transformed to the semi-formal models, which can be made of use to realize decomposition, trace and verification. SysML supports requirement relationship expressed by <<derive>> and <<contain>> to describe the requirement decomposition and analysis. In this study, it is assumed that each root E-requirement is formalized by a corresponding formal specification. Therefore, the remaining issue is to verify that a specification in form of automaton does capture the description nature of the E-requirement. As the description of each requirement is still informal, this issue is indeed to verify the links between informal E-specification and automaton. The temporal logic seems to be a possible solution since it offers simple syntax and semantics for descriptively writing formulae that are paraphrastic in natural language [158]. As a solution introduced in Chapter 2, in [114] [123], the authors proposed a requirement decomposition and validation method which combines both informal and CTL*-based formal specifications. The informal parts in graphical representation are described by SysML requirement diagram. However, the CTL* formulae is clear from the topmost requirement in [114] [123]. As the requirement decomposition in this study

is from ambiguous descriptions to explicit ones, it may be difficult to interpret the requirements (even if the leaf requirements) by formulae of CTL* and therefore leads to the difficulty of formalizing E-requirement and verifying the traceability between narrative requirements and formal specifications.

As the formal model of specification is automaton rather than CTL*, another issue is if temporal logic can be used as the intermediary models for interpretation between informal description and automaton. In [159], the author proposed an approach analyze and synthesize supervised discrete event systems using LTL. In this approach the language generated by automaton is identified and verified the consistency with temporal logic. Based on the proposed algorithm, the automaton which is consistent to the semantic of temporal logic can be constructed. However, the complete formulae to express an automaton will be too complex (the example in the article is just an automaton with three states), which violates to our global purpose for applicability in engineering practice. Secondly, the similar proposition cannot be found so that if there is a direct link between LTL and automaton remains questionable. In fact, the semantic of specification within SCT paradigm is hardly identified, which leads to the difficulty to link to temporal logic.

Indeed, it is still difficult to construct bijective link between narrative requirements and formal specification as the fact that the internal function of specification is difficult to be exhaustively interpreted by natural language. However, it is possible use the SysML requirement diagrams to present the consistency and traceability between narrative description and specification. The solution in this study returns to the semantic nature of automaton. Based on the decomposition of high-level requirements and the description can be detailed as far as possible. When the formal specification is verified its consistency with low-level requirements and the relationships between low-level requirements and high-level requirements, the traceability can be confirmed. For this purpose, two issues should be discussed:

Normalization of narrative description

Previously, the E-specification is specified by a basic description structure. In order to interpret E-specifications formally, the description should be further normalized. Suppose that the narrative description can be formalized by a *deterministic successive string* of events (or called *trajectory*), denoted by $S = e_1 \to e_2 \to ... \to e_n$, where, $\to$ means that e_2 is the direct successor event after occurrence of e_1. To be rigorous, it is necessary to verification the string is the only string generated by specification by starting from event e_1. In other words, whenever the string $\{e_1\sigma \mid \sigma \in \Sigma \wedge \sigma \neq e_2\}$ is generated by specification, it says that the specification is inconsistent with the requirement.

By taken into the consideration the context of control specification, typical description normalization is introduced in Table 4.2.

Table 4.2 Typical descriptions of E-specifications

Description	Formalism	Semantics
A **and** *B*	$S_a \wedge S_b$	Two situations must be satisfied together.
Once *A*, (**then**) *B*	$S = e_a \rightarrow e_b$	The following operation will immediately occurs
Whenever *A*, (**then**)*B*		
B **after** *A*	$S = e_a \circlearrowleft e_b$	The following operation will occurs. (weak)
Repeat *A* and *B*	$S = (e_a \rightarrow e_b) \wedge (e_b \rightarrow e_a)$	*A*, *B* occurs alternatively and repeatedly
Once *A*, (**then**) *B* **or** *C*	$S = e_a \rightarrow (e_b \vee e_c)$	Two alternative following situations
Initially *A*	$S = \rightarrow e_a$	Operation occurs From initially state.
Whenever *A*, **then never** *B*	$S = \neg(e_a \rightarrow e_b)$	Operation never occurs successively.

In the Table 4.2, $s = e_1 \circlearrowleft e_2$ present the weak successor where, it is allowed to occur the events in the self-loop between e_1 and e_2. Furthermore, to extend the expression of the formalism, each atom events can be replaced by string. For example, $S = e_a e_b \rightarrow e_c$ means that whenever $e_a e_b$ occurs, e_c is the unique successor event. We define the equation $(e_a \vee e_b) \rightarrow e_c = e_a \rightarrow e_c \wedge e_b \rightarrow e_c$ holds and $e_a \rightarrow (e_b \vee e_c) \neq e_a \rightarrow e_b \wedge e_a \rightarrow e_c$**.** The semantic of these two formalisms is presented in Figure 4.13. In fact $e_a \rightarrow (e_b \vee e_c)$ means that after occurrence e_a, there are two paths to be studied.

(a) $(e_a \vee e_b) \rightarrow e_c$ (b) $e_a \rightarrow (e_b \vee e_c)$

Figure 4.13 Semantic of two formalisms

By abuse of the notation, the specification is consistent with formalism of requirement is denoted by $H \models S$. Therefore, it is necessary to verify that specification $H \models S$ when given a formalism S interpreted from a narrative requirement. Denote a specification by $H = (Q, E, \delta, \Gamma, q_0, Q_m)$, where,

$\Gamma: Q \to 2^{\Sigma}$ is active event function $\Gamma(q) = \{\sigma \mid \delta(q, \sigma)!\}$. The formalism is defined by $S := e \mid E^* \mid \neg S \mid S \to S \mid S \vee S \mid S \circlearrowleft S \mid \to S$ and $\hat{q}(E^*) = \delta(\Xi, E^*)$ is defined as the state when E^* occurs from arbitrary state. Therefore, in order to verify the consistency between formalism and specification, the Algorithm 4.2 is proposed.

Algorithm 4.2: Consistency verification

Input: Formalism $S = S_1 \to S_2 \to ... \to S_n$ and formal specification $H = (Q, E, \delta, \Gamma, q_0, Q_m)$

Output: $H \vDash S$ or $H \nvDash S$

Begin

(1) $H \vDash \to \vee e_i \Leftrightarrow \Gamma(q_0) = \{e_i, i = 1, 2, ..., n\}$;

(2) $H \vDash \vee e_i \to e_2 \Leftrightarrow \wedge \Gamma[\hat{q}(e_i)] = \{e_2\}, i \in \mathbb{N}$;

(3) $H \vDash e_i \to E^* \Leftrightarrow H \vDash e_i \to e_1 \to e_2 \to ... \to e_n, E^* = e_1 e_2 ... e_n$;

(4) $H \vDash E^* \to \vee e_i \Leftrightarrow \Gamma[\hat{q}(E^*)] = \{e_i, i \in \mathbb{N}\}$;

(5) $H \vDash E^* \to \vee S_i \Leftrightarrow \wedge \Gamma[\hat{q}(E^*)] = \{e_i \mid S_i \text{ is initial by } e_i\} \wedge E^* e_i \to S_i / e_i, i \in \mathbb{N}$;

(6) $H \vDash e_1 \circlearrowleft e_2 \Leftrightarrow \Gamma[\hat{q}(e_1)] / \circlearrowleft = \{e_2\}$;

(7) $H \vDash \neg(e_1 \to e_2) \Leftrightarrow \Gamma[\hat{q}(e_1)] \cap e_2 = \varnothing$;

$i \leftarrow 0$;

Expand S_i and compute by (1) - (7)

If $S_i \vDash S$

  Loop (i <= n)

    Expand $S_i \to S_{i+1}$ compute by (1) - (7) ;

    If $S_i \to S_{i+1} \vDash S$

      $i \leftarrow i + 1$;

    Else

      return $H \nvDash S$;

    End if

  End loop

Else

  return $H \nvDash S$;

End if

return $H \vDash S$

End

E-Requirement decomposition

The high-level requirement description may not be interpreted by the proposed formalism as the description may be unclear. The SysML requirement diagrams provide a series of relationship which help to present the requirement decomposition by different semantics. Two situations should be faced analyze the requirements:

(1) Containment (Composite Requirement)

In this situation, the E-requirement cannot be directly interpreted by formalism but can be decomposed into sub-requirements. It is assumed that the description of each sub-requirement can be identified by a formalism, then the root E-requirement can be identified by conjunction of all the formalism of sub-requirements. A composite requirement can contain sub-requirements in terms of a requirements hierarchy, specified using the namespace containment mechanism. The semantics of a composite requirement may state that the system shall do A and B, which can be decomposed into the child requirements that the system shall do A, and the system shall do B.

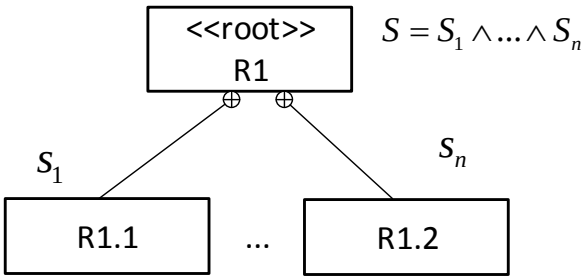


Figure 4.14 Requirement decomposition by <<Contain>>

For precisely definition, suppose there is a set of sub-requirements of the root E-requirement $Sub(R_1) = \{R_1^1, R_1^2, ..., R_1^n \mid n \in \mathbb{N}\}$. For each sub-requirement R_1^n, a corresponding formalism is identified, denoted by S_n. Therefore, the formalism of the root E-requirement can be denoted by S and $S = \wedge_{k=1}^{n} S_k^1$ holds [160] (shown in Figure 4.14). Then the formal specification *H* can be verified when

$$H \vDash \wedge_{k=1}^{n} S_k^1 \Leftrightarrow H \vDash S_1^1 \wedge H \vDash S_2^1 \wedge ... \wedge H \vDash S_n^1$$

When the sub-requirement can also not be interpreted, the requirements should be decomposed to deeper levels. Denote $\wedge(S_i) = \wedge_{k=1}^{n} S_k^1$, then

$$H \vDash S \Leftrightarrow H \vDash \wedge ... \wedge (S)$$

Specification should satisfy all the formalism of leaf E-requirements. As all the sub-requirements are independent with each other, there can be two ways for verification. The first is to directly verify the requirements from the bottom level. When all the bottom level requirements are verified, the root requirement can be regarded as verified. The second is to just identify the formalism from the bottom level and return the results to upper level. The global formalism is verified at the end.

(2) Refinement

In this situation, the E-requirement can be partially interpreted by formalism, but a part of the formalism is still unclear. The problem usually occurs when the description cannot details all the

function as it may be too complex. Therefore, the relationship <<refine>> is used to present the one requirement provides more precise details of the other.

Suppose the requirement R_1 can be identified by a global formalism $S = f(S_1,...,S_n)$, where S_n is the formalism which cannot be identified. Denote the set of requirement, which can refine the root requirement, by $Ref(R_1) = \{R_1^1, R_1^2, ..., R_1^n \mid n \in \mathbb{N}\}$. For each R_1^n, a corresponding formalism is identified, denoted by S_n (shown in Figure 4.15). Therefore, the formalsim S of root E-requirement can also be identified. Different from the previous situation, this root requirement must be verified integrally as each S_n is just a part of the global formalism. However, similarly the refinement can also be analyzed to deeper levels.

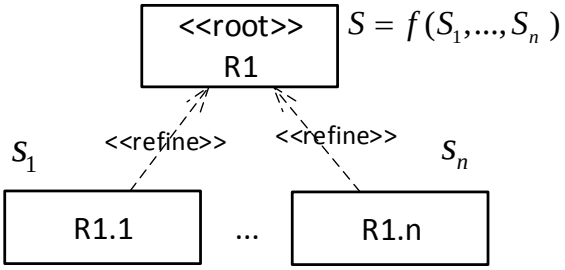


Figure 4.15 Requirement decomposition by <<refine>>

In the requirement diagram, the combination using of relationship <<contain>> and <<refine>> can be applied to present the complex requirement analysis. Example 4.2 shows an analysis of an E-requirement which is decomposed in multi-levels.

Example 4.2 *Example of Analysis of E-requirement R1*

The root requirement R1 can be identified by formalism S*. The formalism cannot be incomplete and two formalisms* S_1, S_2 *remained to be determined. Therefore, two requirements R1.1 and R1.2 are linked to R1 which are identified by* S_1 *and* S_2 *respectively. However, R1.2 can also not be identified and can be decomposed to R1.2.1 and R1.2.2 which can be finally interpreted by* S_3 *and* S_4*. The final result of the global formalism is* $S = f(s_1, s_3 \wedge s_4)$.

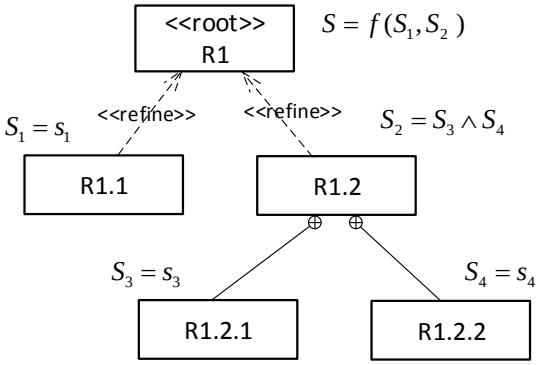


Figure 4.16 Example of analysis R1

4.3.4.2 Traceability Verification Method

The requirement modeling, analysis and verification are performed together within the same iteration involving the global process of Section A. The verification process must indicate the fault information of the verification result. There are two possibilities of the fault of verification. In the first situation, not all the requirements can be interpreted by formalisms, even if the leaf requirements. Therefore, it is impossible to check the consistency and the requirement diagram should be modified. Secondly, when all the requirements can be interpreted, the checking can be performed. The fault of checking must be the problem of formal specifications inconsistent to requirements and it is consequently necessary to modify the corresponding specification models.

The algorithm of verification for one specification is defined in Algorithm 4.3. In the algorithm, a recursion function is defined to identify requirements by formalisms at all levels. When the final return value of $H \models \text{Formal}(R_0^0)$ is **TRUE**, the consistency of E-requirement R_0^0 and formal specification H is verified. Algorithm 4.3 can export the two kinds of error: H, R_i^j. The return value H represents the Error 1 in which the H is inconsistent with R_i^j. Therefore, the activity **AF1.1** (Build formal specifications) should be returned to modify the problem specifications. When the return value R_i^j, the formalism of R_i^j is cannot be defined and the activity **AR1.2** (Analyze system requirements) should be returned to update the requirements.

Algorithm 4.3: Requirement verification

Input: Root E-requirement R_0^0 and formal specification $H = (Q, E, \delta, \Gamma, q_0, Q_m)$

Output: Verified or Error

Begin

$i \leftarrow 0,\ j \leftarrow 0$;

Def Formal (R_i^j):

Identify R_i^j by S_i^j ;

If Identified successfully

return S_i^j ;

Else if $Sub(R_i^j) = \varnothing \wedge Ref(R_i^j) = \varnothing$

return R_i^j ;

exit;

Else if $Sub(R_i^j) \neq \varnothing$

return $\wedge$ Formal ($Sub(R_{++i}^j)$);

Else

return f (Formal ($Ref(R_{++i}^j)$));

End if

End Def

If $H \vDash$ Formal (R_0^0)

return TRUE;

Else

return H;

End if

End

It can be concluded that the requirement verification is an iteration process which involves a series of activities in Section B, including activity **AR1.2** (Analyze system requirements), **AF1.1** (Build formal specifications) and **AF.1.2** (consistency verification). The activity detail of **AF.1.2** is shown in Figure 4.17, which is composed of three activities:

- **AF1.2.1** (Load requirement and specification models): The REQ: System Requirement and specification models are loaded as input of verification algorithm.
- **AF1.2.1** (Perform verification algorithm): The consistency between each root E-requirement in the system requirement diagram and the corresponding specification must be verified in sequence based on Algorithm 4.3.

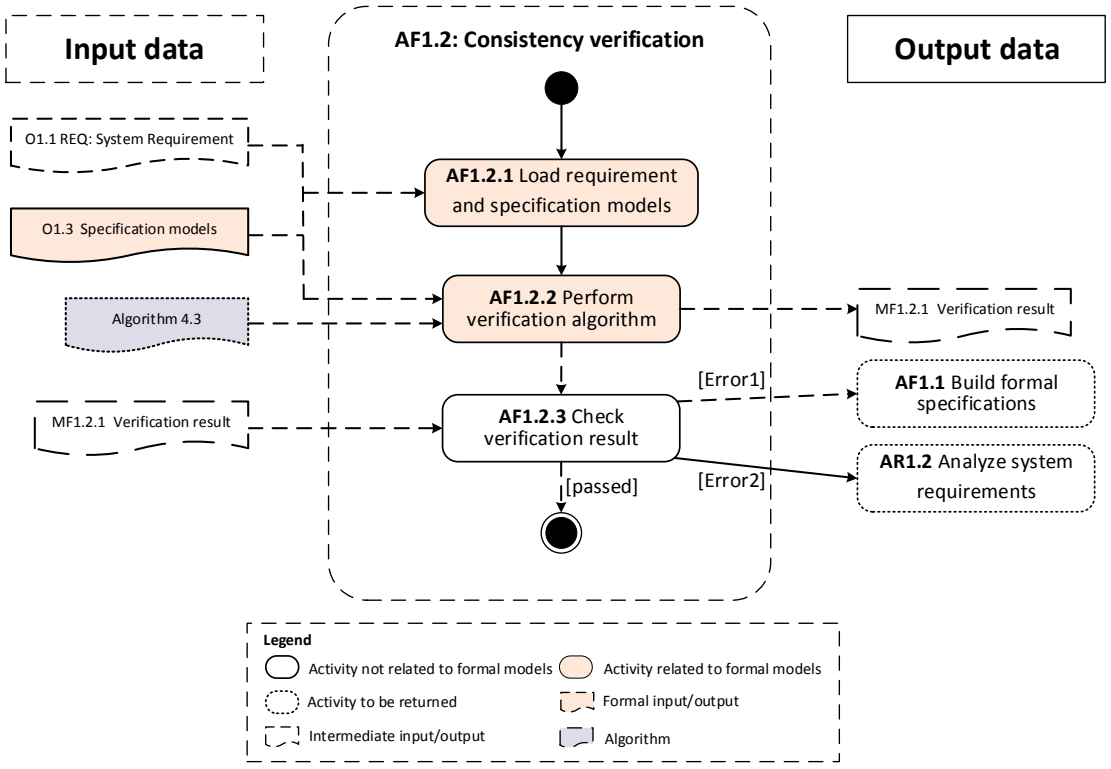


Figure 4.17 Workflow of sub-process of **AF1.2**

- **AF1.2.3** (Check verification result): As Algorithm 4.3 exports the information of verification result, the inconsistent part between requirements and formal specifications can be known and therefore, either requirements or specification can be modified accordingly. If it is indeed verified, then the requirement traceability can be confirmed and the process can be moved to activity **AF3** (Confirm traceability). On the other hand, if an error is found during the verification, the error should be checked and the corresponding activity should be returned for modifying the models. However, there is one situation where no matter how many times the modifications are performed, the error always occurs. It is reasonable to assert that the requirement itself leads to the problem of formalization rather than the formalization process itself. Therefore, it is necessary to perform the communication with stakeholders who provide the original requirements.

4.4 Conclusion

In this chapter, we focus on the formalization within the proposed framework and put forward the solutions to link the formal models and SysML models. Aimed at formalization of plant and specification respectively, the contributions in this study are as follows:

(1) In order to link the plants and physic components, the activity of plant information identification is detailed. The proposed method can be used to decompose the system behaviors

and the corresponding plant models can be built. A template-based approach is proposed to guarantee that the results of plants can correctly reflect the behaviors of physic components and the modeling process can be improved.

(2) With the help of SysML requirement diagram, the informal requirements can be modeled and decomposed. The model E-requirement is proposed to compose the ambiguously-narrative high-level requirements to explicit description so that the formal specifications can be traced. A traceability verification method and algorithm are proposed based on the integration by temporal logic and the consistency between requirements and specifications can be verified.

Chapter 5

Application

5.1 Introduction

In this chapter, the application of the proposed framework introduced in Chapter 3 and Chapter 4 are focused on to show how proposition can be applied into the AC development for real systems. In the case study, a Custom Power Park (CPP) is taken as target system. In [19], the authors addressed the application of SCT to achieve the coordination of various devices in CPP, focused on the supervisor synthesis and simulation. In this case study, we highlight how SCT can be integrated into the AC process from input analysis to concrete controller implementations based on the proposed framework.

5.2 System Description

The upcoming challenges faced by power delivery networks have questioned the capability of the traditional network's topologies and technologies to deliver economical, green, reliable, and high-quality power to customers [19]. Distributed autonomous sub-networks are regarded as the topology of the future power network [161]. Aimed at supplying highly sensitive loads with high quality uninterruptible power, CPP is the state-of-the-art power electronic interfaced equipment, employed in the distributed system topology to solve power quality problems. To simply the structure, the case study supposes that only load LA and LAAA are connected in the system.

A. Target System

CPP can be regarded as a microgrid with a typical structure shown in Figure 5.1. For focusing of on supervisory high-level control of CPP, the system is simplified as being composed of three custom devices: Dynamic Voltage Restorer (DVR), Static Transfer Switch (STS) and Active Power Filter (APF). A combination of three devices is normally fed by two feeders and a backup generator. These devices along with a backup generator cooperate with each other to support the

power supply to the loads.

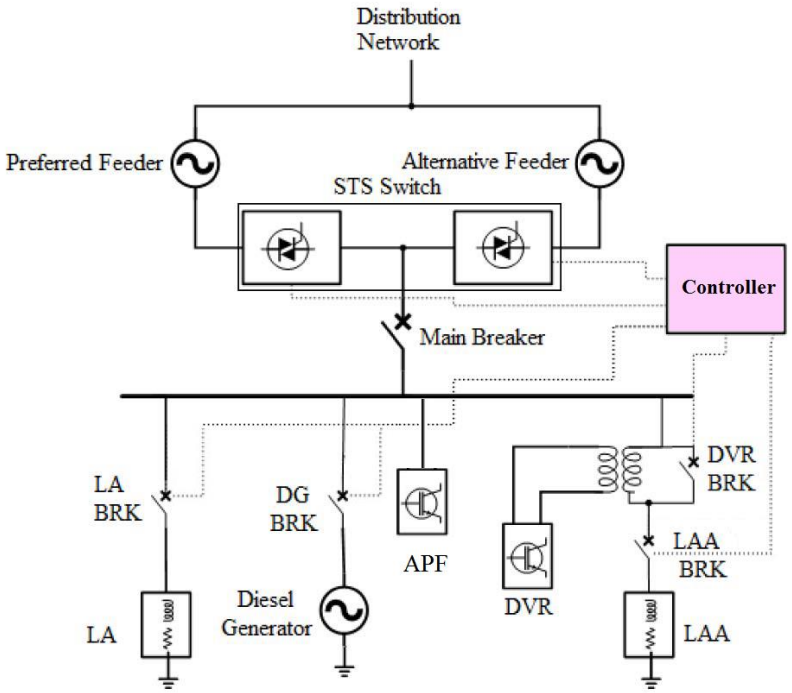


Figure 5.1 Custom Power Park [19]

Static Transfer Switch (STS): CPP is supplied with power from other stations in the distribution network by two feeders to increase the reliability. One feeder is used as alternative feeder whereas the power quality of the preferred feeder drops below certain criteria, the supply is switched to this feeder. The STS is normally a thyristor switched power electronic device which could switch in a time scale of 4-8 milliseconds.

Dynamic Voltage Restorer (DVR) is used to protect sensitive loads from voltage sag/swell or unbalance voltage. It is connected in series with the load in the feeder. In the distributed generation (DG) source connected to DC bus of DVR by using a boost DC/DC converter has been used to transfer voltage and its injection to DC capacitor. The combined operation of a DVR equipped with DG increases the reliability of the device [162].

Active Power Filter (APF) is used to mitigate current harmonic distortion normally injected into the system by nonlinear loads. This power electronic-based device is connected in parallel close to nonlinear loads.

B. Functional Requirement Description

Power quality disturbances refer to any deviation in voltage, current and/or frequency which may result in mal-operation or failure of sensitive equipment. To enhance the quality of power, the problems regarding voltage sag/swell, voltage and current unbalance, and harmonic distortion

should be addressed. In the case of voltage sag or interruption in the preferred feeder, a Static Transfer Switch (STS) is used to transfer the supply from preferred feeder to alternative feeder. This strategy could supply the sensitive loads with continuous high quality power. In the case of the recovery of the power quality in the preferred feeder, it has to switch back to this feeder. The power quality can be Normal, Abnormal or Critical if the voltage sag / swell level is less than 10%, between 10% and 50% or more than 50% respectively, shown in Table 5.1.

Table 5.1 Voltage level in feeders

Voltage Sag/Swell Level	Feeder Condition
Less than 10%	Normal
Between 10 and 50%	Abnormal
More Than 50%	Critical

There are three grades of power in the CPP which is based on the level of quality of power supplied to the loads. Power grade A has the basic power quality upgrade, where STS helps in eliminating voltage sags and swell and APF reduces harmonic distortion injected by nonlinear loads. Power grade AA has a backup generator to supply the loads within 5 to 10 sec following any contingency and includes all features in grade A. The Power grade AAA has all the features in previous grades and has a fast responding DVR delivering higher quality to sensitive loads. This grade has the highest level of power quality and is delivered to the critical sensitive loads. The loads are labeled based on the required grade of power quality, i.e., LAAA means the load needs to be supplied by power grade of AAA.

5.3 Tool Support

As the global modeling process are based on SysML, it is necessary to look for MBSE tool support. *IBM Rational Rhapsody*® provides a collaborative design and development environment for systems engineers and software developers creating real time or embedded systems and supports UML, SysML and AUTOSAR. Besides, it supports automatic generation of complete code for C, C++, Ada, or Java or to export a project to an XMI (XML Metadata Interchange) file which conforms to standard ISO/IEC 19503:2005 [163]. In this study, IBM Rational Rhapsody® is used for modeling all the necessary SysML diagrams in the development process.

For the part of supervisor synthesis and verification, *Supremica* is made use of as the tool support to deal with the formal models. *Supremica* is designed as an Integrated Development Environment (IDE) that allows the creation, verification, and synthesis of discrete event system model [164]. An advantage of Supremica editor is that it provides a graphical user interface to create and modify DES models, which are displayed and edited graphically, with states and transitions labeled by

easy-to-use drag and drop operations. Besides, *Supremica* supports to model import and export by XML files.

5.4 Modeling Results

In this section, the modeling results for each step in the global development process based on the proposed framework are presented. Not all MBSE elements of the architecture will be presented since such elements as viewpoints and concerns can be generally applied in all applications. We highlight the models of the proposed framework which show their improvement compared with typical process.

5.4.1.1 Requirement Analysis

Analyze Stakeholders' Needs (AR1.1):

The intermediate outputs MR1.1.1 should be produced in this step. According to the system description, all the seven components should be under control. Besides, a preliminary analysis result of the stakeholders 'needs for the control mission is presented in Table 5.2. For AC only functional requirement are taken into considerations. The requirements is initially decomposed by three levels in this step:

- Mission level: CPP is deployed in the power network, aimed at supply highly sensitive loads with high quality uninterruptible power. Therefore, a unique mission of the control system can be identified to make sure that CPP can achieve the established target;

- Function level: the mission should be decomposed at this level. Two requirements at this level can be identified: the first requirement is to ensure that the STS performs appropriately according to the STS condition (q.v. Table 5.1). Secondly, CPP should provide qualified power to loads which are sensitive to different grades of power quality.

Table 5.2 MR1.1.1 list of system requirements

Level	**Requirement**	**ID**	**Description**
Mission	Mission Power Supply	M1	*CPP shall supply high quality power.*
Function	STS Condition	F1.1	*CPP should perform appropriately under different STS conditions.*
	Power Grade	F1.2	*CPP should provide power which conforms to the power grade required by loads.*

Analyze System Requirements (AR1.2):

For analyzing and decomposing the high-level requirements, E-requirements are defined in step:

- Execution level: Three requirements can be identified at this level, which are aimed at different power grades respectively to cope with three different cases of STS condition. The requirement for grade A only involves the operations on STS, as described by input "The STS switch should be connected to P-feeder unless it is in abnormal or critical condition and A-feeder is in normal condition". The requirement of grade AA involves such operations that when both feeders are in abnormal condition, the DVR should be turned on and be activated. The requirement of grade AAA handles the situation where both feeders break down. It requires more operations than the others as generator and APF are involved. The input provides the descriptions of the requirements.

To sum up, Table 5.3 list the collection of requirement mentioned above.

Table 5.3 List of requirements

Level	**Requirement**	**ID**	**Description**
Execution	Grade A	E1.1	*The STS switch should be connected to P-feeder unless it is in abnormal or critical condition and A-feeder is in normal condition.*
	Grade AA	E1.2	*Whenever both feeders turn to be abnormal or critical, the DVR should be turned on and be activated. And whenever one of the feeders returns normal, DVR should be shut down.*
	Grade AAA	E1.3	*Whenever both feeders are in critical condition, the generator should be turned on. During synchronization time, all the loads should be disconnected. In the case of recovery of voltage in feeders, the generator should be turned off. The generator is supplying the LAAA load only. When LAAA load is connected, the APF should be turned on.*

In order to make the E-requirements more precise, E-requirements which have more refined and precise description are initially given in the first iteration. E-requirement E1.1 and E 1.3 are

considered to be not precise enough to provide explicit description. For example, the description E1.1 implies two situations:

(1) Whenever P-feeder turns normal or A-feeder is abnormal or critical, A-feeder should be disconnected and connect P-feeder.

(2) Whenever P-feeder turns abnormal or critical and A-feeder is normal, P-feeder should be disconnected and A-feeder connected.

Therefore, E-requirement E1.1 must be decomposed into two sub-requirements whose descriptions should be detailed according to the situations above. Similarly, E-requirement E1.3 should also be decomposed into three sub-requirements. A collection of the sub-requirement is listed in Table 5.4.

Table 5.4 List of sub-requirement

Level	**Requirement**	**ID**	**Description**
Execution	Connect P-feeder	E1.1.1	*Whenever P-feeder turns normal or A-feeder is abnormal or critical, A-feeder should be disconnected and connect P-feeder.*
	STS Condition	E1.1.2	*Whenever P-feeder turns abnormal or critical and A-feeder is normal, P-feeder should be disconnected and A-feeder connected.*
	Initialization	E1.3.1	*Initially, whenever not all feeders are critical, connect LA and LAAA and APF, and whenever all feeders are critical turn on, synchronize generator and connect LA and turn on APF.*
	Generator Start	E1.3.2	*After APF is turned on, then both feeders turn to critical, it should make sure shut down load LAAA, load A and APF and then start and synchronize the generator and start load LAAA and APF.*
	Generator Shut Down	E1.3.3	*After APF is turned on, then at least one of the feeders recovers, the generator should be shut down and then connect load LA.*

To integrate all the requirements, a requirement diagram is preliminarily built for the initial iteration. Apart from presenting the requirements, the relationship between each requirement should also be constructed. Two requirements at function level are contained in mission requirement M.1. As three root E-requirements specify the function at the component level compared with function level, the relationship <<derive>> is use to link the requirements between

these two levels. It can be seen that E1.1 has relationship with both F1.1 and F1.2, which focus on STS control and power grade respectively. Finally, the sub-E-requirements are contained in the corresponding root E-requirement. The modeling result of requirement diagram is shown in Figure 5.2.

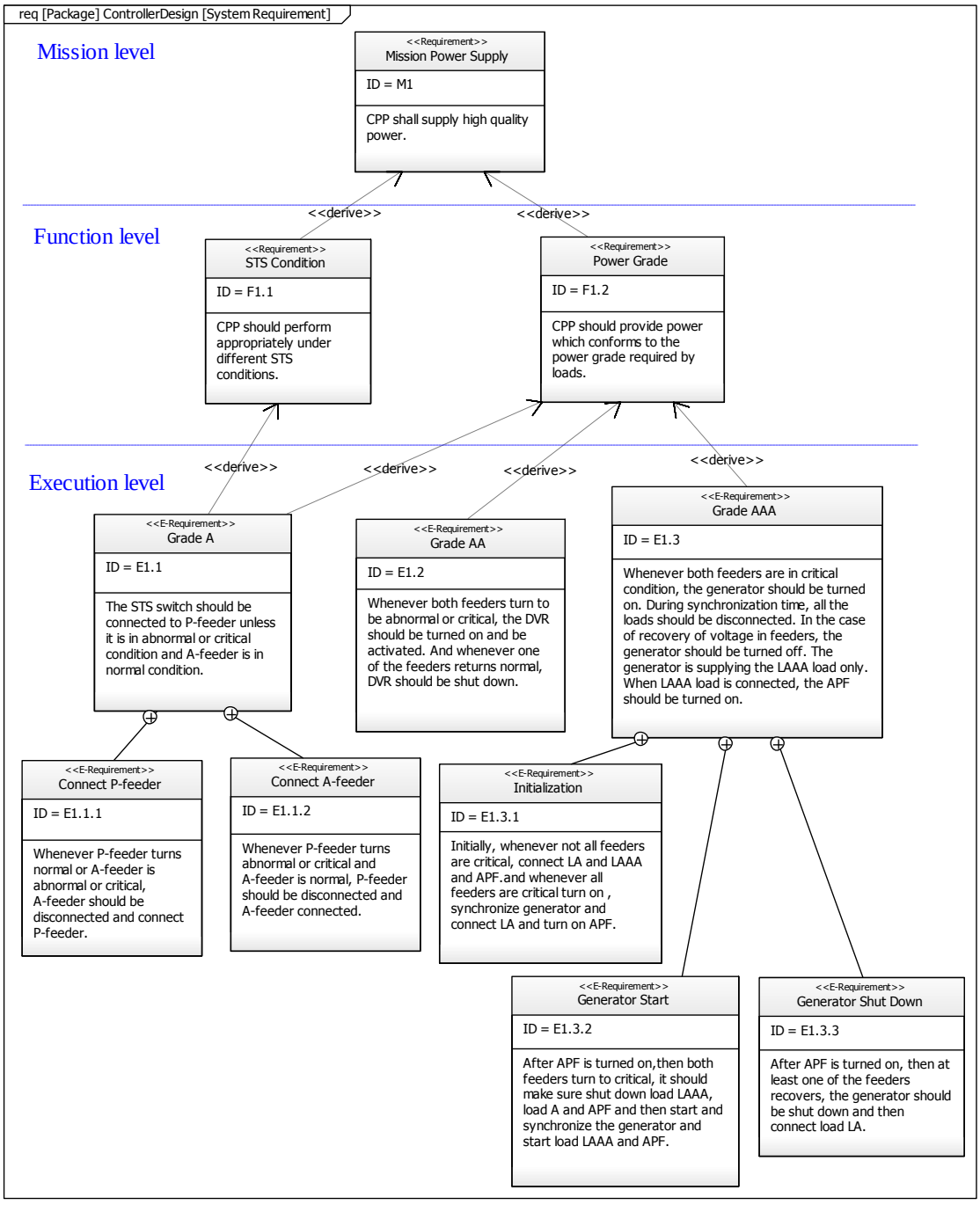


Figure 5.2 REQ: System Requirement

Analyze Context (AR2.1):

In consideration of system description, two elements are defined for composing the context of the

case study. CPP is centralized by the element system-of-interest, for which a controller should design to realize the target of high quality power supply. As a result, CPP is composed of two parts: the system to be controlled and controller. Secondly, CPP is regarded as microgrid of the global power networks and therefore the element represent the external system is designated by power networks.

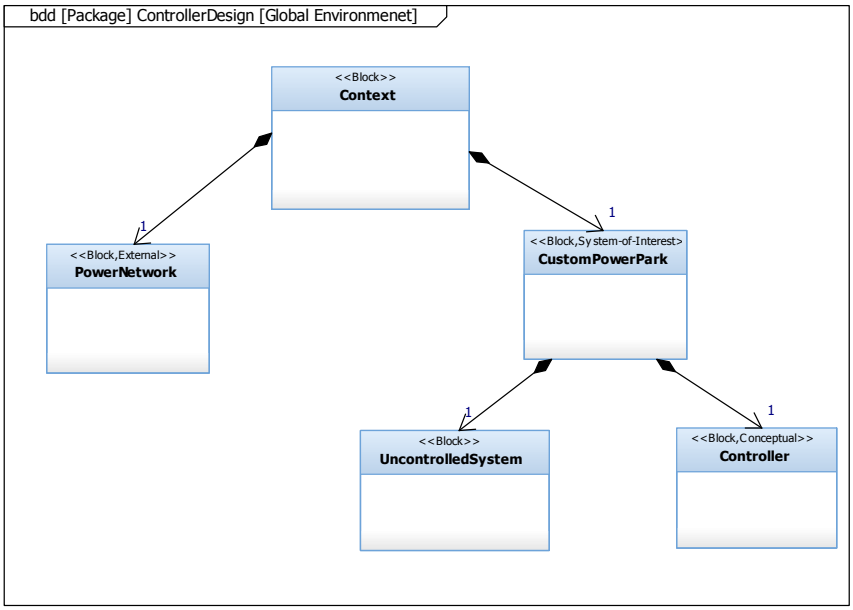


Figure 5.3 BDD: Global Environment

The model represents the global environment of the system to be studied is depicted by a BDD shown in Figure 5.3. The elements in the diagram are labeled by corresponding stereotypies to specify their functions in the context. Since the BDD is the first model to provide a global perspective of the case, the details of each element are unnecessary not, and impossible to, be given in the diagram. The controller in this diagram is labeled by stereotype <<conceptual>> as it still needs to be developed. In particular, the system-of-interest should be exclusively detailed in the activity **AR2.2** and the controller is developed in the design synthesis phase. Besides, it is supposed that the system is automatically controlled and no human participant is involved in the context in this case study.

Analyze System-of-interest (AR2.2):

The system-of-interest is composed of seven components which are corresponding to the seven blocks respectively. Seven kinds of block are defined in the BDD: STS, Power Monitor, LAAA, LA, DVR, APF and Generator and the physic components in the uncontrolled system can be defined by the corresponding blocks with their part names as the plant candidates. There is no hierarchical structure of the system and all components are directly aggregated to the uncontrolled system. In the BDD, the control relationship should be presented by association <<control>> between conceptual controller and parts. A collection of components is shown in Table 5.5. The

modeling result of system structure is presented in Figure 5.4.

Table 5.5 List of components

Part	Block	Composition	Control Target
sts1	STS	Uncontrolled System	X
pm1	Power Monitor	Uncontrolled System	X
l1	LAAA	Uncontrolled System	X
l2	LA	Uncontrolled System	X
dvr1	DVR	Uncontrolled System	X
apf1	APF	Uncontrolled System	X
gen1	Generator	Uncontrolled System	X

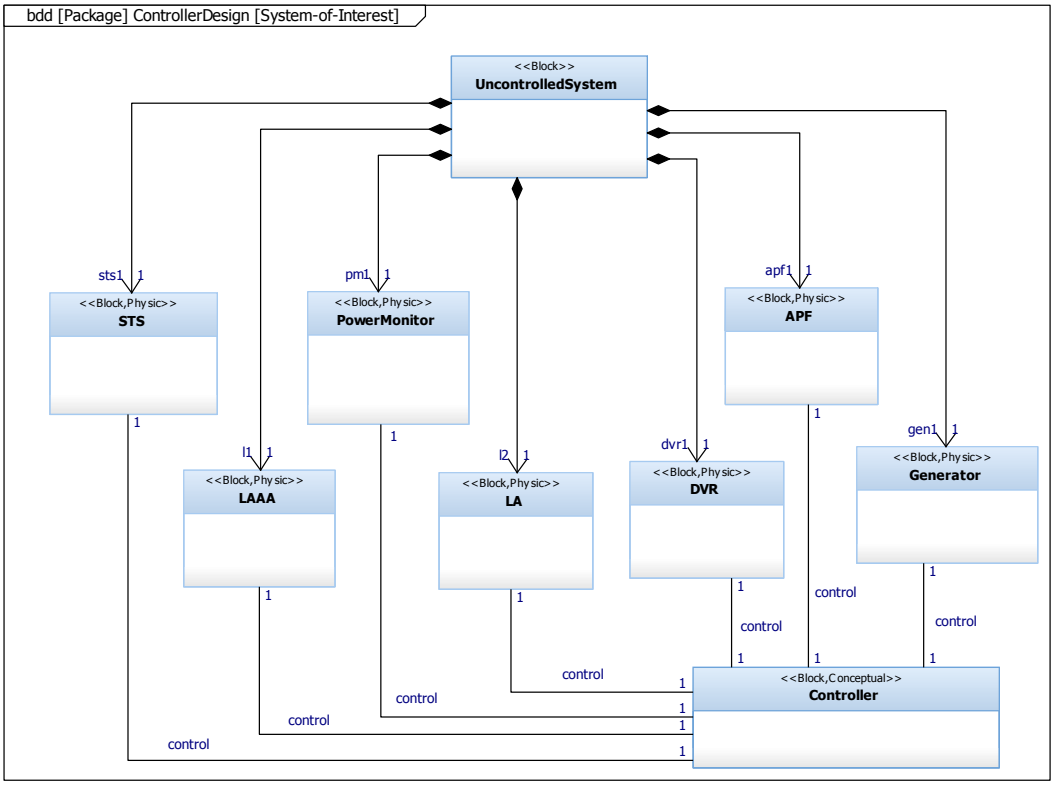


Figure 5.4 BDD: System Structure

5.4.1.2 Functional Analysis

Identify Plant Structure (AF2.1):

Table 5.6 lists the plant information related to the components to be controlled. The formal semantic of actuators and sensors are identified for each component including event name and event type. 15 controllable events and 4 uncontrolled events are identified.

Table 5.6 MF2.1.1 Plant information list

Part	Type	Description	Event Notation	Event Type
sts1: STS	Actuator1	Connect to P feeder	cp1	Controllable
		Disconnect from P feeder	dcp1	Controllable
	Actuator2	Connect to A feeder	ca1	Controllable
		Disconnect from A feeder	dca1	Controllable
pm1: PM	Sensor1	Both feeders normal detected	p1n_a1n_pm1	Uncontrollable
	Sensor2	P feeder abnormal or critical, A feeder normal detected	p1c_a1n_pm1	Uncontrollable
	Sensor3	P feeder abnormal, A feeder abnormal or critical detected	p1a_a1c_pm1	Uncontrollable
	Sensor4	Both feeders critical detected	p1c_a1c_pm1	Uncontrollable
	Actuator1	Initialize monitoring system	int1_pm1	Controllable
l1: LAAA	Actuator1	Connect the load l1	cl1	Controllable
		Disconnect the load l1	dcl1	Controllable
l2: LA	Actuator1	Connect the load l2	cl2	Controllable
		Disconnect the load l2	dcl2	Controllable
dvr1: DVR	Actuator1	Turn on dvr	on_dvr1	Controllable
		Turn off dvr	off_dvr1	Controllable
apf1: APF	Actuator1	Turn on apf	on_apf1	Controllable
		Turn off apf	off_apf1	Controllable
gen1: Generator	Actuator1	Turn on the generator gen	on_gen1	Controllable
		Turn off the generator gen	off_gen1	Controllable
	Actuator2	Generator gen is synchronized	syn_gen1	Controllable

Secondly, the orthogonal sub-system should be identified for each component in Table 5.6. Among them, parts sts1, pm1 and gen1 have several actuators/sensors. However, all of parts cannot be divided into orthogonal sub-systems. Therefore, a block of plants is built for each part. The result is presented in BDD: Plant Structure (Figure 5.5), in which each part is allocated by a

corresponding block representing the formal plant (stereotyped by <<Formal>>).

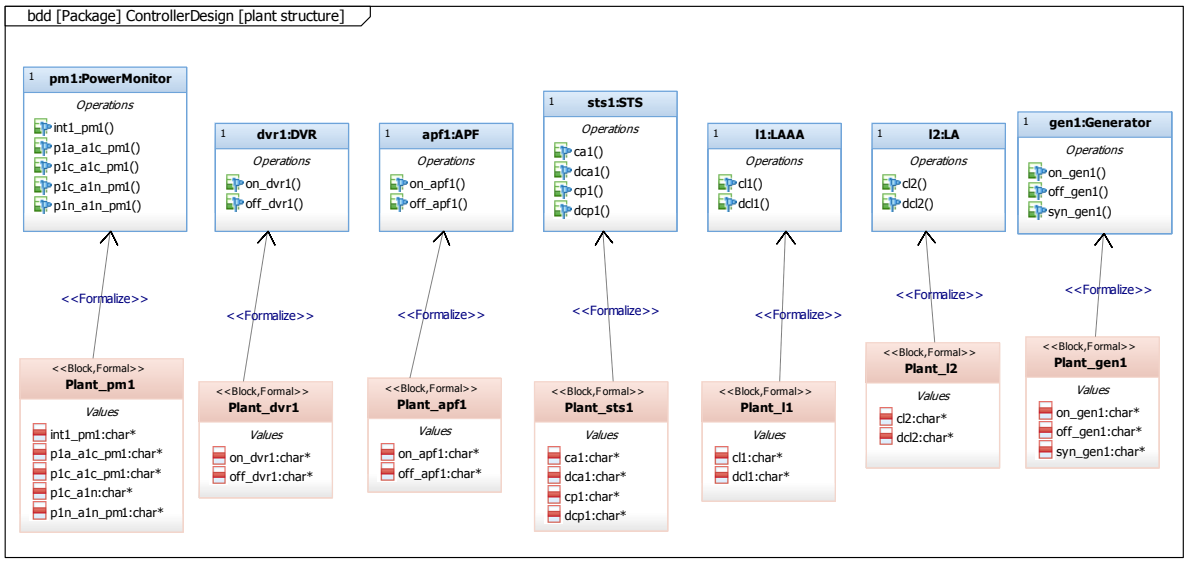


Figure 5.5 BDD: Plant Structure

Build Formal Plant (AF2.2):

As the semantic descriptions of event in the behavior models are given, it is also necessary to define the semantic for each state. In Table 5.7, we list their properties of initial and marked and semantic description for states in each component behavior models involved in the case study. The states which are not marked are those to be regarded as interim sate of components. For example, we must always keep the STS at service state to provide power. However, the switch between A-feeder and P-feeder requires the successive operations of switching off one of them and switching the other on. Therefore, the state of both feeders off cannot be the marked state. So it goes with pm1 and gen1.

Table 5.7 Semantic of state

Component	State	Initial	Marked	Description
sts1: STS	s0	X		Both feeder off
	s1		X	P-feeder on and A-feeder off
	s2		X	A-feeder on and P-feeder off
pm1: PM	s0	X	X	pm1 is initialized
	s1			pm1 received signals
l1: LAAA	s0	X	X	l1 off
	s1		X	l1 on
l2: LA	s0	X	X	l2 off

	s1		X	l2 on
dvr1: DVR	s0	X	X	dvr1 off
	s1		X	dvr1 on
apf1: APF	s0	X	X	apf1 off
	s1		X	apf1 on
gen1: Generator	s0	X	X	gen1 off
	s1			gen1 on
	s2		X	gen1 synchronized

To integrate MF2.1.1 and MF2.2.2, the formal plants are finally determined (shown in Figure 5.6).

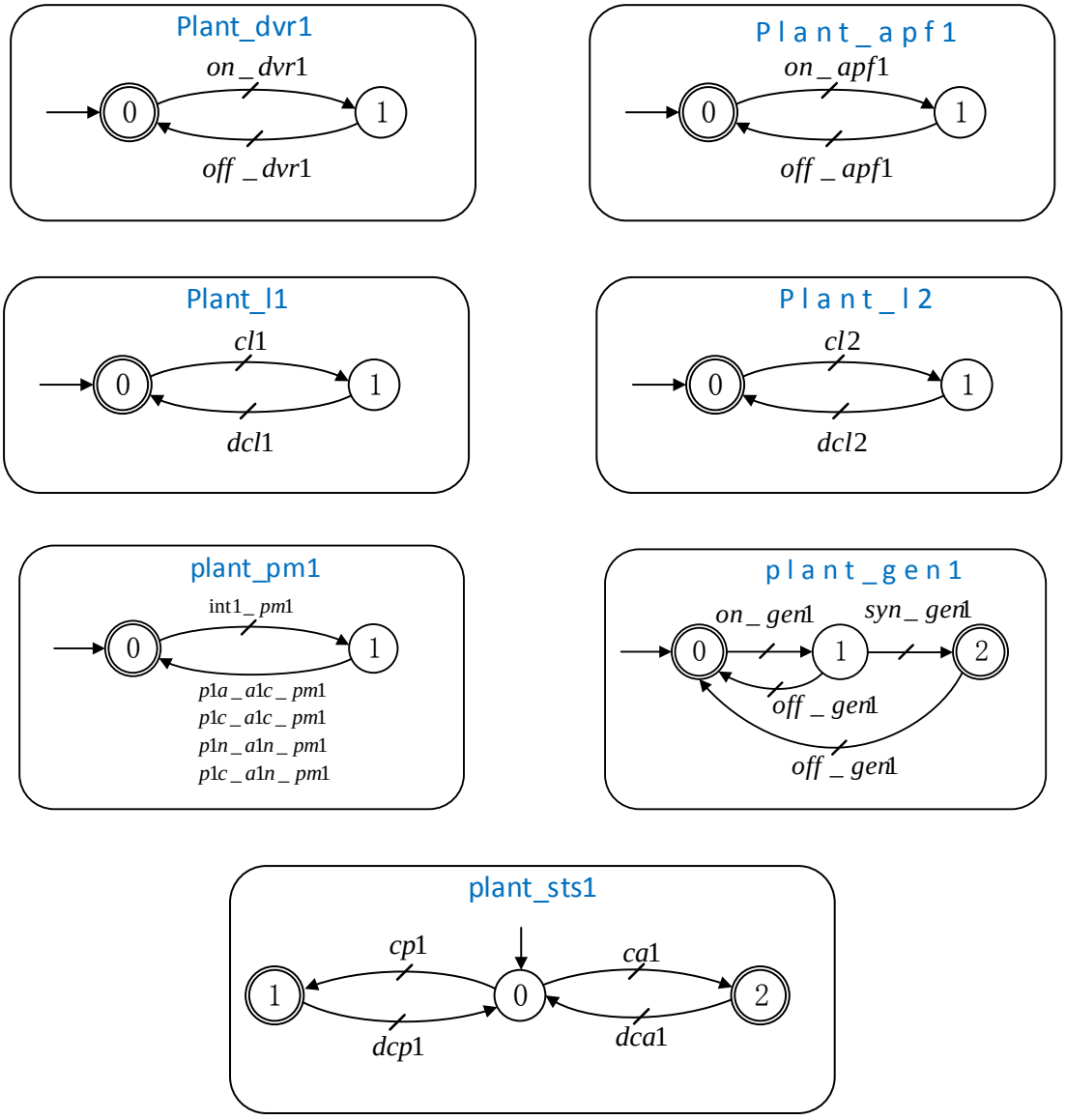


Figure 5.6 Plant model

Remark *In this part, we present the modeling process section A for formalization of plant by activity* ***AF2.1*** *and* ***AF2.2****. As introduced in Chapter 4, a template-based approach can be applied to be instead of the process by Section A'. The details of the adapted modeling process are presented in Appendix B (Section B.4.1).*

Build Formal Specifications (AF1.1):

For the first iteration, all the root E-requirements in the diagram REQ: System Requirements should be formalized. Here, we take formalization of specification 1 as example. Firstly, the events involved in the specification 1 are identified (Table 5.8), by which the event set of specification 1 can be determined.

Table 5.8 List of involved events in specification 1

Event	**Component**
p1n_a1n_pm1, *p1c_a1n_pm1*, *p1a_a1c_pm1*, *p1c_a1c_pm1*, *int_pm1*	pm1
cp1, *dcp1*, *ca1*, *dca1*	sts1

Figure 5.7 shows the specification model which is initially supposed to conform to requirement grade A (E1.1). In order to keep the consistency between specification 1 and requirement E1.1, all the sub-requirement of E1.1 should be check its consistency with specification 1 in activity AF1.2. The other two formal specifications can be seen in Appendix B.

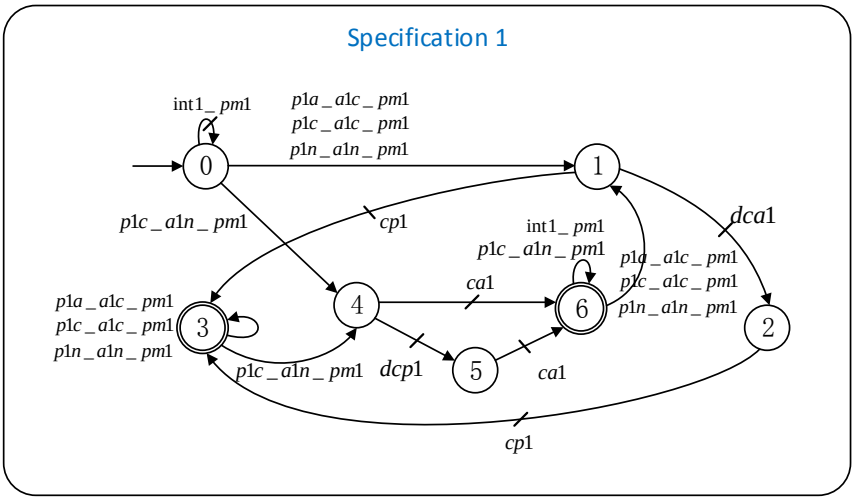


Figure 5.7 Specification 1

Consistency Verification (AF1.2):

As the E-requirement E1.1 can be decomposed into E1.1.1 and E1.1.2, the specification 1 (Spec1) is supposed to satisfy the formalism *S* denoted as follows:

$$Spec1 \models S = S_1^1 \wedge S_2^1$$

Firstly, the satisfaction of specification with S_1^1 should be checked. The requirement E1.1.1 can be interpreted as follows:

$$Spec1 \models S_1^1 = S_1^2 \rightarrow S_2^2$$

However, the formalism of requirement E1.1.1 cannot be fully identified as S_1^2 and S^2 is unclear. As part of formalism is undetermined, it conforms to the situation 2 (refinement). It can be also seen that E1.1.1 doesn't have any sub-requirement, the error 2 is verification result and activity **AR1.2** should be returned to clarify the description of E1.1.1. To this end, the E-requirement E1.1.1 is refined by two sub-requirements, shown in Figure 5.8.

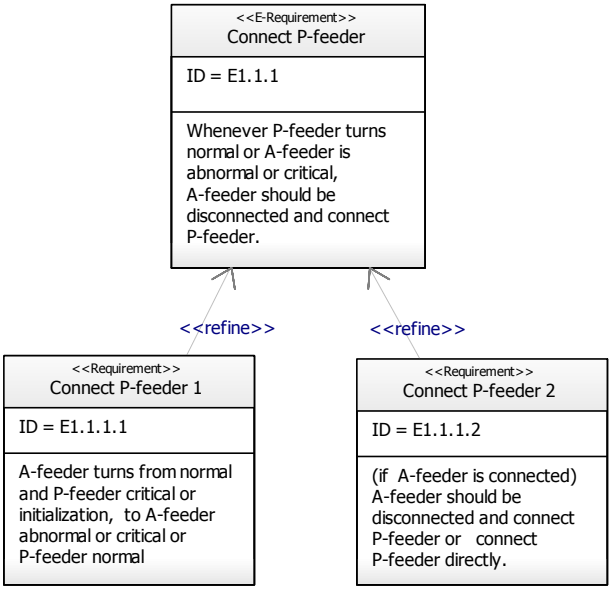


Figure 5.8 Decomposition of requirement E1.1.1

Since the E1.1.1 are refined by two requirements E1.1.1.1 and E1.1.1.2, formalism S_1^2 and S_2^2 are identified respectively. The S_1^2 interpreting E1.1.1.1 can be identified as follows:

$$S_1^2 = (\text{int}1\_pm1 \vee p1c\_a1n\_pm1)(p1a\_a1a\_pm1 \vee p1c\_a1c\_pm1 \vee p1n\_a1n\_pm1)$$

E1.1.1.2 describes two following situations as follows:

$$S_{21}^2 = dca1 \rightarrow cp1 \qquad S_{22}^2 = cp1$$

$$\Rightarrow S_2^2 = S_{21}^2 \vee S_{22}^2 = (dca1 \rightarrow cp1) \vee cp1$$

Therefore, to integrate formalism S_1^2 and S_2^2 into S_1^1:

$$\Rightarrow S_1^1 = ((\text{int}1\_pm1 \vee p1c\_a1n\_pm1)(p1a\_a1a\_pm1 \vee p1c\_a1c\_pm1 \vee p1n\_a1n\_pm1)) \rightarrow ((dca1 \rightarrow cp1) \vee cp1)$$

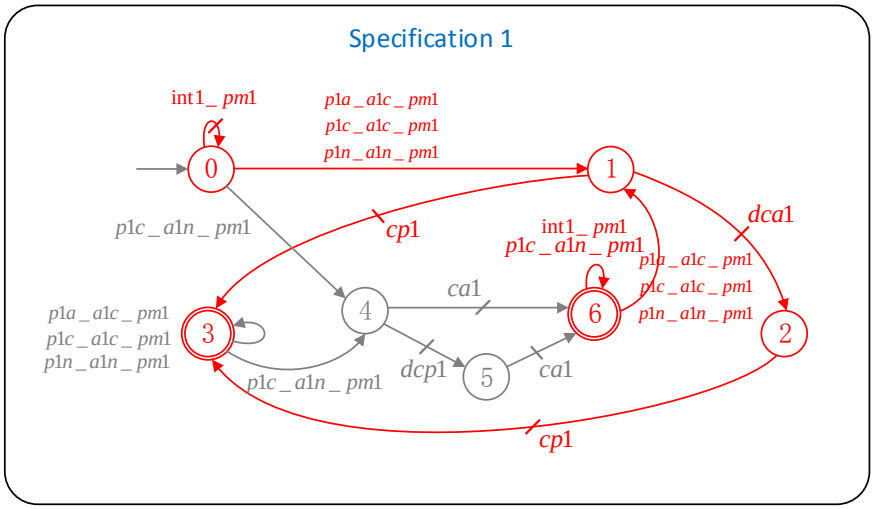


Figure 5.9 Consistency verification for E1.1.1

Presented in Figure 5.9, the partition of specification 1 conforms to the formalism E1.1.1 and no illegal paths can be found. Then, $Spec1 \models S_1^1$ can be verified. The result of pass shows that formal specification is consistency with E-requirement E1.1.1. Similarly, the satisfaction of specification with S_1^2 should be verified. The E1.1.2 can be interpreted by S_2^1 as follows:

$$Spec \models S_2^1 = S_3^2 \rightarrow S_4^2$$

Like requirement E1.1.1, the formula of requirement E1.1.2 also cannot be fully identified. The error 2 is verification result and activity **AR1.2** should be returned for the third iteration to clarify the description of E1.1.2. To this end, the E-requirement E1.1.2 is refined by two sub-requirements, shown in Figure 5.10.

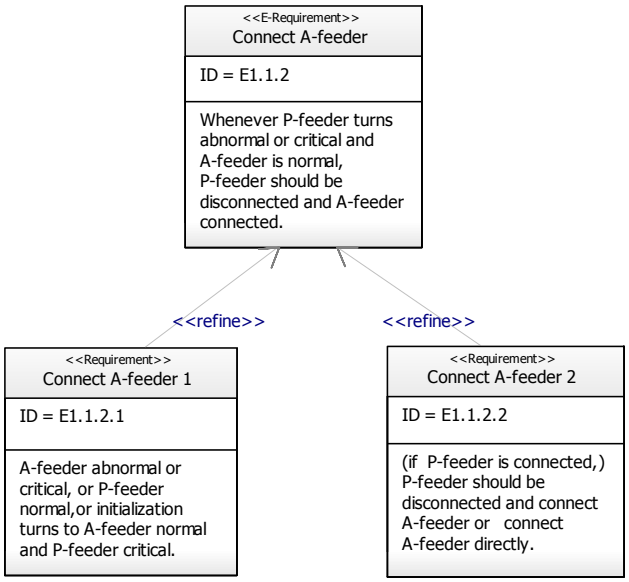


Figure 5.10 Decomposition of requirement E1.1.2

Since the E1.1.2 are refined by two requirements E1.1.2.1 and E1.1.2.2, formula S_3^2 and S_4^2 are

identified respectively:

$$S_3^2 = (\text{int}1\_pm1 \vee p1a\_a1a\_pm1 \vee p1c\_a1c\_pm1 \vee p1n\_a1n\_pm1)(p1c\_a1n\_pm1)$$

E1.1.1.2 describes two following situations as follows:

$$S_{41}^2 = dcp1 \rightarrow ca1 \qquad S_{42}^2 = ca1$$

$$\Rightarrow S_4^2 = S_{41}^2 \vee S_{42}^2 = (dcp1 \rightarrow ca1) \vee ca1$$

Therefore, to integrate formula S_3^2 and S_3^2 into S_2^1:

$$\Rightarrow S_2^1 = ((\text{int}1\_pm1 \vee p1a\_a1a\_pm1 \vee p1c\_a1c\_pm1 \vee$$
$$p1n\_a1n\_pm1)\ p1c\_a1n\_pm1) \rightarrow (dcp1 \vee ca1)$$

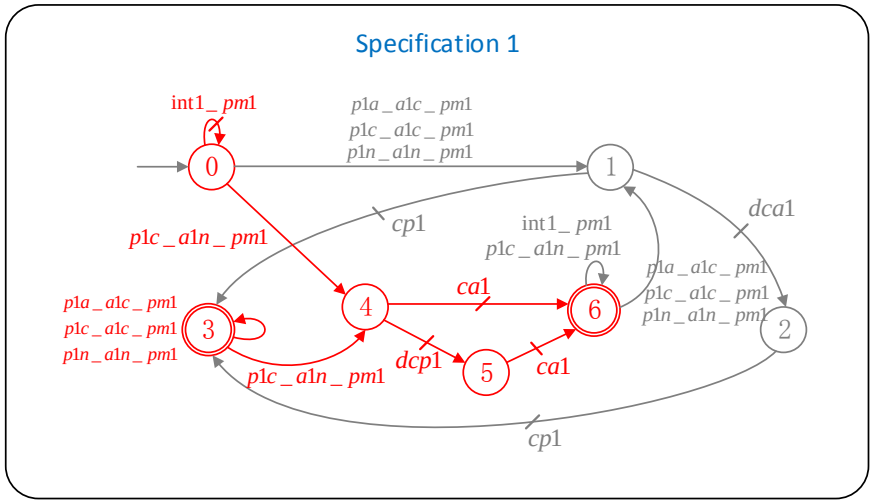


Figure 5.11 Consistency verification for E1.1.2

Presented in Figure 5.11, the partition of specification 1 conforms to the formalism of E1.1.2 and no illegal paths can be found. Then, $Spec1 \models S_2^1$ can be verified. The successful verification result shows that formal specification is consistency with E-requirement E1.1.2. To sum up, the Specification 1 is verified to satisfy formulae both S_1^1 and S_2^1. Specification 1 is verified to conform to both E-requirement E1.1.1 and E1.1.2 and therefore it conforms to requirement E1.1. Apart from Specification 1, models for Specification 2 and Specification 3 are also built, which conforms to E-requirement E1.2 and E1.3 respectively. The details of modeling and verification are given in the Appendix B.

Confirm Traceability (AF3):

Two kinds of traceability are focused on in the diagram. Firstly, the formal specification should be linked to the corresponding E-requirement. As formal specifications are built based on the root E-requirement (Table 5.9), the relationship <<Formalization>> between them can be directly constructed. Secondly, it is also necessary to present the formal plants which are involved in the

function constraints of the E-requirements. The high level E-requirement may not explicitly present all the involved components which leads to the missing of some of them. However, as the formal specifications have already linked to requirements, by checking the event set of each specification and mapping each event to the corresponding plant, the link between plants and requirements can also be constructed (the results can be seen in Table 5.10). Therefore, the diagram can be built as shown in Figure 5.12.

Table 5.9 Allocation of specification to E-requirement

E-Req\Specification	Specification 1	Specification 2	Specification 3
E1.1	↙ formalize		
E1.2		↙ formalize	
E1.3			↙ formalize

Table 5.10 Allocation plant to E-requirement

E-Req\Plant	sts1	Pm1	l1	l2	dvr1	apf1	gen1
E1.1	↙	↙					
E1.2	↙	↙			↙		
E1.3	↙	↙	↙	↙		↙	↙

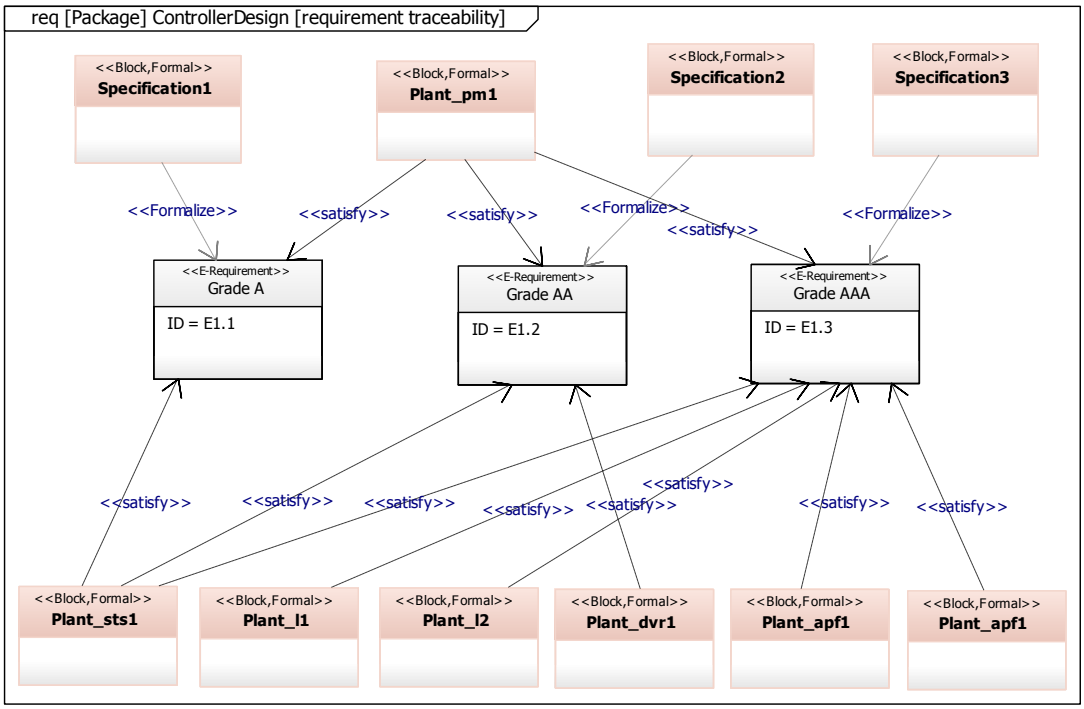


Figure 5.12 REQ: Requirement Traceability

5.4.1.3 Design Synthesis

Define Control Architecture (AD1):

The plant and specification models, including seven plants and three specifications, are obtained in activity **AF2.2** and activity **AF1.1** respectively. In this case study, we make the use of a modular control as control architecture. For this architecture, a global plant should be synthesized by synchronization of all plant models. Each local supervisor is synthesized by the global plant and the corresponding local specification. This structure is presented by the BDD in Figure 5.13. We define three local supervisors and the relation <<SynthesizeFrom>> and <<Synchronize>> is used to link the local supervisor, local specification and global plant. The global plant is aggregated by all the plants defined Figure 5.5.

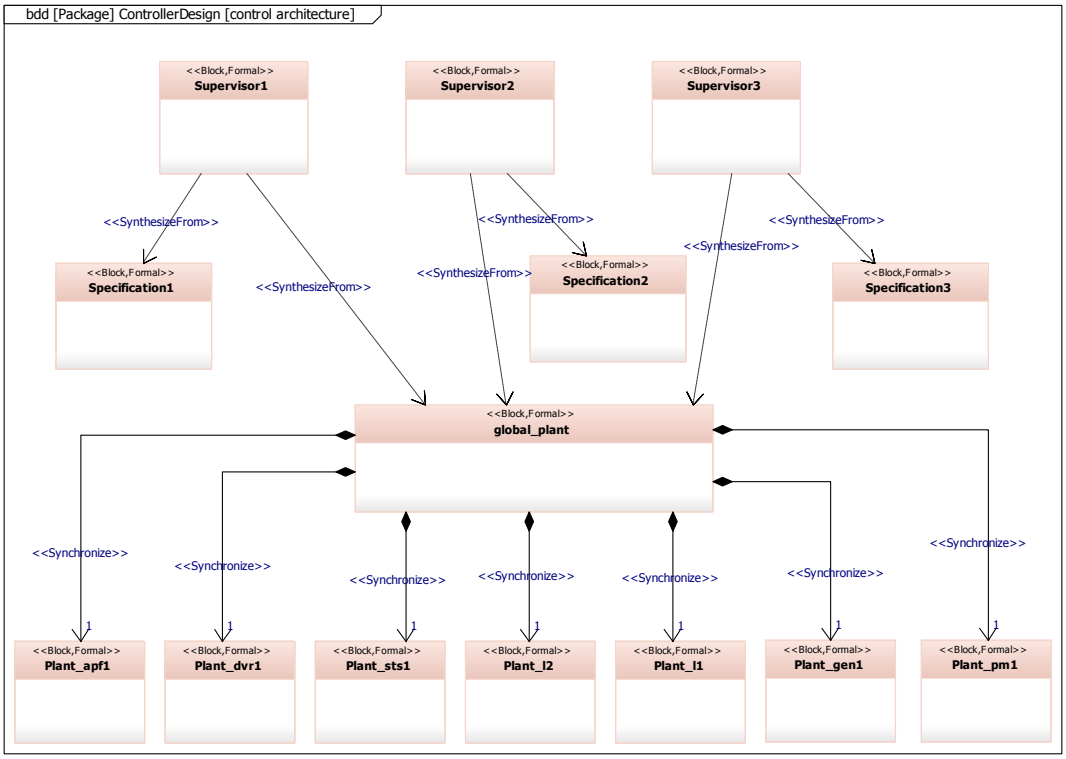


Figure 5.13 BDD: Control Architecture

Build Supervisors (AD2):

The algorithm of modular control synthesis can be found in Appendix A. Here, we denote plant model by $G_j(Q_j, \Sigma_j, \delta_j, q_{j,0}, Q_{j,m})$ and each local plant model is related to the behavior of the corresponding component. Therefore, by interpreting the semantics of <<Synchronize>> in Figure 5.13, the global plant can be synchronized by the seven plants:

$$G \leftarrow \|_{j=1}^{7} G_j$$

According to modular control algorithm, it is not necessary to synchronize the local specifications as a global specification and a local supervisor should be synthesized for each local specification. Table 5.11 shows the computation of for each local supervisor. The supervisors should be verified for their controllability and nonblocking properties and be reduced to final result.

Table 5.11 Supervisor synthesis

Supervisor	**Synthesis**	**Plant**
Supervisor 1	$Sup_1 \leftarrow G \times H_1$	$G \leftarrow \|\|_{j=1}^{7} G_j$
Supervisor 2	$Sup_2 \leftarrow G \times H_2$	
Supervisor 3	$Sup_3 \leftarrow G \times H_3$	

Supervisor 1 represented by SysML state machine is shown in Figure 5.14 and the models of other supervisors can be seen in Appendix B.

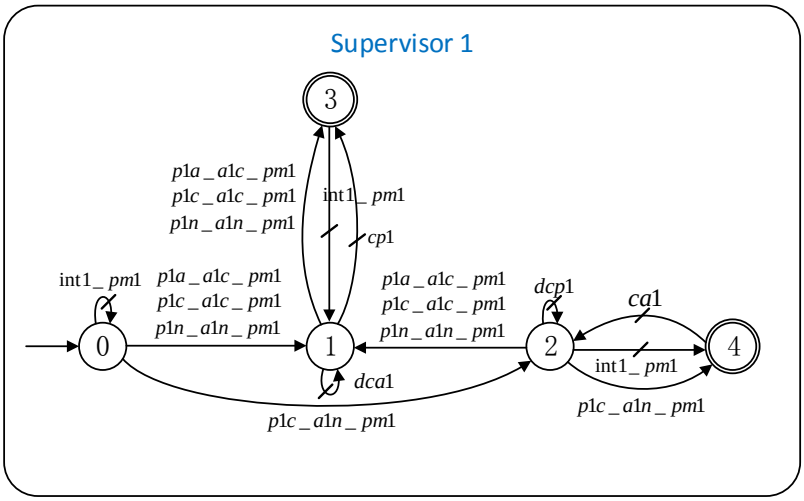


Figure 5.14 Model of supervisor 1

Define Architecture Structure (AD3)

The global plant in modular control is synchronized by all local plants, and therefore, all the events should be defined by the ports for the block of global plant. The event identification of local supervisors is shown in Table 5.12.

Table 5.12 Event identification of supervisor

Supervisor	**Event**
Supervisor 1	*p1n_a1n_pm1, p1c_a1n_pm1, p1a_a1c_pm1, p1c_a1c_pm1, int_pm1, ca1, dca1, cp1, dcp1*

Supervisor 2	*p1n_a1n_pm1, p1c_a1n_pm1, p1a_a1c_pm1, p1c_a1c_pm1, int_pm1, on_dvr1, off_dvr1*
Supervisor 3	*p1n_a1n_pm1, p1c_a1n_pm1, p1a_a1c_pm1, p1c_a1c_pm1, int_pm1, on_gen1, off_gen1, syn_gen1, on_apf1, off_apf1, cl1, dcl1, cl2, dcl2*

To model the port for all events and make connections, the result is shown in Figure 5.15.

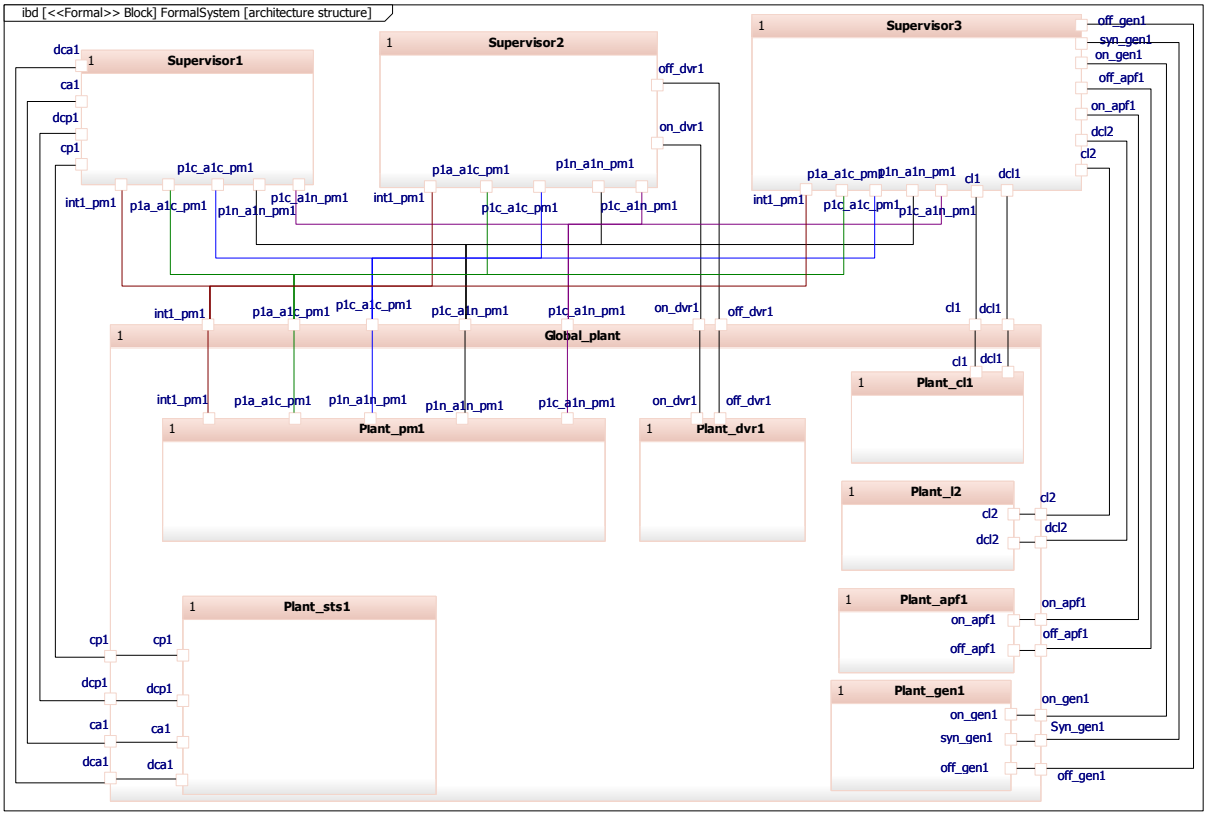


Figure 5.15 IBD: Architecture Structure

Define Controllers (AD4)

In the step, the control strategy is taken into account to transform the supervisors to controllers. Two strategies are applied to determine the choice of events, which are shown in Table 5.13.

Table 5.13 List of control strategies

Control Strategy	**Description**
Strategy 1	*Whenever there are several controllable events that are not disabled (the associated disabling signals are not active) the definition of the next occurrence of event is based on the order.*
Strategy 2	*In one state, the uncontrollable events have the priority over controllable events.*

Three controllers are built and each is allocated by a local supervisor. All the controllers take the control strategies listed in Table 5.13. The relationship between controllers, supervisors and strategies is presented by <<dependency>>. The modeling result of control strategy is shown in Figure 5.16.

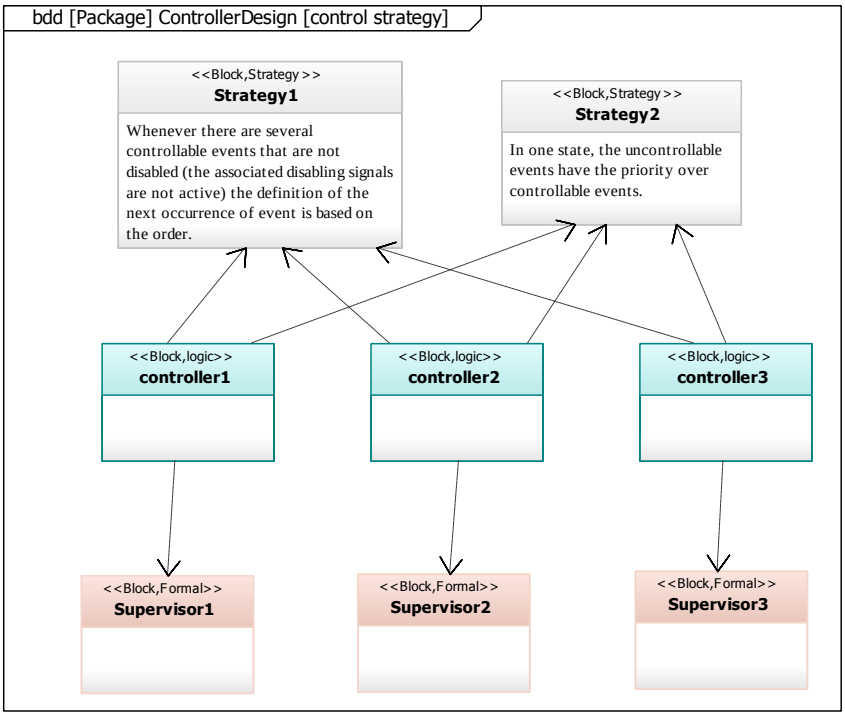


Figure 5.16 BDD: Controller Strategy

Define Concrete Controllers (AD5):

In this case study, we suppose that supervisors are implemented by a monolithic PLC device as the concrete controller and the control program is allocated to the hardware (for activity **AD5.1**). In order to realize the implementation, the ports of the controlled parts should be identified firstly. 12 ports are identified (shown in Table 5.14) which present each actuator/sensor. P.0, P.1 and P.7-P.11 are related to a pair of operations controlled by high or low electric potential of the command (i.e. connect to P feeder and disconnect from P feeder are implemented Boolean signal of P.0).

Table 5.14 MD4.3 Port of controlled components

Controlled component	**Actuator/sensors**	**Port**
sts1: STS	Actuator1	P.0
	Actuator2	P.1
pm1: PM	Sensor1	P.2
	Sensor2	P.3

	Sensor3	P.4
	Sensor4	P.5
	Actuator1	P.6
l1: LAAA	Actuator1	P.7
l2: LA	Actuator1	P.8
dvr1: DVR	Actuator1	P.9
apf1: APF	Actuator1	P.10
gen1: Generator	Actuator1	P.11
	Actuator2	P.12

Secondly, the ports for the concrete controller should be defined. As we use a monolithic concrete controller to implement all the local supervisors, the PLC device should contain all the ports corresponding to supervisors. By arbitrary notation, we use the same port names of controlled parts (P.0~P.12) for convenient connection in the following step. The diagram of control implementation is presented in Figure 5.17. The global PLC device is aggregated by the concrete controller and the control program. The dependency with stereotype <<allocate>> presents the relationship between hardware and software.

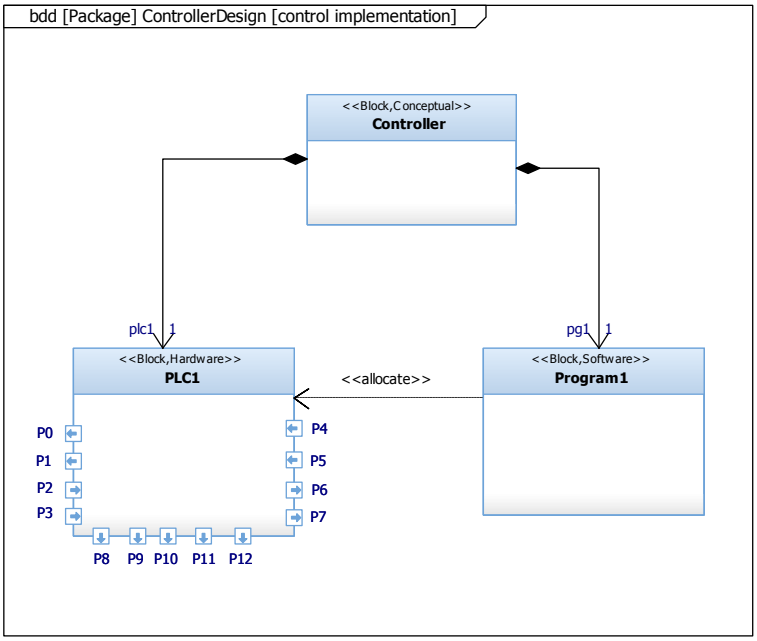


Figure 5.17 GO2 BDD: Control Implementation

The control program is realized by the supervisor implementation method proposed by [37], in which modular control can be implemented by SFC (Sequential Function Chart) representing PLC programs. According to the implementation method, the program is structured by three levels. Figure 5.18, Figure 5.19 and Figure 5.20 shows the program of sts1 at PS level, supervisor 1 at MS

level and sts1 at OP level respectively. The detailed implementation method and entire control programs can be found in Appendix B.

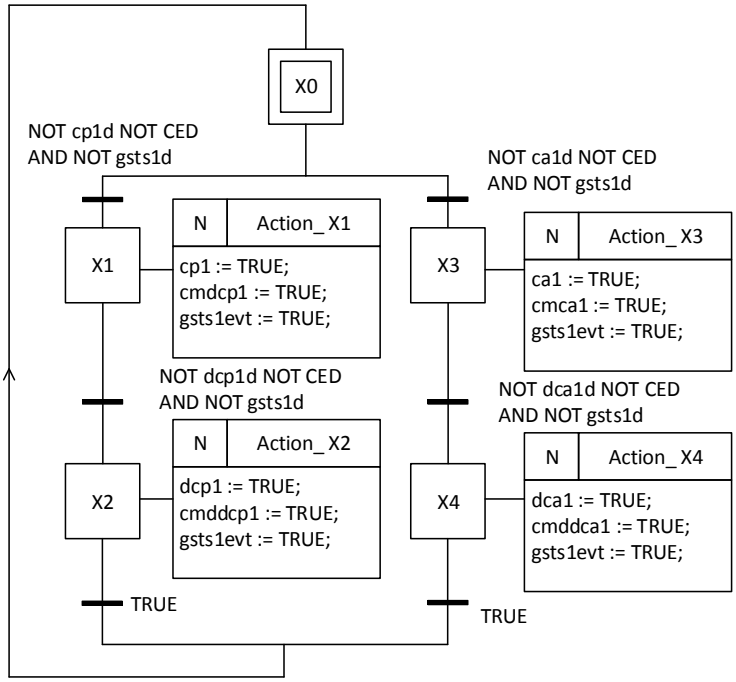


Figure 5.18 Control Program: Part PS sts1

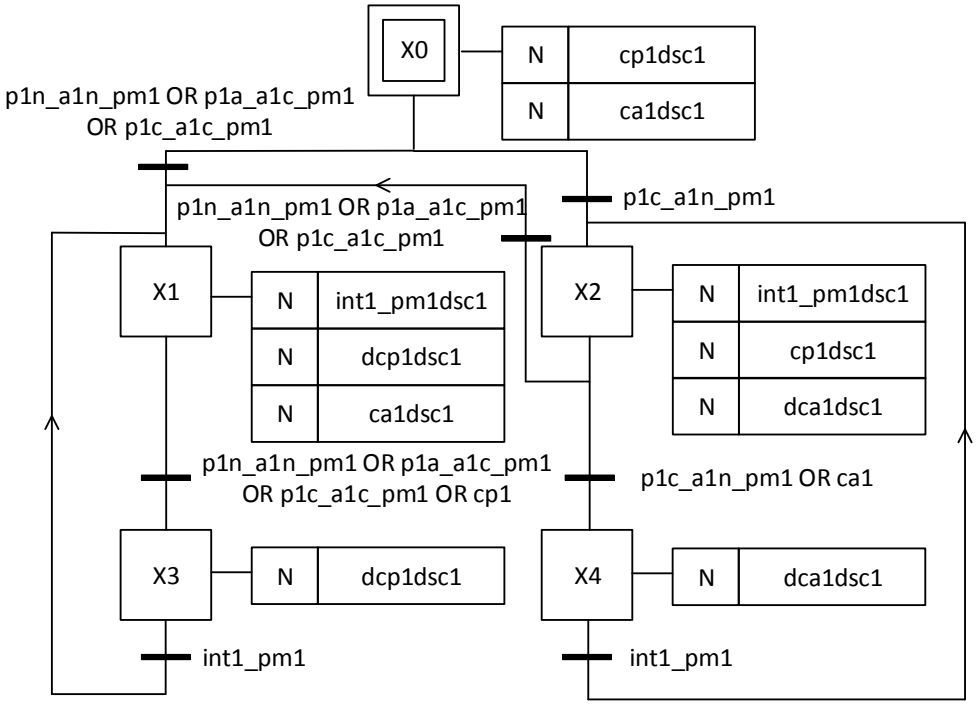


Figure 5.19 Control Program: Part MS Sup1

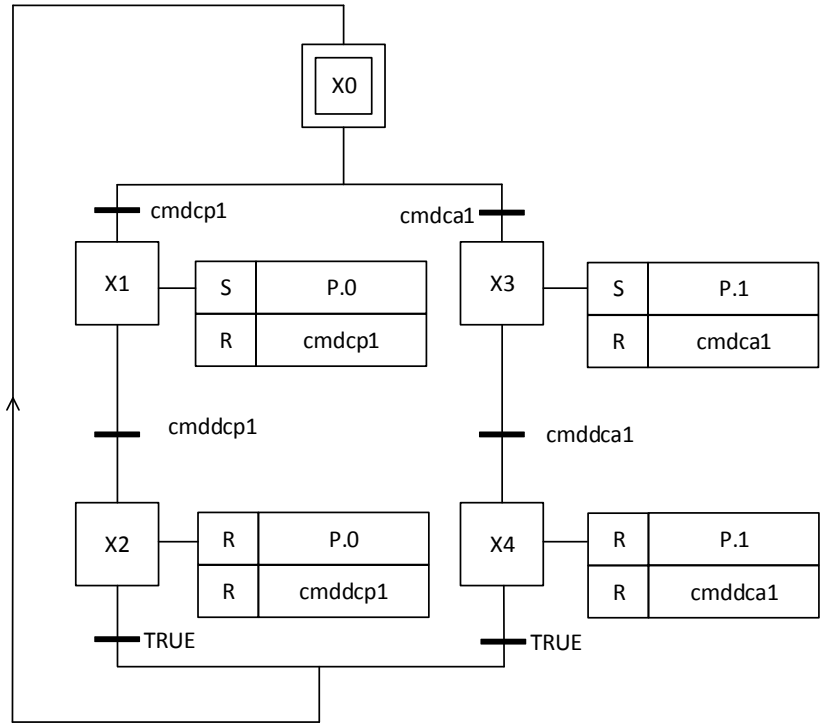


Figure 5.20 GO1 Control Program: Part OP sts1

Define Control Connection (AD6):

As the concrete controller is implemented by a monolithic PLC device, the all the controlled components should be connected to it. In this case study, as we denote the same name for the corresponding ports of concrete controller and components, the connections can be directly constructed. Figure 5.21 show the modeling result of the control structure. In the diagram, the command connections are presented in blue and the censor signal connections are presented in red. All the components are connected to plc1. It should be pointed out that the block Controller and System-to-be-controlled are conceptual models and therefore, we don't need to define the ports for both of them.

Remark *When the controller is implemented by distributed PLC structures, the connection can be more complex. Supposing a situation where each local supervisor is implemented by a proper PLC device, it is necessary to analyze the connections between each local controller and controlled components. The details of this situation can be found in Appendix B.*

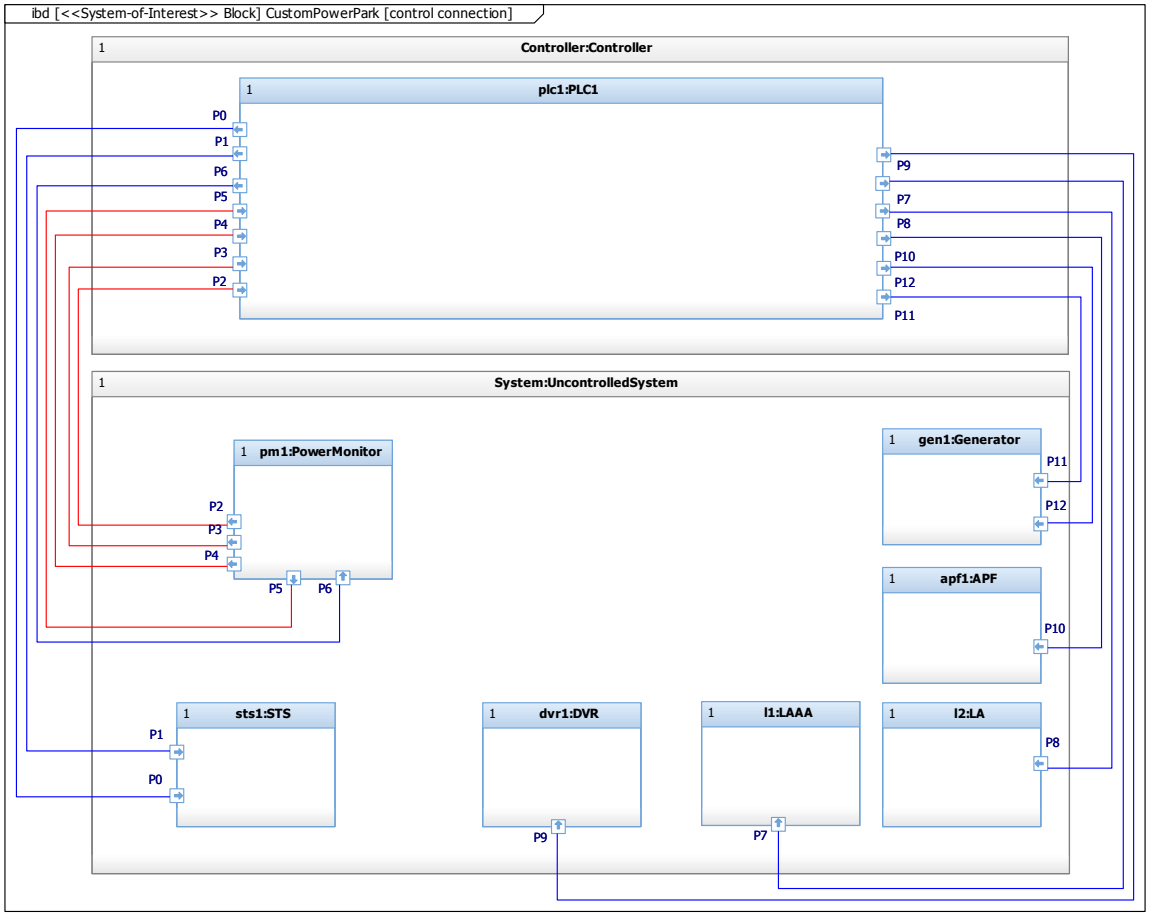


Figure 5.21 GO3 IBD: Control Connection

5.5 Conclusion

This chapter presented the application of the proposed framework by the case study on the system CPP. By this case study, the entire modeling process for AC is introduced in detail. The case study shows the advantages of the proposed framework as follows:

(1) By using SysML structure diagrams, the formal and physic concepts and composition can be described. At the beginning stage of design process, BDDs are built to define the target system which specifies uncontrolled system structure and the components to be controlled. The formal events are identified from physic components through semantic mapping of the operations. Secondly, the structures of formal models are explicitly presented by BDD and IBD, which include the models of plants, supervisor and controllers respectively. Thirdly, BDD and IBD are built to describe concrete controller and control connections to physic controlled systems. The SysML structure diagrams are proved to be able to handle different situations of controller implementation and link the formal supervisors, controllers and physic controllers. In the case study, we've shown the feasibility to model the controller when the supervisor is modular and implement to a monolithic PLC device and another situation where we should implement supervisors to

distributed controller.

(2) The formalization problems can be well coped with the help of SysML requirement diagrams. A requirement diagram is built to represent the analysis and decomposition of input requirements. The three-level requirement models clarified the traceability of requirements from high abstracted mission level to execution level. With the help of formalism, the semantic of description of E-requirements and formal specifications are bridged and the consistency between them can be verified. Therefore, the traceability between requirements and formal specification can be constructed and another requirement diagram is built to explicitly present the traceability.

(3) The architecture of the proposed framework integrates the necessary models and methods to realize a complete AC development process for engineering practice. Compare with typical SCT-based process, the supplement of SysML models extends the modeling scope of just using formal models. For the perspective of MBSE, the case study shows that the proposed framework provides a comprehensive description of the system including target system and controller rather than focusing on supervisor synthesis and implantation in the typical way. With regard to the four limitations of SCT, the case study shows that the proposed framework provides feasible solutions.

To sum up, the comparison between SCT-based process and proposed process are shown in Figure 5.22.

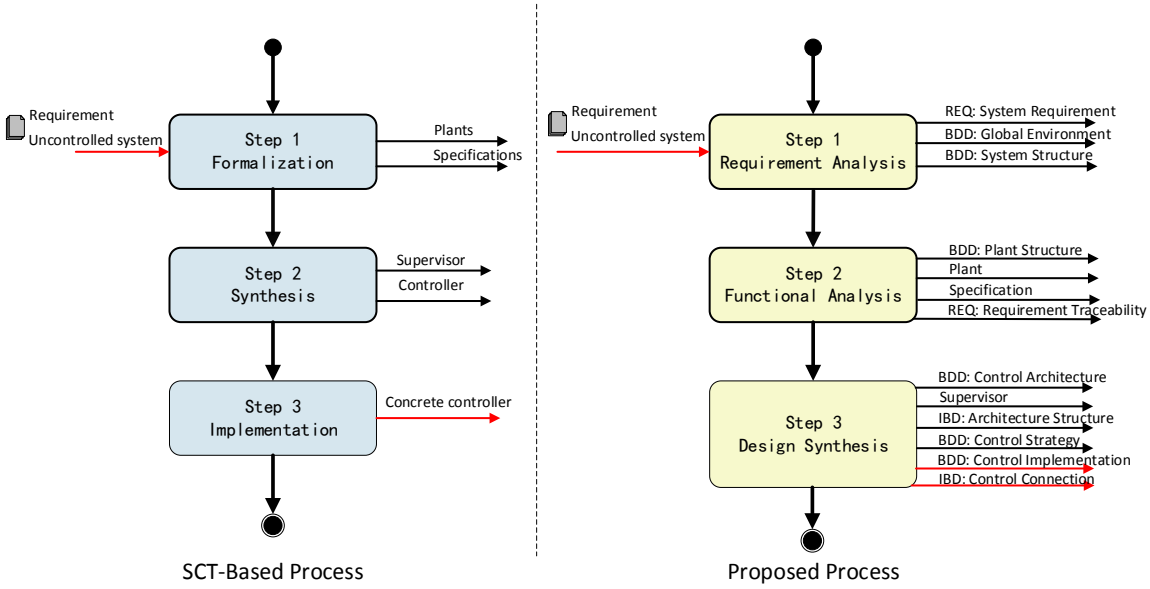


Figure 5.22 Comparison between SCT-based process and proposed process

Chapter 6

General Conclusion

6.1 Conclusion

Formal approaches such as SCT are proven to be more and more important for AC development when facing the emerging challenge of the flexible system or complex system in the industrial domain. In view of the fact that SCT is seldom applied for AC in the engineering practice, this study firstly reviewed the typical SCT-based modeling process and four main limitations are identified: (1) Limitation 1: lack of interpretation of the link between informal requirements and formal specifications, and between free behaviors of physic system and formal plants in typical development process; (2) Limitation 2: lack of structure models (3) Limitation 3: lack of implementation models, and (4) Limitation 4: lack of global framework to standardize the modeling process of AC for engineering practice. Although there are a number of state-of-the-art methods and contributions proposed within the framework of SCT, there are still gaps between academic study and engineering practice. MBSE is investigated as the possible solution for handling SCT limitations. However, the combination of MBSE and formal methods are without any solution for the context of SCT-based AC development process. Focusing on dealing with these issues, this study proposed a novel modeling framework for AC by integrating MBSE and SCT.

The first contribution of this study is the proposition of global modeling process. In order to integrate formal approach in MBSE process and take into consideration the solutions for limitations of SCT, 13 viewpoints are identified to be related to different aspects of the global architecture: (1) for formalization of plants, the focus is on the structure of physic system by the perspective from context level to component level; (2) for formalization of specification, the informal requirements and the traceability should be identified; (3) for supervisor synthesis and implementation, we mainly concern about the description of controller at formal/logic/physic level. 10 proposed SysML models and 3 formal models are defined to reflect the corresponding viewpoints. The identification of viewpoints preliminarily constructs the link between SCT and

MBSE. Based on the allocation of viewpoints to the three main MBSE modeling phases (requirement analysis, function analysis and design synthesis), the global integrated modeling process is developed. In this study, the details of proposed modeling process are presented. 14 main activities are defined along with process (4 activities in requirement analysis phase, 4 activities in function analysis phase and 6 activities in design synthesis phase). The inputs and outputs are identified for each activity. Metamodel is used to define the syntax and semantic of SysML models adapted to the context of AC.

The second contribution of this study is the methods to link SysML models and formal models. Problems (see chapter 2) has to be faced during the formalization of plant and specification. Different methods are proposed in order to keep the consistency between semi-formal SysML models and formal plants and specifications respectively. In order to link the plants and physic components, the activity of plant identification is detailed. A template-based approach is proposed to guarantee that the results of plants can correctly reflect the behaviors of physic components and the modeling process can be improved. On the other hand, we propose E-requirement to compose the ambiguously-narrative high-level requirements to explicit description so that the formal specifications can be traced. A traceability verification method and algorithm are proposed based on the integration by temporal logic and the consistency between requirements and specifications can be verified.

A case study is introduced to present application of the proposed modeling process, in which a Custom Power Park (CPP) is taken as target system. The entire modeling process from stakeholders' needs to the concrete controller implementation is presented in detail by the results of corresponding SysML models and formal models. We also discuss the models for some other situations for different needs of design. Besides, the template-based approach is implemented by an experimental program to realize an automatic analysis and transformation from BDD to plant models. The case study shows the advantages of the proposed modeling framework for engineering practice:

(1) Traceability: The formal models of plant and specification are linked explicitly by corresponding SysML diagrams such as BDDs and requirement diagrams. Besides, at formal level, the concept of supervisor synthesis is described in the SysML diagram to show the link among formal models. In BDD, the link between local and global plants (synchronization), between plants and supervisors and between specification and supervisors are presented (synthesis). We define the semantics of formal concepts for SysML elements are therefore, the concepts such as synthesis and synchronization can be explicitly visualized. To explicit presentation the relationship between supervisor and controller, another BDD is built in which the supervisor, controller and control strategies can be presented.

(2) Reusability: The global modeling framework is proven by the case study and additional

situation study to be reusable and adaptive for engineering practice. The definition of metamodel for each SysML diagram specifies the involved elements and their semantics which should be instantiated in application and provides guides for engineers. Different control architecture of AC can be handled and modeled by the proposed SysML diagrams. The template-based approach, which realizes an automatic transformation from physic system to formal models, improves the reusability of modeling framework as the technique simplify modeling process and guarantee the accuracy of the models which conforms to the engineering practice.

(3) Tool support: Both SysML and SCT can be well supported by modeling tools (IBM Rational Rhapsody® and Supremica in this study). Based on the XMI metadata interchange standard, the information between different tools can be exchanged. The template-based approach implemented by XMI shows the benefit of tool support in the proposed framework for engineering practice.

6.2 Expectation

Based on the contributions in this work, future researches can be furthered in the following directions:

Link Representation

The link between formal elements and SysML elements (e.g. events and operations) is identified in this study. However, an explicit presentation for the link by SysML diagrams is not achieved. The main reason is the link between formal elements and SysML elements are difficult to model by SysML. We intended to model by IBD like the model for architecture structure. However, the link between formal elements is feasible while IBD cannot build the link of two different element types, which can be regarded one of the limitations of SysML. Therefore, it is one of the problems remains in this study.

Library and Template

In this study, we just present a template-based approach based on the tentative library for a case study. It is a prospective technology to be extended in order to realize an efficient and accurate way of plant formalization. To this end, it is necessary to study on an industrial domain rather than a particular case study so that more typical physic components can be parameterized. Secondly, one issue still remaining in this study is that formalization of specification still depend on the expertise although the contributions in this study provide the verification method. The parameterized template for specifications could be another noteworthy research direction. In fact, in [67], the authors have already proposed the model of parameterized specification. However, the model is not too simple and not general enough to be made use of. It is indeed a challenge to parameterize formal specification as it involves in the identification of numbers of behaviors and

the solutions to integrate them by different combinations. An initial idea to realize the proposition is to standardize the narrative description of requirement. Nevertheless, once the solution for this issue is proposed, the formalization problem can be completely handled and the specification verification can also be eliminated. Another proposition for requirement formalization is to develop the verification check program based on the activity **AF1.2** (see Section 4.3.4). The purpose of this proposition is to eliminate the temporal logic verification process and an automatic verification process is friendlier to engineering practice.

Standardized Control Strategy Model

In this study, the models for control strategy cannot be abstracted by a metamodel as no standardized control strategy can be applied. In fact, the boundary among supervisor, controller and control implementation is sometimes ambiguous and therefore, people directly implement the supervisor by their proper methods by skipping controller synthesis, which leads to the lack of systematic study on this point. The controller nowadays is designated in different form such as automaton, algorithm or narrative strategy, according to the particular case studies. It is of great interest to represent the controllers by unified SysML models in the future study.

Control and Implementation Architecture

Due to the scope of the case study in this work, two kinds of control implementation architectures are presented (monolithic controller and modular controller). The SCT provides a number of control and implementation architecture. It is of interest that more controlled systems should be studied to show that proposed framework can be applied to modeling different architecture (such as hierarchical control). In fact, it is a challenge of the implementation of other control architecture. In fact, unlike the modular control which has a systematic implementation approaches the link of formal supervisors of other control architectures and concrete controller still unclear. More efforts are necessary for this issue.

Bibliography

[1] W. M. Wonham, K. Cai, K. Rudie, “Supervisory control of discrete-event systems: A brief history,” Annual Reviews in Control, 45: 250-256, 2018.

[2] W. M. Wonham and P. J. Ramadge, “On the supremal controllable sublanguage of a given language,” SIAM Journal on Control and Optimization, 25 (3): 637-659, 1987.

[3] F. Lin and W. M. Wonham, “On observability of discrete event systems,” Information Sciences, 44(3): 173-198, 1988.

[4] P. J. Ramadge, and W. M. Wonham, “The control of discrete event systems,” Proc. of IEEE, 77(1): 81-98, 1989.

[5] R. Cieslak, C. Desclaux, A. Fawaz and P. Varaiya, “Supervisory control of discrete-event processes with partial observations,” IEEE Transactions on Automatic Control, 33(3): 249-260, 1988.

[6] R. D. Brandt, V. Garg, R. Kumar, F. Lin, S. I. Marcus, and W. M. Wonham, “Formulas for calculating supremal controllable and normal sublanguages,” Systems & Control Letters, 15(2): 111-117, 1990.

[7] S. Lafortune and E. Chen, “The infimal closed controllable superlanguage and its application in supervisory control,” IEEE Transactions on Automatic Control, 35(4): 398-405, 1990.

[8] W. M. Wonham and P. J. Ramadge, “Modular supervisory control of discrete event systems,” Maths. Control, Signals Syst., 1(1): 13-30, 1988.

[9] M. H. de Queiroz and J. E. R. Cury, “Modular supervisory control of composed system,” Proc. of 19th Amer. Control Conf., 6: 4051-4055, 2000.

[10] R. Su, J. H. van Schuppen and J. E. Rooda, “Model abstraction of nondeterministic finite-state automata in supervisor synthesis,” in IEEE Transactions on Automatic Control,

55(11): 2527-2541, 2010.

[11] R. Su, J. H. van Schuppen and J. E. Rooda, “Aggregative synthesis of distributed supervisors based on automaton abstraction,” in IEEE Transactions on Automatic Control, 55(7): 1627-1640, 2010.

[12] M. Skoldstam, K. Akesson, and M. Fabian, “Modeling of discrete event systems using finite automata with variables, decision and control,” 2007 46th IEEE Conference on Decision and Control, Orleans, LA, pp. 3387-3392, 2007.

[13] M. R. Shoaei, L. Feng and B. Lennartson, “Supervisory control of extended finite automata using transition projection,” 2012 IEEE 51st IEEE Conference on Decision and Control (CDC), Maui, HI, pp. 7259-7266, 2012.

[14] T. Xu, H. Wang, T. Yuan and M. Zhou, “BDD-based synthesis of fail-safe supervisory controllers for safety-critical discrete event systems,” in IEEE Transactions on Intelligent Transportation Systems, 17(9): 2385-2394, 2016.

[15] F. Basile, R. Cordone, and L. Piroddi, “A branch and bound approach for the design of decentralized supervisors in Petri Net models,” Automatica, 52: 322-333, 2015.

[16] Y. Chen, Z. Li, K. Barkaoui, N. Wu and M. Zhou, “Compact supervisory control of discrete event systems by Petri Nets with data inhibitor arcs,” IEEE Transactions on Systems, Man, and Cybernetics: Systems, 47(2): 364-379, 2017.

[17] R. C. Hill and S. Lafortune, “Scaling the formal synthesis of supervisory control software for multiple robot systems,” 2017 American Control Conference (ACC), Seattle, WA, pp. 3840-3847, 2017.

[18] A. G. C. Gonzalez et al., “Supervisory control-based navigation architecture: a new framework for autonomous robots in Industry 4.0 environments,” IEEE Transactions on Industrial Informatics, 14(4): 1732-1743, 2018.

[19] A. Kharrazi, Y. Mishra and V. Sreeram, “Discrete-event systems supervisory control for a Custom Power Park”. IEEE Transactions on Smart Grid, 10(1): 483-492, 2019.

[20] R. J. M. Theunissen, “Supervisory control in health care systems”, PhD thesis, Eindhoven: Technische Universiteit Eindhoven, 2015.

[21] M. Uzam and G. Gelen, “An improved hybrid approach for the PLC based implementation of reduced RW supervisors,” The Turkish Journal of Electrical Engineering and Computer Science, 21: 394-419, 2013.

[22] A. D. Vieira, J. E. R. Cury and M. H. de Queiroz, “A model for PLC implementation of supervisory control of discrete event systems,” 2006 IEEE Conference on Emerging Technologies and Factory Automation, Prague, pp. 225-232, 2006.

[23] A. Kharrazi, V. Sreeram and Y. Mishra, “Power admission control of plug-in electric vehicles using supervisory control of discrete event system,” 2017 Australasian Universities Power Engineering Conference (AUPEC), Melbourne, VIC, pp. 1-6, 2017.

[24] W. H. Sadid, S. A. Abobakr and G. Zhu, “Discrete-event systems-based power admission control of thermal appliances in smart buildings,” IEEE Transactions on Smart Grid, 8(6): 2665-2674, 2017.

[25] J. Zaytoon, B. Riera, “Synthesis and implementation of logic controllers - A review”, Annual Reviews in Control, 43: 152-168, 2017.

[26] K. Forsberg and H. Mooz, “The relationship of system engineering to the project cycle,” in Proceedings of the First Annual Symposium of National Council on System Engineering, pp. 57-65, 1991.

[27] S. Mathur and S. Malik, “Advancements in the V-Model,” International Journal of Computer Applications, 1(12): 29-34, 2010.

[28] M. McHugh et al., “An agile V-model for medical device software development to overcome the challenges with plan driven software development lifecycles,” 5th International Workshop on Software Engineering in Health Care (SEHC), San Francisco, CA, pp. 12-19, 2013.

[29] S. C. Spangelo et al., “Applying Model Based Systems Engineering (MBSE) to a standard CubeSat,” 2012 IEEE Aerospace Conference, Big Sky, MT, pp. 1-20, 2012

[30] A. Giorgetti, A. Hammad and B. Tatibouët, “Using SysML for smart surface modeling,” 2010 First Workshop on Hardware and Software Implementation and Control of Distributed MEMS, Besan, TBD, France, pp. 100-107, 2010.

[31] P. David, V. Idasiak, and F. Kratz, “Reliability study of complex physical systems using SysML. Reliability Engineering & System Safety,” 95(4): 431-450, 2010.

[32] L. Piétrac, A. Lelevé, and S. Henry, “On the use of SysML for manufacturing execution system design,” In 16th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, Toulouse, France, 2011.

[33] L. Bassi, C. Secchi, M. Bonfe, and C. Fantuzzi, “A SysML-based methodology for manufacturing machinery modeling and design,” IEEE/ASME Transactions on Mechatronics,

16(6): 1049-1062, 2011.

[34] C. Cassandras and S. Lafortune, “Introduction to Discrete Event Systems”, Springer, 2nd edition, 2007.

[35] M. Ren, “An incremental approach for hardware discrete controller synthesis,” PhD thesis, INSA de Lyon, July 2011.

[36] Y. Chen, “Optimal supervisory control of flexible manufacturing systems,” PhD thesis, CNAM, France, 2015.

[37] A. D. Vieira et al., “A method for PLC implementation of Supervisory Control of Discrete Event Systems,” IEEE Transactions on Control Systems Technology, 25(1): 175-191, 2017.

[38] Y. Tatsumoto, M. Shiraishi, K. Cai and Z. Lin, “Application of online supervisory control of discrete-event systems to multi-robot warehouse automation,” Control Engineering Practice, 81: 97-104, 2018.

[39] C. R. C. Toricco, A. B. Leal, A. T. Y. Watanabe, “Modeling and supervisory control of mobile robots: a case of a sumo robot,” IFAC-PapersOnLine, 49(32): 240-245, 2016.

[40] M. R. Rodr ́guez et al., “Supervisory Control for High-Voltage Direct Current Transmission Systems,” 20th IFAC World Congress, Toulouse, France. 50 (1), 2017.

[41] S. T. J. Forschelen, J. M. van de Mortel-Fronczak, R. Su and J. E. Rooda. “Application of supervisory control theory to theme park vehicles”. Discrete Event Dynamic Systems, 22(4): 511-540, 2012.

[42] F. F. H. Reijnen, M. A. Goorden, J. M. van de Mortel-Fronczak and J. E. Rooda, “Supervisory control synthesis for a waterway lock,” 2017 IEEE Conference on Control Technology and Applications (CCTA), Mauna Lani, HI, pp. 1562-1563, 2017.

[43] F. Yari, S. Hashtrudi-Zad, S. Tafazoli, “Robust supervisory control of a spacecraft propulsion system,” IFAC-PapersOnLine, 49(17): 200-205, 2016.

[44] H. Garavel and S. Graf, “Formal methods for safe and secure computer systems,” Federal Office for Information Security, 2013.

[45] J. Hopcroft, R. Motwani and J. Ullman, “Introduction to automata theory, languages, and computation (3rd Edition),” Pearson, 2013.

[46] R. Leduc, B. Brandin, M. Lawford, and W. M. Wonham, “Hierarchical interface-based supervisory control, Part I: Serial case,” IEEE Trans. Autom. Control, 50(9): 1322-1335,

2005.

[47] R. J. Leduc, M. Lawford and W. M. Wonham, “Hierarchical interface-based supervisory control-part II: parallel case,” in IEEE Transactions on Automatic Control, 50(9): 1336-1348, 2005.

[48] P. Koz ́ak and W. M. Wonham, “Fully decentralized solutions of supervisory control problems”. Report 9310, Systems Control Group Report. Department of Electrical Engineering. University of Toronto, 1993.

[49] G. Barrett and S. Lafortune, “Decentralized supervisory control with communicating controllers,” in IEEE Transactions on Automatic Control, 45(9): 1620-1638, 2000.

[50] J. Komenda, J. H. van Schuppen, B. Gaudin and H. Marchand, “Supervisory control of modular systems with global specification languages”. Automatica 44: 1127-1134, 2008.

[51] K. Cai, W. M. Wonham, “A top-down approach to distributed control of Discrete-Event Systems,” IEEE Transactions on Automatic Control 55(3): 605-618, 2010.

[52] Y.Qamsane, A.Tajer, A.Philippot, “Distributed supervisory control synthesis for discrete manufacturing systems,” IFAC-PapersOnLine, 49(12): 396-401, 2016.

[53] L. Piétrac, S. Chafik and L. Regimbal, “Application de la théorie de la supervision : un exemple de conception de programmes d’API,” Sciences et Technologies de l'Automatique, e-STA, 2003.

[54] A. Morgenstern and K. Schneider, “Synthesizing deterministic controllers in supervisory control,” Informatics in Control, Automation and Robotics II, Springer, Dordrecht, 2007.

[55] D. Paquereau, “Application de la théorie du contrôle par supervision à la gestion automatique du trafic d’une ligne de métro,” Ph.D thesis, INSA Lyon, 2015.

[56] M. Fabian, A. Hellgren, “PLC-based implementation of supervisory control for discrete systems”, Proceedings of the 37th IEEE Conference on Decision and Control, Vol. 3: 3305-3310, 1998.

[57] M. Roth, L. Litz, J.-J. Lesage, “Identification of discrete event systems: implementation issues and model completeness,” 7th International Conference on Informatics in Control, Automation and Robotics (ICINCO), Jun 2010, Funchal, Portugal. 3: 73-80, 2010.

[58] P. Malik, “Generating controllers from discrete-event models,” In F. Cassez, C. Jard, F. Laroussinie, & M. D. Ryan (Eds.), Proceedings of the summer school in modelling and verification of parallel processes (MOVEP), pp. 337-342, 2002.

[59] M. Fabian and A. Hellgren, “PLC-based implementation of supervisory control for Discrete Event Systems,” Proceedings of the 37th IEEE Conference on Decision and Control, Tampa, FL, USA, 3: 3305-3310, 1998.

[60] G. Mušić, D. Matko, “Discrete event control theory applied to PLC programming”, AUTOMATIKA, 43: 1-2 and 21-28, 2002.

[61] P. Malik, “From supervisory control to nonblocking controllers for discrete event systems,” PhD thesis, Kaiserslautern: Department of Computer Science, University of Kaiserslautern, 2003.

[62] L. Andr é, D. L. L. da Cruz, H. Marcelo, “PLC-based implementation of local modular supervisory control for manufacturing systems”, Manufacturing System: 159-182, 2012.

[63] J. Cleland-Huang, “Software and systems traceability,” Springer, London., 2012.

[64] B. Ramesh and M. Jarke, “Toward reference models for requirements traceability,” IEEE Transactions on Software Engineering, 27(1): 58-93, 2001.

[65] O. C. Z. Gotel and C. W. Finkelstein, “An analysis of the requirements traceability problem,” Proceedings of IEEE International Conference on Requirements Engineering, Colorado Springs, CO, USA, pp. 94-101, 1994.

[66] L. Grigorov and K. Rudie, “Problem solving in control of discrete-event systems,” 2007 European Control Conference (ECC), Kos, pp. 5500-5507, 2007.

[67] L. Grigorov et al., “Conceptual design of Discrete-Event Systems using templates,” Discrete Event Dynamic Systems, 21(2): 257-303, 2011.

[68] R. R.H. Schiffelers et al., “Model-based rngineering of supervisory controllers using CIF,” Electronic Communications of the EASST, 21: 1-10, 2009.

[69] J. Markovski, “An integrated systems engineering framework for supervisor synthesis, verification, and performance evaluation,” 2013 European Control Conference (ECC), Zurich: 650-657, 2013.

[70] K. Blind and S. Gauch, “The impact of standardization and standards on innovation,” NESTA. NESTA Working Paper, 2017.

[71] D. Gianni, A. D’Ambrogio and A. Tolk, “Modeling and simulation-based systems engineering handbook,” CRC Press, 2014.

[72] D. Kaslow, G. Soremekun, H. Kim, S. Spangelo, “Integrated Model-Based Systems

Engineering (MBSE) applied to the simulation of a CubeSat mission," Proceedings of IEEE Aerospace Conference, Big Sky, MT, 2014.

[73] D. Kaslow, et al., "Developing a CubeSat Model-Based System Engineering (MBSE) reference model - interim status #3", Proceedings of IEEE Aerospace Conference, Big Sky, MT, 2017.

[74] E. Holger Geise et al., "The EAST-ADL architecture description language for automotive embedded software," P. Cuenot et al., Chapter 11 in Model-Based Engineering of Embedded Real-Time Systems, pp. 297-388, 2010.

[75] P. Pearce and S. Friedenthal, "A practical approach for modeling submarine subsystem architecture in SysML," Proc. 2nd SIA Science, Technology and Engineering Conference, Adelaide, Australia, pp. 347-360, 2013.

[76] M. Z. Iqbal et al., "Applying UML/MARTE on industrial projects: challenges, experiences, and guidelines," Software & Systems Modeling, 14(4): 1367-1385, 2015.

[77] B. Vogel-Heuser et al., "Model-driven engineering of manufacturing automation software projects - A SysML-based approach," Mechatronics, 24(7): 883-897, 2014.

[78] M. Broy and O. Slotosch, "From requirements to validated embedded systems," 2001 International Workshop on Embedded Software, pp. 51-65, 2001

[79] M. Marcos, E. Estevez, "Model-driven design of industrial control systems," In computer-aided control systems, pp. 1253-1258, 2008.

[80] S. Friedenthal, A. Moore and R. Steine, "A practical guide to SysML," 3rd Edition, the MK/OMG Press, 2014.

[81] A. P. Sage and S. M. Biemer, "Processes for system family architecting, design, and integration," IEEE Systems Journal, 1(1): 5-16, 2007.

[82] GEIA/EIA, "EIA-632 Processes for Engineering a System," 2003.

[83] International Organization for Standardization, "ISO/IEC 26702 IEEE Std 1220-2005 - Systems engineering - application and management of the systems engineering process," ISO/IEC/IEEE, 2007.

[84] ISO/IEC/IEEE International Standard - Systems and software engineering - System life cycle processes, in ISO/IEC/IEEE 15288 First edition 2015-05-15, pp. 1-118, 2015.

[85] International Organization for Standardization, "ISO/IEC/IEEE 42010: 2011 Systems and

software engineering - Architecture description," ISO/IEC, 2011.

[86] C. Delp, D. Lam, E. Fosse and C. Lee, "Model based document and report generation for systems engineering," 2013 IEEE Aerospace Conference, Big Sky, MT, pp. 1-11, 2013.

[87] B. Unhelkar, "The Art of Agile Practice: A composite approach for projects and organizations," CRC Press. pp. 56-59, 2016.

[88] H. Lykins, S. Friedenthal and A. Meilich, "Adapting UML for an Object-Oriented Systems Engineering Method (OOSEM)," Proceedings of the INCOSE 2000 International Symposium, Minneapolis, MN, 2000.

[89] H. -P. Hoffmann, "Harmony - SE/SysML deskbook: Model-Based Systems Engineering with Rhapsody," Rev. 1.51, Telelogic/I-Logix white paper, Telelogic AB, 2006.

[90] B. H. Rao, K.Padmaja and P.Gurulingam, "A brief view of Model Based Systems Engineering methodologies," International Journal of Engineering Trends and Technology (IJETT), 8(4): 3266-3271, 2013.

[91] D. Mazeika, A. Morkevicius and A. Aleksandraviciene, "MBSE driven approach for defining problem domain," 2016 11th System of Systems Engineering Conference (SoSE), Kongsberg, pp. 1-6, 2016.

[92] D. A. Wagner, M. B. Bennett, R. Karban, N. Rouquette, S. Jenkins and M. Ingham, "An ontology for State Analysis: Formalizing the mapping to SysML," 2012 IEEE Aerospace Conference, Big Sky, MT, pp. 1-16, 2012.

[93] J. Lu et al., "Towards A service-oriented framework for MBSE tool-chain development," IEEE 13th System of Systems Engineering Conference, Paris France, 2018.

[94] C. Allen, "MDDP: A pragmatic approach to managing complex and complicated MBSE models," IEEE International Symposium on Systems Engineering (ISSE), UK, Edinburgh, 2016.

[95] J.-L. Voirin, "Method & tools to secure and support collaborative architecting of constrained systems," in 27th International Congress of the Aeronautical Sciences, 2010.

[96] L. Tao, F. Jia and S. Yao, "B formal modeling based on UML statechart," 2015 Fifth International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC), Qinhuangdao: 1658-1663, 2015.

[97] S. Kinoshita, H. Nishimura, H. Takamura and D. Mizuguchi, "Describing software specification by combining SysML with the B method," 2014 IEEE International Symposium

on Software Reliability Engineering Workshops, Naples: 146-151, 2014.

[98] J. Liu and Z. Yang, "UML and B method based analysis and refinement for flight control software of unmanned aerial vehicle," 2008 International Conference on Computer Science and Software Engineering, Hubei, pp. 1135-1141, 2008.

[99] L. D. da Silva, A. Perkusich, "A model-based approach to formal specification and verification of embedded systems using colored Petri Nets", Component-Based Software Development for Embedded Systems, LNCS3778: 35-58, 2005.

[100] Garbi, Giuliani Paulineli Loureiro, Geilson, "Petri Nets for systems concurrent engineering", Improving Complex Systems Today, Advanced Concurrent Engineering. Springer, London: 75-82, 2011.

[101] M. Jamro, D. Rzonca, W. Rzasa, "Testing communication tasks in distributed control systems with SysML and Timed Colored Petri Nets model", Computers in Industry , 71: 77-87, 2015.

[102] T. Bouabana-Tebibel, S. H. Rubin and M. Bennama, "Formal modeling with SysML," 2012 IEEE 13th International Conference on Information Reuse & Integration (IRI), Las Vegas, NV: 340-347, 2012.

[103] D. A. Marca, & C. L. McGowan, "SADT structured analysis and design technique," New York: McGraw-Hill, 1988.

[104] OMG, "Business Process Model and Notation (BPMN) V2.0," 2011.

[105] C. Raffaele et al., "BPMN Miner: Automated discovery of BPMN process models with hierarchical structure," Information Systems, 56: 284-303, 2016.

[106] R. T. Kolagari et al., "Model-based analysis and engineering of automotive architectures with EAST-ADL: Revisited," International Journal of Conceptual Structures and Smart Applications, 3(2): 25-70, 2015.

[107] M. Alshayeb, N. Khashan and S. Mahmood, "A framework for an integrated unified modeling language," Frontiers of Information Technology & Electronic Engineering, 17(2): 143-159, 2016.

[108] "OMG Unified Modeling Language (OMG UML), Version 1.4," Object Management Group, September 2015.

[109] "OMG Systems Modeling Language (OMG SysML), Superstructure. Version 2.4.1," Object Management Group, Retrieved 9 April 2014.

[110] S. Lafi et al., “Modeling radio-frequency front-ends using SysML: A case study of a UMTS transceiver,” 1st International Workshop on Model Based Architecting and Construction of Embedded Systems, Toulouse, France, 2008.

[111] D. Emery and R. Hilliard, “Every architecture description needs a framework: Expressing architecture frameworks using ISO/IEC 42010,” Proceedings of Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture, (WICSA/ECSA’ 09), pp. 31-40, Cambridge, UK, 2009

[112] M. Luisa, F. Mariangela, and I. Pierluigi, “Market research for requirements analysis using linguistic tools”, Requirements Engineering, 9(1): 40-56, 2004.

[113] M. dos Santos and Jos L. M. Vrancken. “Model-driven user requirements specification using SysML”, Journal of Software, 3(6):57-68, 2008.

[114] F. Peres, J. Yang, and M. Ghazel. “A formal framework for the formalization of informal requirements,” International Journal of Soft Computing and Software Engineering, 2(8): 14-27, 2012.

[115] Q. A. D. S. Ribeiro, F. G. C. Ribeiro and M. S. Soares, “A technique to architect real-time embedded systems with SysML and UML through multiple views,” In Proceedings of the 19th International Conference on Enterprise Information Systems, Volume 2: 287-294, Porto, Portugal, 2017.

[116] M. R. Curreri, A. Kam and C. Hein, “Modeling cyber threats with SysML,” 28th INCOSE International Symposium, Washington, DC, USA, pp. 1337-1361, 2018.

[117] G. Caltais et al., “SysML to NuSMV model transformation via object-orientation,” 6th International Workshop on Design, Modeling, and Evaluation of Cyber Physical Systems, PA, USA, pp. 31-45, 2016.

[118] C. Brecher, J. A. Nittinger, & A. Karlberger, “Model-based control of a handling system with SysML,” Procedia Computer Science, 16: 197-205, 2013.

[119] F. Mhenni, et al., “Flight control system modeling with SysML to support validation, qualification and certification,” IFAC-PapersOnLine, 49(3): 453-458, 2016.

[120] V.Gervasi and B. Nuseibeh, “Lightweight validation of natural language requirements,” Software: Practice and Experience, 32(2): 113-133, 2002.

[121] G. Cabral and A. Sampaio, “Formal specification generation from requirement documents,” Electronic Notes in Theoretical Computer Science, 195: 171-188, 2008.

[122] D. d. A. Ferreira and A. R. d. Silva, “A controlled natural language approach for integrating requirements and model-driven engineering,” 2009 Fourth International Conference on Software Engineering Advances, Porto, pp. 518-523, 2009.

[123] M. Ghazel, J. Yang, E. -M. El-Koursi, “A pattern-based method for refining and formalizing informal specifications in critical control systems,” Journal of Innovation in Digital Ecosystems, 2:32-44, 2015.

[124] K. Pohl, “Requirements engineering: fundamentals, principles, and techniques,” 1st edition, Springer Publishing Company, Incorporated, 2010.

[125] G. Ekberg and B. H. Krogh, “Programming discrete control systems using state machine templates,” In: Proceedings of the 8th International Workshop on Discrete Event Systems, pp. 194-200. Ann Arbor, MI, USA, 2006.

[126] L. Grigorov and K. Rudie, “Techniques for the parameterization of Discrete-Event System templates,” IFAC Proceedings Volumes, 43(12): 370-375, 2010.

[127] R. Malik, M. Fabian, K. Åkesson, “Modelling large-scale discrete-event systems using modules, Aliases, and Extended Finite-State Automata,” IFAC Proceedings Volumes, 44(1): 7000-7005, 2011.

[128] Z. Chen and D. Zhenhua, “Specification and verification of UML2.0 sequence diagrams using event deterministic finite automata,” 2011 Fifth International Conference on Secure Software Integration and Reliability Improvement - Companion, Jeju Island, 41-46, 2011.

[129] C. Zhang, Z. Duan, B. Yu, C. Tian and M. Ding, “A test case generation approach based on sequence diagram and automata models,” in Chinese Journal of Electronics, 25(2): 234-240, 2016.

[130] R. Grønmo, B. Møller-Pedersen, “From sequence diagrams to state machines by graph transformation”, International Conference on Theory and Practice of Model Transformations, Lecture Notes in Computer Science, 6142. Springer, Berlin, Heidelberg, 2010.

[131] F. Alhumaidan and N.A. Zafar, “Automated semantics treatment of sequence diagram defining grammar rules,” http://worldcomp-proceedings.com/proc/p2013/FCS7057.pdf, 2013.

[132] P. Sugunnasil, “Detecting deadlock in activity diagram using process automata,” 2016 International Computer Science and Engineering Conference (ICSEC), Chiang Mai, 2016.

[133] B. van der Sanden et al., “Modular model-based supervisory AC for wafer logistics in

lithography machines," 2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS), Ottawa, ON, pp. 416-425, 2015.

[134] D. van Beek et al., "CIF 3: Model-based engineering of supervisory controllers," Tools and Algorithms for the Construction and Analysis of Systems, ser. LNCS. Springer, 8413: 575-580, 2014.

[135] J. Markovski, D. A. van Beek, R. J. M. Theunissen, K. G. M. Jacobs and J. E. Rooda, "A state-based framework for supervisory control synthesis and verification," 49th IEEE Conference on Decision and Control (CDC), Atlanta, GA: 3481-3486, 2010.

[136] W. Zhang, et al., "Towards tool integration through artifacts and roles," 2012 19th Asia-Pacific Software Engineering Conference, Hong Kong, pp. 603-613, 2012.

[137] C. Guychard et al., "Conceptual interoperability through models federation," In: Semantic Information Federation Community Workshop, 2013.

[138] Object Management Group, "A proposal for an MDA foundation model," 2005-04-01, http://www.omg.org/docs/ormsc/05-04-01.pdf.

[139] A. Legendre, "Ingénierie système et Sûreté de fonctionnement : Méthodologie de synchronisation des modèles d'architecture et d'analyse de risques," Ph.D thesis, Université Paris-Saclay, 2017.

[140] H. O. Aliyu, O. Maïga and M. K. Traoré, "Un langage graphique pour la modélisation et l'analyse des systèmes réactifs," Modélisation des Systèmes Réactifs conference, Marseille, 2017.

[141] W. Sun, H. Zhang, C. Feng and Y. Fu, "A method based on meta-model for the translation from UML into Event-B," 2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Vienna, pp. 271-277, 2016.

[142] A. Finkelstein, et al. "Viewpoints: a framework for integrating multiple perspectives in system development". In: International Journal of Software Engineering and Knowledge Engineering 2(1): 31-58, 1992.

[143] Y. Takahiro, "A generic method for defining viewpoints in SysML", INCOSE International Symposium, 19(1): 844-852, 2009.

[144] M.Törngren et al., "Integrating viewpoints in the development of mechatronic products," Mechatronics, 24(7): 745-762, 2014.

[145] L. Delligatti, "SysML Distilled: A brief guide to the systems modeling language," first

edition, Addison-Wesley Professional, 2013.

[146] A. Kinder, V. Barot, M. Henshaw and C. Siemieniuch, “System of Systems: “Defining the system of interest”,” 2012 7th International Conference on System of Systems Engineering (SoSE), Genova, pp. 463-468, 2012.

[147] M. H. de Queiroz and J. E. R. Cury, “Synthesis and implementation of local modular supervisory control for a manufacturing cell,” Sixth International Workshop on Discrete Event Systems, Zaragoza, Spain, pp. 377-382, 2002.

[148] A. Aurum and C. Wohlin, “Requirements engineering: Setting the context.” in Engineering and Managing Software Requirements, Springer-Verlag, pp. 1-15, 2005.

[149] R. Fulton, R. Vandermolen, “Chapter 4: Requirements - writing requirements,” Airborne Electronic Hardware Design Assurance: A Practitioner's Guide to RTCA/DO-254. CRC Press. pp. 89–93. ISBN 9781351831420. Retrieved 15 June 2018.

[150] M. Cantarelli and J. Roussel, “Reactive control system design using the Supervisory Control Theory: Evaluation of possibilities and limits,” 2008 9th International Workshop on Discrete Event Systems, Goteborg, pp. 200-205, 2008.

[151] 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, 2007.

[152] A. Yahiaoui, “A distributed dynamic simulation mechanism for buildings automation and control systems,” Ph.D thesis, Technische Universiteit Eindhoven, 2013.

[153] D. B. Silva, “Application of the supervisory control theory to automated systems of multi-product manufacturing,” 2007 IEEE Conference on Emerging Technologies and Factory Automation, Patras, pp. 689-696, 2007.

[154] ISO/IEC 19503:2005 Information technology - XML Metadata Interchange (XMI).

[155] N.G.Leveson, “Engineering a Safer World,” MIT Press, 2011.

[156] R.Sun, D. Zhong, “Translating software requirement from natural language to automaton,” International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC), Shenyang, China, 2013.

[157] A. Fatwanto, “A concern-aware requirements engineering framework,” Ph.D Thesis, Canberra, ACT: The Australian National University, 2011.

[158] K. T. Seow, “Integrating temporal logic as a state-based specification language for

discrete-event control design in finite automata," in IEEE Transactions on Automation Science and Engineering, 4(3): 451-464, July 2007.

[159] F. Lin, "Analysis and synthesis of discrete event systems using temporal logic," Proceedings of the 1991 IEEE International Symposium on Intelligent Control, Arlington, VA, USA, pp. 140-145, 1991A.

[160] A. Goknil, "Semantics of trace relations in requirements models for consistency checking and inferencing," Software & Systems Modeling, 10(1): 31-54, 2011.

[161] N. Hatziargyriou, H. Asano, R. Iravani, and C. Marnay, "Microgrids," IEEE Power Energy Mag., 5(4): 78-94, 2007.

[162] A. Aghazadeh, R Noroozian, A. Jalilvand, and H. Haeri, "Combined operation of dynamic voltage restorer with distributed generation in custom power park," in Proc. Int. Conf. Environ. Elect. Eng. (EEIC), pp. 1-4, Rome, Italy, 2011.

[163] ISO/IEC 19503:2005: Information technology - XML Metadata Interchange (XMI), first edition, 2005.

[164] R. malik et al., "Supremica - an efficient tool for large-scale discrete event systems," IFAC-online, 50(1): 5794-5799, 2017.

Appendix A

Review on Supervisory Control Theory

A.1 Formal Language

More details on the theory of formal language can be found in [1].

Definition A.1 (Formal language). Each DES has an associated underlying event set Σ called an *alphabet*. A sequence of events from the alphabet is called a "word" or "string". An empty event string is denoted by ε. We denote the cardinality of an event set Σ as $|\Sigma|$. For a particular event set Σ, we denote its set of all possible finite strings of events by Σ^*. A *formal language* defined over an event set Σ is a subset of Σ^*.

The usual set operations, such as union, intersection, difference, and complement with respect to Σ^*, are applicable to languages since languages are sets. We define L_1 and L_2 as two languages over a same alphabet Σ. We have operations as follows:

- Union: $L_1 \cup L_2 = \{s \in \Sigma^* \mid s \in L_1 \vee s \in L_2\}$;
- Intersection: $L_1 \cap L_2 = \{s \in \Sigma^* \mid s \in L_1 \wedge s \in L_2\}$;
- Difference: $L_1 \setminus L_2 = \{s \in \Sigma^* \mid s \in L_1 \wedge s \notin L_2\}$;
- Complement: $\Sigma^* \setminus L_1 = \{s \in \Sigma^* \mid s \notin L_1\}$;
- Concatenation: $L_1 L_2 = \{s \in \Sigma^* \mid s = s_1 s_2 \wedge s_1 \in L_1 \wedge s_2 \in L_2\}$;
- Prefix-closure: $\overline{L} = \{s \in \Sigma^* \mid \exists s' \in \Sigma^*, ss' \in L\}$;
- Kleene-closure: Let $L \subseteq \Sigma^*$, then

$$L^* = \{\varepsilon\} \cup L \cup LL \cup LLL \cup ...$$

Definition A.2 (Projection). For a language L defined on event set Σ, its *projection* to a smaller event set $\Sigma_1 \subseteq \Sigma^*$ is a language $Proj_{\Sigma_1}$ constructed by applying the *Proj* operations on each of its

strings:

$$Proj(\varepsilon) = \varepsilon;$$
$$Proj(\sigma) = \begin{cases} \sigma & \text{if } \sigma \in \Sigma_1 \\ \varepsilon & \text{if } \sigma \in \Sigma \setminus \Sigma_1 \end{cases};$$
$$Proj(s\sigma) = Proj(s)Proj(\sigma).$$

As can be seen from the definition, the projection operation takes a string formed from the larger event set (Σ) and *erases* events in it that do not belong to the smaller event set (Σ_1). We will also be working with the corresponding inverse map

$$Proj^{-1} : \Sigma_1^* \to 2^{\Sigma^*}$$

defined as follows:

$$Proj^{-1}(t) = \{s \in \Sigma^* \mid Proj(s) = t\}.$$

A.2 Automaton

Regular languages are seldom used to exhaustively describe the behavior of a reactive system. Sometimes, regular expressions can be an intuitive way for partially specifying repetitive behavior requirements, but they are usually not used to exhaustively design a system. Instead, the most usual modeling artifact is the finite-state automaton.

Definition A.3 (Finite-state automaton). A deterministic Event-driven automaton is defined as a 5-tuple:

$$G = (Q, \Sigma, \delta, q_0, Q_m)$$

Where,

- Q is a finite set of states;
- Σ is a finite set of events associated with G;
- $\delta : Q \times \Sigma \to Q$ is the transition function;
- q_0 is the initial state;
- $Q_m \subseteq Q$ is a set of marked or accepting states.

Note that the transition function δ is usually a partial function. Thus each state can be associated with a set of events which are admissible at that particular state, i.e. for which δ is defined.

Definition A.4 (Active event function). Define active event function $\Gamma : Q \to 2^{\Sigma}$ as following:

$$\Gamma(q) = \{\sigma \mid \delta(q, \sigma) \text{ is defined}\}$$

Definition A.5 (Generated language). The generated language by $G=(Q,\Sigma,\delta,q_0,Q_m)$ is defined as:

$$L(G)=\{s=\sigma_1\sigma_2\ldots\sigma_n\in\Sigma^*\mid\delta(q_0,s)\text{ is defined}\}$$

Definition A.6 (Marked language). The marked language by $G=(Q,\Sigma,\delta,q_0,Q_m)$ is defined as:

$$L_m(G)=\{s=\sigma_1\sigma_2\ldots\sigma_n\in L(G)\mid\delta(q_0,s)\in Q_m\}$$

Remark Accepting (or marked) states Q_m are used in order to distinguish between desired undesired behaviors. Desired behaviors are execution paths which end with an accepting state $q_m\in Q_m$. Thus, accepting states q_m are used to specify the point where a finite behavior should terminate. □

In event-driven models, events can only occur one at a time. Simultaneous occurrence of two or more events is not allowed. This paradigm is used to model concurrent systems where events cannot arrive simultaneously. Concurrency is a fundamental property of most control systems: a collection of building blocks are composed together in order to achieve more complex control behaviors. The composition operation expresses a requirement of simultaneous run of the blocks composed together. This is formally defined by Milner's synchronous product recalled below.

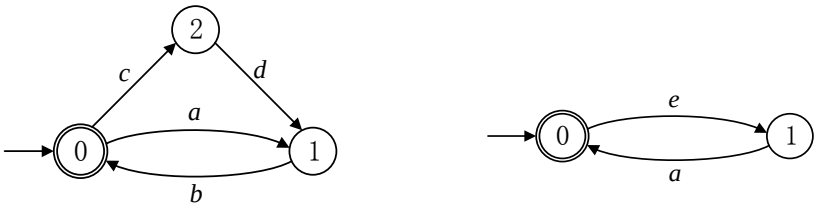
Definition A.7 (Synchronous composition). The synchronous composition of two finite-state automaton $G_1=(Q_1,\Sigma_1,\delta_1,q_{01})$ and $G_2=(Q_2,\Sigma_2,\delta_2,q_{02})$ is an automaton denoted as $G_1\|G_2$ and defined by:

$$G_1\|G_2=(Q_1\times Q_2,\Sigma_1\cup\Sigma_2,\delta_{G_1\|G_2},(q_{01},q_{02}))$$

Where, the transition function $\delta_{G_1\|G_2}:Q_1\times Q_2\times(\Sigma_1\cup\Sigma_2)\to Q_1\times Q_2$ is defined by:

$$\delta((q_1,q_2),\sigma)=\begin{cases}(\delta_1(q_1,\sigma),q_2) & \text{if } \sigma\in\Gamma_1(q_1)\setminus\Sigma_2\\ (q_1,\delta_2(q_2,\sigma)) & \text{if } \sigma\in\Gamma_2(q_2)\setminus\Sigma_1\\ (\delta_1(q_1,\sigma),\delta_2(q_2,\sigma)) & \text{if } \sigma\in\Gamma_1(q_1)\cap\Gamma_2(q_2)\\ \text{undefined} & \text{otherwise}\end{cases}$$

Example A.1. *Synchronous composition of two simple machines.*



(a) Machine G1 (b) Machine G2

Figure A.1 Two simple automata

*Figure A.*1 *shows two simple machines, with shared event set* $\Sigma_1 \cap \Sigma_2 = \{b\}$. *The synchronous composition of such two machines is illustrated in Figure A.*2.

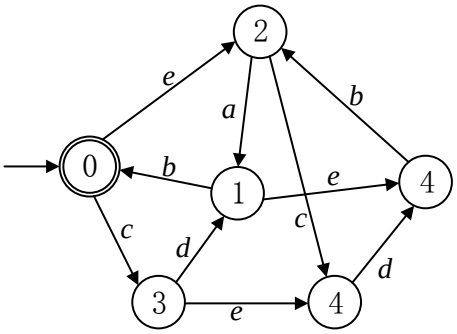


Figure A.2 Synchronous composition of two simple machines

A.3 Examples of Specification

The examples of four typical specifications are shown as follows:

Example A.2. *Illegal state. If a specification identifies certain states of G as illegal, then it suffices to delete these states from G, that is, remove these states and all the transitions attached to them, then do the Ac operation.*

Example A.3. *State splitting. Consider the example shown in Figure A.3. The specification prevents the occurrence of string* $a_1a_2b_2b_1$ *and* $a_2a_1b_1b_2$. *To remember how state 3 in the system model is achieved, we have to split state 3 into two states: state 3 and 4 in the specification.*

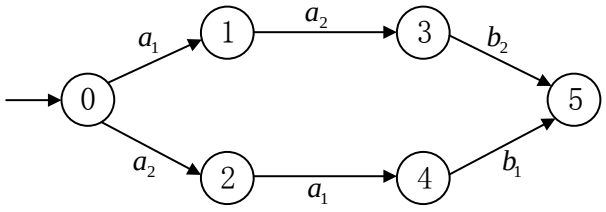


Figure A.3 State splitting specification

Example A.4. *Event alternation. Consider a specification which requires that event a and b occur alternately, starting by a. This specification can be recognized by an automaton shown in Figure A.4.*

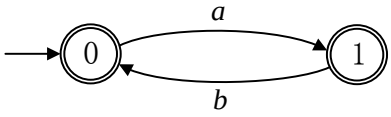


Figure A.4 Event alternation specification

Example A.5. *Illegal substring. Figure A.5 shows a specification expressed by an automaton. This automaton recognize a language which includes all strings {a, b, c, d}* in except those contain the substring abcd.*

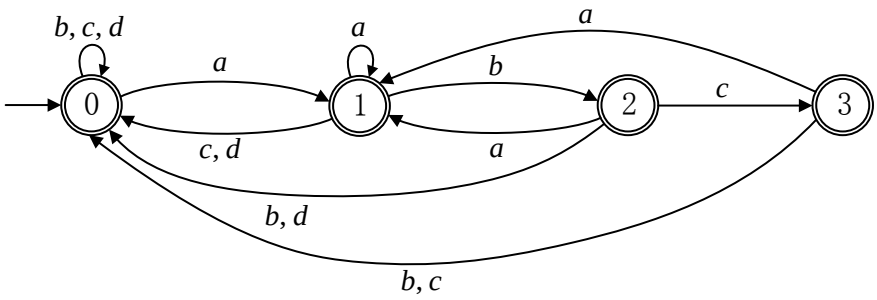


Figure A.5 Illegal substring specification

A.4 Supervisor Properties

The supervisor cannot be synthesized directly to the system without verification because it may lead to big problems. For example, the system behavior may block under supervisory control. There are two basic properties of supervisor should be verified during synthesis: the controllability and nonblocking.

Definition A.8 (Controllability). Consider DES $G=(Q,\Sigma,\delta,q_0,Q_m)$ where $\Sigma_{uc}\subseteq\Sigma$ is the set of uncontrollable events. Let $K\subseteq L(G)$, where $K\neq\varnothing$. Then there exists supervisor S such that $L(S/G)=\bar{K}$ only and if only

$$\bar{K}E_{uc}\cap L(G)\subseteq\bar{K}$$

This condition on K is called the *controllability* condition.

Definition A.9 (Nonblocking). Consider DES $G=(Q,\Sigma,\delta,q_0,Q_m)$ where $\Sigma_{uc}\subseteq\Sigma$ is the set of uncontrollable events. Let $K\subseteq L_m(G)$, where $K\neq\varnothing$. Then there exists a nonblocking supervisor S for G such that

$$L_m(S/G)=K \quad \text{and} \quad L(S/G)=\bar{K}$$

if and only if the condition L_m - closure holds:

$$K=\bar{K}\cap L_m(G)$$

Definition A.10 (Supremal Controllable Sublanguage). Let us define the class of controllable sublanguages of K by $C_{in}(K):=\{L\subseteq K\mid \bar{K}E_{uc}\cap L(G)\subseteq\bar{K}\}$. Then the supremal controllable

sublanguage denoted by $K^{\uparrow C}$ is :

$$K^{\uparrow C} := \bigcup_{L \in C_{in}(K)} L$$

A.5 Supervisor Synthesis Algorithm

Given a number of plant models denotes by $G_j(Q_j, \Sigma_j, \delta_j, q_{j,0}, Q_{j,m})$ and specification models denoted by $H_k(X_k, \Sigma, \Delta_k, x_{k,0}, X_{k,m})$, then the monolithic supervisor which generates the supreme controllable language can be computed by Algorithm A.1.

Algorithm A.1: Algorithm for computing *MonSup*

Input: Formal model of plants $Plant = G_{j=1}^{r}(Q_j, \Sigma_j, \delta_j, q_{j,0}, Q_{j,m})$

Formal model of Specifications $Spec = H_{k=1}^{n}(X_k, \Sigma, \Delta_k, x_{k,0}, X_{k,m})$

Output: Monolithic supervisor $MonSup = (Y, \Sigma, \varphi, y_0, Y_m)$

Begin

Init

$G \leftarrow \|_{j=1}^{r} G_j$;

$H \leftarrow \|_{k=1}^{n} H_k$;

$MonSup_0 = (Y_0, \Sigma, \varphi, (q_0, x_0), Y_{0,m}) \leftarrow G \times H$ with $Y = Q \times X$;

$Y_0 \subseteq Q_0 \times X_0$;

$i \leftarrow 0$;

loop $MonSup_{i+1}$ is not an empty automaton

$Y_i' \leftarrow \{(q,x) \in Y_i \mid \Gamma_G(x) \cap \Sigma_{uc} \subseteq \Gamma_{MonSup_i}((q,x))\}$

with $\Gamma_G(x) = \{\sigma \in \Sigma \mid \delta(q,\sigma)!\}$ and $\Gamma_{MonSup_i}((q,x)) = \{\sigma \in \Sigma \mid \varphi_i((q,x),\sigma)!\}$;

$\varphi_i' \leftarrow \varphi_i \mid Y_i'$;

$Y_{i,m}' \leftarrow Y_{i,m} \cap Y_i'$;

$MonSup_{i+1} \leftarrow Trim(Y_i', \Sigma, \varphi_i', (q_0, x_0), Y_{i,m}')$

with $MonSup_{i+1} = (Y_{i+1}, \Sigma, \varphi_{i+1}, (q_0, x_0), Y_{i+1,m})$;

if $MonSup_{i+1} == MonSup_i$

$MonSup \leftarrow MonSup_{i+1}$;

Break;

else

$i \leftarrow i+1$

end loop

End

A modular supervisor consists of several local supervisors. Each local supervisor can be computed by the global plant and one specification, instead of the global specification in monolithic supervisor. Therefore, the local supervisor should be computed successively on the basis of specifications. The adapted computation algorithm of modular supervisor is shown in Algorithm A.2.

Algorithm A.2: Algorithm for computing *ModSup*

Input: Formal model of abstract plant $Plant = G_{j=1}^{r}(Q_j, \Sigma_j, \delta_j, q_{j,0}, Q_{j,m})$

Formal model of abstract Specification $Spec = H_{k=1}^{n}(X_k, \Sigma_k, \Delta_k, x_{k,0}, X_{k,m})$

Output: Modular supervisor $ModSup = LocSup_{i=1}^{n}(Y_i, \Sigma, \varphi_i, y_{i,0}, Y_{i,m})$

Begin

Init

$G \leftarrow \|_{j=1}^{r} G_j$;

$l \leftarrow 1$;

loop $l <= n$

$LocSup_{0}^{l} = (Y_i^l, \Sigma_k, \varphi^l, (q_0, x_0), Y_{0,m}^l) \leftarrow G \times H_l$ with $Y^l = Q \times X$;

$Y_0^l \subseteq Q_0 \times X_0$

$i \leftarrow 0$

loop $LocSup_{i+1}$ is not an empty automaton

$Y_i^l{}' \leftarrow \{(q,x) \in Y_i^l \mid \Gamma_G(x) \cap \Sigma_{uc} \subseteq \Gamma_{LocSup_i^l}((q,x))\}$

with $\Gamma_G(x) = \{\sigma \in \Sigma \mid \delta(q,\sigma)!\}$ and $\Gamma_{LocSup_i^l}((q,x)) = \{\sigma \in \Sigma \mid \varphi_i^l((q,x),\sigma)!\}$;

$\varphi_i^l{}' \leftarrow \varphi_i^l \mid Y_i^l{}'$;

$Y_{i,m}^l{}' \leftarrow Y_{i,m}^l \cap Y_i^l{}'$;

$LocSup_{i+1}^l \leftarrow Trim(Y_i^l{}', \Sigma, \varphi_i^l{}', (q_0, x_0), Y_{i,m}^l{}')$

with $LocSup_{i+1}^l = (Y_{i+1}^l{}', \Sigma, \varphi_{i+1}^l{}', (q_0, x_0), Y_{i+1,m}^l{}')$;

if $LocSup_{i+1}^l == LocSup_i^l$

$LocSup^l \leftarrow LocSup_{i+1}^l$;

Break;

else

$i \leftarrow i+1$

end loop

$l \leftarrow l+1$

end loop

End

Appendix B

Supplementary Results for Case Study

B.1 Specifications

B.1.1 Specification 2

(1) Modeling

Figure B.1 shows the specification model conforming to requirement grade AA. The description of requirement E1.2 specifies a cycle behavior of switching on/off dvr1 with monitor sensor conditions. Therefore, the interim states s1 and s3 are used to connect successive occurrences of sensor signal and switching event.

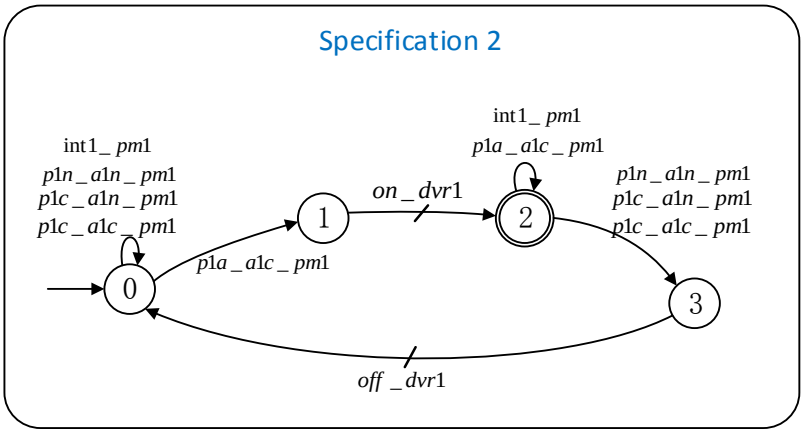


Figure B.1 Specification 2

(2) Verification

There is no sub-requirement initially defined for the E-requirement E1.2. However, two formalisms can be identified, as there is the key word “and” in the description. Therefore, the

requirement can be interpreted by the formalism as follows:

$$Spec2 \models S_2 = S_1^1 \wedge S_2^1$$

Each formalism can be identified as follows:

$$S_1^1 = ((\text{int}1\_pm1 \vee p1c\_a1n\_pm1 \vee p1c\_a1c\_pm1 \vee$$
$$p1n\_a1n\_pm1) \; p1a\_a1c\_pm1) \quad o$$
$$S_2^1 = ((\text{int}1\_pm1 \vee p1a\_a1c\_pm1)(p1c\_a1n\_pm1$$
$$\vee p1c\_a1c\_pm1 \vee p1n\_a1n\_pm1)) \rightarrow off\_dvr1$$

Therefore, the global formalism is explicit. $Spec2 \models S_2 = S_1^1 \wedge S_2^1$ can be verified.

B.1.2 Specification 3

(1) Modeling

The requirement E1.3 is traced by specification 3. To confirm the traceability between them, the specification 3 should be verified to conform to all the three sub-requirement of E1.3.

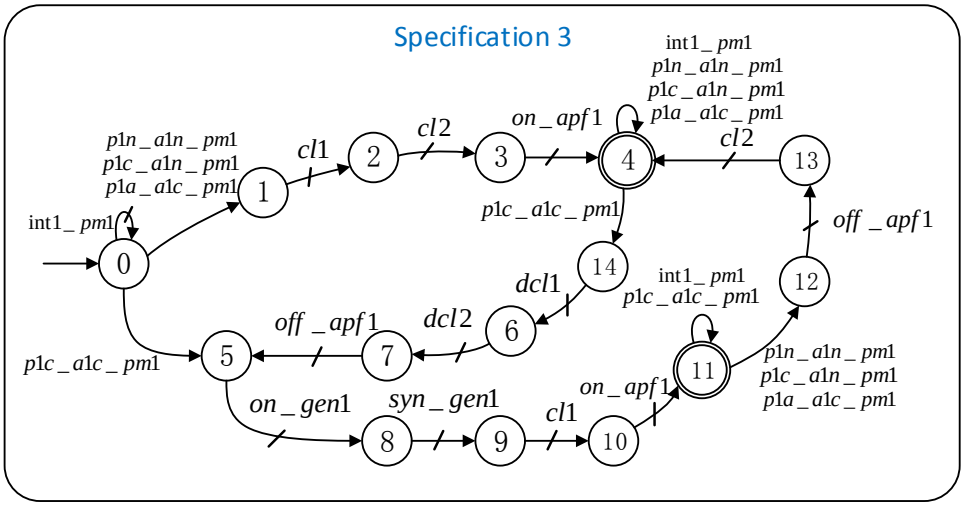


Figure B.2 Specification 3

(2) Verification

As the E-requirement E1.3 can be decomposed into E1.3.1 and E1.3.2 and E1.3.3 the specification 3 (Spec3) is supposed to satisfy the formalism S_3 denoted as follows:

$$Spec3 \models S_3 = S_1^1 \wedge S_2^1 \wedge S_3^1$$

The E1.3.1 can be identified as follows:

$$S_1^1 = \rightarrow((p1c\_a1n\_pm1 \vee p1n\_a1n\_pm1 \vee p1a\_a1c\_pm1) \rightarrow S_1^2) \vee (p1c\_a1c\_pm1 \rightarrow S_2^2))$$

Where,

$$S_1^2 = cl1 \rightarrow cl2 \rightarrow on\_apf1$$

$$S_2^2 = on\_gen1 \rightarrow syn\_gen1 \rightarrow cl1 \rightarrow on\_apf1$$

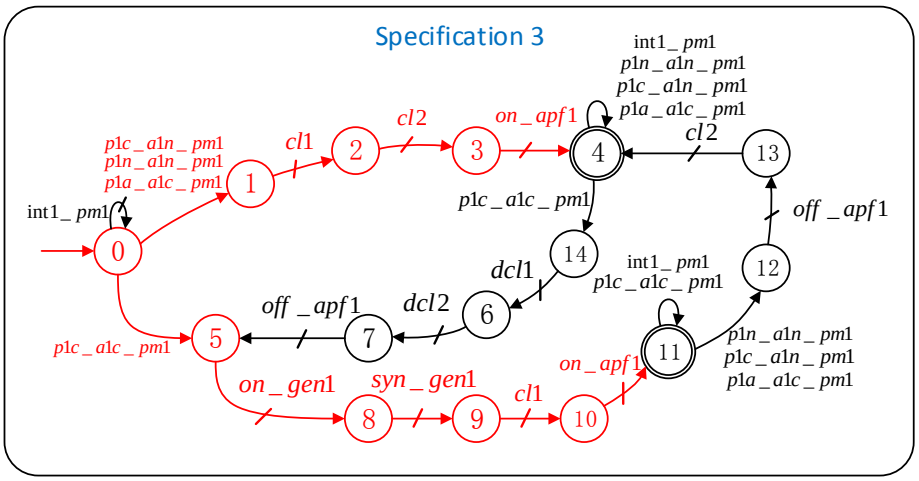


Figure B.3 Consistency verification for E1.3.1

The consistency verification for E1.3.1 is present in Figure B.3. Secondly, The E1.3.2 can be identified as follows:

$$S_2^1 = on\_apf1 \circlearrowleft p1c\_a1c\_pm1 \rightarrow S_3^2$$

$$S_3^2 = dcl1 \rightarrow dcl2 \rightarrow off\_apf1 \rightarrow S_2^2$$

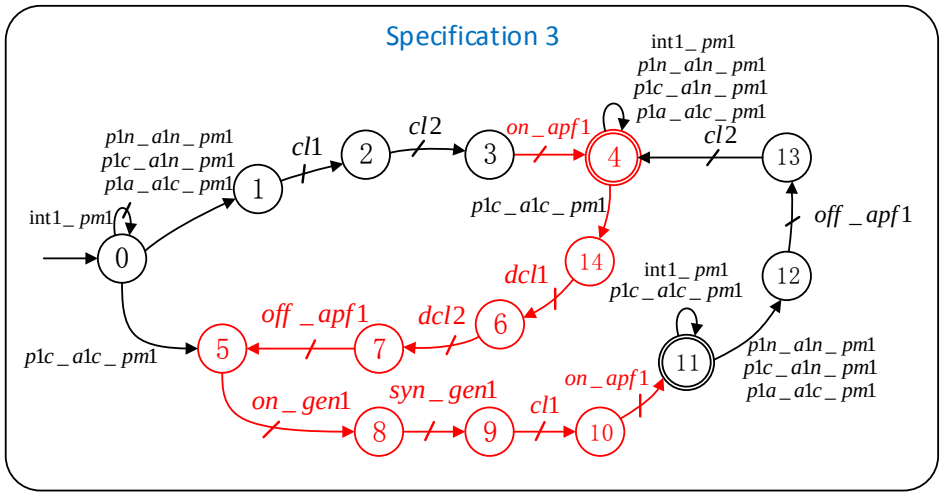


Figure B.4 Consistency verification for E1.3.2

The consistency verification for E1.3.2 is present in Figure B.4. The E1.3.3 can be identified as

follows:

$$S_3^1 = on\_apf1 \circlearrowleft (p1c\_a1n\_pm1 \vee p1n\_a1n\_pm1 \vee p1a\_a1c\_pm1) \rightarrow cl2$$

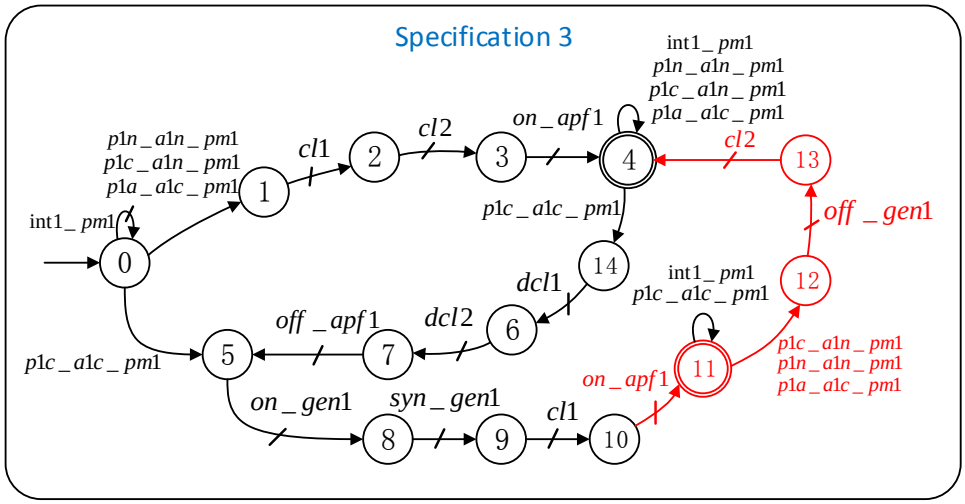


Figure B.5 Consistency verification for E1.3.3

The consistency verification for E1.3.3 is present in Figure B.5. To integrate all, the global formula S_3 is determined and can be verified and the traceability is confirmed.

B.2 Supervisors

The models of supervisor 2 and supervisor 3 after computation of supervisor synthesis, represented by SysML state machine, are shown in Figure B.6 and Figure B.7.

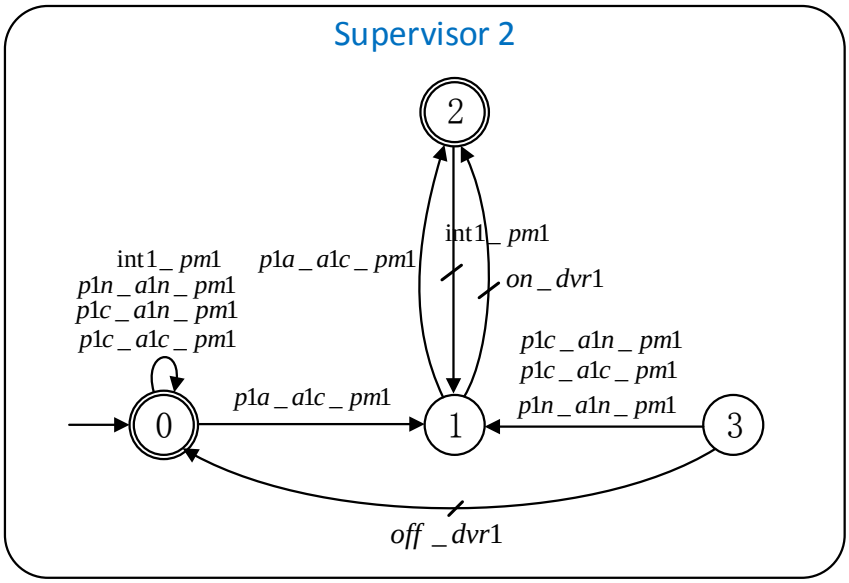


Figure B.6 Model of supervisor 2

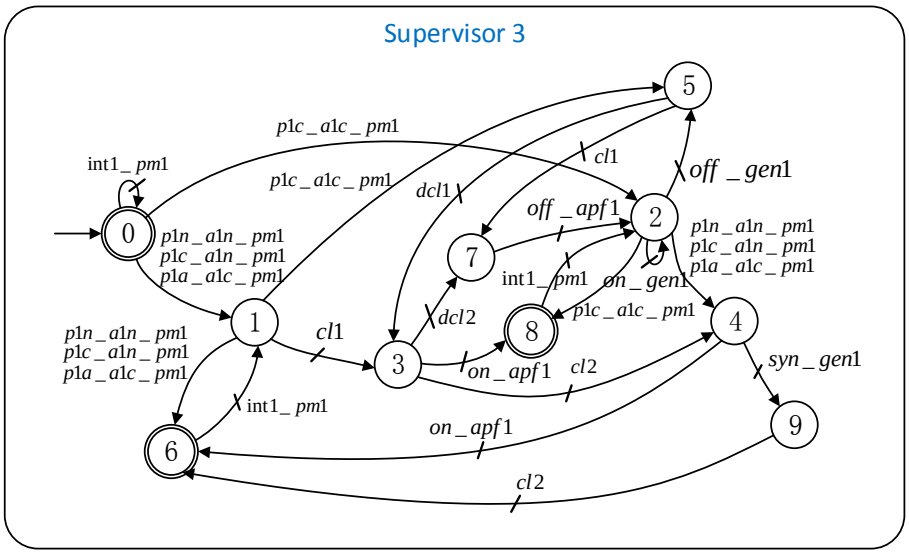


Figure B.7 STM: Model of supervisor 3

B.3 Control Program

The Supervisor implementation architecture for modular control, proposed by Queiroz and Cury (shown in Figure B.8), deals with most of the PLC implementation problems. The introduction of an interface between the supervisors and the physical system allows the designer to deal with the above-mentioned implementation aspects. This interface is composed of the levels named Product System (PS) and Operational Procedures (OP).

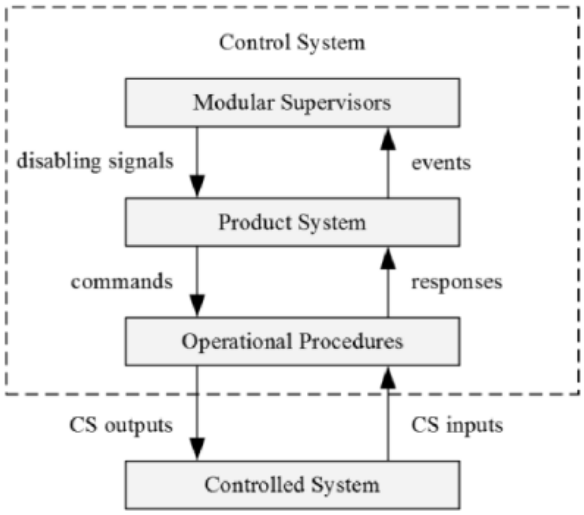


Figure B.8 Implementation approach of local modular supervisory control

(1) PS level is responsible for generating controllable events and signaling the occurrence of uncontrollable events within the alphabet of events of the corresponding subsystem. The

self-looping transitions don't need to be programmed. Treated events (i.e., generated or signaled) are sent to the MS level and commands are sent to the OP level.

(2) MS level performs the sequential call of all FBs, which updates the corresponding supervisor with the sequence of events already generated. The MS also establishes the state of the disabling signals, which is done based on the control action of each supervisor.

(3) OP level are designed considering sensors and actuators that constitute the corresponding piece of equipment, specifying the sequence of activities that must be performed upon the occurrence of controllable events and recognizing the occurrence of uncontrollable events.

PLC device perform activities sequentially along each PLC scan cycle in Figure B.9. The implementation method prioritizes the treatment of responses in detriment of the generation of controllable events (i.e., the occurrence of controllable events must be suspended while responses are to be treated).

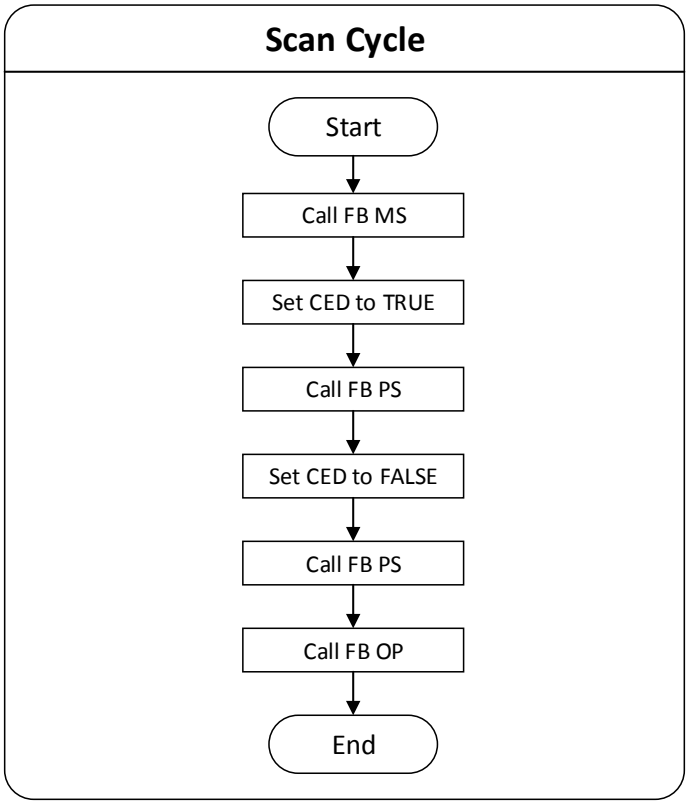
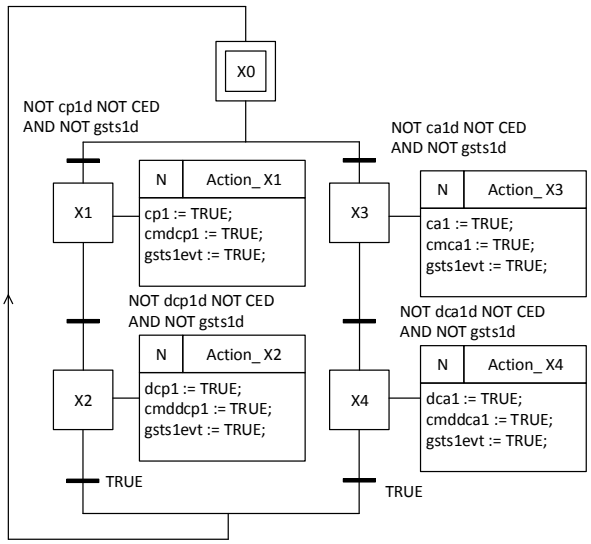
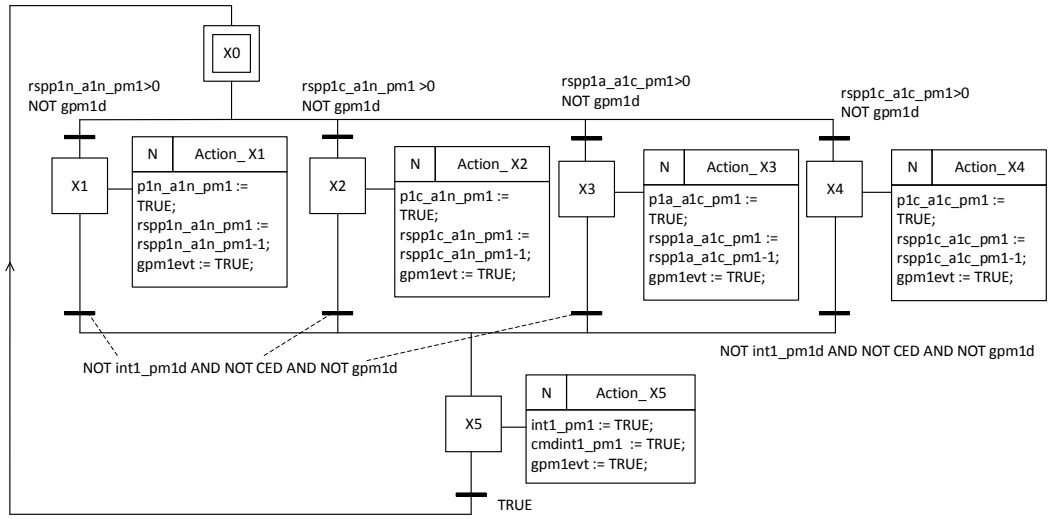


Figure B.9 Activities in one PLC scan cycle

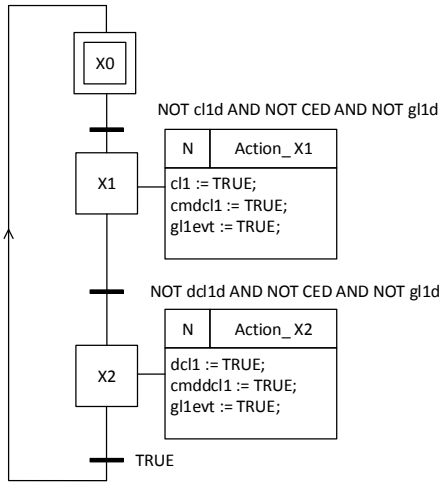
(1) PS level



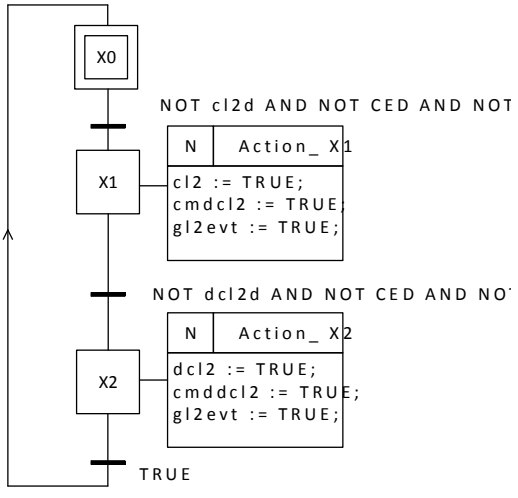
(a) Control Program: Part PS sts1



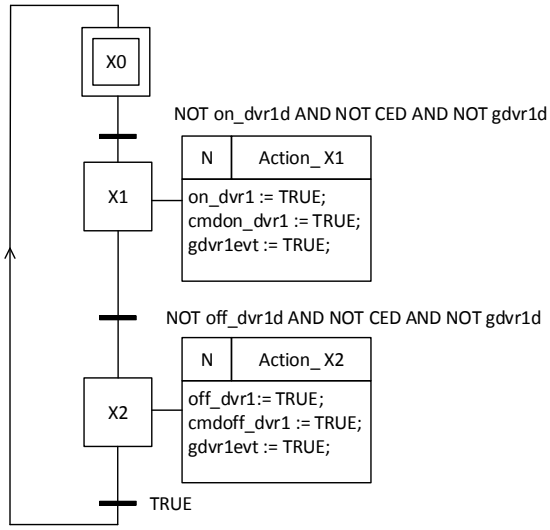
(b) Control Program: Part PS pm1



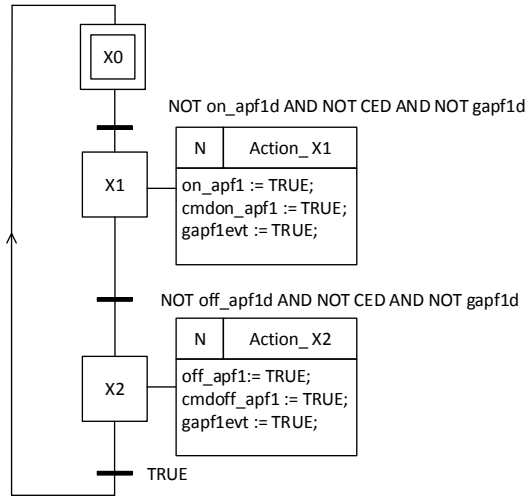
(c) Control Program: Part PS l1



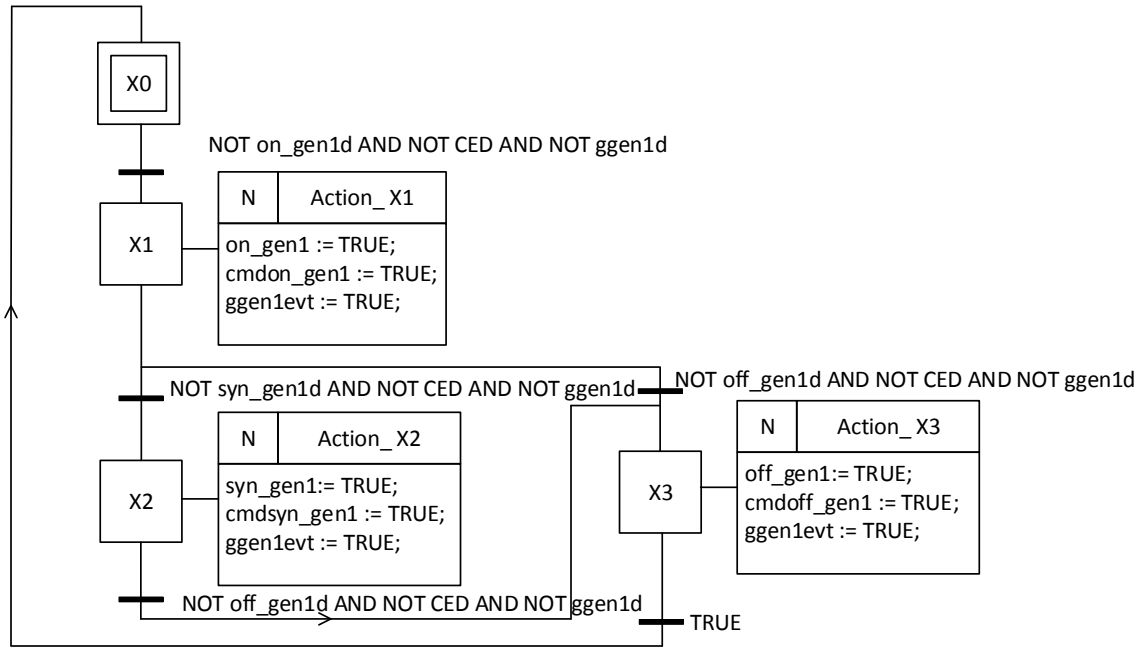
(d) Control Program: Part PS l2



(e) Control Program: Part PS dvr1



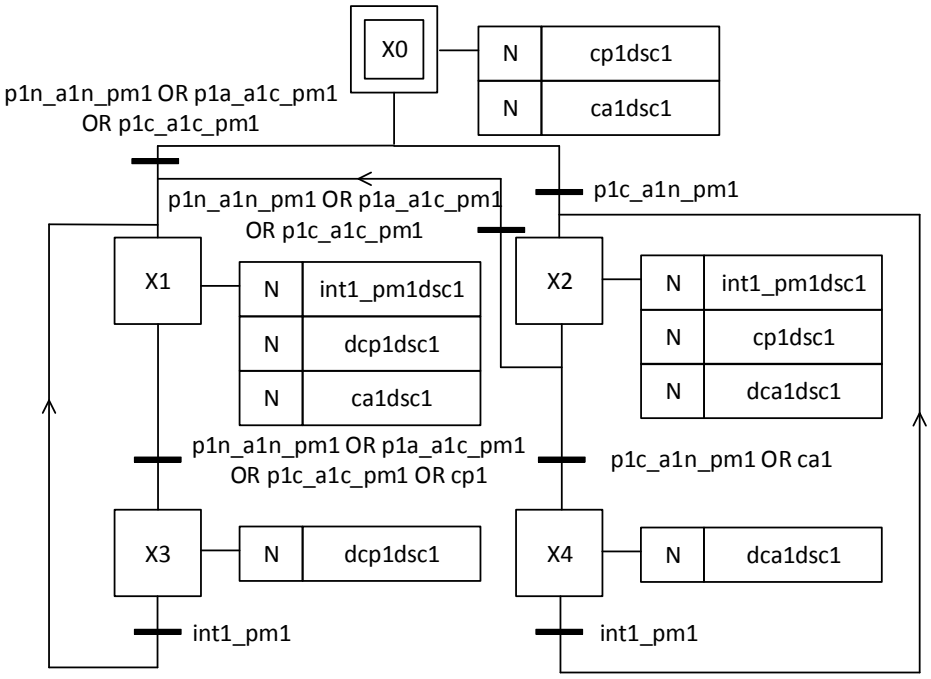
(f) Control Program: Part PS apf1



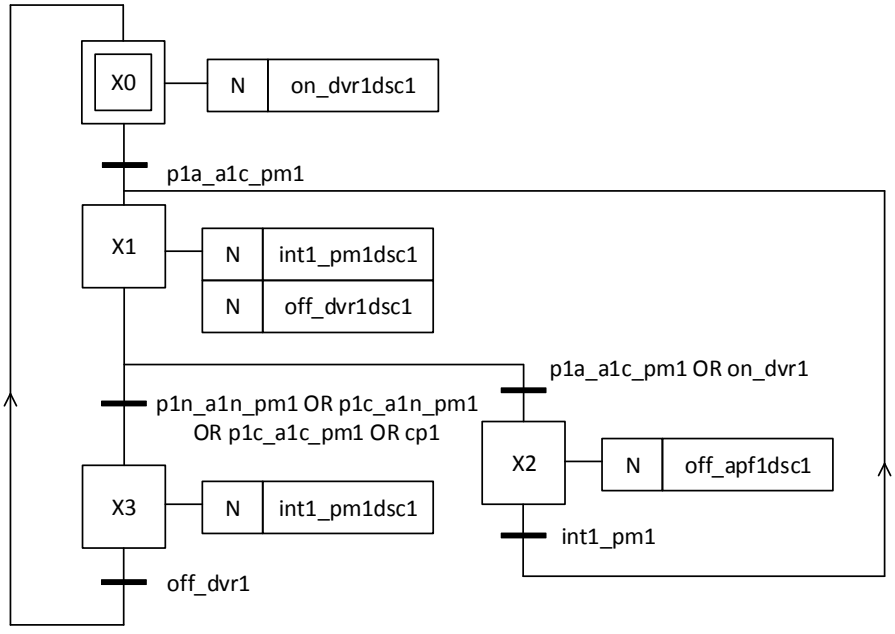
(g) Control Program: Part PS gen1

Figure B.10 Control Program: PS level

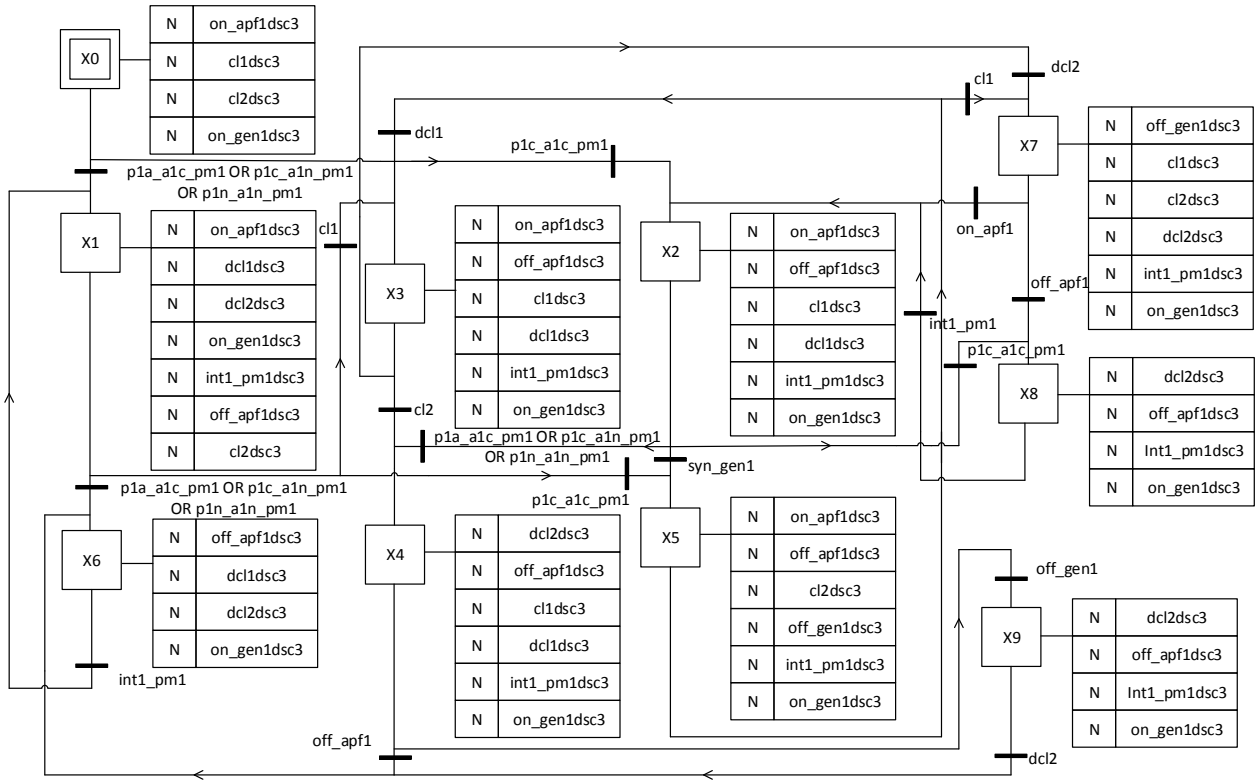
(2) MS level



(a) Control Program: Part MS Sup1



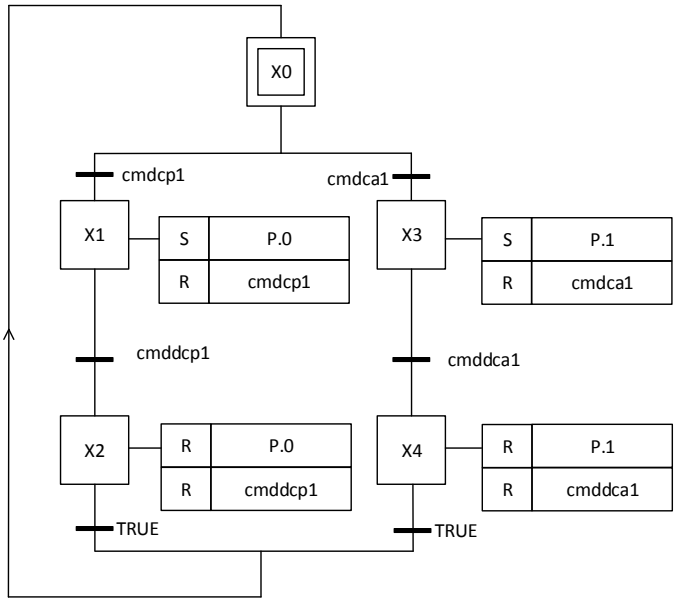
(b) Control Program: Part MS Sup2



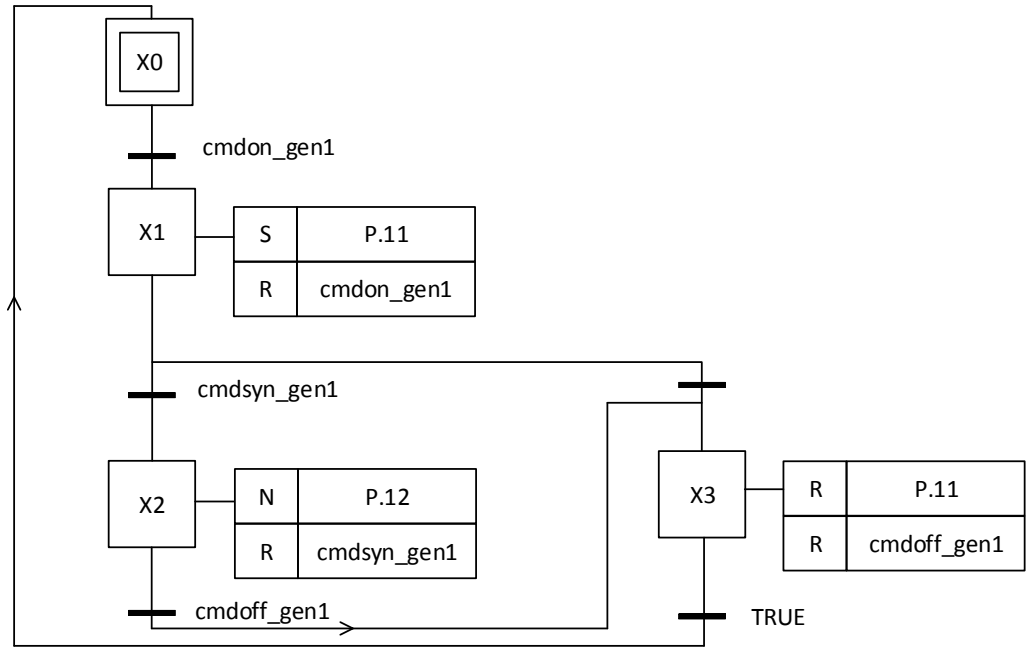
(c) Control Program: Part MS Sup3

Figure B.11 Control Program: MS level

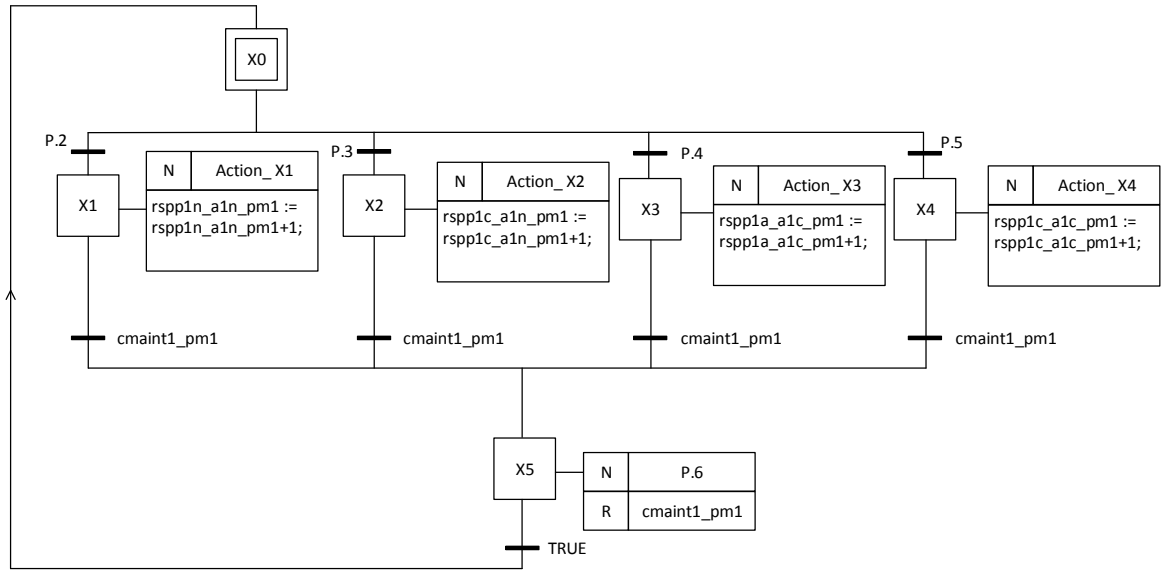
(3) OP level



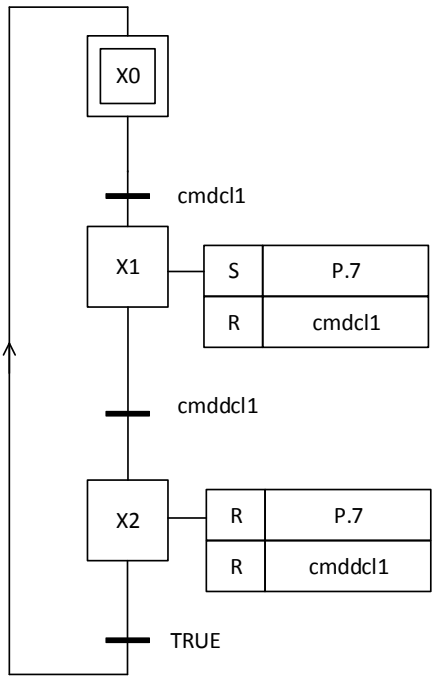
(a) Control Program: Part OP sts1



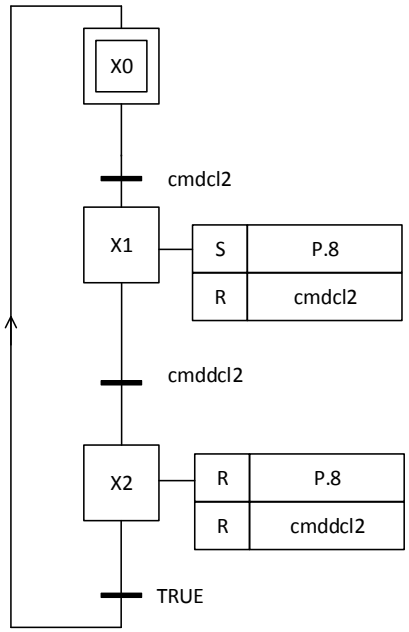
(b) Control Program: Part OP gen1



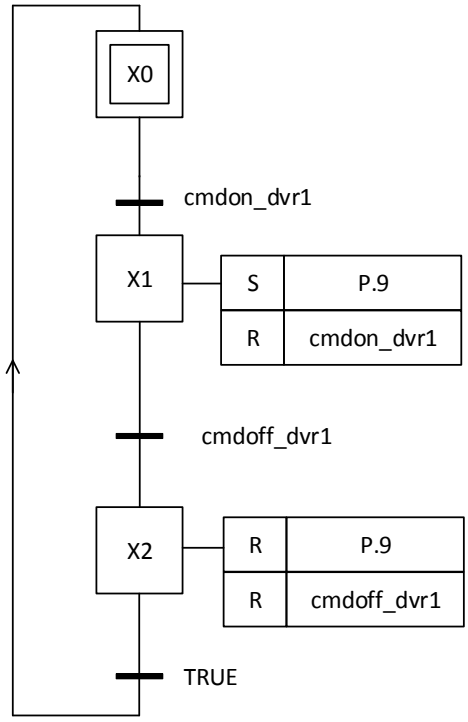
(c) Control Program: Part OP pm1



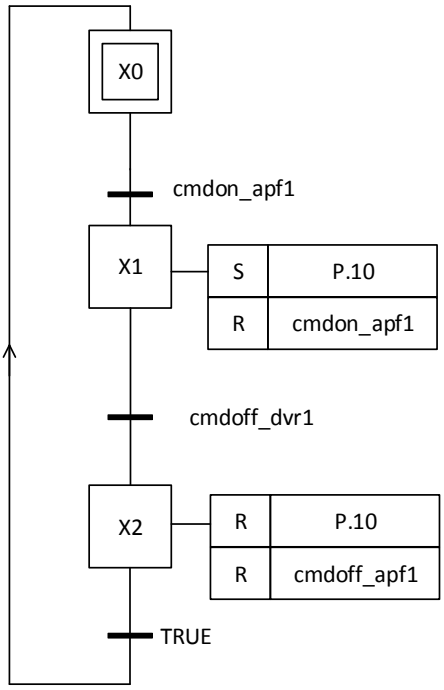
(d) Control Program: Part OP l1



(e) Control Program: Part OP l2



(f) Control Program: Part OP dvr1



(g) Control Program: Part OP apf2

Figure B.12 Control Program: OP level

B.4 Discussion on Supplementary Situations

In this section, we will propose four supplementary situations which are not discussed in the main body of case study. In the first situation study, we will present the example of structured component to explain why STS is identified by one unique plant. Secondly, we will present the template-based approach by which the plant models can be built automatically based on the BDD: System Structure. Thirdly, we present models for the control architecture by local modular control. In the fourth situation, we will introduce implementation models when the controller is distributed instead of monolithic.

B.4.1 Structured Component

In the case study, we supposed the situation where the some of the components are structured. An example is shown in Figure B.13, in which it is supposed that the STS becomes no more a standardized part and the two feeders (attention to the difference with standardized STS in the case study) are identified as two actuators. In the BDD, we also define the control relationship and the STS is under control as a whole rather than each feeder respectively under control. In consequence, this situation requires the behavior models of the entire STS as a plant.

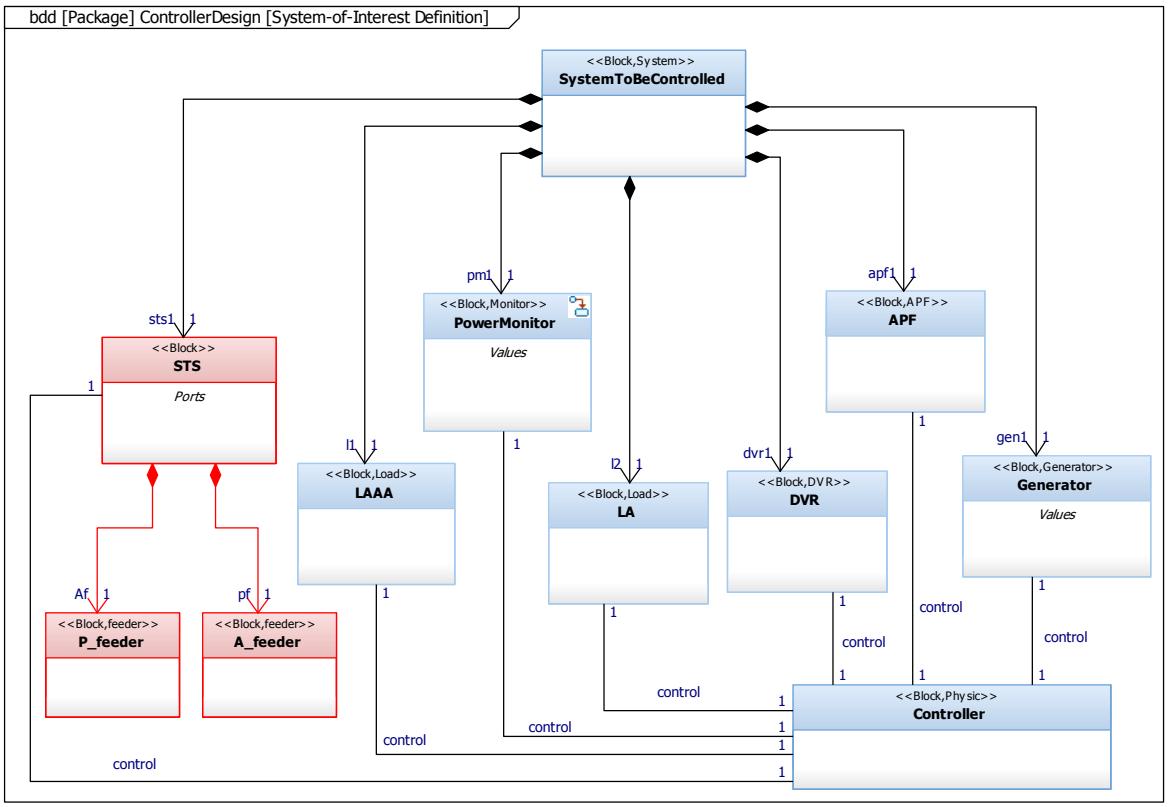
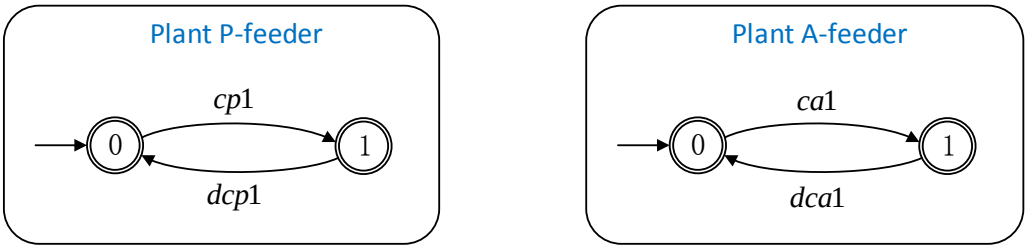
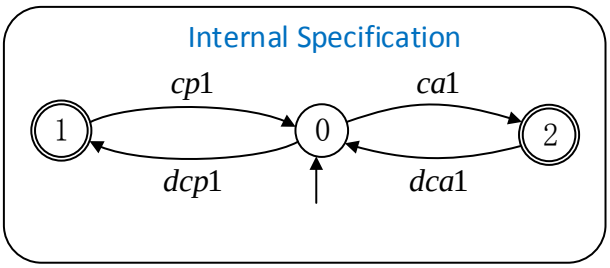


Figure B.13 BDD: System Sturecture (2)

For each actuator, an independent plant is related to (shown in Figure B.14 (a) (b)), which present the connection and disconnection switch behaviors of each actuator. However, the STS must connect one feeder at a time and one relationship (shown in Figure B.14 (c)) exists in order to coordinate the switch between feeders in STS. Therefore, P-feeder and A-feeder are not orthogonal sub-system. The global plant is the synchronization of Plant P-feeder, Plant A-feeder and relationship. The result is the same as the plant of STS in case study.



(a) Plant P-feeder (b) Plant A-feeder



(a) Relationship R

Figure B.14 Plant of STS

B.4.2 Modeling Process of Section A'

In the Section A', the template-based approach is applied instead of plant information identification and formalization. The input of this section is the BDD: System Structure. In this diagram, all the block which represent the templates of the plant should be defined their parameters (Figure B.15). The templates of relevant models in system-of-interest of CPP are listed in Table B.1. We assign six templates mapped to the corresponding blocks. Hereinto, block load LAAA and block load LA share the same template since they conform to the typical behavior of template load. Among the blocks, only block Power Monitor should be parameterized which indicate the sts1 to be monitored. Besides, the part name of the relationship composition is another parameter for each plant candidate. When the parts are transformed to the corresponding plants, the elements of the models are determined by the parameters.

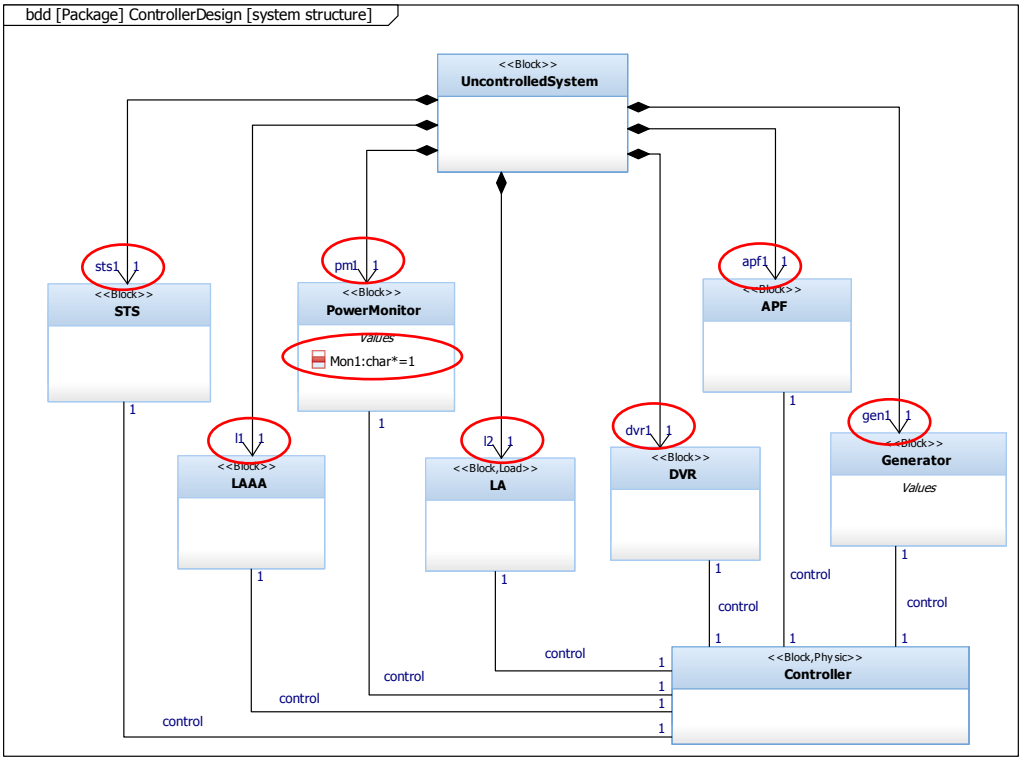


Figure B.15 BDD: System Structure (2)

Table B.1 List of templates

Parameterized Part	**Plant**
Block: STS argument: Component name	Automaton = *plant sts_i*, $i = 1, 2, ..., n$: $X = \{s0, s1, s2\}$; $\Sigma = \{\text{cpi, cai, dcpi, dcai}\}$; $\delta = \{(s0, cpi, s1), (s1, dcpi, s0) \mid i = 1, 2\}$; $x_0 = \{s0\}$, $X_m = \{s1, s2\}$.
Block: Load A and Load AAA argument: Component name	Automaton = *plant l_i*, $i = 1, 2, ..., n$: $X = \{s0, s1\}$; $\Sigma = \{\text{cli, dcli}\}$; $\delta = \{(s0, cpi, s1), (s1, dcpi, s0)\}$, $i = 1, 2, ..., n$; $x_0 = \{s0\}$, $X_m = \{s0, s1\}$.
Block: DVR argument: Component name	Automaton = *plant dvr_i*, $i = 1, 2, ..., n$: $X = \{s0, s1\}$; $\Sigma = \{on\_dvri, off\_dvri\}$; $\delta = \{(s0, on\_dvri, s1), (s1, off\_dvri, s0)\}$; $x_0 = \{s0\}$, $X_m = \{s0, s1\}$.

Block: APF argument: Component name	Automaton = *plant apf_i*, $i = 1, 2, ..., n$: $X = \{s0, s1\};$ $\Sigma = \{on\_apfi, off\_apfi\};$ $\delta = \{(s0, on\_apfi, s1), (s1, off\_apfi, s0)\};$ $x_0 = \{s0\}$, $X_m = \{s0, s1\}$.
Block: Genetor argument: Component name	Automaton Type = *plant gen_i*, $i = 1, 2, ..., n$: $X = \{s0, s1, s2\};$ $\Sigma = \{on\_geni, off\_geni, syn\_geni\};$ $\delta = \{(s0, on\_geni, s1), (s1, syn\_geni, s2), (s1, off\_geni, s0), (s2, off\_geni, s0), \};$ $x_0 = \{s0\}$, $X_m = \{s0, s2\}$.
Block Monitor argument: Component name Detect: sts_k	Automaton Type = *plant pm_i*, $i = 1, 2, ..., n$: $X = \{s0, s1\};$ $\Sigma = \{pkn\_akn\_pmi, pkc\_akn\_pmi, pka\_akc\_pmi, pkc\_akc\_pmi\} \cup \{intk\_pmi\};$ $\delta = \{(s0, intk\_pmi, s1), (s1, pkn\_akn\_pmi, s0), (s1, pkc\_akn\_pmi, s0),$ $(s1, pka\_akc\_pmi, s0), (s1, pkc\_akc\_pmi, s0)\};$ $x_0 = \{s0\}$, $X_m = \{s0\}$.

In order to the implementation of the model transformation, an experimental transformation program is developed whose framework is shown in Figure B.16. In this study, the SysML models are built by IBM Rhapsody® and it supports to export the project as XMI. The output of the program is the XML file which can be supported by Supremica. Three points are taken into account for the reasons of transformation to Supremica file:

(1) Rhapsody® integrates all elements and models in the project in the XMI file other than a particular diagram, it is necessary to modify the original XMI file;

(2) In the following steps, the supervisors should be synthesized. In this study, the computation of supervisor is achieved by Supremica. Therefore, the direct transformation to Supremica XML file can be a solution;

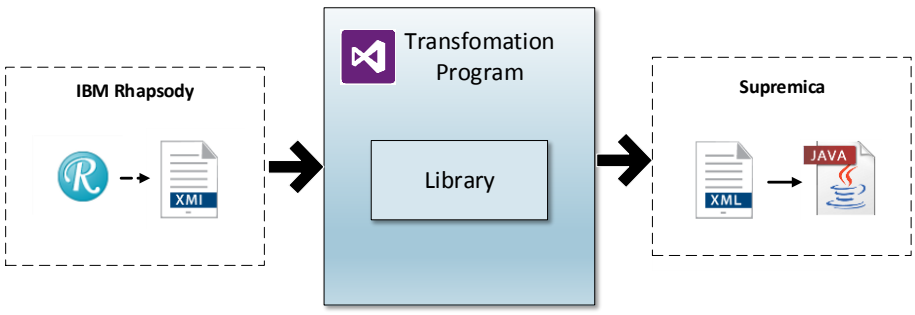


Figure B.16 Implementation framework of transformation

The transformation program is composed of two modules: the pre-process module, which transforms the input to a dada structure conforming to the templates, and the transformation modules, which process the data based on the template and exports the XML file.

The key points of the transformation are presented as follows:

(1) The input XMI file needs to be pre-processed before transformation. As IBM Rhapsody® can only export the whole project as XMI files in which all elements and models defined are contained, we keep the data related to BDD and delete all the unnecessary data.

(2) We can record eight blocks from the BDD. The part data is dynamically stored in the list, in which four kinds of information of each part are recorded: block name, part ID, a static variable and an array of additional parameters. We assign the additional parameter array to store the information necessary for constructing the automata from component models. For example, the monitor is parameterized by the arguments which clarify to target STS to be monitored. The monitor pm1 has only one target STS sts1 and therefore the parameter is recorded in the array.

(3) The library which contains the template particular for the CPP components is constructed. It should be pointed out that we can substitute other template library for that in this case study to realize the universality of the transformation program. The transformation process read the parts successively from part data structure and match the corresponding models in the library according to the stereotype.

Figure B.17 shows a part of the XMI of BDD describing the system structure exported by Rhapsody®.

```
<packagedElement xmi:type="uml:Stereotype" xmi:id="GUID+55c9cbe3-773e-4437-afce-0bf0d6258036" name="Monitor">
  <ownedAttribute xmi:type="uml:Property" xmi:id="_DeRqjE0BEem_9bB20X3KeA" name="base_Class" visibility="public" association="_DeRqp00BEem_9bB20X3KeA">
    <type xmi:type="uml:Class" href="http://www.omg.org/spec/UML/20090901/UML.xmi#Class"/>
  </ownedAttribute>
  <ownedAttribute xmi:type="uml:Property" xmi:id="_DeRqjU0BEem_9bB20X3KeA" name="base_Interface" visibility="public" association="_DeRqqU0BEem_9bB20X3KeA">
    <type xmi:type="uml:Class" href="http://www.omg.org/spec/UML/20090901/UML.xmi#Interface"/>
  </ownedAttribute>
</packagedElement>
<packagedElement xmi:type="uml:Stereotype" xmi:id="GUID+36c841f7-be4d-4816-ae16-23196163255a" name="DVR">
  <ownedAttribute xmi:type="uml:Property" xmi:id="_DeRqj00BEem_9bB20X3KeA" name="base_Class" visibility="public" association="_DeRqq00BEem_9bB20X3KeA">
    <type xmi:type="uml:Class" href="http://www.omg.org/spec/UML/20090901/UML.xmi#Class"/>
  </ownedAttribute>
  <ownedAttribute xmi:type="uml:Property" xmi:id="_DeRqkE0BEem_9bB20X3KeA" name="base_Interface" visibility="public" association="_DeRqrU0BEem_9bB20X3KeA">
    <type xmi:type="uml:Class" href="http://www.omg.org/spec/UML/20090901/UML.xmi#Interface"/>
  </ownedAttribute>
</packagedElement>
```

Figure B.17 Partial of the XMI of BDD

Figure B.18 presents a part of the XML of plant models to be read by Supremica. The XML is transformed automatically by the experimental program.

```
<Automaton name="Gdvr1" type="Plant">
  <Events>
    <Event id="0" label="on_dvr1" controllable="true" />
    <Event id="1" label="off_dvr1" controllable="true" />
  </Events>
  <States>
    <State id="0" name="s0" accepting="true" initial="true" />
    <State id="1" name="s1" accepting="true" />
  </States>
  <Transitions>
    <Transition source="0" dest="1" event="0" />
    <Transition source="1" dest="0" event="1" />
  </Transitions>
</Automaton>
```

Figure B.18 Partial of the XML of formal models

B.4.3 Local Modular Control Architecture

Suppose that the control architecture is local modular. As there are three specifications, three local plants are synchronized by corresponding plants respectively. Therefore, the supervisor synthesis result can be presented in Table B.2.

Table B.2 Supervisor synthesis (2)

Supervisor	**Synthesis**	**Plant**
Supervisor 1	$Sup_1 \leftarrow G_{loc1} \times H_1$	$G_{loc1} \leftarrow G_{sts1} \times G_{pm1}$
Supervisor 2	$Sup_1 \leftarrow G_{loc2} \times H_2$	$G_{loc1} \leftarrow G_{dvr1} \times G_{pm1}$
Supervisor 3	$Sup_1 \leftarrow G_{loc2} \times H_3$	$G_{loc1} \leftarrow G_{apf1} \times G_{pm1} \times G_{gen1} \times G_{cl1} \times G_{cl2}$

The modeling result can be seen in Figure B.19 and Figure B.20.

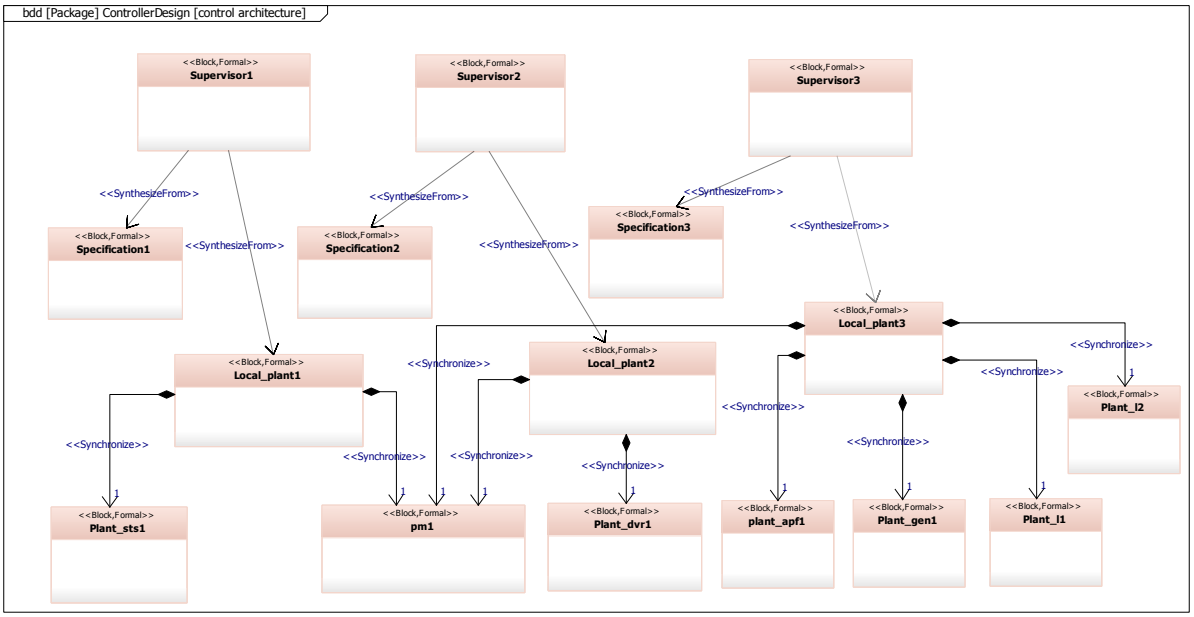


Figure B.19 BDD: Control Architecture (2)

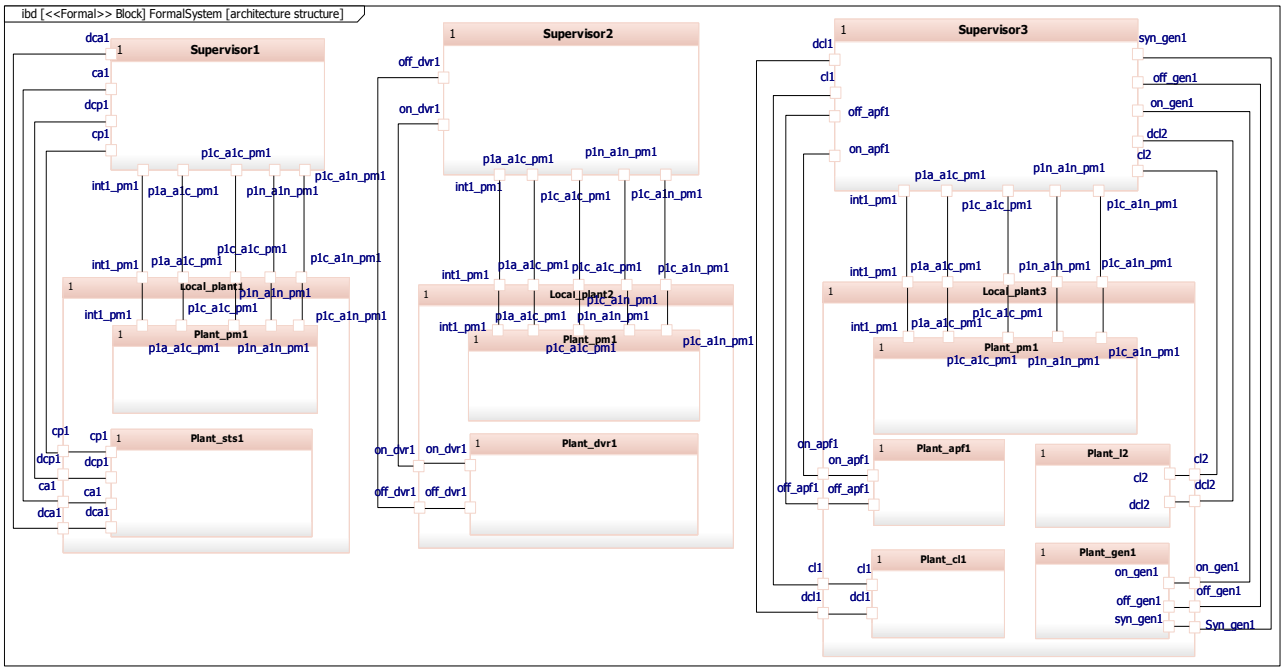


Figure B.20 IBD: Architecture Structure (2)

B.4.4 Distributed Concrete Controller

In the Situation 2, the supervisor is supposed to be implemented by distributed controller. Each local supervisor is realized by a local PLC device and a control program is allocated to each device. Based on the mapping from events in formal models, the ports for each controller can be defined as Table B.3.

Table B.3 Port definition for controller

Hardware	Software	Event	Port
plc1	pg1	p1n_a1n_pm1, p1c_a1n_pm1, p1a_a1c_pm1, p1c_a1c_pm1, int1_pm1, cp1, dcp1, ca1, dca1	P.0, P.1, P.1, P.2, P.3, P.4, P.5, P.6
plc2	pg2	p1n_a1n_pm1, p1c_a1n_pm1, p1a_a1c_pm1, p1c_a1c_pm1, int1_pm1, on_dvr1, off_dvr1	P.2, P.3, P.4, P.5, P.6, P.9
plc3	pg3	p1n_a1n_pm1, p1c_a1n_pm1, p1a_a1c_pm1, p1c_a1c_pm1, int1_pm1, cl1, dcl1, cl2, dcl2, on_apf1, off_apf1, on_gen1, off_gen1, syn_gen1	P.2, P.3, P.4, P.5, P.6, P.7, P.8, P.10, P.11, P.12

The control implementation model is presented in Figure B.21. Three PLC devices are aggregated as the global controller and three control programs are allocated to PLC devices respectively. The ports for each concrete controller defined in Table B.3 are also presented in the diagram.

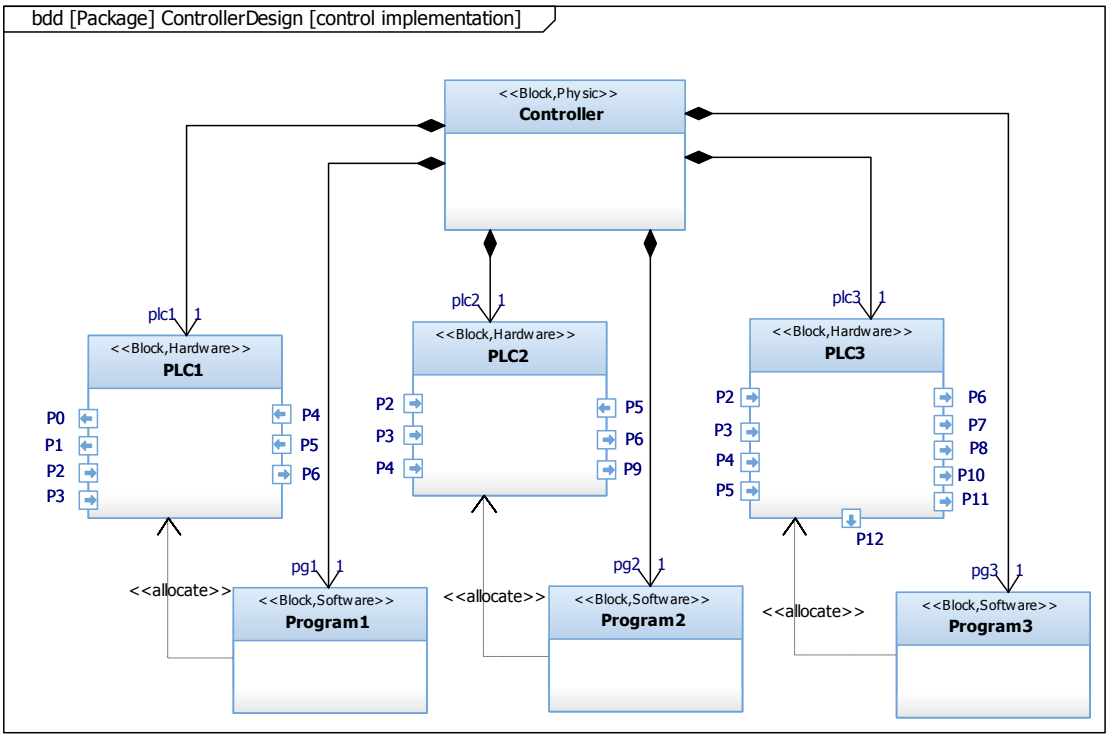


Figure B.21 GO2 BDD: Control Implementation (2)

The connection between local controller and components are presented in the IBD shown in Figure B.22. Compared with monolithic controller, the connections are more complex. Two relationships can be presented in the diagram: the control relationship between local controller and target component and connections between ports. In the IBD, the connections of local controllers 1, 2 and 3 are distinguished by red, green and blue respectively.

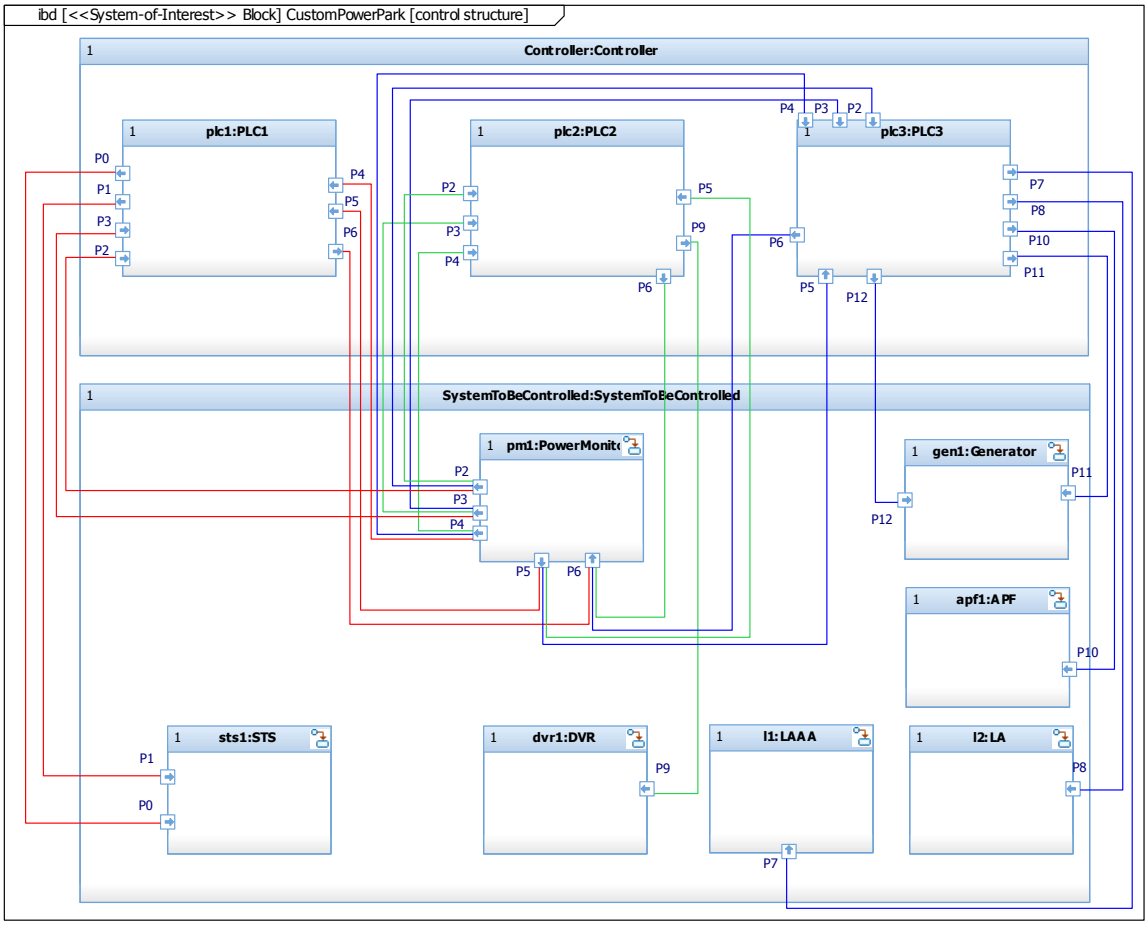


Figure B.22 GO3 IBD: Control Connection (2)

Appendix C

Résumé

C.1 Introduction à la Théorie du Contrôle par Supervision

C.1.1 Contexte

La complexité croissante des contrôleurs, la nécessité de réduire le temps de développement et la réutilisation possible des modules logiciels existants nécessitent une approche formelle dans la pratique de l'ingénierie.

La Théorie du Contrôle par Supervision (TCS) est l'un des paradigmes formels les plus importants pour le développement de contrôleurs dans le cadre de l'étude des systèmes à événements discrets (SED). Le grand nombre de contributions scientifiques montre que la TCS suscite un intérêt académique considérable et il a été prouvé que cette théorie était applicable dans divers domaines industriels tels que les systèmes manufacturiers, les systèmes embarqués ou les systèmes de transport [17-24], [38-43]. Avec cette théorie, les exigences qui sont traditionnellement vérifiées postérieurement à la conception du contrôleur sont utilisées comme données d'entrée à la génération de ce contrôleur, qui est alors juste par construction. La conséquence immédiate est que l'étape de vérification n'a plus lieu d'être, ce qui élimine en partie les cycles de reconception et de vérification nécessaires à la mise au point du système sous contrôle.

C.1.2 Concepts Théoriques

Le paradigme de modélisation de SED définit le comportement comme un ensemble de séquences d'événements. Le système est considéré comme asynchrone, ce qui signifie qu'un seul événement peut se produire à chaque instant. Pour modéliser le comportement d'un SED, le cadre TCS d'origine tire parti de deux formalismes : le langage formel et l'automate à états finis.

Le langage formel spécifie toutes les séquences d'événements admissibles que le SED est capable de "traiter" ou de "générer", tout en évitant le recours à une structure supplémentaire. En fait, les langages normaux sont rarement utilisés directement pour présenter le système et conviennent mieux pour prouver propriétés de SED. La raison est que les langages normaux ne sont pas une manière intuitive de modéliser et de présenter un comportement répétitif du système. Au lieu de cela, l'artefact de modélisation le plus habituel est l'automate à états finis [35].

Définition C.1 (Langage formel) : chaque DES est associé à un ensemble d'événements Σ sous-jacents appelé *alphabet*. Une séquence d'événements de l'alphabet est appelée «mot» ou «chaîne». On note la cardinalité d'un ensemble d'évènement Σ par $|\Sigma|$. Pour un ensemble d'événements particulier Σ, nous désignons son ensemble de toutes les chaînes finies possibles d'événements par Σ^*. Un *langage formel* défini sur un ensemble d'événements Σ est un sous-ensemble de Σ^*.

Définition C.2 (Automate à états): Un automate à états est défini comme un 5-tuple :

$$G = (Q,\ \Sigma,\ \delta,\ q_0,\ Q_m)$$

Ou,

Q est un ensemble fini d'états ;

Σ est un ensemble fini d'événements associés à G ;

$\delta : Q \times \Sigma \to Q$ est la fonction de transition ;

q_0 est l'état initial ;

$Q_m \subseteq Q$ est un ensemble d'états marqués ou acceptants.

Le langage formel et l'automate à états ne sont pas deux modèles non corrélés. En fait, chaque langage normal défini sur un ensemble fini d'événements peut être généré par un automate à états finis déterministe. Il existe donc une équivalence entre ces deux modèles [45].

En TCS, le modèle de comportement du système incontrôlé est appelé *procédé*, ce qui représente le comportement libre du système. Un système complexe peut avoir plusieurs modèles des procédés modulaires basés sur les composants ou les sous-systèmes qui le composent. TCS fournit une méthode de composition synchrone pour composer procédés multiples en vue de constituer un procédé monolithique globale. Le choix des procédés modulaires ou monolithiques dépend de la complexité du système et de l'architecture de contrôle dans chaque cas particulier.

Afin de limiter le libre comportement du système et de faire en sorte que ses comportements respectent les attentes, il est nécessaire de définir des spécifications. En TCS, la spécification est également une modélisation par automate.

Le paradigme de la théorie du contrôle des superviseurs considère tout système comme la

composition d'un procédé à contrôler et comme un superviseur assurant la logique de contrôle du respect des spécifications. Nous présentons généralement la synthèse du superviseur démarrée par le système consistant en un procédé monolithique et un superviseur monolithique. La boucle de réaction de contrôle entre le procédé et le superviseur est illustrée à la Figure C.1.

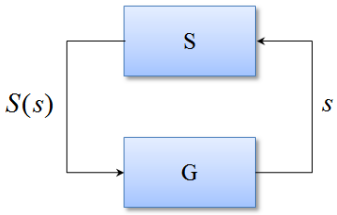


Figure C.1 La boucle de réaction du contrôle de supervision

Il y a deux propriétés du superviseur qui doivent être vérifiées lors de la synthèse : la contrôlabilité et le non blocage. L'algorithme de synthèse du superviseur est le noyau de la TCS, sur la base duquel le contrôle de supervision résultant des procédés par rapport aux spécifications données est prouvé être contrôlable et vivant. TCS fournit un algorithme itératif pour la synthèse du superviseur et la vérification des propriétés.

Généralement, si le système n'est pas complexe, l'approche de contrôle centralisé / monolithique suffit pour synthétiser un superviseur unique sur un procédé. Malheureusement, pour modéliser un système à grande échelle, l'approche centralisée ne peut pas résoudre le problème de l'explosion de l'espace d'états. Le nombre total d'états d'un modèle de procédé augmente rapidement lorsque le nombre de composants locaux augmente, en raison du produit synchrone engendrant un produit cartésien [11].

Un certain nombre d'efforts ont été consacrés aux méthodes permettant de surmonter cette difficulté. Une contribution propose le concept de modularité [8], qui est étendu au concept de modularité locale [9]. Dans cette architecture de contrôle, où le superviseur global est modulaire local, un non blocage global du contrôle de supervision doit être vérifié. Dans [46] [47], les auteurs ont introduit une méthode hiérarchique qui décompose un système en deux sous-systèmes et limitation l'interaction des sous-systèmes au moyen d'une interface. Ainsi, le modèle de système complet n'a jamais besoin d'être construit, offrant des économies potentiellement considérables en termes de calcul.

C.1.3 Mise en Œuvre

Les études de cas présentent des types de contrôleurs logiques concrets pouvant être utilisés pour réaliser un contrôle de supervision. L'Automate Programmable Industriel (API) étant l'un des principaux exécutants de l'automatisation industrielle, de nombreuses études ont été impliquées

dans la mise en œuvre du TCS par le programme de contrôle du API, qui sont écrites dans des langages normalisés, tels que le schéma à contacts, la traduction textuelle IL) ou SFC (tableaux de fonctions séquentielles) (IEC 61131-3). Une autre étude de cas [39] présente le contrôle de supervision mis en œuvre par le microcontrôleur des familles Microchip PIC18F et dsPIC30F. Dans [55], une application de contrôle de supervision a mis en œuvre l'ATS (Automatic Train Supervision), qui vise à aider la gestion des lignes de métro de la RATP au centre de contrôle de lignes pour réaliser le contrôle du trafic et la sécurité

En raison de certaines caractéristiques communes telles que le cycle de balayage d'entrée, presque tous les types de contrôleurs rencontreront des difficultés, qui sont systématiquement introduites dans [56]. L'étude a conclu à cinq préoccupations principales lors du passage du modèle d'automate fini asynchrone du superviseur à la nature synchrone, sans toutefois s'y limiter, du système de contrôle API. Un aperçu intéressant des phénomènes connexes est donné dans [57].

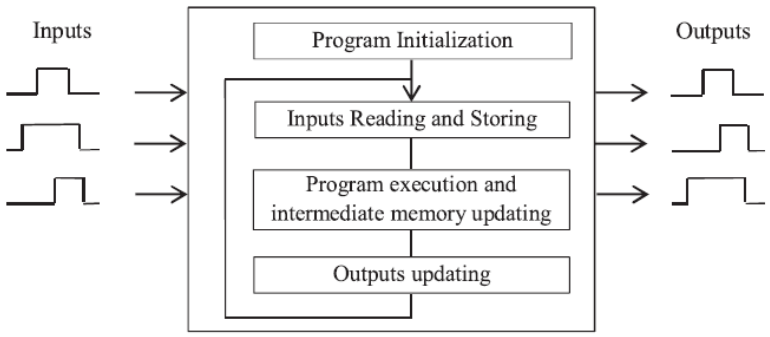


Figure C.2 Principe du contrôleur logique

C.2 Problématique

[25] introduit la démarche de développement typique basé sur TCS en passant en revue l'état de l'art des méthodes de synthèse et de mise en œuvre. Le processus global, illustré à la Figure C.3, comprend trois activités génériques: la formalisation, la synthèse (synthèse du superviseur et synthèse du contrôle) et la mise en œuvre comme suit :

(1) Formalisation

Deux types de modèles formels doivent être construits: le modèle de spécification et le modèle de procédé. Les modèles de spécification formalisent les contraintes de comportement que le contrôleur ou le processus contrôlé doivent respecter conformément aux exigences. Formaliser les propriétés requises peut également être une tâche difficile, générant de nombreux essais et erreurs, et nécessitant une expertise spécifique liée à la fois à la centrale (pour exprimer les contraintes liées aux opérations et aux choix technologiques de l'actionneur et du capteur) et au modèle comportemental requis pour la suite procédure de synthèse.

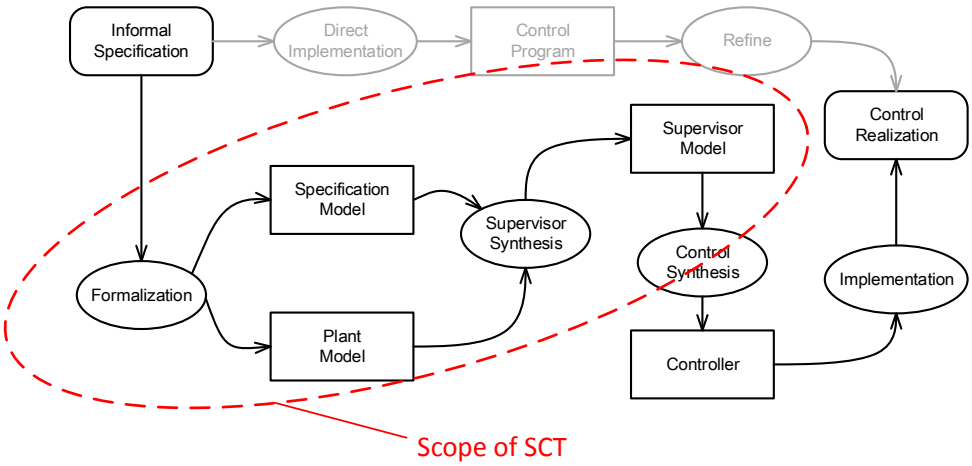


Figure C.3 Activités et modèles concernés dans la démarche de développement pour SED

(2) Synthèse

En phase de synthèse, deux sous-activités sont nécessaires: synthèse du superviseur et synthèse du contrôle. La différence fondamentale entre un superviseur et un contrôleur est qu'un superviseur interdit la survenue de certains événements contrôlables en vue de maintenir le comportement de la centrale dans des états et des séquences légaux, tandis qu'un contrôleur agit en forçant les entrées de la centrale de manière à atteindre l'objectif souhaité [25]. Dans le paradigme TCS, l'approche de synthèse consiste à synthétiser un superviseur à partir d'une description de le procédé et de la spécification. Un tel superviseur vise à faire respecter les spécifications tout en offrant une flexibilité comportementale maximale. Le contrôleur peut être considéré comme intermédiaire entre le superviseur officiel et le contrôleur concret. Le contrôleur est construit en fonction de la stratégie de contrôle. Bien qu'il n'y ait pas d'étude systématique sur la stratégie de contrôle pour transformer le superviseur en contrôleur, un certain nombre d'études de cas ont proposé leurs solutions appropriées. Dans [53], les auteurs proposent une approche de choix de trajectoire afin de rendre le comportement de contrôle «déterministe». Le principe du choix de trajectoire est de s'assurer que les états dans l'automate sont tous accessibles et co-accessibles. Dans [54], les auteurs ont développé un algorithme pour la génération de contrôleurs valides qui renforçait la propriété de coreachabilité afin de garantir qu'un état marqué soit finalement atteint, quel que soit le comportement de la centrale. Dans [18], les auteurs ont proposé un planificateur dans l'architecture de navigation sous contrôle. Le planificateur, composé de plusieurs algorithmes (y compris l'algorithme de Dijkstra), aide le superviseur modulaire à sélectionner le meilleur choix de chemin d'accès.

(3) Mise en œuvre

Quel que soit le contexte académique ou industriel, le superviseur / contrôleur est généralement mis en œuvre par un contrôleur concret pour la validation, qui est la cible de cette étape. En

fonction des différents objectifs, la mise en œuvre peut être classée en deux types: un contrôleur logique concret et une simulation, qui seront discutés en détail dans la partie précédente.

Bien que la TCS soit construit sur une théorie systématique, le paradigme a ses limitations du point de vue du contexte d'ingénierie. Comme le montre la Figure C.3, la portée de la TCS est limitée à la synthèse de la formalisation et du superviseur. Même si la mise en œuvre du contrôleur est considérée comme l'extension de la TCS, lorsqu'un processus basé sur la TCS est appliqué dans la pratique de l'ingénierie, plusieurs problèmes doivent être résolus :

Limitation 1 : Formalisation

La qualité des résultats de la synthèse dépend fortement de la pertinence des exigences proposées. La formalisation des exigences informelles et des comportements libres du système à contrôler est l'une des difficultés majeures à laquelle sont confrontées les approches de synthèse de contrôle existantes [25].

Premièrement, Les exigences des systèmes sont généralement écrites sous forme textuelle, car cela signifie généralement une meilleure compréhension entre les différentes parties prenantes. D'autre part, les modèles formels à bases d'automate ont une sémantique sans ambiguïté, ce qui signifie qu'ils ne peuvent pas être compris de différentes manières. La plupart des études de cas aboutissent à la modélisation et à la mise en œuvre de leurs contrôleurs en formalisant directement l'exigence au tout début. Il est encore difficile de présenter les liens entre les exigences de formalisation et de narration informelle. En fait, le processus de modélisation basé sur la TCS accorde peu d'attention à la vérification de la cohérence et la traçabilité entre les exigences informelles et les spécifications formelles sous forme d'automates. En ingénierie, les exigences doivent être suivies et un certain nombre d'outils tels que la matrice de traçabilité des exigences (RTM) ou des modèles graphiques (diagramme d'exigences SysML) sont utilisés dans le processus de développement [63] [64]. La représentation de la traçabilité des exigences pour les modèles formels dans le paradigme TCS reste non résolue.

Deuxièmement, la formalisation des comportements possibles du système est également discutable. Il n'existe pas de méthode systématique pour construire ce modèle. Par exemple, le stock, en tant que composant physique d'un système de production, est généralement formalisé par une spécification plutôt que par un procédé. En revanche, dans [18], le comportement du robot mobile est formalisé par plusieurs procédés. On peut constater une contradiction en comparant ces deux situations et une question peut être posée : comment des procédés formels peuvent-elles être liées à un système physique à contrôler ? En effet, dans un certain nombre de contributions, les procédés sont supposés être formalisés en identifiant les signaux des capteurs et des commandes aux actionneurs, conformément aux lois de commande implémentées dans un programme utilisateur [20] [25] et [30]. Cependant, il n'y a pas d'étude pour ce point. Deuxièmement, à l'instar des

spécifications, la vérification de l'exactitude des procédés n'est toujours pas résolue, ce qui est également important pour le résultat de la synthèse par le superviseur.

En effet, la traçabilité des exigences est reconnue comme un sujet de préoccupation dans un nombre croissant de normes et de directives [65]. Dans la pratique de l'ingénierie, la relation de traçage explicite est d'une importance vitale pour garantir la conformité du produit à nos besoins. Comparé à l'approche basée sur la TCS, le problème de formalisation impliquant à la fois des procédés et des spécifications conduit à un manque de traçabilité entre les modèles formels et les cahiers des charges. La raison de ce problème est due à l'absence de modèles au sein de l'approche formelle pouvant présenter la traçabilité.

Limitation 2 : Modèle de structure

Comme le modèle formel ne peut représenter que le comportement du système à étudier, il est difficile de décrire l'aspect de la structure, qui inclut à la fois la physique et la logique. Plusieurs problèmes se posent à ce stade. Premièrement, les modèles de procédé qui représentent le comportement libre des composants physiques sont totalement incapables de contenir des informations sur la structure du système. Dans les études de cas susmentionnées, les systèmes sont simples et sans structure. Cependant, le comportement d'un composant physique peut être modélisé par un certain nombre de procédés (par exemple, dans [18], les comportements de robots sont formalisés par plusieurs procédés indépendants selon différents modules). Le fossé entre les procédés et les composants physiques pose le problème: la cohérence entre les procédés et les composants physiques n'est pas claire. Deuxièmement, le superviseur / contrôleur peut également être structuré. Il est toujours vrai que tous les types de superviseurs sont structurés, à l'exception du monolithique lorsque l'architecture de contrôle est modulaire ou distribuée. Malheureusement, aucun modèle ne peut être utilisé pour décrire explicitement les détails de la structure, ce qui conduit à une incompréhension ou une confusion pour les personnes qui se concentrent sur le AC dans la pratique du génie De manière similaire, c'est toujours le cas pour le contrôleur car le contrôleur peut également être structuré selon différentes stratégies de contrôle. Dans le contexte académique, il est acceptable que la structure puisse être illustrée simplement par des schémas; Cependant, cela est inacceptable dans la pratique d'ingénierie qui requiert généralement des formes de représentation standardisées (telles que SADT, SysML, BPMN, etc.). Les problèmes ci-dessus suscitent peu d'attention de la part des universitaires et peu de contributions peuvent être trouvées.

Limitation 3: Modèle de mise en œuvre

Dans les processus basés sur SCT, le contrôleur est implémenté sur la base de l'architecture de contrôle de supervision. Bien qu'un certain nombre de méthodes aient été proposées pour transformer superviseur / contrôleur en contrôleur concret, il existe toujours un écart entre les deux.

Les contributions existantes se concentrent sur la manière de transformer le superviseur en le programme de contrôle et le modèle de la mise en œuvre du contrôleur concret est absent. Par exemple, dans l'étude [37], les superviseurs modulaires locaux sont implantés par des programmes de contrôle d'API par la SFC (Sequential Function Chart). D'après la description de l'étude, les programmes de contrôle sont déployés dans un contrôleur monolithique concret. Cependant, le contrôle modulaire peut également être mis en œuvre par contrôleur concret distribué. En outre, le système contrôlé n'est pas claires car aucun modèle ne peut représenter les détails internes du système de contrôle physique dans le paradigme TCS. En fait, ces deux situations sont dues à l'absence de modèles de mise en œuvre. L'architecture de contrôle de supervision existante et les méthodes de mise en œuvre ne permettent pas de spécifier le lien entre les modèles de superviseur / contrôleur et les modèles de contrôleur concret, ce qui est inacceptable dans le contexte d'ingénierie.

Limitation 4 : Absence de processus d'ingénierie

Les paradigmes TCS fournissent les solutions formelles pour conception des contrôleurs, ou plutôt contrôle automatique (AC). Cependant, il n'existe pas de cadre global permettant de normaliser le processus de modélisation. En fait, pour chaque étape de modélisation de la Figure C.3, pour la formalisation, la synthèse ou la mise en œuvre, un grand nombre d'approches différentes ont été proposées. Par contre, il semble toujours que la plupart des contributions scientifiques soient classées en deux parties: la synthèse par le superviseur et la mise en œuvre du contrôleur. En revanche, les ingénieurs ont généralement tendance à tirer parti des processus de développement dans lesquels les ingénieurs suivent une série d'étapes méthodiques pour trouver une solution à un problème. Par conséquent, l'absence de processus de modélisation globale de la SCT pose des difficultés pour la pratique de l'ingénierie.

En conclusion, les pratiques d'ingénierie montrent que les processus de conception normalisés sont essentiels à l'amélioration de l'efficacité de la conception et de leur mise en œuvre, un grand nombre de normes techniques et de méthodes normalisées étant proposées dans le domaine de l'ingénierie [70]. En revanche, les propositions de processus de pointe basées sur TCS se situent toujours dans le cadre du processus de la Figure C.3 et montrent plus ou moins leurs inconvénients en ce qui concerne la pratique de l'ingénierie.

C.3 MBSE

L'initiative MBSE (Model Bades System Engineering) a été proposée dans le cadre de la Vision SE 2020 du Conseil International pour l'Ingénierie des Systèmes (INCOSE) à Albuquerque en janvier 2007 [71]. MBSE est une méthodologie d'ingénierie des systèmes axée sur la création et l'exploitation de modèles de domaine comme principal moyen d'échange d'informations entre

ingénieurs, plutôt que sur l'échange d'informations fondé sur des documents. INCOSE SE Vision 2020 a donné la définition de MBSE : l'ingénierie de systèmes basée sur un modèle est l'application formalisée de la modélisation pour prendre en charge les activités, la conception, l'analyse, le tragage, la vérification et la validation des systèmes; les phases. Malgré leur lancement, il y a seulement dix ans, les méthodes MBSE sont largement appliquées dans différentes applications d'ingénierie:

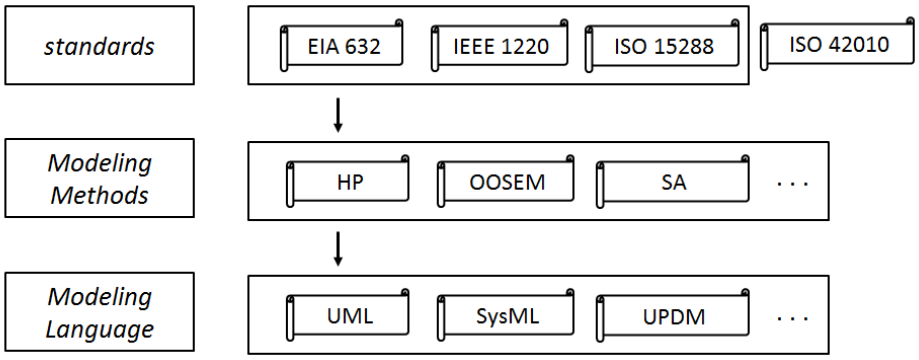


Figure C.4 Taxonomie partielle des normes d'ingénierie des systèmes [80]

(1) Standard

Au cours des dernières années, des efforts importants ont été consacrés au développement, au perfectionnement et, finalement, à l'acceptation des processus pour les systèmes d'ingénierie ou aux processus d'ingénierie des systèmes [81]. Les normes décrivant un processus SE incluent les normes EIA 632 [82], IEEE 1220 [83], ISO 15288 [84] et ISO 42010 [85], qui répondent aux besoins généraux de l'industrie et reflètent les principes fondamentaux de l'ingénierie des systèmes qui fondent la une approche d'ingénierie des systèmes.

(2) Méthode

Le cycle de développement d'un système (SDLC) est un terme utilisé en ingénierie des systèmes, systèmes d'information et en logiciel pour décrire un processus de développement d'un système. Un cycle de vie de développement de système est composé d'un certain nombre de phases de travail clairement définies et distinctes, utilisées par les ingénieurs système et les développeurs de systèmes pour planifier, concevoir, construire, tester et livrer un système. Un certain nombre de modèles SDLC ont été créés, notamment en cascade, en fontaine, en spirale, de construction et de réparation, de prototypage rapide, incrémental, de synchronisation et de stabilisation [87]. Le premier paradigme de cycle de vie connu, introduit en 1956, était l'approche en cascade, qui est une approche de conception séquentielle relativement linéaire pour certains domaines de la conception technique. Le modèle en Vee (Figure C.4) [26] est proposé pour représenter un processus de développement considéré comme une extension du modèle en cascade plutôt que de

descendre de manière linéaire. Les étapes du processus sont pliées vers le haut après la phase de codage pour former la forme en Vee. Le modèle en Vee original ne précise pas les étapes spécifiques ni les modèles à utiliser. Par conséquent, un certain nombre de méthodes et de cadres MBSE sont proposés par la communauté de l'ingénierie des systèmes sur la base du modèle Vee et des normes MBSE. Ces méthodes fournissent les étapes de modélisation spécifiques et les cahiers des charges / produits dans les cadres.

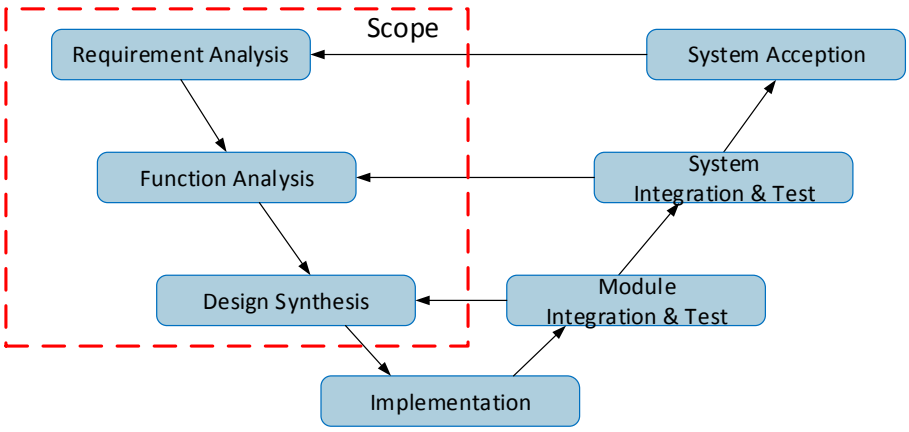


Figure C.5 Modèle en Vee

(3) Langage de modélisation

SysML est l'un des langages de modélisation graphique pour les applications d'ingénierie des systèmes, proposé pour la première fois par Object Management Group (OMG) et le Conseil international pour l'ingénierie des systèmes (INCOSE) en 2001 et adopté comme norme en mai 2006 [29]. SysML est une extension du sous-ensemble du langage UML (Unified Modeling Language) avec neuf types de diagrammes, permettant à SysML de représenter les systèmes et chaque composant, ainsi que leurs comportements et leurs structures. SysML comprend neuf diagrammes, comme indiqué dans la taxonomie du diagramme de la Figure C.6. Ici, le diagramme de séquence, le diagramme d'activité, le diagramme de machine à états, le diagramme de séquence et le diagramme de cas d'utilisation sont dérivés du langage UML. Le diagramme des exigences, le diagramme de définition de bloc, le diagramme de bloc interne et le diagramme paramétrique sont nouvellement définis dans SysML [80] :

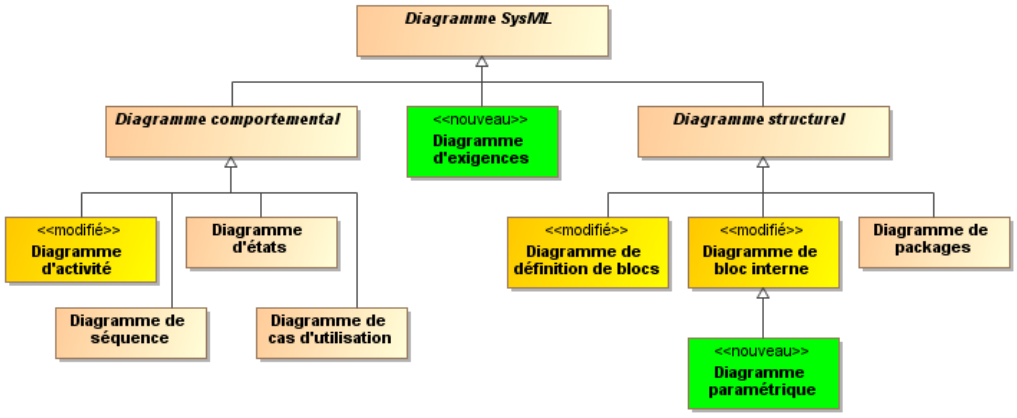


Figure C.6 Taxonomie du diagramme SysML

Comparé avec les modèles formels de la TCS, SysML fournit une variété de modèles semi-formels, qui peuvent bénéficier à l'AC et étendre le champ de modélisation de la TCS. De ce point de vue, les limitations du TCS peuvent être traitées :

(1) Exigence

Pour MBSE, l'analyse des exigences se situe toujours au tout début du cycle de vie. Les exigences initiales des parties prenantes sont généralement écrites en langage naturel, parfois à l'aide de modèles spécifiques à un domaine ou même de modèles informels non liés à une méthode ou à un langage [112]. Comme indiqué dans la limitation de la formalisation, le problème dans le cadre de la TCS est que les modèles du procédé et de la spécification sont toujours modélisés par une interprétation directe du cahier des charges. Le diagramme des exigences SysML peut être une solution pour aider à la formalisation des exigences textuelles. Les exigences en langage naturel peuvent être modélisées en blocs d'exigence, ce qui permet de gérer et de les organiser.

(2) Modèles de structure

Du point de vue de MBSE, la définition de la structure du système cible est indispensable au processus de conception global. Les modèles de structure peuvent être bien réalisés avec des diagrammes de structure SysML, incluant BDD (Block Definition Diagram) et IBD (Internal Block Diagram). Par rapport au modèle formel, BDD et IBD fournissent une représentation graphique explicite de la structure du système, ce que l'automate n'est pas en mesure de réaliser. Sur la base des blocs et des relations définis dans les BDD, les types d'entités physiques (c'est-à-dire le système, les composants du système et la structure hiérarchique) dans le système de contrôle peuvent être présentés. Grand nombre d'études de cas montre que l'utilisation de diagrammes de structure SysML est une solution possible de la description de la structure du système [115-117]. D'autre part, la structure des modèles formels, par exemple l'architecture de contrôle, peut également être présentée. Le bloc dans BDD est un élément de modèle qui n'est pas

simplement limité aux entités physiques, mais également une représentation étendue comprenant des concepts, une définition ou même des modèles dynamiques (par exemple, une activité). Par conséquent, le bloc peut être utilisé pour représenter les concepts formels, tels que générateur, spécification ou superviseur, de sorte que la structure de ces modèles formels puisse être présentée par un diagramme de structure SysML. Un certain nombre d'éléments de relation de modèle sont définis dans SysML pour visualiser le lien entre les modèles. Dans BDD, la composition, l'agrégation, la généralisation et l'association des relations sont généralement utilisées pour présenter la relation ou la structure, à l'aide de laquelle un système complexe ou structuré peut également être modélisé.

(3) Mise en œuvre

L'implémentation du contrôleur et les modèles de déploiement font également défaut dans les processus basés par la TCS. Comme indiqué précédemment, étant donné que le système de contrôle est un système matériel / logiciel, la constitution du superviseur basée sur architectures différentes n'implique pas directement la structure du contrôleur concret. Pour être plus précis, les ingénieurs souhaitent à avoir non seulement les modèles formels qui représentent la logique de contrôle du programme, mais également le lien entre le contrôleur physique et le système contrôlé. Basé sur SysML BDD et IBD, des modèles d'implémentation et de déploiement de contrôleurs peuvent être réalisés. L'architecture de superviseur peut être implémentée sur un contrôleur et des programmes concrets définis et décrits par BDD. De plus, le déploiement du contrôle peut être défini dans IBD, ce qui permet de présenter la connexion entre le contrôleur et les composants de l'architecture de contrôle, même si un contrôleur complexe tel qu'un contrôle modulaire est disponible.

C.4 Verrous Scientifiques

Afin d'approfondir l'étude des méthodes permettant de prouver que le diagramme SysML est capable de gérer les problèmes liés à la génération de contrôleur suivant l'AC, les contributions à la modélisation de système par SysML sont examinées.

En fait, on peut trouver un grand nombre d'études de cas [32], [89], [118], [119], mais il est impossible de les détailler ici, ce qui est la limitation de longueur de cette étude. Considérant que, en comparant les études de cas mentionnées avec l'approche basée sur la TCS, on peut tirer les conclusions suivantes :

(1) La problématique des limitations du TCS peut être validée. On voit dans les contributions que les modèles de structure et les modèles de mise en œuvre sont indispensables dans le processus de modélisation. En outre, les diagrammes des besoins sont généralement utilisés dans MBSE pour présenter l'analyse, la décomposition et la traçabilité des besoins informels. Le paradigme TCS

manque à tous ces modèles.

(2) Les contributions offrent des solutions alternatives pour traiter la problématique. Différents types de diagramme SysML sont introduits pour modéliser différents aspects du système étudié.

(2) Les contributions offrent des solutions alternatives pour traiter la problématique. Différents types de diagramme SysML sont introduits pour modéliser différents aspects du système étudié.

Du point de vue scientifique et de la pratique d'ingénierie, nous sommes encore intéressés par trois aspects :

- **Traçabilité**: La traçabilité désigne la situation où l'on dispose de l'information nécessaire et suffisante pour connaître la composition des produits tout au long de sa chaîne de développement. Parce que nous pouvons avoir produits ou modèles différents au cours du processus de développement, il faut de présenter explicitement le lien des produits du début à la fin.

- **Cohérence** : La cohérence signifie que les modèles différents n'entraînent pas une contradiction. Garder la cohérence entre les modèles est important non seulement lorsque nous construisons des modèles, mais également que nous les modifions.

- **Réutilisabilité**: La réutilisabilité signifie l'utilisation des actifs existants sous une forme ou une autre pour un autre processus de développement afin de réduire les doublons de développement et augmenter la productivité et l'interopérabilité.

La discussion susmentionnée montre l'intérêt et la possibilité d'intégrer TCS et MBSE pour AC. Afin de proposer une solution correspondante, il convient de relever certains défis et problèmes locaux :

Verrous 1 : Comment intégrer les langages de modélisation différents ?

La formalisation des exigences est toujours l'un des objectifs du processus de CA basé sur TCS. Bien que les modèles semi-formels tels que le diagramme SysML puissent aider à décrire l'analyse, la décomposition et la modélisation de la spécification informelle, une approche permettant de rapprocher les exigences textuelles et les spécifications formelles est nécessaire.

Pour la formalisation des exigences, un processus, appelé Natural Language Processing (NLP), de traduction automatique des exigences exprimées en langage naturel dans des modèles a été développé par [120-122]. Malgré l'intérêt de ce travail, ils ne s'appliquent qu'aux exigences qui ne sont pas écrites en langage naturel libre. En fait, les exigences traitées doivent être très structurées et régulières, utiliser un langage précis ainsi qu'un vocabulaire bien défini. Des autre auteurs proposent une méthodologie basée sur CTL * pour affiner la description en langage naturel [114]

[123]. Le processus de raffinement est pris en charge par une structure graphique formelle appelée graphique de pseudo-exigence ou dans un diagramme SysML. Cependant, il n'est pas encore clair que l'approche puisse être utilisée pour lier la spécification informelle à la spécification basée sur un automate. Par ailleurs, la sémantique particulière des exigences pour le contexte de l'AC doit également être discutée. En fait, les spécifications de la TCS devraient formaliser diverses exigences qui présentent des différences considérables selon l'application, même si les systèmes étudiés sont les mêmes.

Un autre problème concerne les liens entre modèles SysML et modèles formels. Certaines références traitent de la transformation entre automate et diagramme de comportement de SysML, tel qu'un diagramme de séquence et un diagramme d'activité [128-132]. Cependant, les propositions sont générales sans prendre en compte le contexte de la TCS. En fait, l'automate dans la TCS a sa sémantique particulière et une étude plus approfondie devrait être menée pour en vérifier la faisabilité. Par exemple, la spécification dans TCS comporte généralement des boucles à chemins multiples, ce qui peut rendre difficile l'utilisation du diagramme de séquence pour gérer cette situation.

En résumé, les approches formelles utilisées pour l'interprétation et la vérification formelles ont pu être intégrées aux modèles MBSE. Cependant, les approches de pointe ont leurs limitations. Premièrement, aucune proposition correspondant au contexte particulier du paradigme TCS ne peut être trouvée. Par exemple, le lien entre la structure du système et les modèles formels n'est toujours pas clair. Deuxièmement, les contributions existantes se concentrent sur l'utilisation d'approches formelles pour le calcul et la vérification. Peu de propositions prêtent attention à la représentation des concepts formels par les modèles MBSE, ce qui est l'un des objectifs de cette étude. Les limitations des modèles formels nécessitent la nécessité d'utiliser des modèles MBSE pour décrire explicitement les modèles formels de TCS, tels que l'architecture de contrôle. La traçabilité entre modèles formels et modèles physiques peut donc également être présentée.

Verrous 2 : Quel est le processus global de modélisation ?

Du point de vue de l'ingénierie système, il existe un certain nombre de méthodes dans lesquelles des modèles formels et des modèles MBSE sont combinés ou un modèle formel est intégré dans la méthode MBSE. [41] and [133-135] ont proposé une approche de modélisation similaire associant TCS et processus basé sur un modèle. Cette approche intègre les méthodes TCS dans le processus de modélisation afin de structurer le processus de synthèse du contrôle de supervision. Cependant, les approches ne fournissent pas les méthodes pour le processus global de l'AC pour l'analyse des besoins à la mise en œuvre. En fait, les approches essaient simplement d'utiliser le modèle TCS pour remplacer les documents pendant le processus de conception, ce qui ne peut être considéré comme un processus de modélisation intégré.

Le méta modèle, ou un modèle du modèle, est un type spécial de modèle qui spécifie la syntaxe abstraite d'un langage de modélisation [138]. L'architecture dirigée par un modèle est l'une des branches les plus actives de MBSE proposées par OMG. Le méta modèle peut être considéré comme un résumé de haut niveau et un modèle est toujours conforme à un méta modèle unique. En fait, les langages de modélisation MBSE tels que UML et SysML sont des méta modèles typiques proposés par OMG. L'approche par méta modèle est généralement utilisée en ingénierie des systèmes pour l'analyse et la construction de modèles et la transformation de modèles.

L'approche fondée sur les points de vue attire l'attention et des travaux utiles et pertinents peuvent être trouvés dans certaines contributions. [142] décrit un cadre d'ingénierie de systèmes orienté point de vue (VOSE) qui prend en charge l'utilisation de perspectives multiples dans le développement de systèmes, qui utilise des «points de vue» pour partitionner la spécification système, la méthode de développement et les représentations formelles utilisées pour exprimer les spécifications système. Cela montre que le cadre peut gérer le problème des perspectives multiples. Dans [143], propose une méthode générique pour définir des points de vue dans SysML en généralisant les concepts utilisés pour définir des points de vue dans le modèle de référence du traitement réparti ouvert (RM-ODP). En utilisant cette méthode, les points de vue à utiliser dans différents domaines problématiques peuvent être définis de manière systématique, et la réutilisation et / ou le partage de points de vue entre différents domaines problématiques peuvent également être facilités. Dans [86], les auteurs décrivent une technique développée par JPL consistant à appliquer des points de vue et des vues SysML afin de générer des documents et des rapports par XML. Cette technique montre qu'une grande variété de vues peuvent être générées et donne un aperçu de la façon dont la collaboration et l'intégration sont activées. Dans [144], les auteurs ont introduit une application de points de vue afin de pouvoir traiter la portée et les exigences en constante évolution des produits mécatroniques, grâce auxquelles les parties prenantes et les modèles de développement peuvent être intégrés. Ces contributions montrent que les approches basées sur les points de vue peuvent fournir de bonnes solutions pour le développement de systèmes lorsque nous avons besoin de perspectives et de préoccupations différentes. Les éléments d'architecture peuvent être bien organisés sous des points de vue. Par conséquent, prenant cela comme une avancée majeure, nous étudions la possibilité de redéfinir l'architecture de modélisation pour le développement de contrôleurs. Par exemple, pour proposer un diagramme SysML pour la conception contrôlée, la focalisation et le problème peuvent être spécifiés en fonction du point de vue correspondant.

Verrous 3 : Comment intégrer le contexte de la TCS ?

Dans le contexte de l'AC, la sémantique des modèles doit être prise en compte. Les concepts et les éléments entre SysML et la TCS doivent être liés. Par exemple, les problèmes suivants doivent être traités : (1) Quel est le lien entre les modèles SysML et les modèles de procédé, de spécification et de superviseur dans TCS? (2) Quels événements contrôlables et incontrôlables peuvent être

représentés par des éléments SysML? (3) Comment visualiser les concepts de synchronisation, de formalisation et de synthèse du superviseur? Ces questions sont importantes pour combler le fossé entre la formalisation et l'ingénierie alors que les réponses sont loin d'être claires pour ce qui est des contributions les plus récentes. Deuxièmement, le guide normalisé des processus MBSE pour les modèles à construire. Cependant, le processus doit être adapté au contexte particulier de la AC. En outre, les éléments de modèle de chaque diagramme doivent également être spécifiés. Comme indiqué précédemment, il est nécessaire de clarifier la sémantique du diagramme plutôt que d'utiliser le méta modèle d'origine du diagramme fourni par SysML, qui spécifie simplement la syntaxe. En raison de l'objectif de haute réutilisabilité, le méta modèle de chaque diagramme SysML proposé doit spécifier les éléments de modèle nécessaires et les relations entre ces éléments. Par exemple, lorsque nous proposons à un BDD de décrire l'architecture de contrôle, les modèles des superviseurs, des procédés et spécifications correspondantes ainsi que leurs relations doivent être détaillés par méta modèle. D'autre part, il convient de définir la méthode de modélisation de chaque modèle du cadre, qui n'est pas fournie par le processus normalisé MBSE. Habituellement, les méthodes peuvent être présentées par méta modèle. Dans TCS, la synthèse du superviseur peut être considérée comme une méthode de modélisation du superviseur. Toutefois, lorsque la TCS est étendue par des modèles SysML, aucune méthode de modélisation de pointe ne peut être trouvée pour ce contexte. C'est donc un autre défi à relever dans cette étude.

C.5 Cadre de Modélisation Proposé

Sur la base de la discussion ci-dessus, un nouveau cadre de modélisation est proposé dans cette étude. Afin de réaliser les objectifs visant à réduire l'écart entre le TCS et la pratique d'ingénierie et à faire face aux limitations du TCS, tout en faisant face aux défis de l'intégration du TCS et du MBSE, les réponses aux questions suivantes devraient être abordées dans la proposition Quels types de modèles devraient être définis dans le cadre proposé? (2) Quelle est la sémantique des éléments à définir dans les modèles particuliers au contexte de l'AC? (3) Comment les modèles et méthodes TCS peuvent-ils être intégrés dans le processus MBSE? (4) Quel est le processus de modélisation MBSE global pour AC?

C.5.1 Définitions de Modèle

Le modèle doit être défini pour réaliser la vue qui est régie par le point de vue correspondant. Les modèles formels de procédé, de spécification et de superviseur peuvent être directement utilisés comme modèles de comportement à la place du diagramme de comportement SysML (diagramme de séquence, diagramme d'activité et machine à états), car les modèles formels sont rigoureux en termes de syntaxe et de sémantique. La syntaxe est l'ensemble des règles, principes et processus qui régissent la structure des phrases et la sémantique signifie le sens du langage, des langages de

programmation, de la logique formelle et de la sémiotique. Par une étude plus approfondie des résultats de chaque phase de modélisation, seuls les modèles de comportement représentés par un procédé, une spécification et un superviseur officiels sont impliqués dans le processus TCS, en particulier dans les phases de formalisation et de synthèse. Les modèles impliqués dans le processus MBSE dépendent des langages de modélisation à utiliser. En prenant SysML comme exemple, une série de modèles est impliquée dans différentes étapes de modélisation. Dans l'analyse des exigences, les exigences système et les cas d'utilisation sont modélisés par des diagrammes d'exigences et des diagrammes de cas d'utilisation. Afin de garantir que toutes les exigences de performances fonctionnelles et associées sont couvertes par les cas d'utilisation, une traçabilité respective entre exigence et cas d'utilisation est établie. Dans l'étape d'analyse fonctionnelle, des diagrammes de comportement sont utilisés pour représenter les scénarios de cas d'utilisation. Il n'y a pas de modèle défini pour représenter la fonction et chaque diagramme de comportement (c'est-à-dire un diagramme d'activité, un diagramme de machine à états et un diagramme de séquence) peut jouer un rôle spécifique dans l'élaboration du comportement de cas d'utilisation. Dans l'étape de synthèse de la conception, plusieurs modèles doivent être construits. Le modèle d'analyse architecturale doit présenter une étude commerciale afin de déterminer le meilleur moyen de réaliser de manière rationnelle la capacité d'une fonction particulière, généralement sous forme de BDD. Les modèles de comportement de boîte noire devraient être réalisés par des modèles de comportement de boîte blanche. Les modèles d'architecture système et les modèles de sous-systèmes physiques représentent les résultats finaux, qui sont modélisés par BDD et IBD.

En conclusion, MBSE est plus performant dans l'analyse des exigences car des méthodes et des modèles sont fournis pour l'analyse, la décomposition et la traçabilité systématiques des exigences. En revanche, la TCS fonctionne mieux que MBSE pour l'analyse fonctionnelle en raison de l'approche formelle. Pour la conception et la synthèse, TCS et MBSE ont leurs propres avantages.

D'autre part, les diagrammes SysML doivent être utilisés pour réaliser d'autres types de modèles tels que la structure, les exigences et la traçabilité. Sur la base de la discussion sur le type de modèle, des modèles pour tous les points de vue peuvent être déterminés. Outre la modèle formelle, 10 SysML diagrammes sont définis pour être alloués aux points de vue dans le cadre proposé. Pour comparer les modèles formels dans un processus de modélisation typique, les modèles peuvent être logiquement regroupés en trois divisions centrées respectivement par division procédé (en rouge), spécification (en noir) et superviseur (en bleu).

Les relations entre les modèles sont présentées à Figure C.7. La modélisation des procédés dépend de trois BDDs qui représentent la structure du système, le contexte et le lien entre les procédés et les composants. Les BDD décrivent le système non-contrôlé du niveau de contexte au niveau du système et des composants. La décomposition du système et des composants d'intérêt est identifiée, sur la base de laquelle le lien entre les modèles de comportement et les modèles de

structure et le lien interne entre les systèmes physiques peuvent être suivis. Deux diagrammes d'exigences sont modélisés aux fins de la spécification formelle de liaison et du procédé. La formalisation de la spécification est basée sur le REQ : Exigence du système qui détaille l'analyse et la relation entre les exigences descriptives des parties prenantes. Par conséquent, les exigences peuvent être tracées par des spécifications formelles. Un autre diagramme d'exigences présente explicitement la traçabilité entre les modèles d'exigences et formels, y compris les procédés et les spécifications. Etant donné que l'approche formelle de la synthèse par le superviseur nécessite des procédés et des spécifications correspondantes, le diagramme des exigences fournit également une relation explicite des modèles formels, indispensable pour une architecture de contrôle. En se concentrant sur la synthèse du superviseur et la mise en œuvre du contrôleur, les BDDs et les IBDs sont modélisés pour présenter le AC au niveau formel / logique / physique, respectivement. Pour le modèle de chaque niveau, les modèles sont conformes à la sémantique particulière :

<<Formalize>> : Comme le procédé et la spécification sont deux modèles qui devraient être formalisés à partir des cahiers des charges , deux situations peuvent rendre l'utilisation de la relation «Formalize» dans les diagrammes SysML. Des composants physiques et des blocs de plantes pour présenter la sémantique de la formalisation, le lien entre les exigences et les blocs de spécifications peut également être lié par la relation.

<< SynthesizeFrom >>: Les résultats de calcul de la synthèse du superviseur dépendent des modèles des procédé et des spécifications, ce qui permet d'étendre la dépendance des relations afin de relier les blocs de modèles formels. Doit décrire l'architecture de contrôle dans laquelle la sémantique de la synthèse du superviseur basée sur les procédés, la spécification et les superviseurs doit être présentée dans le diagramme. A cette fin, la relation «SynthesizeFrom» fournit une connexion directe entre les blocs Plantes et spécifications correspondantes.

<< Synchronize >>: La sémantique de ce stéréotype est la présentation de la synchronisation des plantes locales aux plantes globales. Au fur et à mesure que la relation s'étend de la composition, les plantes locales font donc partie intégrante des plantes globales. Soyez explicitement décrit dans le schéma IBD: Architecture Structure, dans lequel les installations locales peuvent être directement insérées dans les installations globales.

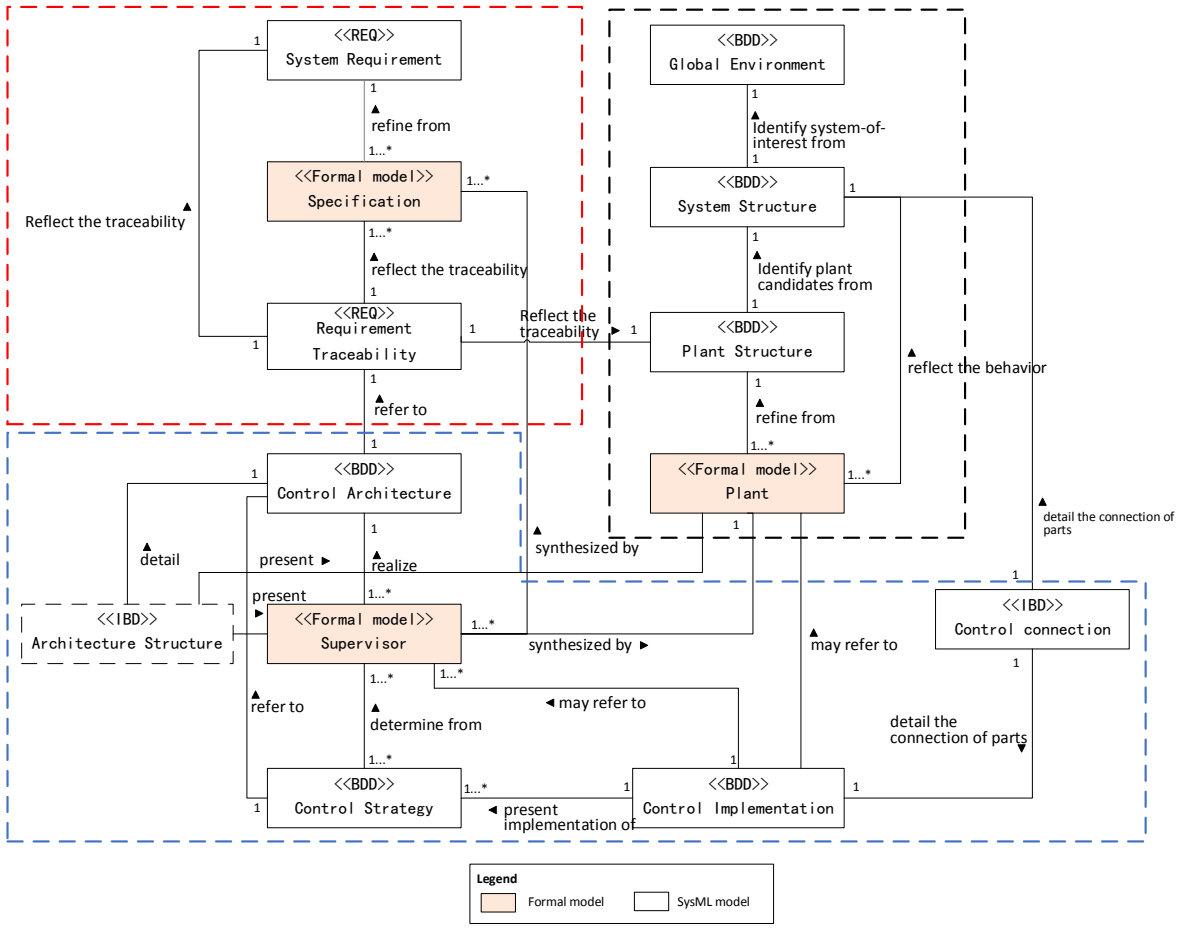


Figure C.7 Définition et liens de modèles

C.5.2 Processus de Modélisation Proposé

Un processus de modélisation globale est proposé pour intégrer l'architecture adaptée à AC (voir Figure C.8), y compris les modèles formels, les modèles SysML proposés et une série de méthodes de modélisation, de synthèse et de vérification. En prenant en compte la pratique d'ingénierie de l'AC, les cahiers des charges et produits globales dans cette étude sont supposées comme suit :

Cahier des charges globales :

(1) Besoins des parties prenantes : Les parties prenantes sont des personnes qui ont un intérêt ou sont en charge du projet. Dans MBSE, les besoins des parties prenantes impliquent généralement un grand nombre de conditions à différents aspects, y compris les conditions fonctionnelles et les conditions non fonctionnelles. Afin de simplifier les exigences et de se concentrer sur le contexte de AC, cette entrée est supposée spécifier uniquement les besoins en spécifications de contrôle.

(2) Contexte : le contexte est supposé être l'environnement global du système à étudier, y compris les systèmes externes et les participants.

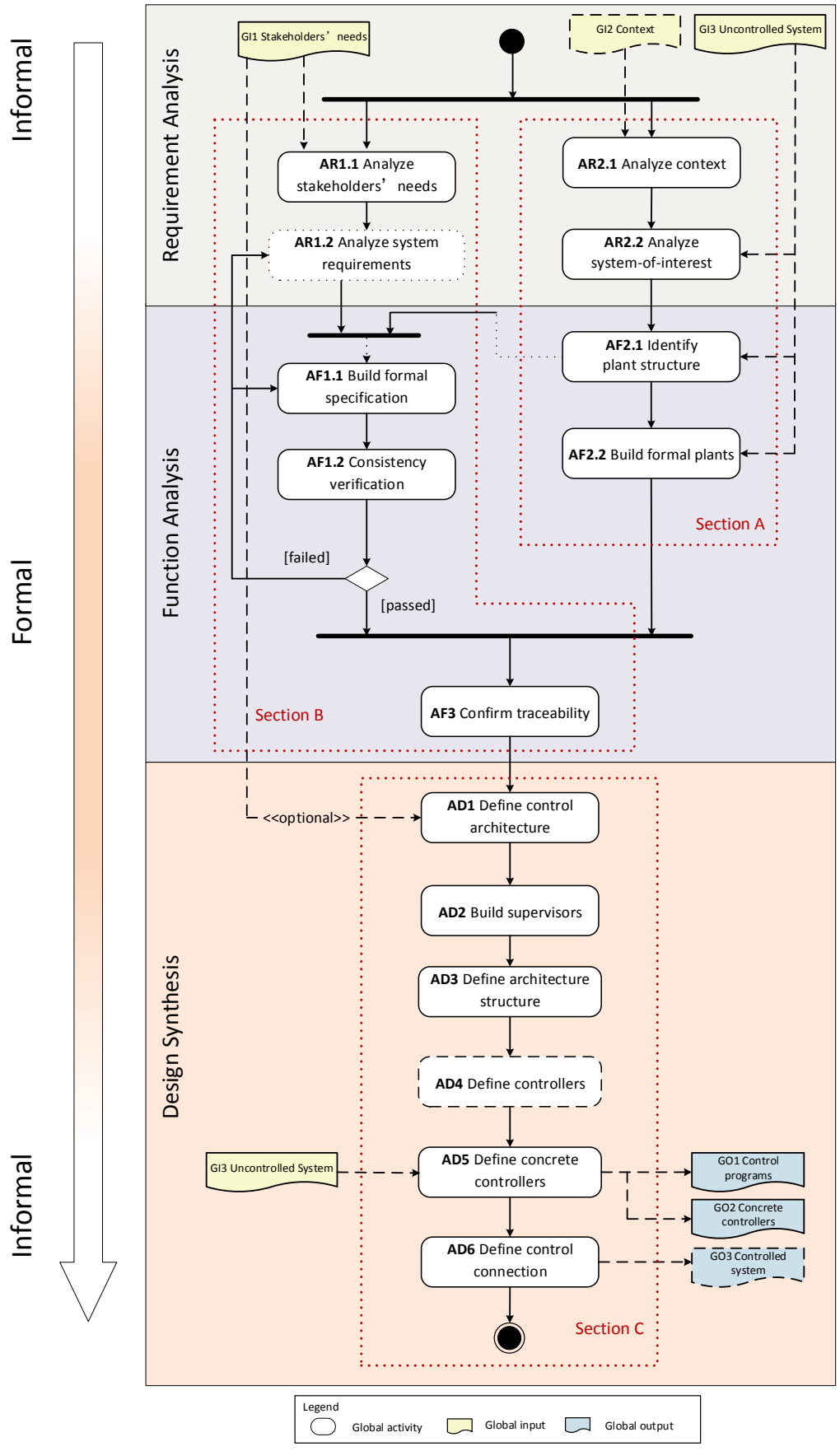


Figure C.8 Processus de modélisation globale

(3) Système non-contrôlé : la structure et les composants physiques du système incontrôlé doivent être fournis en détail, y compris les fonctions et les opérations.

Produits globaux :

(1) Contrôleur concret : Le contrôleur concret doit être désigné par un type de contrôleur (par exemple, un automate programmable, un microcontrôleur, etc.). Dans cette étude, BDD : Control Implementation est utilisé pour représenter ce résultat global, préconisé par MBSE.

(2) Programme de contrôle : le programme de contrôle doit être généré par la méthode de mise en œuvre appropriée et pouvoir être exécuté sur le contrôleur concret défini.

(3) Système contrôlé : la sortie nécessite un système complet comprenant un contrôleur et un système à contrôler. Semblablement, il est supposé être présenté par le modèle IBD : Control Connection.

(1) Section A

L'objectif de la section sous-processus est axé sur la formalisation des procédés. Par conséquent, le système non-contrôlé et le contexte doivent être analysés. Étant donné que la structure du système joue un rôle important dans la modélisation des résultats de la centrale, il est nécessaire de porter une attention particulière au lien entre les modèles formels et les composants physiques dans cette section. Quatre sorties seront produites dans cette section : trois SysML BDDs et des procédés formels. Les principaux objectifs des activités dans Section A sont axés sur :

- analyser l'environnement global de l'ensemble du système à étudier et de construire le modèle d'environnement global
- détailler la structure du système d'intérêt
- lier les composants physiques et les plantes formelles
- construire les modèles d'automates pour tous les procédés

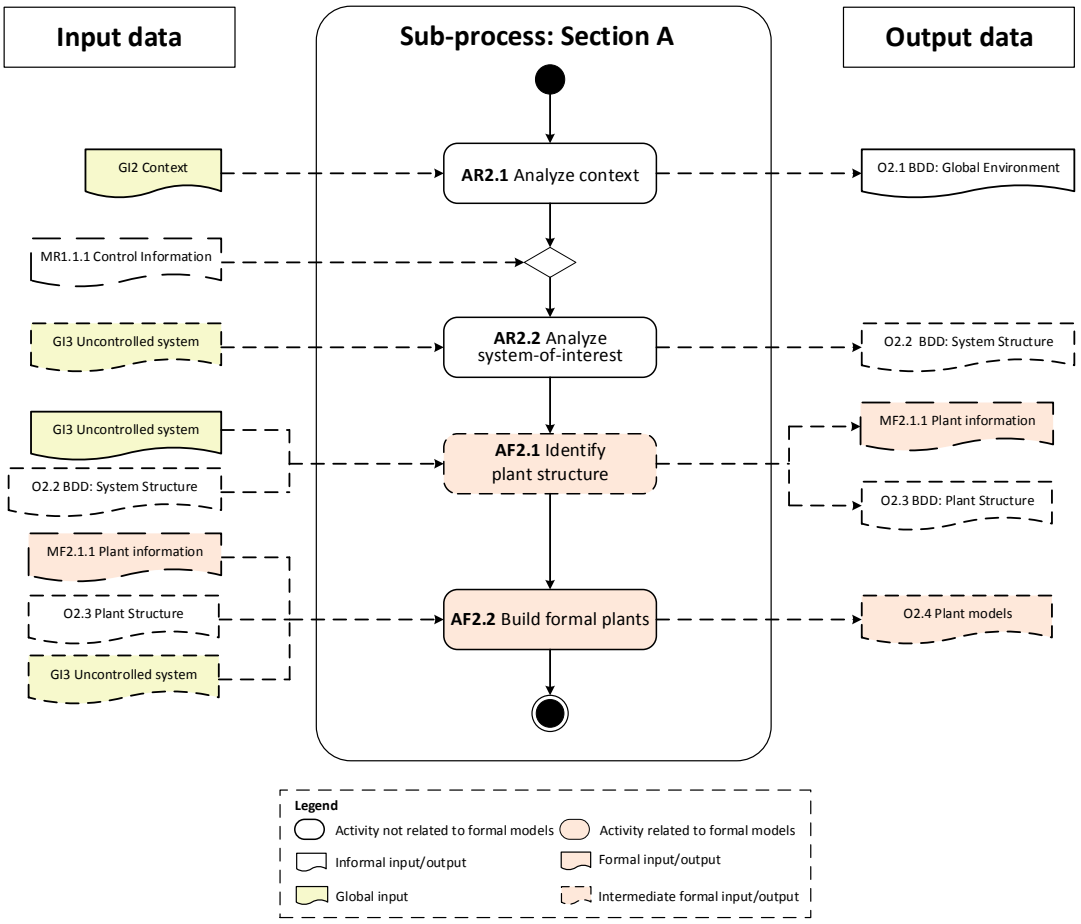


Figure C.9 Sous-processus de la section A

(2) Section B

L'objectif du sous-processus section A est d'analyser le besoin des parties prenantes en spécifications de contrôle, de décomposer les exigences, de formaliser et de vérifier les spécifications. A cet effet, la section A est composée d'activités couvrant les aspects suivants : (a) le prétraitement des informations dans les besoins des parties prenantes, (b) l'analyse et la décomposition des exigences narratives, (c) la formalisation des exigences, (d) la vérification de la cohérence entre les exigences et les spécifications formelles et (e) la traçabilité et la cohérence entre des exigences et des modèles formels. Nous devons présenter le lien entre les exigences narratives et les automates qui partagent la même sémantique. De plus, nous devons vérifier la cohérence entre exigence informelle et spécifications formelles selon la sémantique partagée.

Trois sorties sont produites dans la section, y compris trois diagrammes SysML et modèles de spécification. Les principaux objectifs des activités dans Section A sont axés sur :

- analyser et décomposer les exigences narratives de haut niveau en exigences de bas niveau pour la formalisation et la traçabilité dans les activités suivantes

- construire les modèles formels selon les exigences narratives

- vérifier la cohérence entre les exigences et les spécifications formelles ; intégrer les exigences, les procédés et les spécifications et présenter explicitement dans un diagramme les liens entre ces modèles

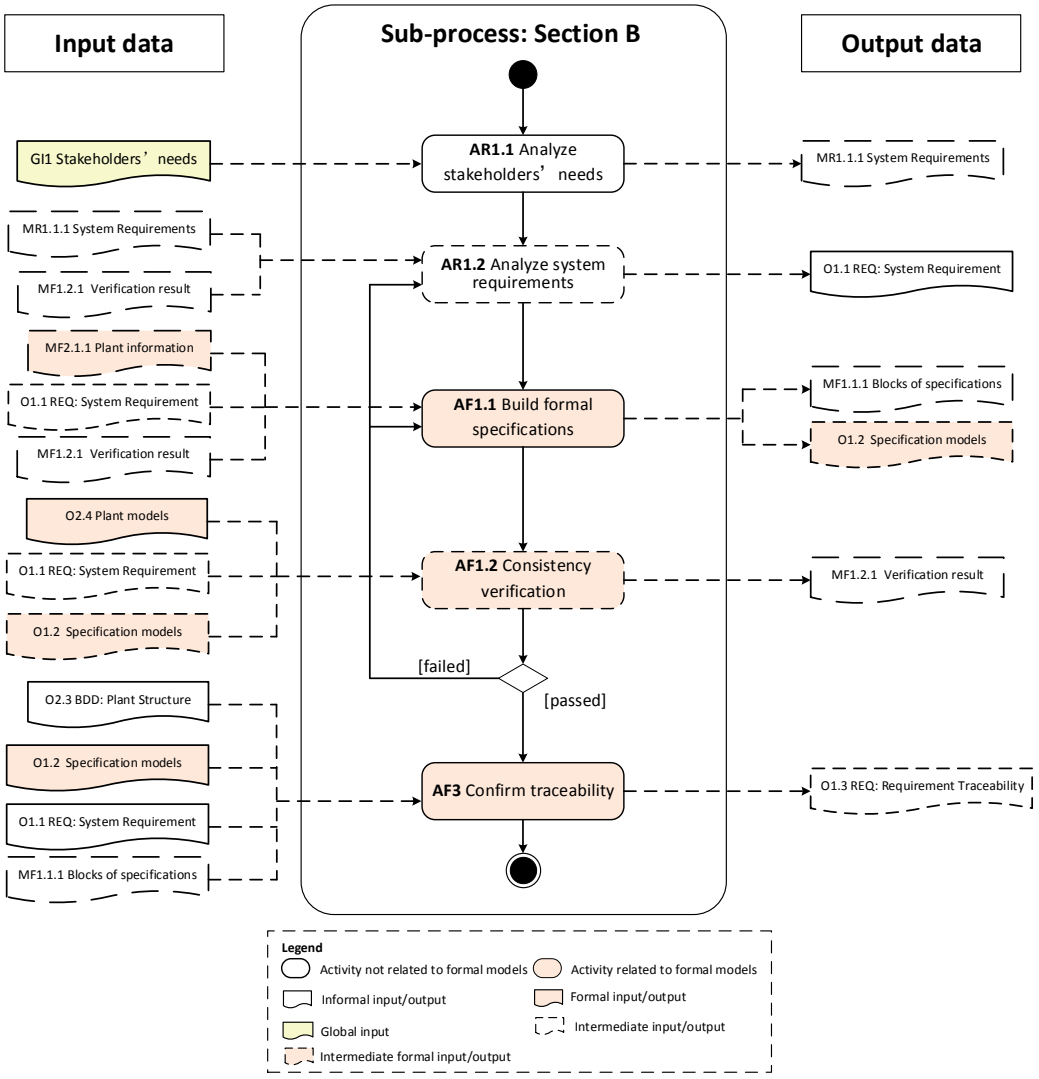


Figure C.10 Sous-processus de la section B

(3) Section C

La section C remplace la phase de synthèse de conception MBSE. Dans cette section, le fil du processus de développement est rigoureusement à la lumière de l'ordre de synthèse du superviseur, synthèse du contrôleur et mise en œuvre du contrôleur. En fait, les deux sous-phases suivantes peuvent être considérées comme une analyse architecturale et la mise en œuvre du contrôleur peut être considérée comme une conception architecturale. Tous les résultats globaux doivent être produits dans la section. Sept sorties sont produites dans la section, y compris des modèles pour les superviseurs, les contrôleurs et la mise en œuvre des contrôleurs. Parmi ceux-ci, l'implémentation du contrôleur est la sortie globale du processus de modélisation, impliquant le programme de

contrôle GO1, le contrôleur concret GO2 et le système contrôlé GO3. Les principaux objectifs des activités dans Section A sont axés sur :

- déterminer et présenter l'architecture de contrôle ; détailler la structure des superviseurs ainsi que les relations avec les procédés et les spécifications correspondantes.
- construire les modèles de superviseur basés sur l'architecture de contrôle définie
- construire les modèles pour les contrôleurs concrets et générer des programmes de contrôle
- Présenter explicitement la connexion de contrôle entre les contrôleurs concrets et les composants physiques afin de clarifier la structure de contrôle physique du système contrôlé global

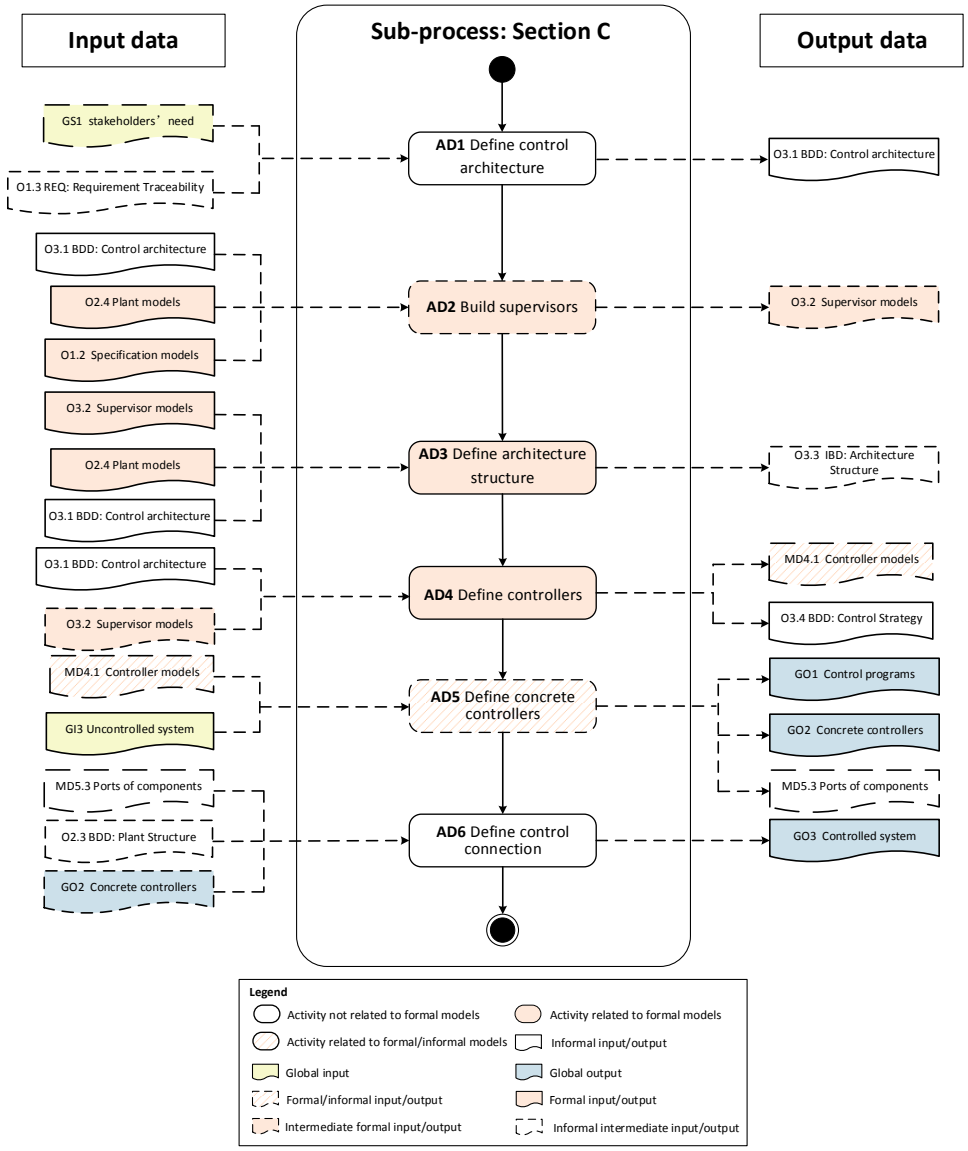


Figure C.11 Sous-processus de la section C

C.6 Cohérence des Modèles Formels et des Modèles SysML

Dernier problème à résoudre : le lien entre les modèles formels et les modèles SysML. Ce problème a été abordé, dans lequel le problème de formalisation doit être résolu pour le procédé et les spécifications. Dans le processus de modélisation proposé, les activités des sous-processus Section A et Section B sont impliquées dans le problème de la formalisation. Les problèmes et solutions suivants liés à la formalisation seront présentés en détail :

(1) La cohérence entre les procédés formelles et les composants physiques : dans cette partie, les liens entre les éléments physiques et formels seront définis, sur la base desquels l'identification de modèles de procédé sera introduite. Deuxièmement, afin d'éliminer complètement le risque d'incohérence entre le comportement réel et le procédé, une approche basée sur des modèles sera proposée basée sur les modèles SysML.

(2) La méthode de cohérence et de traçabilité entre spécifications formelles et exigences. Nous devons vérifier si les exigences informelles et les spécifications formelles correspondantes représentent la même sémantique. Quand ils sont incohérents, comment gérer la situation ? Dans cette partie, afin de tirer le meilleur parti du diagramme d'exigences SysML pour l'analyse et la formalisation des exigences, la méthode de vérification de traçabilité est proposé.

C.6.1 Cohérence des Procédés et du Système Physique

L'identification et la formalisation de le procédé est l'une des étapes difficiles, qui mérite beaucoup d'attention dans le cadre du processus de modélisation globale. Les procédures de synthèse du superviseur nécessitent l'utilisation de modèles de procédé cohérents avec les comportements des composants. Si ces aspects ne sont pas exprimés dans le modèle de procédé, le contrôleur synthétisé pourrait ne pas fonctionner comme prévu [150]. Dans cette section, nous nous concentrons sur deux aspects de la formalisation d'un procédé : (1) quels types de composants physiques doivent être identifiés comme étant un procédé provenant d'un système incontrôlé ? et (2) s'il existe une solution pour que les modèles de procédé reflètent strictement le comportement des composants physiques cibles ? Figure C.12 montre un schéma de lien entre le système physique et les modèles formels dans des études de cas typiques. Le comportement du système physique incontrôlé est formalisé par des actionneurs et des capteurs identifiés. Toutefois, les détails du lien ne sont pas spécifiés dans les contributions existantes. En fait, le problème pose des difficultés pour décomposer et formaliser le système physique incontrôlé et le traçabilité entre le composant physique et le procédé formelle reste ambiguë. Dans ce but, nous proposons notre solution dans cette étude. Le lien sémantique des événements formels et des modèles SysML peut être défini comme suit :

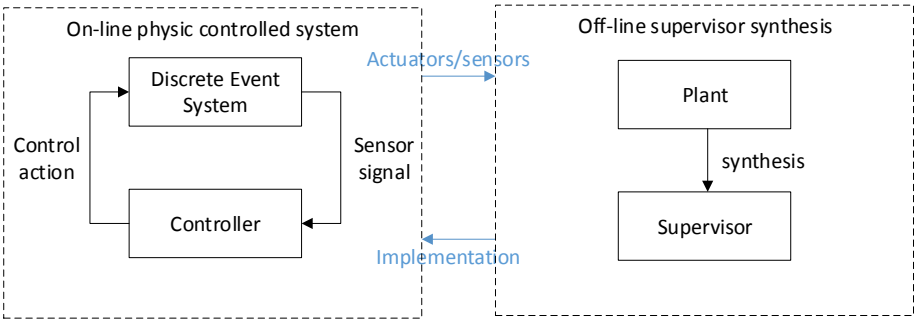


Figure C.12 Schéma du lien entre système physique et modèles formels

Evénement contrôlable : un événement contrôlable peut être lié à un actionneur d'un composant. L'actionneur est la partie exécution du composant. La sémantique d'un événement contrôlable est la représentation d'un fonctionnement de l'actionneur déclenché par le signal de commande du contrôleur.

Evénement incontrôlable : un événement incontrôlable peut être lié à un capteur de composant. La sémantique d'un événement incontrôlable est la représentation d'un signal d'un changement intervenant dans le système détecté par le capteur et envoyé au contrôleur.

Pour résumer, les événements formels ont une correspondance bijective pour commander / signaler. Cependant, la correspondance entre événement formel et actionneur / capteur est injective. En d'autres termes, l'actionneur / capteur peut recevoir / envoyer plusieurs commandes / signaux et chaque commande / signal peut être formalisé à l'événement correspondant.

Il est toujours vrai que les modèles de procédé peuvent contenir à la fois des événements contrôlables et des événements incontrôlables. En d'autres termes, le composant physique formalisé par le procédé peut être composé à la fois d'actionneurs et de capteurs. Par conséquent, nous définissons que le procédé peut être formalisé pour un composant physique lorsqu'au moins une opération d'actionneur ou un signal de capteur identifié pour interagir avec le contrôleur. À cette fin, afin de relier les composants physiques aux procédés formels, toutes les opérations des actionneurs et les signaux des capteurs doivent être identifiés.

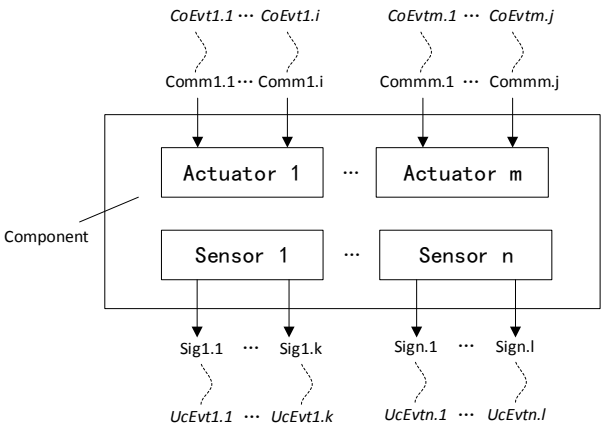


Figure C.13 Identification de la composition du composant physique

Bien que le processus d'identification améliore effectivement la formalisation du procédé, il ne peut toujours pas garantir de manière approfondie l'exactitude du modèle du procédé. Contrairement à la formalisation de spécification, qui peut tracer des exigences narratives, les procédés doivent être formalisées directement en analysant le comportement de composants physiques incontrôlés donnés. L'expertise en modélisation détermine toujours la qualité des modèles dans la pratique de l'ingénierie. Le modèle paramétré peut constituer une solution pour améliorer la possibilité de réutilisation du processus de modélisation. Avec l'aide du diagramme BDD SysML, la méthode de paramétrage peut être encore améliorée. En pratique, les composants du système à contrôler sont ce qui est «sélectionné» plutôt que ce qui est «conçu», ce qui est conforme à la tendance du COTS (Commercial Off-The-Shelf). Les propriétés de chaque composant peuvent être identifiées à l'avance. Les composants du système à contrôler peuvent généralement être normalisés et les comportements peuvent être paramétrés avant la modélisation. Pour chaque système spécifique (par exemple, un système de fabrication, un système électrique, etc.) à contrôler, une bibliothèque de modèles peut être construite sur la base des modèles qui représentent des comportements typiques de composants paramétrés. La bibliothèque peut être réalisée par le bloc paramétré dans SysML. En d'autres termes, lors de la construction de la structure du système BDD, les modèles de comportement des composants sont automatiquement construits en même temps. Nous pouvons donc bénéficier de cette approche :

(1) Les modèles des procédés étant construits automatiquement, le problème fastidieux de la formalisation peut être résolu. Comparez avec les méthodes de paramétrage existantes, une combinaison avec SysML BDD élimine la modélisation des modèles paramétrés abstraits;

(2) L'approche basée sur des modèles garantit que les modèles de procédé peuvent refléter strictement les comportements des composants physiques, quelle que soit la performance humaine. La bibliothèque construite à l'avance contient un certain nombre de modèles de comportement paramétrés prédéfinis pour les composants concrets, qui sont élaborés par des experts.

Pour réaliser l'approche proposée, un modèle à trois niveaux est proposé pour chaque composant. Une collection de modèles constitue la bibliothèque. Le principe du modèle est qu'il paramètre le comportement typique du composant physique correspondant et qu'il est instancié lorsque le composant est fourni. Le modèle de modèle à trois niveaux comprend un modèle SysML, une interface et un modèle de procédé (Figure C.14). Le modèle de bloc SysML est directement utilisé par le concepteur, qui encapsule le modèle de procédé. Entre les deux niveaux, l'interface doit être construite pour transformer le modèle SysML en modèle TCS. Lors de la modélisation d'un système à contrôler, l'utilisateur modélise chaque composant par modèle SysML et les organise dans BDD. Ensuite, ce modèle sera transformé automatiquement en modèle formel.

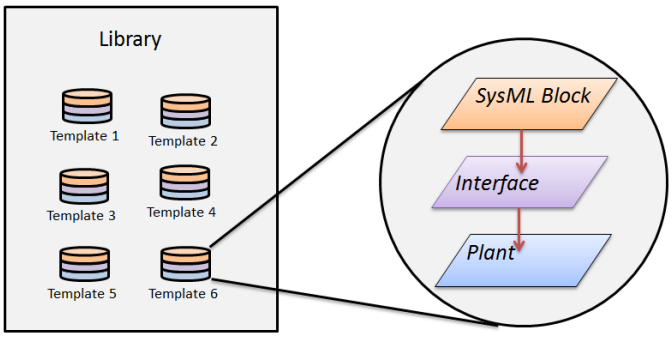


Figure C.14 Schéma du modèle à trois niveaux et de la bibliothèque

C.6.2 Cohérence des spécifications formelles et des exigences

Selon les études de la contribution, il existe un écart entre les exigences informelles décrites par le langage naturel et les spécifications représentées par un modèle formel et il est nécessaire de construire la cohérence des exigences et des spécifications formelles. À l'aide du diagramme d'exigences SysML, les exigences narratives informelles peuvent être analysées et décomposées par des modèles SysML semi-formels, qui augmente fortement la réutilisabilité de notre cadre proposé.

Afin de prendre en compte le contexte de l'AC et le lien avec les spécifications formelles, la description du besoin devrait contenir :

- Sur ou sur plusieurs sujets impliqués : les sujets représentent la composante physique ou le sous-système cible, qui déterminent quelle composante ou quel système doit remplir la fonction désignée.

- Evénement : l'événement décrit dans le texte doit être une abstraction de haut niveau en tant qu'événement dans un modèle formel plutôt que celui orienté composant. Par conséquent, l'événement peut être directement mappé (la relation bijective) aux homologues dans une

spécification formelle.

- Fonction : fonction indique la contrainte de comportement sur le comportement libre des sujets pour atteindre les objectifs fonctionnels désignés.

- Condition : la condition représente les restrictions à respecter par la fonction de subjectif. La description peut être la fonction d'un autre sujet.

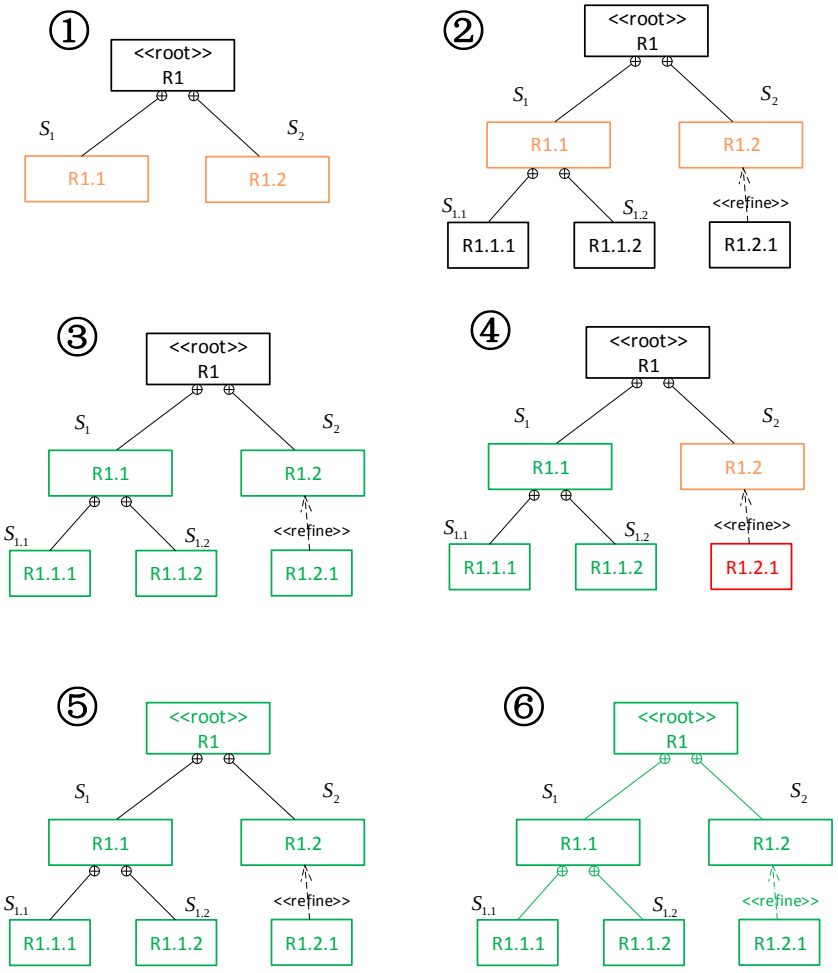


Figure C.15 Processus de vérification d'exigence

La condition est écrite en commençant par la conjonction, telle que «après», «quand», «une fois» ou «si». Chaque E-spécification devrait décrire la fonction logique définie, qui exprime de manière articulée un objectif unique de la contrainte de comportement. Deuxièmement, afin de maintenir la cohérence entre l'E-exigence et la spécification formelle, la E-spécification devrait être identifiée par un formalisme, qui revêtira une importance vitale pour la vérification de la traçabilité dans l'activité suivante. Les exigences narratives informelles peuvent être modélisées par des modèles d'exigences SysML. Par modèle d'exigence, nous pouvons effectuer la décomposition et décrire

explicitement les relations entre les exigences. Par conséquent, nous devons définir un modèle d'E-exigence qui contient la E-spécification.

On peut en conclure que la vérification des exigences est un processus d'itération impliquant une série d'activités dans la section B, notamment l'activité AR1.2 (Analyser les exigences système), AF1.1 (Construire des spécifications formelles) et AF.1.2 (Vérification de la cohérence). Le détail des activités de AF.1.2 est présenté à la Figure C.15, composée de trois activités :

- Modèles de spécification de charge et de spécification : Le modèle REQ : Exigence du système et spécification est chargé en tant qu'entrée de l'algorithme de vérification.

- Effectuer un algorithme de vérification : la cohérence entre chaque E-exigence fondamentale dans le diagramme des exigences système et la spécification correspondante doit être vérifiée dans l'ordre de séquence de l'algorithme.

- Vérifier le résultat de la vérification : puisque l'algorithme exporte les informations du résultat de la vérification, la partie incohérente entre exigences et spécifications formelles peut être connue et, par conséquent, les exigences ou les spécifications peuvent être modifiées en conséquence. Par exemple, lorsque la description de l'exigence R1.2.1 (Figure C.15(4)) ne correspond pas à la même sémantique des spécifications formelles, l'erreur doit être vérifiée et l'activité correspondante doit être retournée pour la modification des modèles. S'il est effectivement vérifié, la traçabilité des exigences peut être confirmée et le processus peut être déplacé vers l'activité AF3 (Confirmer la traçabilité). Nous devons présenter le lien entre les exigences informelles et les modèles formels (procédés et spécifications) : Quelle exigence est conforme à la spécification formelle ? et Quel procédé est impliquée dans les exigences ? Cependant, il existe une situation où, peu importe le nombre de fois où les modifications sont effectuées, l'erreur se produit toujours. Il est raisonnable d'affirmer que l'exigence elle-même pose le problème de la formalisation plutôt que le processus de formalisation lui-même. Par conséquent, il est nécessaire d'effectuer la communication avec les parties prenantes qui répondent aux exigences initiales.

C.7 Étude de Cas

Dans l'étude de cas, une centrale de puissance personnalisé (CPP) est prise comme système cible. Dans [19], les auteurs ont abordé l'application du TCS pour réaliser la coordination de divers dispositifs dans la CPP, axée sur la synthèse et la simulation par le superviseur. Dans cette étude de cas, nous montrons comment intégrer la TCS dans le processus AC, depuis l'analyse des cahiers des charges jusqu'à la mise en œuvre concrète de contrôleurs basée sur le cadre proposé.

La CPP peut être considérée comme un micro-réseau avec une structure typique illustrée à Figure C.16. Pour se concentrer sur le contrôle de niveau élevé de supervision du CPP, le système est

simplifié comme étant composé de trois dispositifs personnalisés : distributeur de tension dynamique (DVR), commutateur de transfert statique (STS) et filtre de puissance active (APF). Une combinaison de trois appareils est normalement alimentée par deux distributeurs et un procédé de secours. Ces dispositifs ainsi qu'un procédé de secours coopèrent les uns avec les autres pour supporter l'alimentation des charges.

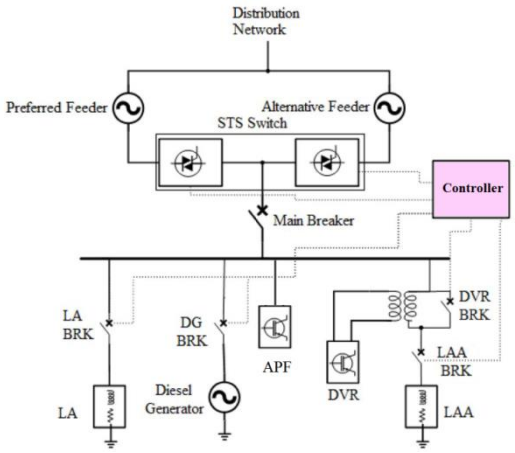
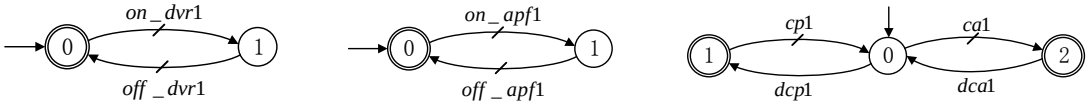


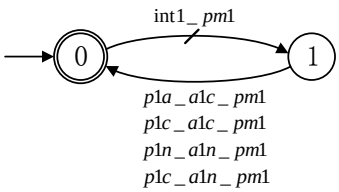
Figure C.16 Custom Power Park [19]

Les perturbations de la qualité de l'alimentation se rapportent à tout écart de tension, de courant et / ou de fréquence susceptible d'entraîner un dysfonctionnement ou une défaillance des équipements sensibles. Pour améliorer la qualité de l'alimentation, il convient de s'attaquer aux problèmes d'affaissement / gonflement de la tension, de déséquilibre de tension et de courant et de distorsion harmonique. En cas de chute de tension ou d'interruption dans le distributeur préféré, un commutateur de transfert statique (STS) est utilisé pour transférer le distributeur préféré au distributeur alternatif. Cette stratégie pourrait fournir aux charges sensibles une puissance continue de haute qualité. Dans le cas de la récupération de la qualité de l'énergie dans le distributeur préféré, celui-ci doit repasser sur ce distributeur. La qualité de l'alimentation peut être normale, anormale ou critique si le niveau d'affaissement / gonflement de la tension est inférieur à 10%, compris entre 10% et 50% ou supérieur à 50%, respectivement.

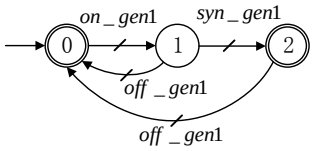
Les modèles formels de contrôle automatique essayant pour le système cible sont modélisés et calculés par le paradigme TCS, qui est donné dans (Kharrazi, Mishra et Sreeram, 2019). Figure C.17 présente certains des modèles officiels, y compris les procédés, les spécifications et les superviseurs.



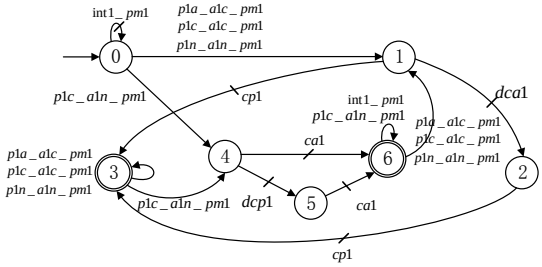
(a) Procédé DVR (b) Procédé APF (c) Procédé STS



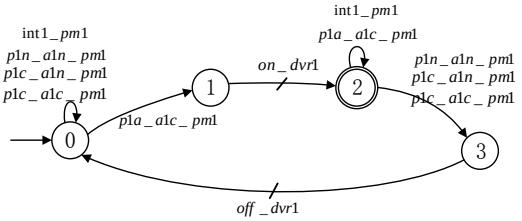
(d) Procédé moniteur de puissance



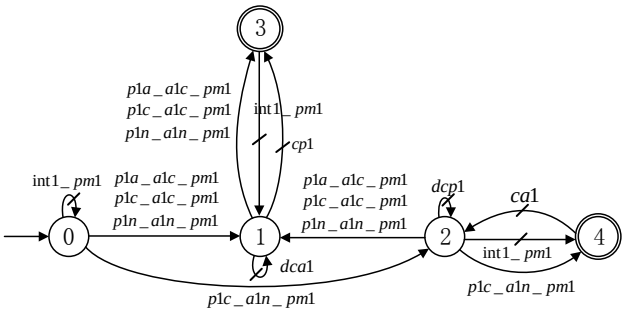
(e) Procédé générateur



(f) Spécification 1



(g) Spécification 2



(g) Superviseur 1

Figure C.17 Parties de modèles formels pour tentative d'AC

Selon cette proposition, les diagrammes SysML devraient être modélisés en complément des modèles formels. Compte tenu de la description du système, deux éléments sont définis pour composer le contexte de l'étude de cas. Le CPP est centralisé par le système d'éléments, pour lequel un contrôleur doit concevoir pour atteindre l'objectif d'une alimentation de haute qualité. En conséquence, le CPP est composé de deux parties : le système à contrôler et le contrôleur. Deuxièmement, le CPP est considéré comme un micro-réseau des réseaux électriques mondiaux et l'élément représentant le système externe est donc désigné par les réseaux électriques. Par conséquent, le résultat de la modélisation du contexte peut être présenté par BDD illustré à Figure C.18.

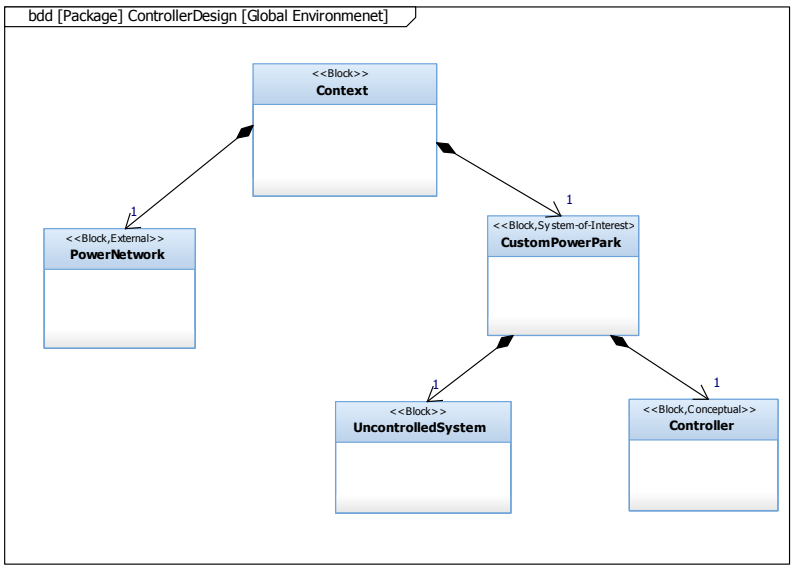


Figure C.18 BDD: Environnement global

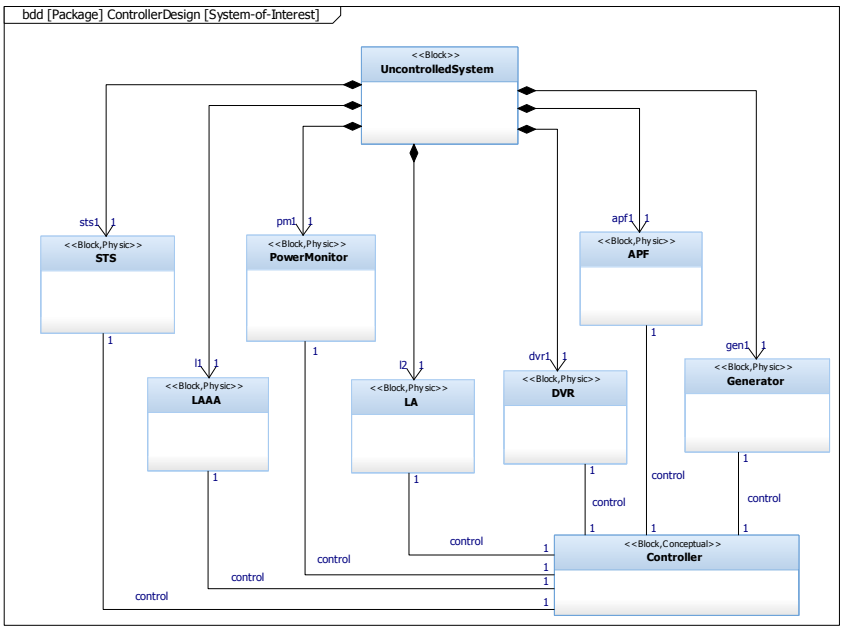


Figure C.19 BDD: Structure du système

Le système d'intérêt est composé de sept composantes correspondant aux sept blocs. Le BDD définit sept types de bloc (Figure C.19) : STS, moniteur de puissance, LAAA, LA, DVR, APF et procédé, et les composants physiques du système incontrôlé peuvent être définis par les blocs correspondants avec leurs noms de composants correspondant au procédé candidats. Il n'y a pas de structure hiérarchique du système et tous les composants sont directement regroupés dans le système incontrôlé. Dans le BDD, la relation de contrôle doit être présentée par l'association «control» entre le contrôleur conceptuel et les pièces.

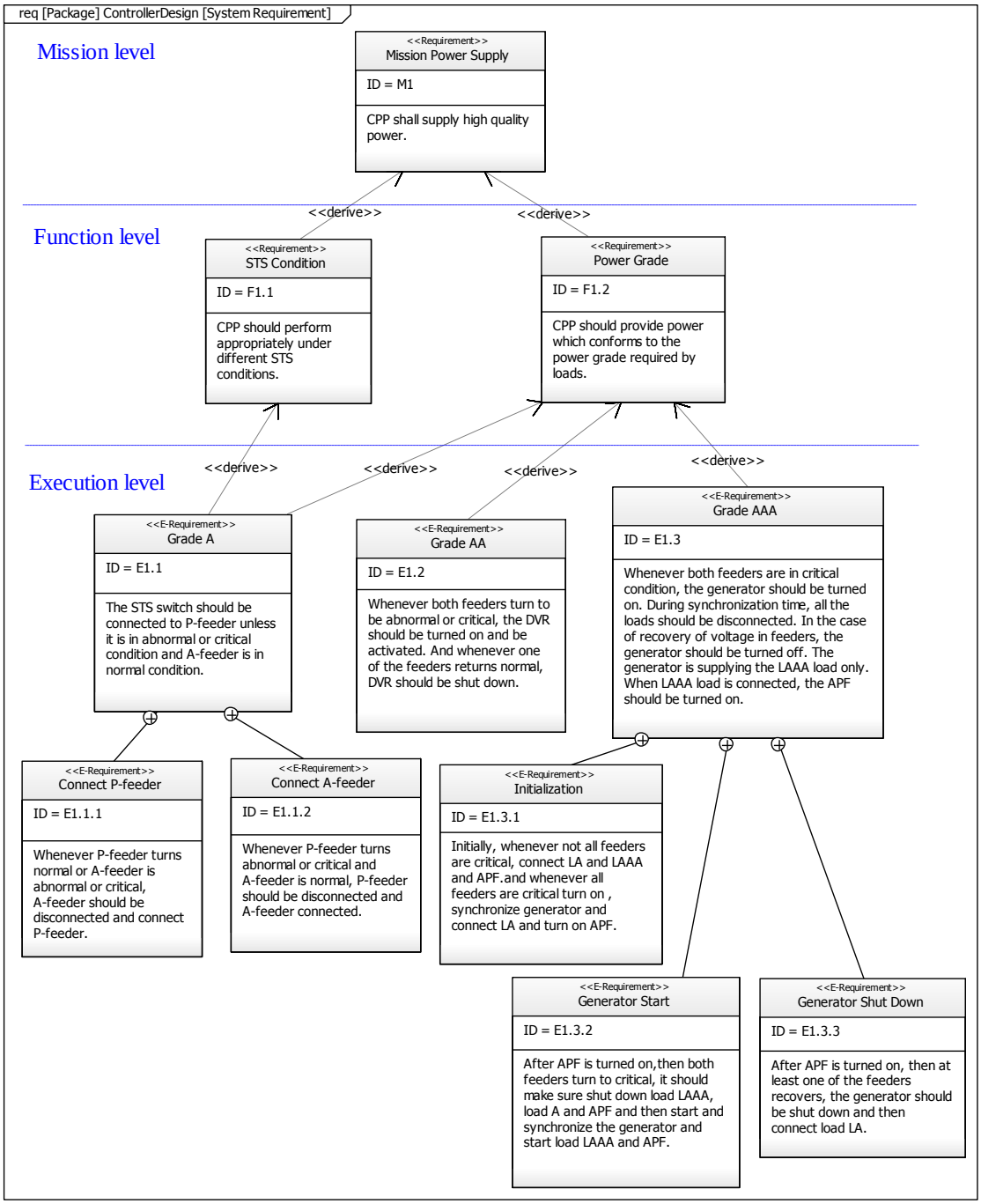


Figure C.20 REQ: Exigences du système

Les exigences sont divisées en trois niveaux (voir Figure C.20). Le niveau d'exigence le plus élevé définit la cible de la mission de contrôle. A ce niveau, une ou plusieurs des principales exigences dont les descriptions spécifient quels objectifs globaux du système doivent être atteints sous contrôle. La description des exigences de la mission doit être concise en utilisant plusieurs mots clés et aucun détail de la fonction ne doit être présenté. Les exigences définies au niveau de la fonction sont désignées pour spécifier la fonctionnalité du système contrôlé pour atteindre l'objectif global des exigences de la mission. Les descriptions peuvent être soit logiques soit

quantitatives tant que la mission peut être bien expliquée. En outre, la décomposition des exigences à ce niveau peut être hiérarchisée. Au niveau de l'exécution, des exigences sont spécifiées pour la réalisation de l'exigence de la fonction au niveau du composant. Destinée à la traçabilité pour la spécification formelle, l'exigence d'exécution est définie avec la description dans laquelle le récit en langage de nature doit être interprété en tant que spécification formelle.

Le BDD : Structure des procédés (Figure C.21) est conçu pour présenter un lien graphique entre les procédés formels et les composants physiques. Comme les procédés sont des modèles formels qui représentent le comportement des composants, la structure du procédé dépend de la structure des composants. Pour établir le lien entre les procédés et les composants (pièces), la sémantique des éléments du diagramme doit être identifiée comme suit :

(1) Pour les parties représentant les composants, les opérations / signaux sont présentés. En tant que lien, l'événement contrôlable et l'événement incontrôlable au fonctionnement de l'actionneur et du signal du capteur du composant physique sont construits sur la base de l'identification.

(2) Chaque composant en interaction avec le contrôleur doit avoir un modèle de procédé pour représenter le comportement. Cependant, un procédé est un modèle formel qui ne peut pas être présenté directement dans le diagramme. Un bloc nommé par chaque procédé et stéréotypé par «formel» est défini pour représenter les modèles de procédé dans le diagramme. Par conséquent, le procédé en bloc doit être affecté aux composants physiques correspondants, liés par la relation «Formaliser».

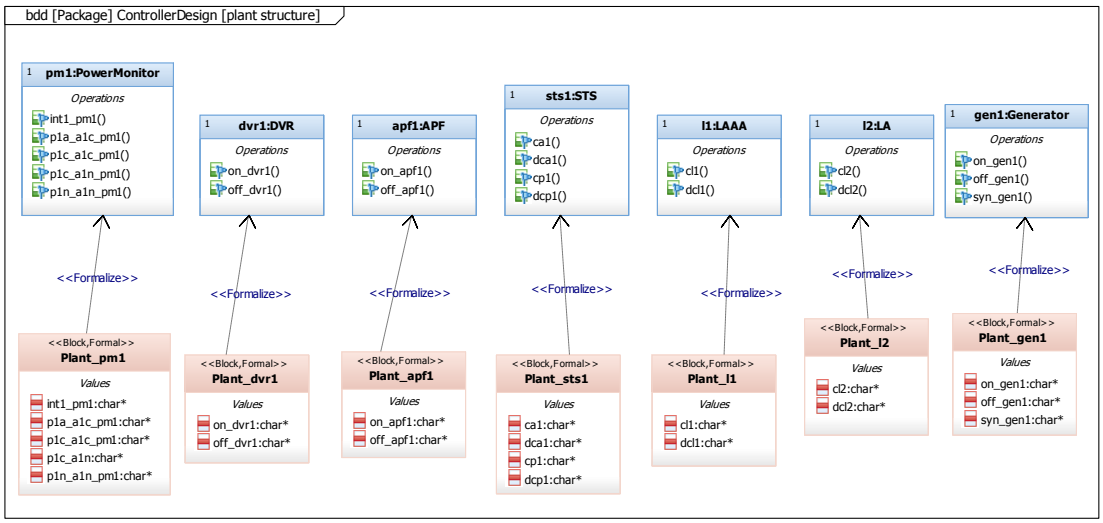


Figure C.21 BDD: Structure des procédés

Le REQ : Traçabilité des exigences (Figure C.22) présente le lien entre les exigences et les procédés et spécifications formelles. SysML prend en charge les relations entre les exigences et les autres modèles. Afin de présenter le lien entre les exigences et les modèles formels, la sémantique

des relations est définie comme suit :

(1) Pour représenter le lien entre les exigences et les blocs de spécification, la relation «formalize» est utilisée. Pour chaque exigence fondamentale, un bloc de spécification correspondant est lié. La sémantique de «formalize» dans ce diagramme indique que la spécification formelle est cohérente avec l'exigence.

(2) La relation entre le procédé et les exigences est présentée par «satisfaire» car le procédé est la représentation du comportement de la composante physique. La sémantique de «satisfy» dans ce diagramme est que le comportement représenté par un procédé formel est sous contrainte par l'exigence.

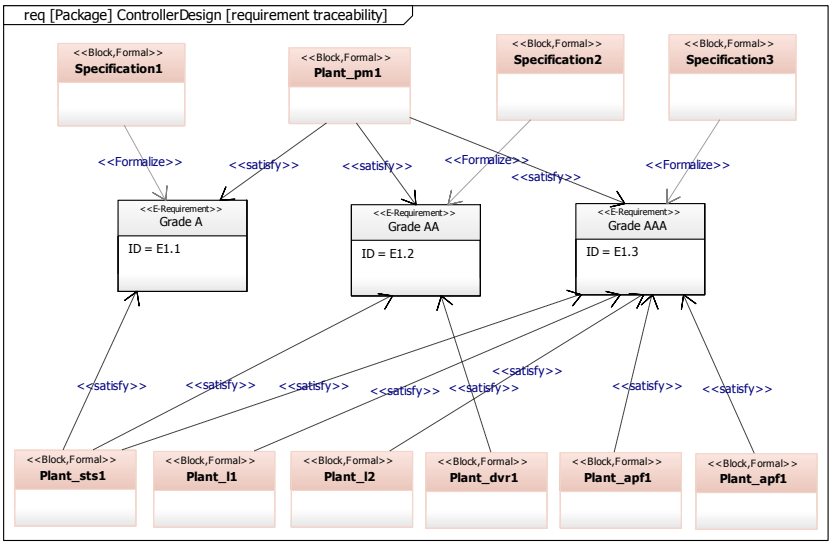


Figure C.22 REQ: Traçabilité des exigences

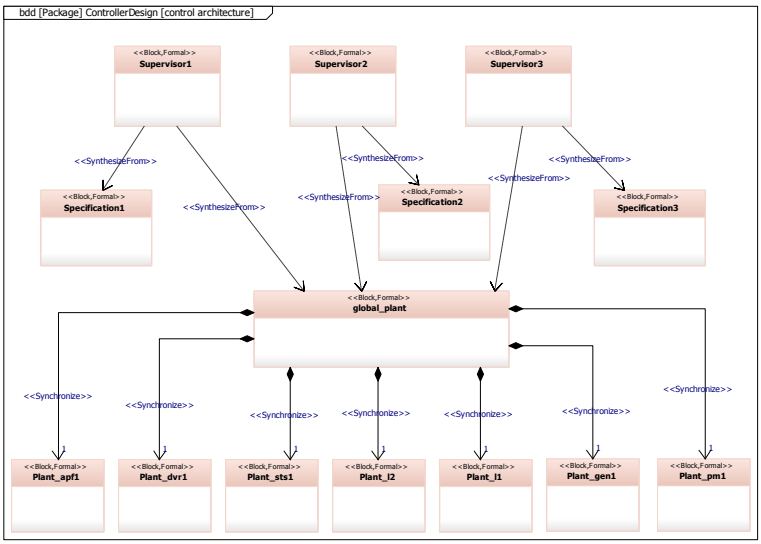


Figure C.23 BDD: architecture de contrôle

Dans cette étude de cas, nous utilisons une commande modulaire comme architecture de commande. Pour cette architecture, un procédé global doit être synthétisé par une synchronisation de tous les modèles de procédé. Chaque superviseur local est synthétisé par le procédé global et les spécifications locales correspondantes. Cette structure est présentée par BDD à Figure C.23. Nous définissons trois superviseurs locaux et la relation «SynthesizeFrom» et «Synchronize» est utilisée pour relier le superviseur local, la spécification locale et le procédé global. Les procédés globaux est agrégée par tous les procédés.

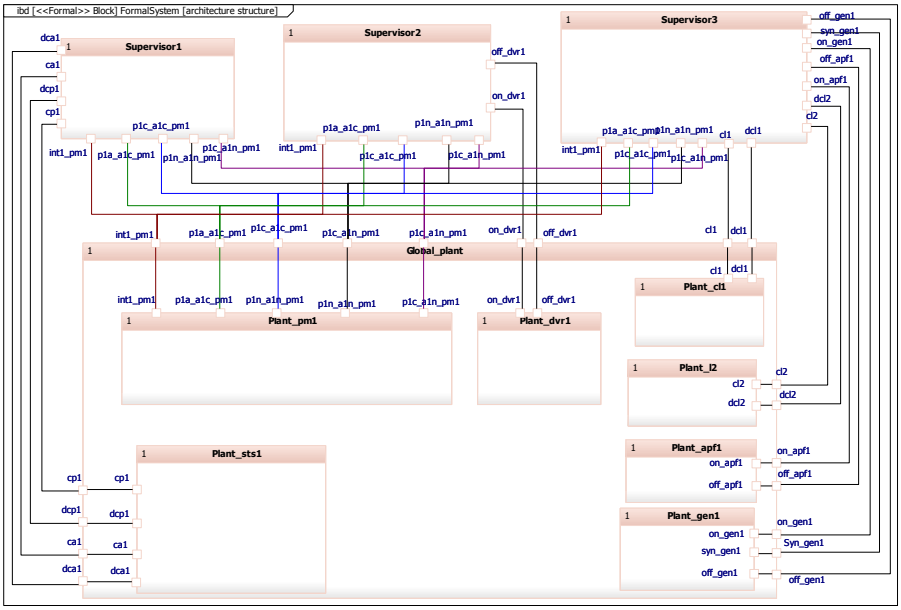


Figure C.24 IBD: structure d'architecture

IBD : La structure d'architecture est construite pour détailler la structure de l'architecture de contrôle (Figure C.24). Le procédé global sous contrôle modulaire est synchronisé par l'ensemble des procédés locaux. Par conséquent, tous les événements doivent être définis par les ports du bloc du procédé global. Deux types de connexion représentant la sémantique formelle sont construits : (1) la connexion entre des procédés locaux et des procédés globaux : la sémantique de cette connexion indique que l'ensemble d'événements globaux comprend l'ensemble d'événements de centrales locales; (2) la connexion entre les procédés globaux et les superviseurs globaux / locaux : la sémantique de cette connexion représente les ensembles d'événements observés et contrôlés par les superviseurs globaux / locaux.

Dans cette étude de cas, nous supposons que les superviseurs sont mis en œuvre par un automate monolithique en tant que contrôleur concret et que le programme de contrôle est attribué au matériel. 12 ports sont identifiés qui présentent chaque actionneur / capteur. P.0, P.1 et P.7-P.11 sont liées à une paire d'opérations commandées par le potentiel électrique élevé ou faible de la commande (c.-à-d. connecter au distributeur P et déconnecter de l'distributeur P sont implémentés par signal booléen de P. 0) Deuxièmement, les ports pour le contrôleur concret doivent être définis.

Comme nous utilisons un contrôleur monolithique concret pour implémenter tous les superviseurs locaux, le dispositif d'API doit contenir tous les ports correspondant aux superviseurs. Par notation arbitraire, nous utilisons les mêmes noms de port que les parties contrôlées (P.0 ~ P.12) pour une connexion pratique à l'étape suivante. Le schéma de mise en œuvre du contrôle est présenté à Figure C.25. Le dispositif d'API global est agrégé par le contrôleur concret et le programme de contrôle. La dépendance avec le stéréotype «allocate» présente la relation entre matériel et logiciel, par laquelle la sémantique selon laquelle les programmes de contrôle sont exécutés par le matériel peut être présentée.

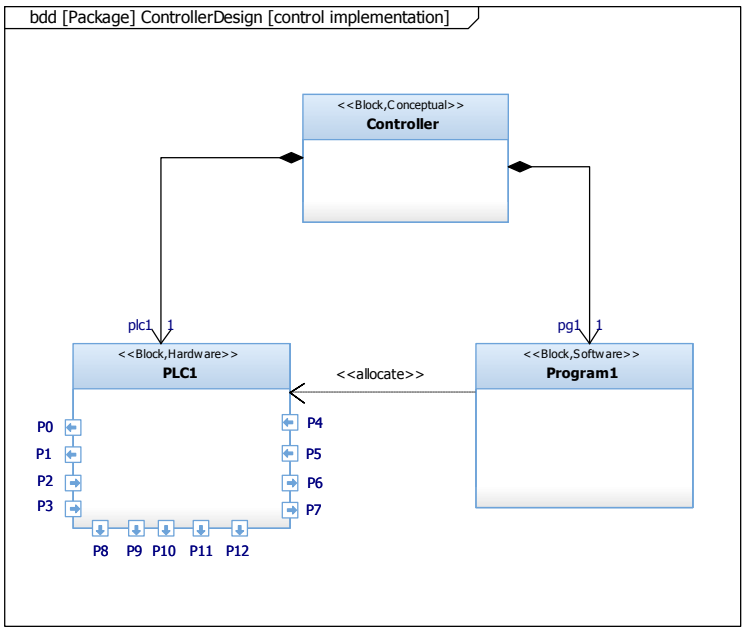


Figure C.25 BDD: Mise en œuvre du contrôle

Comme le contrôleur concret est implanté par un automate monolithique, tous les composants contrôlés doivent y être connectés. Deux éléments physiques sont présentés dans le diagramme : les composants physiques et le contrôleur concret. Comme nous désignons le même nom pour les ports correspondants du contrôleur concret et des composants, les connexions peuvent être directement construites. Figure C.26 montre le résultat de la modélisation de la structure de contrôle. Dans le diagramme, les connexions de commande sont présentées en bleu et les connexions du signal de censure sont présentées en rouge. Tous les composants sont connectés à plc1. Il convient de souligner que le contrôleur de bloc et le système à contrôler sont des modèles conceptuels. Par conséquent, il n'est pas nécessaire de définir les ports pour les deux. Afin de différencier la connexion de commande et la connexion de signal, le port doit suivre le principe suivant : pour chaque connexion de commande, le port des composants doit être entré et le port du contrôleur concret doit être généré. Pour chaque connexion de signal, le port des composants doit être la sortie et le port du contrôleur concret doit être entré.

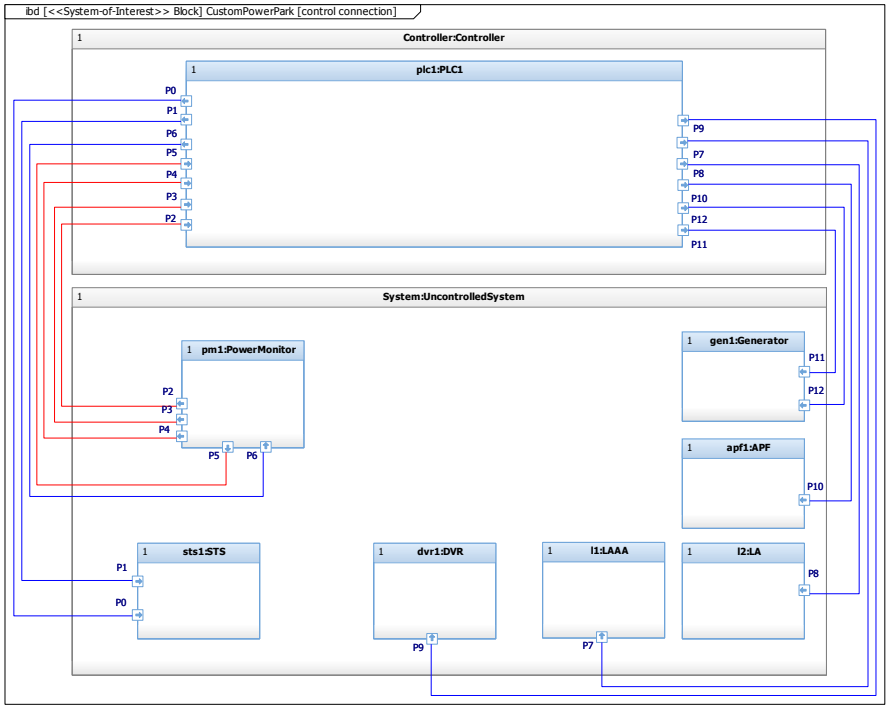


Figure C.26 IBD: Connexion de contrôle

C.8 Conclusion

Les approches formelles telles que la TCS se révèlent être de plus en plus importantes pour conception des contrôleurs lorsqu'elles sont confrontées au nouveau défi du système flexible ou du système complexe dans le domaine industriel. Étant donné que la TCS est trop rarement appliquée à l'AC dans la pratique de l'ingénierie, cette étude a tout d'abord examiné le processus de modélisation typique basé sur la TCS et a identifié quatre principales limitations : (1) Limitation 1 : manque d'interprétation du lien entre les exigences informelles / spécifications formelles, et entre comportements libres du système physique et des procédés formelles dans un processus de développement typique; (2) Limitation 2 : absence de modèles de structure (3) Limitation 3 : absence de modèles de mise en œuvre, et (4) Limitation 4 : absence de cadre global permettant de normaliser le processus de modélisation de l'AC pour une pratique en ingénierie. Bien qu'un certain nombre de méthodes et de contributions récentes soient proposées dans le cadre de la TCS, il existe encore des écarts entre les études universitaires et la pratique. Le processus MBSE est étudié comme solution possible pour gérer les limitations TCS. Cependant, la combinaison de MBSE et de méthodes formelles ne présente aucune solution directe dans le contexte du processus de développement d'AC basé sur TCS. En se concentrant sur ces questions, cette étude a proposé un nouveau cadre de modélisation pour l'AC en contribuant à l'intégration MBSE et TCS.

La première contribution de cette étude est la proposition d'un processus de modélisation globale.

Afin d'intégrer l'approche formelle dans le processus MBSE et de prendre en compte les solutions aux limitations du TCS, 13 points de vue sont identifiés pour être liés aux différents aspects de l'architecture globale: (1) pour la formalisation des procédés, le foyer est mis sur la structure des système physique par la perspective du niveau de contexte au niveau de composant; (2) pour la formalisation de la spécification, les exigences informelles et la traçabilité devraient être identifiées; (3) pour la synthèse et la mise en œuvre par le superviseur, nous nous intéressons principalement à la description du contrôleur au niveau formel / logique / physique. 10 modèles SysML proposés et 3 modèles formels sont définis pour refléter les points de vue correspondants. L'identification des points de vue construit de manière préliminaire le lien entre TCS et MBSE. Sur la base de la répartition des points de vue entre les trois phases principales de la modélisation MBSE (analyse des besoins, analyse des fonctions et synthèse de la conception), le processus de modélisation intégrée globale est développé. Dans cette étude, les détails du processus de modélisation proposé sont présentés. 14 activités principales sont définies avec le processus (4 activités en phase d'analyse des besoins, 4 activités en phase d'analyse des fonctions et 6 activités en phase de synthèse de la conception). Les cahiers des charges et les produits sont identifiées pour chaque activité. Le méta modèle est utilisé pour définir la syntaxe et la sémantique des modèles SysML adaptés au contexte de l'AC.

La première contribution de cette étude est la proposition d'un processus de modélisation globale. Afin d'intégrer l'approche formelle dans le processus MBSE et de prendre en compte les solutions aux limitations du TCS, 13 points de vue sont identifiés pour être liés aux différents aspects de l'architecture globale : (1) pour la formalisation des procédés, le foyer est mis sur la structure des système physique par la perspective du niveau de contexte au niveau de composant; (2) pour la formalisation de la spécification, les exigences informelles et la traçabilité devraient être identifiées; (3) pour la synthèse et la mise en œuvre par le superviseur, nous nous intéressons principalement à la description du contrôleur au niveau formel / logique / physique. 10 modèles SysML et 3 modèles formels sont définis pour refléter les points de vue correspondants. L'identification des points de vue construit de manière préliminaire le lien entre TCS et MBSE. Sur la base de la répartition des points de vue entre les trois phases principales de la modélisation MBSE (analyse des besoins, analyse des fonctions et synthèse de la conception), le processus de modélisation intégrée globale est développé. Dans cette étude, les détails du processus de modélisation proposé sont présentés. 14 activités principales sont définies avec le processus (4 activités en phase d'analyse des besoins, 4 activités en phase d'analyse des fonctions et 6 activités en phase de synthèse de la conception). Les cahiers des charges et les produits sont identifiées pour chaque activité. Le méta modèle est utilisé pour définir la syntaxe et la sémantique des modèles SysML adaptés au contexte de l'AC.

La deuxième contribution de cette étude concerne les méthodes permettant de lier les modèles SysML et les modèles formels. Les problèmes doivent être résolus lors de la formalisation du procédé et de la spécification. Différentes méthodes sont proposées afin de maintenir la cohérence entre les modèles SysML semi-formels et les procédés et spécifications formelles, respectivement. Afin de relier les procédés et les composants physiques, l'activité d'identification des procédés est détaillée. Une approche basée sur des modèles est proposée pour garantir que les résultats des procédés puissent refléter correctement les comportements des composants physiques et que le processus de modélisation puisse être amélioré. D'autre part, nous proposons E-exigence pour composer les exigences de haut niveau ambiguës-narratives en description explicite afin de permettre la traçabilité des spécifications formelles. Une méthode et un algorithme de vérification de la traçabilité sont proposés sur la base d'une intégration par logique temporelle et la cohérence entre exigences et spécifications peut être vérifiée.

Une étude de cas est présentée pour présenter l'application du processus de modélisation proposé, dans lequel une centrale de puissance personnalisé (CPP) est prise comme système cible. L'ensemble du processus de modélisation, depuis les besoins des parties prenantes jusqu'à la mise en œuvre concrète du contrôleur, est présenté en détail par les résultats des modèles SysML et des modèles formels correspondants. Nous discutons également des modèles pour d'autres situations et pour différents besoins de conception. En outre, l'approche basée sur des modèles est mise en œuvre par le développement d'un logiciel pour réaliser une analyse et une transformation automatiques de modèles BDD en modèles de procédé. L'étude de cas montre les avantages du cadre de modélisation proposé pour la pratique en ingénierie :

(1) Traçabilité : les modèles de procédé et les spécifications sont liés explicitement par les diagrammes SysML correspondants, tels que les BDD et les diagrammes des exigences. En outre, au niveau formel, le concept de synthèse du superviseur est décrit dans le diagramme SysML pour montrer le lien entre les modèles formels. Dans les BDD, le lien entre les procédés locaux et globaux (synchronisation), entre les procédés et les superviseurs et entre la spécification et les superviseurs est présenté (synthèse). Nous définissons la sémantique des concepts formels pour les éléments SysML. Par conséquent, les concepts tels que la synthèse et la synchronisation peuvent être explicitement visualisés. Pour présenter de manière explicite la relation entre superviseur et contrôleur, un autre BDD est intégré dans lequel les stratégies de superviseur, de contrôleur et de contrôle peuvent être présentées.

(2) Réutilisabilité : l'étude de cas et l'étude de situation supplémentaire ont prouvé que le cadre de modélisation global était réutilisable et adaptable aux pratiques d'ingénierie. La définition du méta modèle pour chaque diagramme SysML spécifie les éléments impliqués et leur sémantique, qui doit être instanciée dans l'application, et fournit des guides aux ingénieurs. Différentes

architectures de contrôle de AC peuvent être manipulées et modélisées par les diagrammes SysML proposés. L'approche basée sur des modèles, qui réalise une transformation automatique du système physique en modèles formels, améliore la réutilisabilité du cadre de modélisation car la technique simplifie le processus de modélisation et garantit la précision des modèles conformes à la pratique d'ingénierie.

(3) Prise en charge des outils : SysML et TCS peuvent être bien pris en charge par des outils de modélisation (IBM Rational Rhapsody et Supremica dans cette étude). Sur la base de la norme d'échange de métadonnées XMI, les informations entre différents outils peuvent être échangées. L'approche basée sur des modèles mise en œuvre par XMI montre les avantages de la prise en charge des outils dans le cadre proposé pour les pratiques d'ingénierie.

Sur la base des contributions à ce travail, les recherches futures peuvent être approfondies dans les directions suivantes :

Sur la base des contributions à ce travail, les recherches futures peuvent être approfondies dans les directions suivantes :

(1) Le lien entre les éléments formels et les éléments SysML (par exemple, des événements et des opérations) est identifié dans cette étude. Cependant, une présentation explicite du lien par les diagrammes SysML n'est pas obtenue. La raison principale en est que le lien entre les éléments formels et les éléments SysML est difficile à modéliser par SysML. Nous avions l'intention de modéliser par un IBD comme modèle de structure d'architecture. Cependant, le lien entre les éléments formels est réalisable, alors qu'IBD ne peut pas établir le lien entre deux types d'éléments différents, ce qui peut être considéré comme l'une des limites de SysML. Par conséquent, il demeure l'un des problèmes vacants de cette étude.

(2) Dans cette étude, nous présentons une approche basée sur des modèles basée sur la bibliothèque provisoire pour une étude de cas. Il s'agit d'une technologie prospective à étendre afin de réaliser un moyen efficace et précis de formalisation des procédés. À cette fin, il est nécessaire d'étudier sur un domaine industriel plutôt que sur une étude de cas afin de pouvoir paramétrer des composants physiques plus typiques. Deuxièmement, un problème qui subsiste dans cette étude est que la formalisation de la spécification dépend toujours de l'expertise bien que les contributions dans cette étude fournissent la méthode de vérification. Le modèle paramétré pour les spécifications pourrait constituer une autre orientation de recherche. En fait, dans [67], les auteurs ont déjà proposé le modèle de spécification paramétrée. Cependant, le modèle n'est ni trop simple ni assez général pour pouvoir être exploitable. Il est en effet difficile de paramétrer la spécification formelle dans la mesure où elle implique l'identification de nombres de

comportements et les solutions pour les intégrer selon différentes combinaisons. Une idée intéressante consisterait à normaliser la description narrative de l'exigence. Néanmoins, une fois que la solution à ce problème est proposée, le problème de formalisation peut être entièrement traité et la vérification des spécifications peut également être éliminée. Une autre proposition de formalisation des exigences consiste à développer le programme de contrôle de vérification basé sur l'activité AF1.2. Le but de cette proposition est d'éliminer le processus de vérification de la logique temporelle et un processus de vérification automatique est plus convivial pour la pratique de l'ingénierie.

(3) Dans cette étude, les modèles de stratégie de contrôle ne peuvent pas être globalisés par un méta modèle, aucune stratégie de contrôle standardisée ne pouvant être appliquée. En fait, la frontière entre la mise en œuvre des superviseurs, des contrôleurs et des contrôleurs est parfois ambiguë et, par conséquent, les ingénieurs implantent directement le superviseur par leurs propres méthodes en ignorant la synthèse des contrôleurs, ce qui conduit à l'absence d'étude systématique sur ce point. De nos jours, le contrôleur est désigné sous différentes formes telles qu'automate, algorithme ou stratégie narrative, en fonction des études. Il est très intéressant de représenter les stratégies de contrôle par modèles SysML dans l'étude future. Par exemple, par quel type de diagramme SysML peut-on utiliser pour présenter différentes stratégies de contrôle ? Comment définir les concepts de stratégie de contrôle par SysML ? Ce problème n'est encore pas résolu.

(4) En raison de la portée de l'étude de cas dans ce travail, deux types d'architectures d'implémentation de contrôle sont présentés (contrôleur monolithique et contrôleur modulaire). Le TCS fournit un certain nombre d'architectures de contrôle et d'implémentation. Il est intéressant d'étudier davantage de systèmes contrôlés pour montrer que le cadre proposé peut être appliqué à la modélisation d'architectures différentes (telles que le contrôle hiérarchique). En fait, c'est un défi de la mise en œuvre d'une autre architecture de contrôle. En fait, contrairement au contrôle modulaire qui a une implémentation systématique, le lien des superviseurs formels des autres architectures de contrôle et du contrôleur concret reste flou. Plus d'efforts sont nécessaires pour résoudre ce problème.

FOLIO ADMINISTRATIF

THESE DE L'UNIVERSITE DE LYON OPEREE AU SEIN DE L'INSA LYON

NOM : Lu

DATE de SOUTENANCE : 18/09/2019

Prénoms : Xiaoshan

TITRE : Contribution to the Integration of Supervisory Control Theory in a Model-Based System Engineering method

NATURE : Doctorat

Numéro d'ordre 2019LYSEI072

Ecole doctorale : Electronique Electrotechnique Automatique (EEA)

Spécialité : Automatique

RESUME :

Supervisory Control Theory (SCT) is one of the most important formal paradigms for developing controllers for Discrete Event Systems (DESs). The large number of scientific contributions shows that SCT catches extensive academic interest and this theory has been proved to be applicable in various industrial domains such as manufacturing systems, embedded systems, transportation systems and energy systems. With SCT, the requirements which are checked afterward in conventional engineering are used as input for generation of the design of the controller the verification of this model can be eliminated. However the SCT suffers from an important lack of integration in a global design process which leads to the gaps between the theoretical development and applications of SCT within engineering practice.
The Model-Based System Engineering (MBSE) provides the solutions to deal with the limitations of SCT. The objective of this study is to propose a novel framework for automatic control (AC) which integrates both SCT and MBSE to bridge the gaps formal paradigm and engineering process. In the proposed framework, different SysML diagrams are used to as complementary models which present the indispensible views of the system to be studied in the global modeling process. Secondly, in order to keep the consistency between SysML models and formal models, methods for formal modeling and verification are proposed. A case study introduced at the end prove that the proposed framework which provides a global development process from requirement analysis to controller implementation can well meet the needs of engineering practice.

MOTS-CLÉS : Supervisory control theory, model-based system engineering methods, SysML, formal approach, integrated modeling process

Laboratoire (s) de recherche : Ampère

Directeur de thèse: Éric Niel
Co-directeur de thèse : Laurent Piétrac

Président de jury :

Composition du jury : Pascal Berruet, Véronique Carré-Ménétrier, Éric Niel, Laurent Piétrac