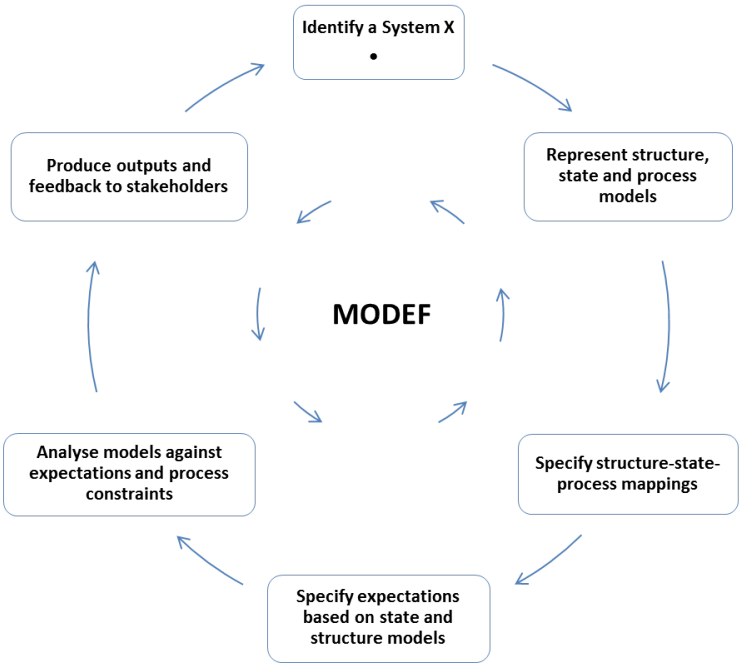
Model-based federation of systems of modelling

Freddy Kamdem Simo

par Freddy Kamdem Simo

Model-based federation of systems of modelling

Thèse présentée pour l'obtention du grade
de Docteur de l'UTC



Soutenue le : 26 Septembre 2017
Spécialité : Technologies de l'Information et des Systèmes,
Informatique: Unité de recherche HeuDiaSyc (UMR 7253)

Model-based federation of systems of modelling

Freddy Kamdem Simo

Thèse soutenue le 26 Septembre 2017 pour l'obtention du grade
de Docteur de l'UTC devant le jury composé de :

Rapporteurs:

Eric BONJOUR, Professeur des Universités, Université de Lorraine

Dominique LUZEAUX, PhD. HDR, Ecole Polytechnique

Examinateurs:

Isabelle BORNE (Présidente), Professeur des universités, Université de Bretagne-Sud

Mohamed SALLAK, PhD. HDR, Université de Technologie de Compiègne

Directeurs de Thèse:

Dominique ERNADOTE, PhD. MBSE Senior Expert chez Airbus Defence & Space

Dominique LENNE, Professeur des universités, Université de Technologie de Compiègne

---

Université de Technologie de Compiègne

Laboratoire Heudiasyc UMR CNRS 7253

LabEx MS2T

26 - 09 - 2017

Technologie de l'information et des systèmes, Informatique

Résumé

L'ingénierie des systèmes complexes et systèmes de systèmes conduit souvent à des activités de modélisation (MA) complexes. Les problèmes soulevés par les MA sont notamment : comprendre le contexte dans lequel elles sont exécutées, comprendre l'impact sur les cycles de vie des modèles qu'elles produisent, et finalement trouver une approche pour les maîtriser. L'objectif principal de cette thèse est d'élaborer une approche formelle pour adresser ce problème.

Dans cette thèse, après avoir étudié les travaux connexes en ingénierie système et plus spécifiquement ceux qui portent sur la co-ingénierie du système à faire (le produit) et du système pour faire (le projet), nous développons une méthodologie nommée MODEF pour traiter ce problème. MODEF consiste en: (1) Caractériser les MA comme un système et plus généralement une fédération de systèmes. (2) Construire de manière itérative une architecture de ce système via la modélisation du contenu conceptuel des modèles produits par MA et leur cycle de vie, les tâches réalisées au sein des MA et leurs effets sur ces cycles de vie. (3) Spécifier les attentes sur ces cycles de vie. (4) Analyser les modèles (des MA) — par rapport à ces attentes (et éventuellement les contraintes sur les tâches) pour vérifier jusqu'à quel point elles sont atteignables — via la synthèse des points (ou états) acceptables.

D'un point de vue pratique, l'exploitation des résultats de l'analyse permet de contrôler le déroulement des tâches de modélisation à partir de la mise en évidence de leur impact sur les modèles qu'elles produisent. En effet, cette exploitation fournit des données pertinentes sur la façon dont les MA se déroulent et se dérouleraient de bout en bout. A partir de ces informations, il est possible de prendre des mesures préventives ou correctives. Nous illustrons cela à l'aide de deux cas d'étude (le fonctionnement d'un supermarché et la modélisation de la couverture fonctionnelle d'un système).

D'un point de vue théorique, les sémantiques formelles des modèles des MA et le formalisme des attentes sont d'abord données. Ensuite, les algorithmes d'analyse et d'exploitation sont présentés. Cette approche est brièvement comparée avec des approches de vérification des modèles et de synthèse de systèmes.

Enfin, deux facilitateurs de la mise en oeuvre de MODEF sont présentés. Le premier est une implémentation modulaire des blocs de base de MODEF. Le second est une architecture fédérée (FA) des modèles visant à faciliter la réutilisation des modèles formels en pratique. La formalisation de FA est faite dans le cadre de la théorie des catégories. De ce fait, afin de construire un lien entre abstraction et implémentation, des structures de données et algorithmes de base sont proposés pour utiliser FA en pratique. Différentes perspectives sur les composantes de MODEF concluent ce travail.

Mots clés: Complexité, Ingénierie Système Dirigée par les Modèles; Modélisation interopérable; Architecture; Système-Exigences-Analyse; Modèles de structure, processus et états; Sémantique des modèles; Espace d'états; Formalisme Assomption/Préférence; Réutilisation des modèles; Théorie des Catégories.

Abstract

The engineering of complex systems and systems of systems often leads to complex modelling activities (MA). Some challenges exhibited by MA are: understanding the context where they are carried out and their impacts on the lifecycles of models they produce, and ultimately providing a support for mastering them. How to address these challenges with a formal approach is the central challenge of this thesis.

In this thesis, after discussing the related works from systems engineering in general and the co-engineering of the system to be made (product) and the system for make (project) systems specifically, we position and develop a methodology named MODEF, that aims to master the operation of MA. MODEF consists in: (1) characterizing MA as a system (and more globally as a federation of systems) in its own right; (2) iteratively architecting this system through: the modelling of the conceptual content of the models produced by MA and their life cycles, the tasks carried out within MA and their effects on these life cycles; (3) specifying the expectations over these life cycles and; (4) analysing models (of MA) against expectations (and possibly tasks constraints) – to check how far expectations are achievable – via the synthesis of the acceptable behaviours.

On a practical perspective, the exploitation of the results of the analysis allows figuring out what could happen with the modelling tasks and their impacts on the whole state of models they handle. We show on two case studies (the operation of a supermarket and the modelling of the functional coverage of a system) how this exploitation provides insightful data on how the system is end-to-end operated and how it can behave. Based on this information, it is possible to take some preventive or corrective actions on how the MA are carried out.

On the foundational perspective, the formal semantics of three kinds of involved models and the expectations formalism are first discussed. Then the analysis and exploitation algorithms are presented. Finally this approach is roughly compared with model checking and systems synthesis approaches.

Last but not least, two enablers whose first objectives are to ease the implementation of MODEF are presented. The first one is a modular implementation

of MODEF's buildings blocks. The second one is a federated architecture (FA) of models which aims to ease working with formal models in practice. Despite the fact that FA is formalised within the abstract framework of category theory, an attempt to bridge the gap between abstraction and implementation is sketched via some basic data structures and base algorithms. Several perspectives related to the different components of MODEF conclude this work.

Key Words: Complexity; Model-based Systems Engineering; Interoperable Modelling; Architecture; System-Requirements-Analysis; Structure, Process and State models; Semantics of models; State Space; Assumption/Preference formalism; Model reuse; Category Theory.

Remerciements

Ce travail a été réalisé et financé dans le cadre du LabEx MS2T (Laboratoire d'Excellence Maîtrise des Systèmes de Systèmes Technologiques) au sein du laboratoire HeuDiaSyc (Heuristique et Diagnostic des Systèmes Complexes) UMR CNRS 7253 à l'Université de Technologie de Compiègne (UTC) et de Airbus Defence & Space (ADS), MBSE Services. Je remercie de ce fait tous les personnels au sein de l'UTC et ADS qui ont, d'une certaine manière, rendu possible ce travail et contribué à son bon déroulement.

Je remercie mes directeurs de thèse Dominique Ernadote et Dominique Lenne pour la confiance qu'ils m'ont accordée pour réaliser ce travail, ensuite pour les discussions que nous avons eues, tous leurs conseils et regards critiques au cours de ce travail. Dominique Ernadote m'a accordé cette confiance depuis mon stage de fin d'études Ingénieur qui s'est déroulé au sein d'ADS et qui a conduit à ce travail.

Je remercie également Mohamed Sallak et Claude Moulin (à l'UTC) et David Spivak (au MIT) pour leur disponibilité, leur regard critique et les échanges que j'ai eus avec chacun d'entre eux au cours de ce travail.

Je remercie Messieurs Dominique Luzeaux et Eric Bonjour qui ont accepté de rapporter sur ce travail et pour toutes les suggestions qu'ils ont émises et les échanges que nous avons eus. Je remercie Madame Isabelle Borne qui a accepté de présider mon jury de soutenance.

Je remercie tous les membres du Jury (en particulier, mes rapporteurs) de m'avoir fait l'honneur d'évaluer ce travail.

Je remercie les collègues d'HeuDiaSyc et plus généralement de l'UTC et d'ADS avec qui j'ai passé des moments agréables et chaleureux.

Je remercie celles et ceux qui ont contribué (ou contribuent) favorablement à la construction et au fonctionnement des systèmes Humanité et Nature.

Acronyms

AM	Structure models
C	Constraints on processes
CPM	Critical Path Method
CT	Category Theory
DSM	Design Structure Matrix
EVM	Earned Value Method
FA	Federated Architecture (the proposed architecture of models)
FMI	Functional Mockup Interface
HFSM	Hierarchical Finite State Machine
M	Models produced by the Modelling Activity
MA	Modelling Activity
MBSE	Model-Based Systems Engineering
MG	Mappings
MODEF	Model-based Federation of systems of modelling
OPM	Object-Process Methodology
PA	Programmatic Activity
PERT	Program Evaluation and Reviewing Technique
PM	Process models
R	Requirements
SD	System Dynamics
SE	Systems Engineering
SEMP	SE Management Plan
SEMS	SE Master Shcedule
SM	State models
SMC	Symmetric Monoidal Category
SMuC	Symmetric MultiCategory
SOI	System-Of-Interest
SoM	System of Modelling

SoS	System of Systems
SoSoM	System of Systems of Modelling
SS	State Space
SSG	State Space Graph
TA	Technical Activity
TP	Technical Processes
TMP	Technical Management Processes
WBS	Work Breadown Structure
WD	Wiring Diagrams

List of Figures

Contents

Introduction

1.1 Complex systems and complex modelling activity

The social, scientific and technological development has led human beings to the construction of complex systems. These systems are complex in the sense that they are difficult to describe, design, validate, implement, operate, repair etc. Arguably, there are no fixed, exhaustive and *a priori* ways to master them. Besides, their goals, the size and heterogeneity of their components and their components' relations and interactions are increasingly growing.

A central approach to master the engineering of complex systems is the modelling activity. A model is a means that enables an understanding of something for some goal. Such an understanding, hence the associated model, is generally a partial, sometimes incomplete, view of the actual modelled thing. One old yet topical challenge is the assembly of heterogeneous models for a global purpose (e.g., verification). Nonetheless, with respect to the reality they precisely represent, models make it possible to understand, analyse, optimize and operate complex systems. As a consequence, models are means to preserve, share and reuse knowledge about the thing they relate to. What about the process that operates the models?

Today, models produced by the modelling activities are spread over different locations within one or several companies. Due to the complexity of engineered systems, they often remain specific to a domain of study (hardware architecture, software, electrical engineering, mechanics, etc.) that addresses a particular view of the system. This means they are also of different kinds.

At the same time, there are several projects and programs within most large engineering companies. They evolve in parallel with different modelling activities' life cycles. The produced models are taken into account on various periods. These periods extend on a scale of days to several months. In this context, the modelling

activities are in turn complex.

It turns out that there are two main levels of complexity: one associated to the engineered systems and the other relating to all entities and practices that contribute to the modelling of those systems.

In spite of the complexity of the engineered systems, once the engineering process is carried out by multiple autonomous stakeholders and (sub-)projects evolving in parallel, with different time-scales, there is a need to ensure that the modelling activity and subsequent activities and the resulting models are rightly operated. In fact, this is a necessary condition to ensure that the projects and programs will succeed. Some challenging questions which inevitably arise in practice here are:

(1) How to better understand and use models in this context? i.e. what models are present in a particular location and what upon they stand for?

(2) How to analyse and identify the impact of their changes? i.e. what is the current state of models and in which states are they likely to end up? What does a state means for the project and program?

(3) How to help in mastering their evolution? i.e what is necessary to guarantee that models might reach some expected states?

The central question here is how to to formally reason on the MA to answer the above three issues.

1.2 Towards mastering such a complexity

This dissertation addresses these issues through the proposal of a methodology that mainly consists in:

(a) **Characterizing** the modelling as a *federation* of Systems of Modelling–SoM: A System of Modelling–SoM is an autonomous system where the components are the people, processes, methods, tools and models resulting from the interaction of these components. Roughly, a federation is a form of organisation where the components have a certain capability and autonomy to address some problems. Additionally, these components together define interoperation mechanisms for reciprocal exchanges. Federation is useful in order to characterise the modelling activity in such a way that the autonomy and capabilities of stakeholders are explicitly recognized and observed. We named such a federation a System of System of Modelling–SoSoM.

It is at the level of the SoSoM that the interrelationships between several SoM are studied. Fundamentally, the challenge addressed is: how to see and characterize MA?

(b) **Modelling** a SoM and SoSoM via structure state and process models and their mappings and **Specifying** the requirements that these models should satisfy: Fundamentally, the models of an SoM (resp. SoSoM) answer the question: What are the adequate levels of abstraction to study the modelling activity (resp. the relationships between SoM)? What are the appropriate models?

In this thesis the models (M) produced by the modelling activity are abstracted at a conceptual level. We will argue on the benefits expected from conceptual models as a means to abstract away the main content of models. We call such models, structure models. The state models are used to describe the life cycles of M and the transitions between the states during the life cycle. Finally it is the process models that capture the modelling tasks that bring about the changes of states of M.

Once the models of a SoM (resp. SoSoM) are described, we define the requirements expected from the SoM (resp. SoSoM) as expectations on the states of M. The expectations are expressions relative to the appreciations of the possible states of M, e.g., this state is preferred to that one when another state appears. The idea is that, an expectation is the expression of a preference over some states of M in a specified context.

(c) **Analysing** the models of a SoM (resp. SoSoM) with respect to its expected properties and **Providing** stakeholders with some exploitations of analysis' results. Fundamentally, the challenge here is twofold: how to reason on the models against the requirements and what outputs should be given to the stakeholders?
The analysis algorithms are proposed to explore how far expectations are achievable against structure, state and process models and their mappings. Such algorithms are means to demonstrate the relevance or not of the modelling tasks, i.e. whether some problem may occur or not. The exploitation algorithms provide stakeholders with data that reveal such a demonstration regarding the SoM or SoSoM.

As a result, the proposed methodology aims to support the ability: to (1) understand the current global state of the modelling activity, (2) check whether the modelling activity is going toward a good direction, (3) assist the stakeholders into the building of processes ensuring continuous appropriate changes of the models

they produce.

Therefore, we are not explicitly dealing with the internal practices of the modelling activity. That is the design techniques, methods and tools involved in the modelling activities are not explicit i.e. modelled. We deal with the conceptual contents of M produced by the modelling activity. The internal practices are black boxes for the proposed methodology. Even though the internal practices of the modelling activity are relevant and unavoidable to obtain M, within the context mentioned above (Section 1.1), the successive processes and their relevance for going towards a successful state of M are necessary and mandatory from the foregoing. This latter concern is formally addressed by the proposed six-step methodology: **MODEF** which is a methodology for model-based federation of systems of modelling. The steps of the procedural part of MODEF are summarized on Figure 1.1. These steps are not intended to be always applied altogether and/or sequentially. At a given step, it is possible to go backwards to another step. A return to another step for its re-execution, can be justified by a problem detected at the level of one of the six main steps of MODEF.

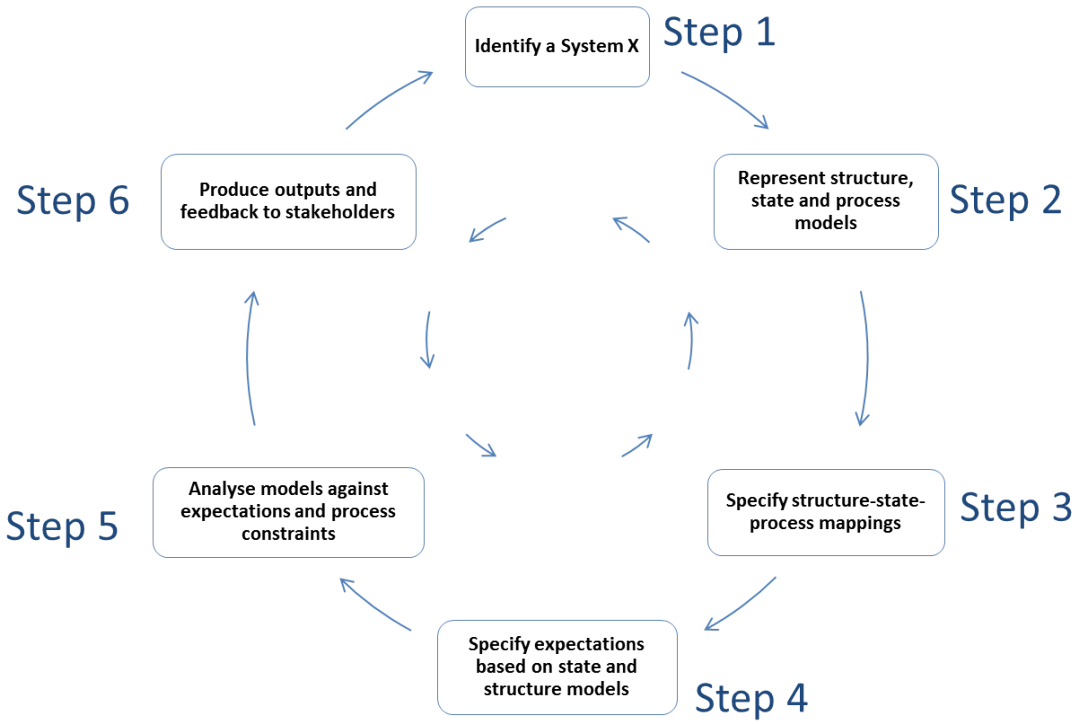


Figure 1.1: The procedural structure of MODEF

1.3 Contributions

In order to demonstrate the soundness and feasibility of the proposed methodology, we argue on both the theoretical and practical elements related to MODEF. Generally, for each of the 6 main steps of MODEF, where applicable, we present the activity carried out at this step, methods that support the carrying out of the activity and the implementation (not prescriptive) of methods in practice.

The proposed methodology is not limited to the specific concern addressed in this thesis. In fact, any system that can be understood and modelled with the input models of the methodology, may benefit from its outputs. In this perspective, MODEF might be applicable for the model-driven techniques i.e., internal practices of the modelling activity. Nonetheless, this is not the central objective of this thesis.

After reviewing the relevant related work and following the components of MODEF, the main contributions of this thesis, which to the best of our knowledge are not published elsewhere (apart from published ones) are:

Characterisation of the Modelling Activity: A System of Modelling (SoM) and a System of Systems of Modelling (SoSoM)
We characterise the modelling activity (MA) in a new way: a federation of several SoM giving rise to a SoSoM. The SoSoM is a means to understand the evolving relationships between different SoM. These SoM have a certain autonomy and capabilities. But agreements on interoperation mechanisms are required to get an added-value at the federation level. In order to formally understand and study the SoM and SoSoM, we then consider to model them.

Modelling the Modelling Activity: structure, state and process models and their mappings
We newly use the structure, state and process models together with their mappings to support a modelling of the architecture of MA from the perspective of a SoM and SoSoM:

a) The structure models describe the structure of the studied elements. For the SoM, the studied elements refer to the data models (resp. the modelling objectives) that describe the main content (resp. expected goals of the modelling) of the models (M) produced by MA. The studied elements at the federation level are typically about the interrelationships of studied elements within several SoM.

b) The finite-state models describe the lifecycle of studied elements via their

states and transitions between these states.

c) The process models represent the modelling tasks. Some events generated by the execution of those tasks are intended to trigger the expected transitions between the states of studied elements. For this connection to exist between state models and process models, we define the mappings from events of the latter to transitions of the former.

In order to analyse models, it is also important to specify the requirements or properties they should guarantee or satisfy.

Specification of expectations (or requirements) of MA: Assume/Preference formalism inspired by Assume/Guarantee contracts

An expectation consists in expressing preferences on the life cycle of a studied elements given some context or assumption. Especially, those preferences are defined on the states of the studied elements given an assumption (A). In the style of Assume(A)/Guarantee(G) contracts, the guarantee of a contract is transformed in preferences (P) by giving to P a pre-order structure. A pre-order structure is general enough to describe preferences among the elements of a given set.
Both A and G (P) are defined on a domain (D) in which an element is a couple: (*object, state*) meaning *object* is in state *state*. An assumption is theoretically a formula of zeroth-order logic where atomic propositions are elements of D. Their use is also motivated by the fact that they enable consistency and compatibility verification.

In the spirit, the inspiration from contracts here is close to their first uses in programming where pre-conditions/assumptions and post-conditions/guarantees are defined for programs (system models here). In this thesis the expectations are exploited in the analysis procedure to synthesise the behaviours of the studied system.

(Re) analysis of models and exploitation of analysis' outputs

The analysis mainly consists in exploring what would be the behaviours of the system and in verifying how far the expectations from the system are achievable with respect to the models. It is a support for providing the effective means for a better (in relation to the preferences defined by expectations) execution of MA. Since MA is an iterative and evolutive process, i.e. subject to continuous changes, the analysis routine is intended to be executed whenever necessary.

We introduce a synthesis procedure which applies on-the-fly the uniform-cost-search (UCS) algorithm on the state space described by the co-exploration of state models of the studied elements and the process models both constrained by the mappings. The novelty of this procedure lies in the fact that it exploits the expectations (and potentially constraints on processes) in UCS to guide the co-exploration throughout the discovered state space and eventually prune some of its regions. Moreover it is possible to replace UCS by another search algorithm or tuning it by introducing a heuristics.

To make the results of analysis understandable by a human, some exploitation algorithms are provided to build synthetic data. For example, it shall be possible to figure out i) whether from a given configuration (combination of active states and processes or tasks) of the system, it will be no longer improvable (in the sense of expectations and process constraints) given the input models, ii) the successive configurations necessary to reach a target configuration, iii) critical paths, etc. Such data will therefore help the stakeholders to operate the system, prevent issues and take corrective actions. The analysis and exploitation are demonstrated on two case studies.

Dealing with models in practice: A federated architecture for external exploitation of models coming from modelling tools

We attempt to automate the use (especially for the analysis purposes) of models coming from a given tool. This yields us to define, based on the framework of category theory, an architecture for models so that they are easily exploitable outside the modelling tool. The devised architecture separates and naturally maps the structure of models and their semantics and the usages (or instances) of the structure such that any semantics of models can be operationally implemented for exploitation purposes. This is particularly important if we are about to consider another tool for graphically building models, then exploting them outside that tool. All models are basically considered as composite structures a basic way to deal with heterogeneity and complexity.

Implementation based on a modular architecture

MODEF has been set up with the support of a modelling tool: MEGA (see mega.com), and a prototype implemented with the Scala programming language. The modelling tool support the ability to execute Step 2, Step 3 and Step 4 of MODEF, whereas the prototype allows via its modules, to implement Step 5 and Step 6.

The choice of the modelling tool and the prototype API are not prescriptive insofar another tool and an implementation could be considered using the specifications, models and algorithms corresponding to the steps of MODEF, all presented in this thesis. The setting up of MODEF demonstrates its implementation feasibility. The originality and relevance of the implementation come from the fact that: i) we do not introduce a new modelling tool or language ii) the definition of the exploration semantics of models is explicitly implemented and typically has to be available as code or library.

Despite the theoretical exponential complexity of the main algorithm involved in MODEF, we shall argue on its advantages. Therefore, the implementation architecture and the main algorithm's performance are used as the factors to evaluate the feasibility of MODEF .

Many applications of MODEF in real *situ* will be necessary to relevantly discuss its usability i.e. its effectiveness, efficiency, and satisfaction against the stakeholders and the objectives MODEF addresses. A first step towards the demonstration of the usability of MODEF has been the application examples and two case studies considered in this thesis.

1.4 Organisation of this document

The logical organisation of this dissertation is depicted on Figure 1.2. Each box on that figure represents a section of this document and the main question it answers. The arrows indicate the logical sequencing of sections i.e. the source of an arrow supplies the target of the same arrow. The rest of this thesis is organised as follows.

In Section 2, we discuss the relevant related work that addresses the mastering of complex modelling activity. We first focus on Systems Engineering body of knowledge then specific approaches that can be compared with MODEF. We conclude this section by positioning MODEF. Specific related research is also discussed in different sections.

In Section 3, the SoM and SoSoM are presented. We first argue on the characterization of MA as SoM and SoSoM. Then, some application examples that illustrate the SoM and SoSoM are presented. The application examples are used in the following sections to illustrate the concepts.

In Section 4, the principles and semantics of three kinds of models (structure, process, state) and their mappings, considered to model the SoM and SoSoM, are first introduced. Then, the formalism that supports the specification of the

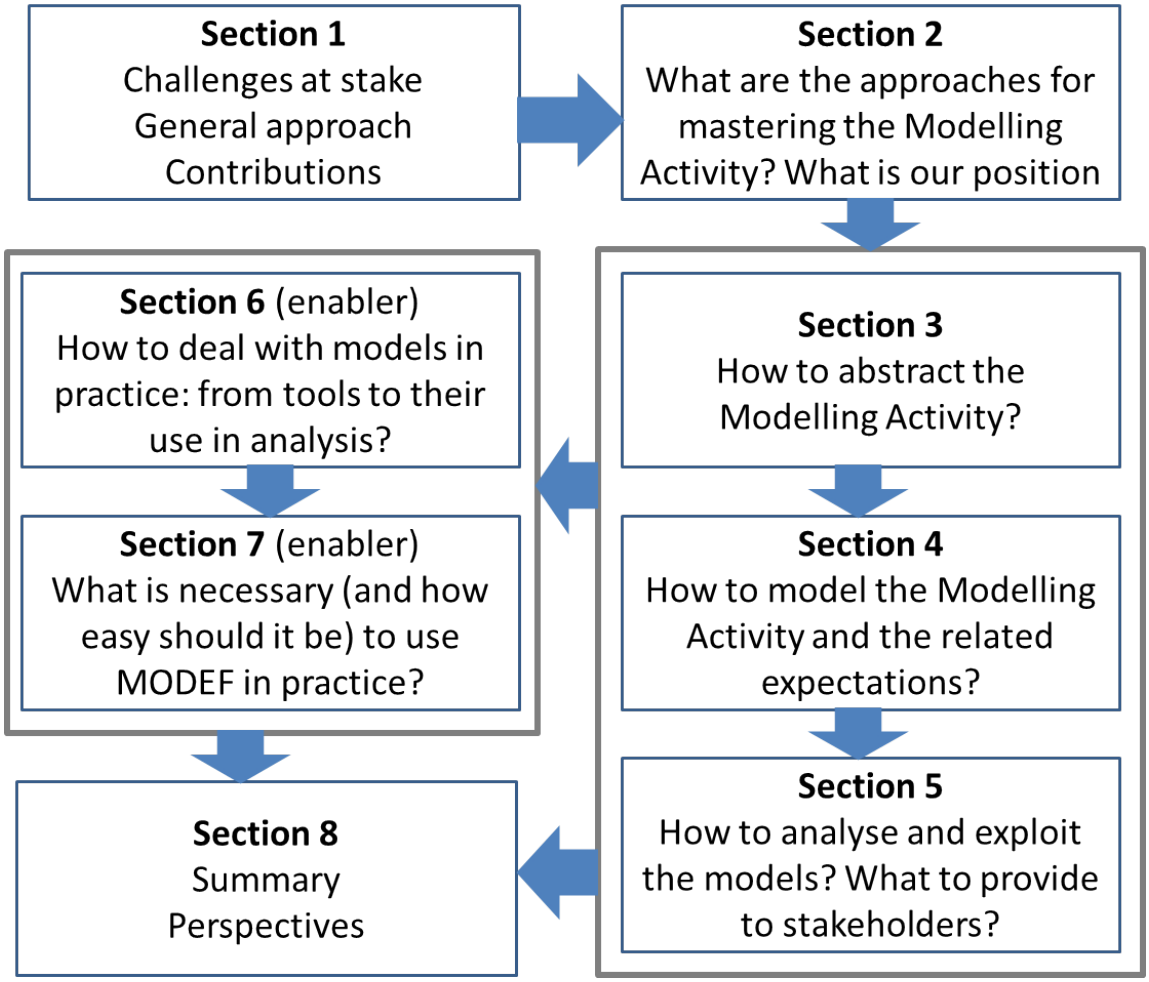


Figure 1.2: The organisation of this document

expectations that models should ensure is described.

In Section 5, the analysis procedure to exploring and verifying models against the expectations (and potentially constraints of process models) is discussed. Then an exploitation of the analysis procedure is presented following by two case studies.

Sections 6 and 7 can be seen as the enablers of MODEF.

In Section 6, the devised architecture that allows to deal (for analysis purposes) with models in practice is discussed. We present the components of this architecture and their relations, a categorical definition that we motivate and the data structures to take advantage of this architecture in practice.

In Section 7, the implementation architecture and its qualities are presented. We present the components (that are not prescriptive) involved to setting up MODEF in practice. Then some qualities that could be considered in an evaluation or comparison of MODEF are presented.

In Section 8, we finally present the conclusions and perspectives of this work.

Approaches for mastering the Modelling Activity

In this section, we discuss the relevant related work that addresses the mastering of complex modelling activities. We first focus on Systems Engineering body of knowledge (Section 2.1) then model-based approaches (Section 2.2) that can be compared with MODEF. The specific related research related to the different components of MODEF is also discussed in the other sections.

2.1 Systems Engineering

2.1.1 Overview

Systems Engineering—SE has been considered for nearly eighty years (see e.g., [52] for the evolution and definitions of SE) to manage the complexity of both the engineered systems and their engineering process.

A system is a set of interrelated elements that accomplishes a function within an environment. A system is also (ISO/IEC/IEEE 15288 [85]) "a combination of interacting elements organized to achieve one or more stated purposes."

SE has been building on two significant disciplines [64]: "the technical knowledge domain in which the systems engineer operates, and systems engineering management." The former is generally related to technical activities (requirement engineering, architecting, design, etc.) where as the latter is concerned with the planning and management of the former. Note that, SE management is different yet complementary to general project management (and to a large extent business processes) since it focuses on technical and engineering aspects [85] [82] [11].

It can also be said that SE is concerned with two main systems: the engineered, developed or studied system and the system that enables the engineering of the studied system. These two systems are also called the "system to be made" and the

"system for making" respectively [40]. Since a system is by definition located within an environment, it is also surrounded by other systems that may contribute to its existence. The "system to be made" also refers to the System-Of-Interest—SOI.

As a result, it is worth emphasizing here that SE is mainly built on technical and programmatic interrelated activities. Basically, the former answers the question: are we rightly engineering the right SOI? And the latter answers the question: are we rightly engineering and operating the right "system for making"? The programmatic activities might generally refers to SE management.

Since the outputs directly correlated to the SOI in the engineering process are produced by the technical activities (TA), they are unavoidable. Obviously, without TA no tangible artefact related to the SOI is going to be produced. The programmatic activities (PA) are useful for mastering the development of complex TA. PA are unavoidable and critical if the engineering is spread over a long period (from a scale of weeks or years), and, is composed of several autonomous and distributed stakeholders, as it is the case for many large engineering programs. By programs we mean a set of projects contributing to the engineering or development of (a) complex system(s), (see [40] for more details on programs). In this situation or in presence of complex TA, successful PA is largely, if not inevitably, a necessary condition to the success of TA. Likewise, PA aims to guarantee and enforce that the engineering of an SOI could eventually succeed or not. Finally, PA and TA influence mutually.

SE is a well-documented discipline and practice by the means of standards and handbooks [52] [64] [85] [82] [40] [11]. One approach to efficiently deal with SE has been the use of models and modelling instead of documents as the main support of the SE application. This does not mean that documents are no longer used, they are rather generated from models. Such an approach refers to Model-Based SE—MBSE [52] and to a large extent model-driven engineeering—MDE [101] [10] [84] [60]. The efficiency comes from the ability of models and modelling to improve communications among the stakeholders of a SE approach, to improve the representations of systems and to enhance preservation, share and reuse of knowledge related to the SE processes. This means that models and modelling are central in an MBSE approach.

2.1.2 SE and the Modelling Activity

Now, we analyse how SE aims to support the mastering of MA, a core activity in MBSE.

One of the main standard related to SE is ISO/IEC/IEEE 15288 [52] which defines the SE processes and life cycles. The four main types of processes identified in ISO/IEC/IEEE 15288:2015 are: Agreement processes, Organizational project-enabling processes, Technical processes and Technical management processes. Of all of these processes, 14 are technical and 16 are non-technical (in the sense that, they are not TA). For each type of process, Purpose, Description, Inputs/Outputs, Process activities and Process elaboration have been identified [52]. This identification is a way to perform the SE process. Typically, TA and PA are mainly located within TPs and TMPs respectively.

Technical processes are used to design the system and consist of several processes from the business analysis process to the disposal process.

Technical management processes are [52]: "... used to establish and evolve plans, to execute plans, to assess actual achievement and progress against the plans and to control the execution through to fulfilment."

Agreement processes are upstream and downstream processes for acquiring (by an acquirer) a system and setting up the contracts and environment within which TPs, TMPs, and OEPs will be carried out by the supplier.

Organizational project-enabling processes deal with organisational (infrastructures and resources) aspects related to the support of other processes throughout the system's life cycle.

To accommodate to a particular situation or circumstance, the Tailoring process enables the selection of relevant processes (generally artefacts) or parts thereof for their application such that too much unnecessary (perhaps formal) processes are avoided [52].

The results of Technical management processes are specified in the SE Management Plan (SEMP). The SEMP is a top-level plan and outcome in planning, organising, and controlling SE activities [52]. Examples of entries of the SEMP are: the SE Master Schedule (SEMS) that describes the overall project schedule, the Work Breakdown Structure (WBS) that contains the hierarchy of all work packages and associated SE processes and their control, for the project.

Among the Technical management processes, there is a Measurement process which is a process of elicitation and provision of data. It enables to figuring out

and demonstrating the quality of products, services and processes involved during the engineering of a system [52] ISO/IEC/IEEE 15288. Therefore, a measurement would be typically tailored for a given need, the problem at stake and the available data. There are several approaches and standards that deal with the measurement process. One standard is ISO/IEC/IEEE 15939 Systems and software engineering—Measurement process. One approach is the SE leading indicators [77].

In [77], 13 leading indicators for evaluating (via their implementation, analysis and interpretation) the effectiveness of SE activities are discussed. "[...] leading indicators are intended to provide insight into the probable future state, allowing projects to improve the management and performance of complex programs before problems arise." [77] As a result, leading indicators are predictive by nature. They are based on historical trend data. As indicated in [77], the indicators have to be used appropriately, for instance regarding their scope and the frequency of measures to avoid pitfalls in using leading indicators and measures in general. The authors finally argue that trend data are neither always easily available nor stored in central repository. As argued in [52] , a measure itself does not improve process performance. Even a model like CMMI [94] which is useful for process improvement, states what is necessary to reach a level of maturity, but not how to implement it.

Some other useful standards, materials [58], [82] and models related to SE and particularly PA are described in [52]. The standard IEEE Std 1220-2005 (ISO/IEC 26702) provides guidances on the application and management of a SEMP; the standards EIA/IS 731.1 (Systems Engineering Capability Model), ISO/IEC 15504 (Information Technology - Process Assessment) and the model CMMI (Capability Maturity Model Integration) [94] provide guidances in SE processes implementation, assessment and improvement. The standards ANSI/EIA 632 (Processes for Engineering a System) and ISO/IEC/IEEE 42010 (Architecture Description) deal with the description of artefacts related to processes and systems.

To summarize the foregoing, ISO/IEC/IEEE 15288 aims to support SE process via a description of processes involved in a SE approach. The SEMP captures the artefacts (SEMS, WBS) necessary for mastering such processes via (their planning and control). Other standards (ANSI/EIA-632, ISO/IEC 15504:2004, IEEE Std 1220-2005, etc.) provide guidance for dealing (i.e., describing, improving, assessing) with such processes. It follows that the SE body of knowledge does contribute to the mastering of the SE processes regardless of how they are actually implemented.

Although the aforementioned materials and standards are clearly useful, they

often remain documents and generally deal with rules and best practices. They generally address the What (description) To Make but not the How (prescription) to Make the What. Even though the Tailoring process allows to adapt to a particular situation, it nonetheless does not provide the "How to Make" the selected artefacts; furthermore it is not intended to that. As a consequence, they are not sufficient in the perspective of a model-based mastering of MA. In fact, in an attempt to solely and directly use them, one is going to face the problems encountered with document-based SE. Those problems are: analysis, consistency, traceability, sharing, reuse etc. Moreover, the diversity and autonomy of different stakeholders of the MA might prevent such an attempt and simply might not make it possible. And if this happens, it would likely lead to uncontrolled, non-optimal, endless and unsuccessful programs. Besides, in this situation, even with models, one may encounter the same problems as those encountered with documents. Nonetheless models offer several advantages (reuse, share, analysis etc.) over documents if they are well operated. By considering PA as a system or the "system for making", SE (MBSE) is applicable for its engineering and exploitation [40].

Model-based techniques have been advocated and introduced to engineer the programmatic aspect of the SE process [87] [44] [100] [42]. We come back to the latter in Section 2.2.

While over the years SE has been documented, many times successfully applied, particularly in defence and aerospace industry, today it is still based on heuristics and informal practices as argued in [53]. Despite the evolution of SE and its rich available body of knowledge, there have been many large engineering projects which failed: see e.g., in [7] for a list of past projects, this work concludes failures are to be attributed to the complexity of the projects themselves; see also the story in [43] of some recent failed projects based on a waterfall lifecycle where agile approach was called for help subsequently.

Meanwhile, it is expected that SE application will expand across other industries [53]. For this expansion and the future of SE to be successful, formal supports of SE process are necessary by taking advantage of the current SE's body of knowledge. In this dissertation, PA (in tandem with TA) are considered as a proper system that needs to be engineered with the support of models.

2.2 Model-based mastering of the Modelling Activity

In this section we start by presenting some relevant model-based approaches to the mastering of a modelling activity then we conclude by given the position of MODEF with respect to these approaches.

2.2.1 The OPM approach

In [87], the tandem (project-product dimension) composed of the project (the programmatic aspect) and the product (the technical aspect) and their specificity is firstly emphasised. The authors compare the methods of project management that are common in SE management together with the Object-Process methodology OPM [30] used for project planning and product modelling and design. Some of these methods are: Earned Value Method (EVM) for project control, Critical Path Method (CPM), Program Evaluation and Reviewing Technique (PERT) and Gantt Chart for project planning and scheduling, System Dynamics (SD) for project planning and dynamic modelling, and Design Structure Matrix (DSM) for project planing and product design.

Using a simplified unmanned aerial vehicle (UAV) as a system use case, it turns out from their empirical comparaison that some methods are relevant with respect to particular product and project factors. Some factors are: Budget/Schedule measurement/tracking, Stakeholders/agents tracking, Performance quality, Product measurement/tracking. Moreover only SD, DSM and OPM methods were found to handle the project-product dimension. SD is a way to correlate factors (schedule, budget) related to project planning in way that can be plotted. DSM represents the interactions among elements (components, tasks, teams) of both the project and product. They finally conclude that OPM is the only suited for the function-structure-behaviour modelling (i.e. Project-Product Model-Based [88]) of both the project and the product inside an integrated conceptual model.

All the compared methods except OPM ought to be derived from models that represent the product and project, since they address a particular and specific concern. The OPM approach is required for both TA and PA. The authors claim that such a choice is particularly suitable to combine PA and TA within the single and same foundation: OPM. It follows from this choice that the structure, function and behaviour OPM models respectively, describe the structure the function and

behaviour of both the product and the project. The structure models together with their states are the possible inputs and outputs of the project's processes. The OPM model of the simplified UAV use case is depicted on Figure 2.1. On Figure 2.1 is an Object-Process Diagram, a graphical representation of an OPM model. Ellipses and rectangles correspond to processes and deliverables. The links correspond to whole-part (black triangles), and characterization like relations.

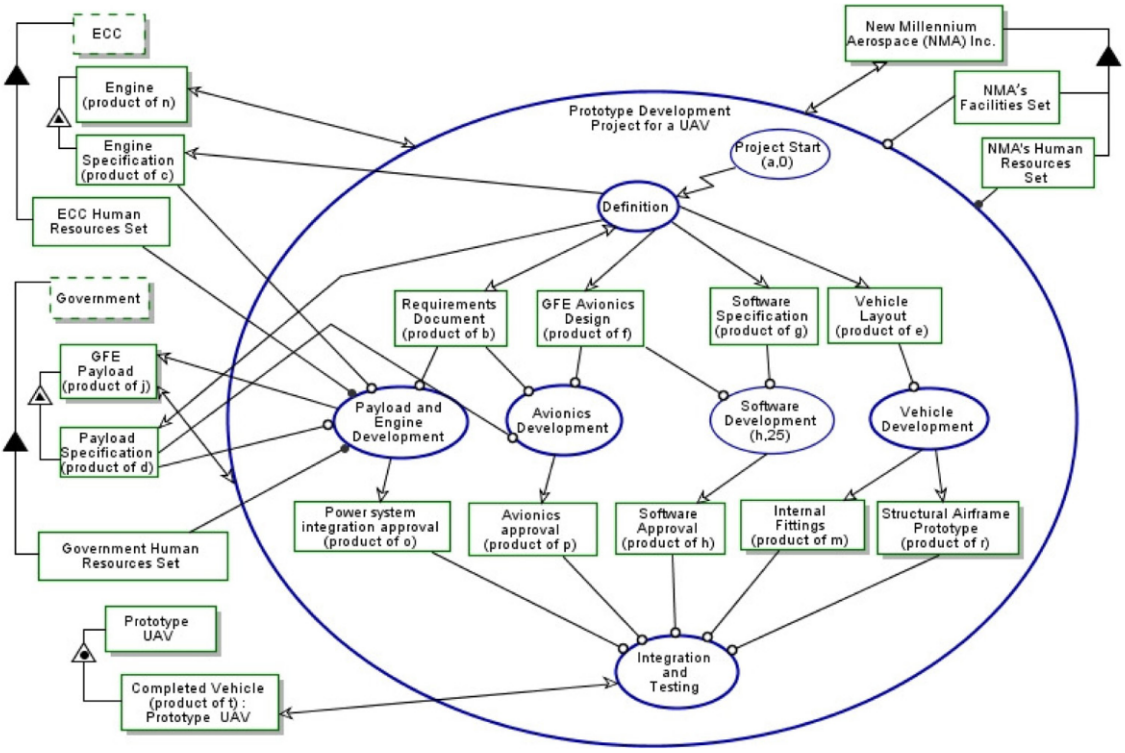


Figure 8. Object-Process Diagram (OPD) of the UAV case. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

Figure 2.1: OPM model of the simplified UAV adapted from [87]

2.2.2 Coupling of TA and PA

In [100], a model and rules for the management of the multi-level interaction between system design processes (typically TA) and project planning processes (typically PA) are discussed. The rules have been integrated in the ATLAS IT platform. After the failures of the A380 Program and OlKiluoto Nuclear Power Plant projects, which were executed within a concurrent engineering environment, and based on their empirical survey, the authors first argue on the vital need to formalise the interactions between the design of a system and its design project. Further, they highlighted there has been no work that formally addressed such a need from the

perspective of planning and controlling of design activities. However few works made explicit these interactions.

Then they establish a bijective link at a structural level between a System S and a project P, system requirements SR and the requirement task definition PR, and system alternative SA and alternative developement task PA. At the behavioural level, the two processes (TA and PA) are interrelated via their so-called feasibility and verification attributes of elements at the structural level. A meta-model supports the realization of links.

The feasibility attributes are equipped with 3 states: undetermined UD, feasible OK, and unfeasible KO. The state of an attribute is computed by a design manager and a planning manager based on requirements, constraints, risks and schedule, resources. Based on those states, precedence rules are established between structural elements and their states. As an example: it is not possible to start working on a solution SA if SR are KO.

The verification attributes are equipped with 3 states: undetermined UD, verified OK, unverified KO. The same way as feasibility, precedence rules are established. An example: it is not possible to verify PA before PR.

Based on the two kinds of attributes and their states, 9 synchronisation (for S and P) rules are finally defined to guarantee the consistent evolution of system design and project design. This yields to a 27-state state diagram (see Figure 2.2) which supports the synchronisation of S and P. A state in this diagram is a seven-tuple (SR.Fa, SA.Fa, SA.Ve, PR.Fa, PR.Ve, PA.Fa, PA.Ve) where Ve and Fa are related to verification and feasibility attributes respectively. The initial state is given by (UD,UD,UD,UD,UD,UD,UD). The transitions between states are logically determined from rules.

Finally, as a use case, a landing gear system decomposed in a wheel and brake subsystem and associated projects is considered. This system has 1 SR related to the weight of the system and 1 PR related to the duration of the project. Other works in the same spirit are: [1] [23].

In [44], the use of MBSE as the foundation of SE management for planning, reviewing and measuring the development process is discussed. The authors claim the approach is supported by a metamodel implemented by the software tool: Mechatronic Modeller. This tool allows to define the abstract and concrete syntax and the static and dynamics semantics for the proposed models. These models are those of

i) the system of objectives about processes, products and objects

ii) the operation system which transforms the product objectives into objects

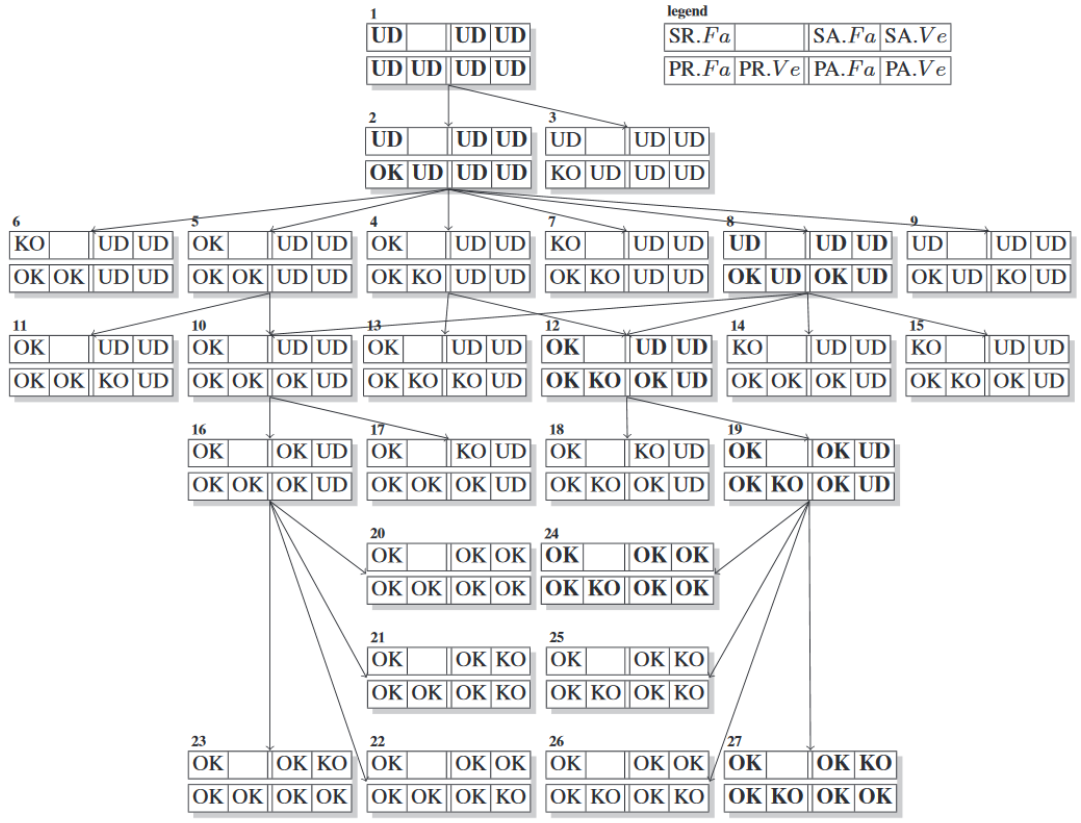


Figure 2.2: 27-state state diagram adapted from [100]

under the constraints of time and cost objectives. Examples of the components of the operation system are: engineers, methods and tools.

iii) the system of objects which is the output (CAD-models, test results, etc.) of the development process.

Together these systems yield the system of product development. These models are then the basis for planning (allocation of resources), monitoring and controlling (time, cost and quality) and alignments on technical and organizational interfaces of the development progress. This approach is mainly about quantitative insights for TA and PA.

In [56], some modelling activities have been tailored following two perspectives: what must be modelled, and how far this modelling must be performed against modelling objectives. This has been possible with the support of a Modelling Management Plan fed by a Modelling Planning Process (MPP) [35], itself automated to ease modelling operations. The MPP enables the alignment of the design tasks on the projects' needs. The MPP ensures the definition and prioritization of modelling objectives, their association with different modelling artefacts (project

concepts, standards and deliverables) and the evaluation of the modelling activity progress. The authors conclude on the necessity to federate the modelling since the proposed approach does not consider the autonomous and distributed nature of MA. Models do target specific purposes, are involved in different MBSE projects in large engineering companies such as Airbus Defence & Space.

2.2.3 Other works

To a large extent, other works in the fields of product (and process) design development and even production see e.g., [3, 15, 4, 17, 44, 21, 69, 16, 73, 103, 27, 75, 93] and [1, 33, 104] for surveys, have addressed the association, integration or coupling (explicitly or not) of the two systems: "system to be made" or the product and the "system for making" or the project, the development system; by different means and for different purposes.

Such a necessity has been introduced for software development projects [93]. The functional requirements and design parameters of the product are mapped into tasks of a Gantt chart project plan following an Axiomatic Design paradigm [93]. The authors claim that such an association between tasks and design enables the rapid delivery of product. [3], [4] and [17] deal with product/process models configuration using constraint-based frameworks. [15] addresses the coupling (via a matrix) to co-evolve the product architecture and the conception project given by DSM and to optimize the organisation of the conception project. [1] provides a summary of few approaches that associate design and planing after which the author concludes that these approaches are little used in the industrial world because their tooling is quite limited. Furthermore they do not take into account the dynamic of design process.

[33] discusses some approaches to the integration of product and process models in engineering design from the points of view of the purposes (Visualisation, Planning, Execution, Synthesis, Analysis etc.) of the Project/Product models and modelling formalisms (Design Structure Matrices, IDEF (Integrated Definition), etc.) and their level of integration (Isolated, coupled, integrated). It is mentioned that few works address the integration of product and process domains. It is also mentioned that, to bring approaches that seem theoretical closer to the industrial context, challenges regarding the scope, focus, development and visualisation of models need to be overcome.

[104] provides a survey on process models and modelling approaches for design and development process (DDP). The authors firstly argue on three features (nov-

elty, complexity and iteration) of DDP. DDP are different from business (typically also production and manufacturing) processes since they call to creativity; they are iterative by nature. They are carried out in large-scale concurrent evolving engineering environments. These features also apply for MA. They discuss the different approaches following two axes: the model scope (micro, meso and macro levels) and the approach type (procedural, analytic, abstract and management science/operations research techniques). ([104, Fig. 1] provides a positioning of the surveyed approaches.) Models are indeed used for different objectives and with different requirements.

Some issues related to the iterative, complex and creative nature of DDP are also pointed out. A central challenge that one may take out from these issues is: how models are/should be exploited against this iterative, complex and creative nature of DDP to solve product/project development challenges in practice? For instance, tooling issues (modelling notations and tools) should be addressed. Finally, they argue on the one hand that DDP modelling is a challenging task of which practitioners should understand the importance. On the other hand, "models should be easy to understand and deliver clear benefit."

2.3 Position of MODEF

In this section we position the methodology (MODEF) proposed in thesis in relation to the presented related work.

2.3.1 On the abstraction of the Modelling Activity

All the above approaches do not explicitly deal with the concurrent nature of the engineering environments (built of autonomous and even independent, stakeholders). Although [100] clearly recognizes the influence of such a nature, the presented multi-level approach does not explicitly consider it. Note also that integrated approaches may not be effective in such a context with regard to capability and autonomy of different stakeholders. This is even more true with engineering activities of Systems of Systems–SoS [55]. Generally, all traditional approaches for SE and management ought to be reviewed to cope with SoS engineering concerns. Indeed, [81] discusses evolutionary principles and implications of the *federalism* concept for SE and management of SoS. This federalism is important because engineering development alliances have been clearly taking the form of virtual organizations [81]

[46]. It encourages autonomy and loosely coupled systems but requires well defined interfaces between autonomous systems. It is argued that the components of those system may be: "locally managed and optimized independently". It is also argued that such a federation of engineering development projects should be considered as Complex Adaptive Systems. The need to deeply understand federated organisations is outlined. Request for modelling, simulations and analyses are finally pointed out.

Unlike the approaches developed in [100] and [87], we do not focus on a particular methodology within the TA; we do not make any assumption on the content (paradigms, methods, languages, tools etc.) necessary to the internal means of TA. We abstract away details and concentrate on the relevant content of models (M) produced by MA. We believe such assumptions do not take into account the diversity of approaches in MBSE. For instance, by considering that the SOI or the "system to be made" should be also modelled with a function structure behaviour approach [88] [87], this might be restrictive, since TA are generally characterize by several model-based domain-specific practices.

The approach in [100] (and even [88] [87]) imposes a structure for TA and PA. In doing so, it also imposes, an *a priori* interesting dynamic for TA and PA. However, this mitigates the flexibility of the approach. However, while MODEF is intended to deal with PA of MA, it might be applicable for TA that address the engineering of a specific SOI (see Section 1.3). We believe a bijective link between the system S and and the project P is a too strong assumption and might not be always relevant given the structure of S and its granularity. From the PA side, we only consider the scheduling of MA and their impacts on the state of M. The link between PA and TA is therefore modelled via the mappings (see Section 1.2).

On the other hand, the verification and validation attributes [100] are interesting since they emanate from quantitative data that provide insights on the states of elements they relate to. In our proposed methodology, we believe such insights could be considered either as constraints for processes or useful to corroborate the states of M in practice. In the approach [100], these attributes are use for both TA and PA.

2.3.2 On the modelling of the Modelling Activity

The models we consider for representing the abstraction of MA (see Section 4) have to enable the modelling of concurrent MA. We also make the choice of explicitly modelling the expected states (and transitions between states) of the conceptual

content of M, something which is rarely considered in the literature. The conceptual content of M is considered because, again, we do not consider any internal content (methodology, paradigms, methods, languages, tools etc.) of TA as necessary to run TA. On the other hand, the models of those states are not integrated with the process models representing modelling tasks. We argue on the benefits expected from this latter choice in Section 4. Indeed there is a trade-off between loosely and deeply coupled models as indicated in [33].

Regarding models themselves, a similarity with [87], is that we use structure and behaviour (process) models. But in our methodology, as argued before, the structure models are not used for the same purpose OPM structure models are used. Likewise, in the OPM-based approach the structure models and their states are the possible mandatory inputs and outputs of process models, in our proposed methodology the processes bring about changes of states of M.

Just like the OPM-based approach, the methodology developed in this thesis is rather general in the sense that it is not tailored for particular techniques (Critical Path Method–CPM, Program Evaluation and Reviewing Technique–PERT, etc.) and factors (Budget measurement, Schedule tracking etc.) studied and compared in [87]. The latter ought to be either derived from the models and their analysis or considered as constraints for models. For instance, like specific approaches focusing on quantitative insights (such as leading indicators), they could help determining the constraints of MA, insofar particular insights need to be considered in an analysis procedure. Indicators are generally quantitative insights. We believe they should be used in conjunction with the models that describe the processes. Indeed the processes give procedural insights that may be useful to anticipate the bad states or deviation of thresholds. More interestingly, it could be possible to specify the required actions to remedy or adjust afterwards.

Unlike the previous approaches, we do not assume a modelling notation or tool to obtain and manipulate the models of MA. We rather provide the formal semantics (of models) which can be mapped to some modelling notations implemented by modelling tools. This choice aims to tackle the challenge related to the tooling of approaches in practice pointed out by [1] [33] [104].

The way (see later Sections 4 and 6) we use models and their purposes are different from all the above approaches. We rely on state-of-the-art standard models languages and notations for a structure-state-process modelling of the modelling activity.

2.3.3 On the analysis carried out with models

From the above approaches, we could not find an evidence of an explicit formal analysis of involved models against some expected formalised requirements. Except perhaps the work [100] where the analysis emanates from the synchronisation rules and the OPM approach [88] [87] where the simulation of OPM models are intended to be exploited to detect various problems (product and project parameters feasibility, deviations and impacts) and take appropriate actions in turn. Indeed the question: what are the properties (or requirements) expected from the system (MA here)? is not formally addressed for MA and more generally in the field of process (and product) design development like (see e.g., [6] [74]) in the model-checking and systems synthesis fields.

Regarding the analysis algorithm itself, the UCS algorithm we use (see Section 5) to guide the exploration of models against requirements might be replaced by another algorithm from operations research and artificial intelligence fields. Therefore, the analysis is dissociated of the provenance of models and can be tailored for a specific concern (see Sections 5 and 8). This contributes to the building a flexible methodology while dealing with the tooling issues.

2.3.4 On the implementation of approaches

It follows from the foregoing that the methodology presented in this dissertation is not intended to be specifically built for a modelling tool or a modelling language. Instead all principles, foundations and algorithms are given. Nonetheless, modelling tools are at least necessary to build models. Therefore, to make the connection with tools in order to reuse models for analysis concerns, we specify (see Sections 6 and 7) an architecture and means for its implementation for exploiting models coming from a modelling tool. We could not find such an implementation choice from the aforementioned approaches. We believe such a choice is very useful for the practicality of the methodology. This could also holistically contribute to tackle the tooling issues in practice. We will show that this architecture might be useful to deal with model reuse (for analysis purposes) in model-driven engineering.

Abstraction of the Modelling Activity

This section elaborates the first step of MODEF(see Figure 1.1). We present the System of Modelling (SoM) and the System of Systems of Modelling SoSoM (Sections 3.1.1 and 3.1.2). The application examples (Section 3.2) are presented to illustrate the concepts and ground the ideas for the next sections.

3.1 SoM and SoSoM

For many current complex systems and even more for systems-of-systems–SoS, it is not possible for an individual even a domain-specific community to understand all aspects of these systems. Different stakeholders tend to be expert in their domain. It is a reason why the shift has been continuously operated from the document-based to the model-based approaches [51]. Arguably, even with model-based engineering, other issues (namely representation, computation and utilisation) relating to model arise. The growing complexity of the modelling environment is characterized by autonomous stakeholders with different practices (paradigms, methods, tools, standards, etc.). Finally, several projects and programs are common in most large companies. They often evolve in parallel, addressing different issues or views related to a given problem, finally they have each their own life cycle. To characterize the modelling activity (MA) in such a context, we clarify in the following the SoM then we follow [81] on the *New Federalism* [19] principle to describe the SoSoM.

3.1.1 System of Modelling–SoM

The SoM depicted on Figure 3.1, is to a large extent a set of stakeholders and their practices which are autonomous with their own capability. It is roughly a system

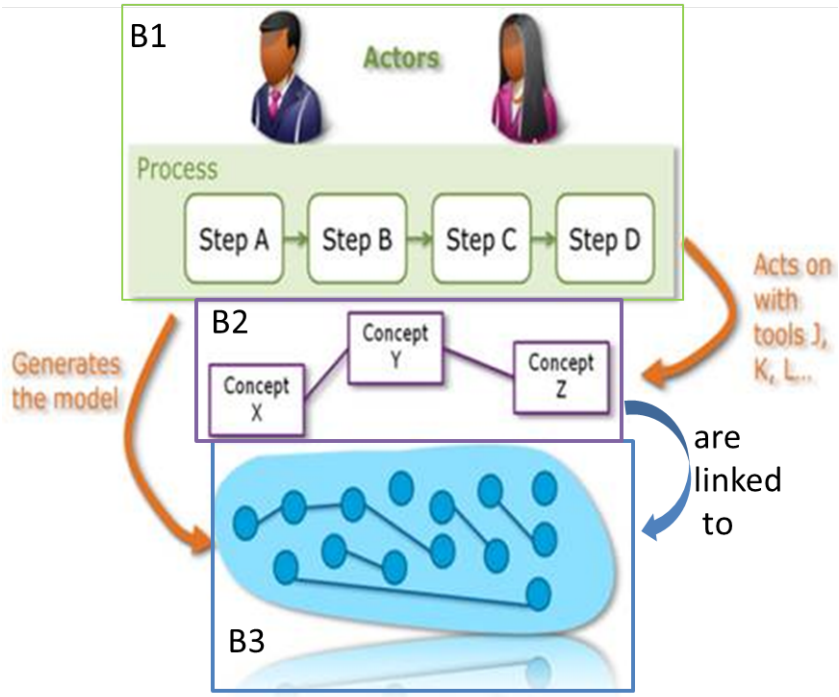


Figure 3.1: SoM

of people, methods, tools, processes, standards and models (M) in interaction that addresses the engineering/modelling of a system-of-interest. We selected System of Modelling instead of System of Engineering to stress on the fact that we are in a MBSE framework. Let us now discuss on which elements of SoM we will focus to study it.

On Figure 3.1, we have three main blocks.

- At the top of Figure 3.1, we have people and a scheduling of tasks (Step A, Step B, etc.) they carry out. This block (B1) represents the scheduling of MA.

- The block B2 contains three boxes seen as conceptual models that represent the level where M are abstracted.

- The block B3 contains the actual models (M) that represent the SOI that is being modelled. The blue arrow on Figure 3.1 indicates the connection between M and the conceptual models. Such a connection, its definition and implementation, applications are partly reported in [35] [36] [56] [37]. Apart from the fact that conceptual models are abstractions of M, they are useful as a means to involve different stakeholders in the modelling process and ease it typically via an approach that combines metamodels and ontologies [36]. For instance,

the stakeholders that are unfamiliar with specific metamodels concepts are involved via domain specific concepts related to their view points. The two kinds of concepts are therefore related via the combination of metamodels and ontolgies.

In order to study the SoM, we focus on scheduling of modelling tasks, the conceptual (or structure) models that capture the main content of M. To understand the impacts of modelling tasks on M, we associate on the one hand, the conceptual models to state models that characterise the expected (or lifecyle) states of M. On the other hand, the events from the tasks' execution are be mapped on the transitions of state models to indicate the effects of these tasks on states. Therefore, the SoM is studied at these three architectural views of a modelling project.

Actually, several modelling projects are run in parallel. Indeed, many engineering programs involve several organisations located on multiple sites (geographically distributed). Regardless of the geographical distribution, the engineering projects often address concerns that overlap. At the same time the SoM are generally autonomous. The question which arises here is: how to understand these SoM and their evolution?

3.1.2 System of Systems of Modelling–SoSoM

The SoSoM, depicted on Figure 3.2, is built of different SoM that could together participate to several projects and programs. The SoSoM is a way to understand the evolving relationships between several SoM. Therefore it is more than just the sum of SoM. In fact, on Figure 3.2, the dashed line (with arrows directed towards the SoM) means that there are interactions or commonalities between SoM. The added-value of the SoSoM emerges from those interactions. With the autonomy and the proper operational capabilities of SoM, it is difficult to impose an integrated approach. Instead, we argue in the following that the federation is adequate to characterize the SoSoM. We eventually recall that Federation has been considered as a type of System of Systems by [61] [81].

Following [81], the federation can be defined as an organisation form that meet the federalism principles. The federalism principles indicate, on the first hand, the autonomy and leadership of SoM are recognized and unavoidable. On the other hand, shared interfaces and agreements on some interoperation mechanisms are required to get an added-value at the federation level. The federalism principles are of particular importance because engineering development alliances have been

clearly taking the form of virtual organizations [81] [46]. Federalism encourages autonomy and loosely coupled systems but requires well defined interfaces between autonomous systems. Federalism is based on five principles defined by [19] and summarized by [81] in the perspective of SE and management:

1 *Subsidiarity*–This means that the power is distributed throughout the lower levels of the federation of SoM where there is some assumed responsibility.

2 *Interdependence*–This means that services are combined whenever and wherever necessary.

3 *Uniform and Standardized Way of Doing Business*–The interdependence is possible only if there are agreements with interoperation protocols.

4 *Separation of Powers*–Management, monitoring and governance aspects of Federation of SoM should be considered as distinct functions.

5 *Dual Citizenship*–This means that parts of SoM are both in their local SoM and in the Federation of SoM.

It follows from the foregoing that, the federation is adequate to qualify the SoSoM. That is, the SoSoM level is obtained by the federation of several SoM. To study the

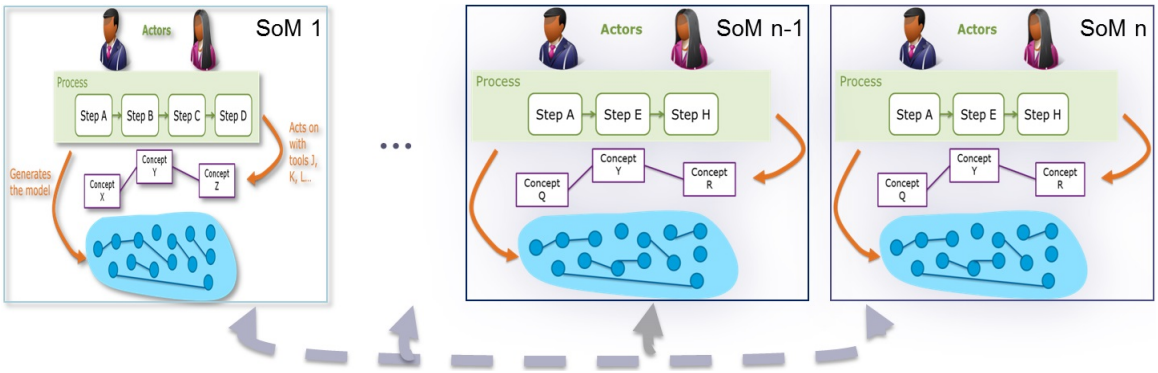


Figure 3.2: SoSoM

SoSoM, we consider the same views as with the SoM: structure, state and process of MA related several SoM. The actual content of these views will depend on the problem at hand at the SoSoM level. But the relations between the different models remain the same: we have elements of the structure view that have states that evolve under the effects of processes.

3.1.3 Related research

Federation of systems has been considered as a type of Systems of Systems–SoS [61] [81]. In this kind of SoS, there is no central authority, but the components of the SoS collaborate and cooperate to meet the objectives of the federation [61].

Regarding the SoS themselves, there has been a huge literature for SoS definition without a globally accepted definition. The five principle characteristics of an SoS are [68] :

(a) *Operational independence* of the component systems – The SoS is composed or assembled of systems which are initially independent and useful in their own right.

(b) *Managerial independence* of the component systems – They can operate independently outside the SoS or participate in several SoSs without losing their operational identity.

(c) *Evolutionary development* – The configuration of the SoS and its design evolve continuously.

(d) *Emergent behavior* – It is the result of the dynamic assembly of components systems which is not observable in any of them.

(e) *Geographic distribution* – the component systems are geographically distributed. It is worth emphasising here that in the case of the SoSoM, although the geographic distribution is unavoidable, what really matters in MODEF are the three views on which we focus: the modelling activity, the produced models and their expected states and transitions.

The discussions of those characteristics, other definitions of SoS and the challenges (mainly related to TA) in the model-based engineering of SoS can be found, e.g., in [83] [81] [55] [66] [18] and [41].

We addressed the necessity to consider MA or models produced by the MA as a federation in [56]. The SoM and SoSoM as representations of MA are introduced in [57]. The SoSoM might be also considered as a multi-agent system (MAS) [89]. But, since we do not focus on issues related to organisation and communication which are central in MAS, we prefer to stay at system level which is more generic and to focus on some architecture views relevant to the objective of MODEF.

3.2 Application examples

In this section we introduce some application examples to illustrate the concepts and ground the ideas for the following sections. We start by presenting actual elements that constitute a SoM (Section 3.2.1), then we discuss the problems one may need to address at the SoSoM level (Section 3.2.2) that comprises several SoM.

3.2.1 An SoM: Modelling the functional coverage of a SOI

Different aspects of the SOI are generally modelled by different stakeholders. And it might be required that at some stage in the development process, models satisfy some criteria for verification, test or integration purposes for example. Suppose we are only interested in the functional architecture whose objective is to ensure that the modelled SOI covers the functional needs. We need to figure out what are the elements allowing to define this modelling project as an SoM say SoM0.

Following the main elements (see Section 3.1.1) considered to study an SoM, one needs to identify

- the sequencing of modelling tasks at stake to carry out this modelling project
- the conceptual models that abstract away the main contents of models (M)
- the expected states of M and transitions between them represented by elements of the conceptual models
- the effects (mappings) of modelling tasks on these states.

Figure 3.3 is an illustration of such elements. We can observe on that figure the following data:

• At the right of Figure 3.3, is a process model (PM) that represents a high level view of the tasks carried out in the SoM0. This process model contains six tasks starting with the the task named "Model high level functions" and ending with the task named "Validate RefinedFunctions".

• At the top left of Figure 3.3, is a conceptual model (AM) that represents the entities (named "System Component" and "System Function") and their relation taken into account in the SoM0.

• At the bottom left of Figure 3.3, is a state model (SM) that has 3 states. The initial state is named "Maturity<30". Typically, a state in this model describes the maturity level of something that should be known after a mapping.

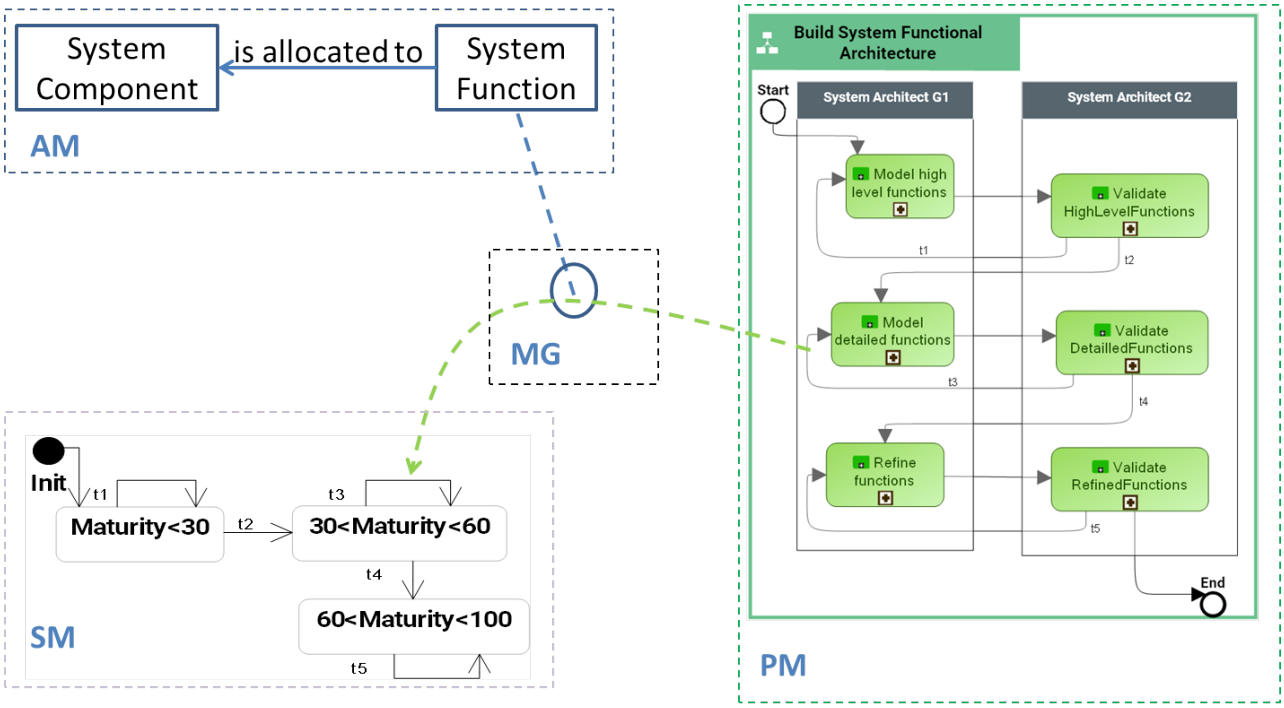


Figure 3.3: At the top left, the bottom left, the right, and the middle are a structure, state, process model and a mapping respectively of the SoM0

• At the middle of Figure 3.3, is a mapping (MG) which roughly means: once the task named "Validate DetailedFunctions" is finished, the state of a model abstracted away by the entity "System Function" remains "30<Maturity<60" if it was "30<Maturity<60". Such a mapping typically indicates in practice that, the outcomes of the task "Validate DetailledFunctions" do not allow to evolve the state of some models (M) to the "60<Maturity<100".

Given the presentation of these models of the SoM0, it is clear that they might be obtained separately, but must be connected via the mappings to get the whole view of the SoM0.

Other SoM could be defined the same way. For instance, a modelling project regarding a different viewpoint of the SOI.

3.2.2 Examples of SoSoM

The SoSoM is about the relationships of at least two SoMs. The local view of the SoSoM is characterized by the different SoM whereas the global view depends upon the SoSoM's objective. Suppose the different objectives at the SoSoM level are as follows.

• Objective 1: We want to harmonize the models (M) produced by several SoM.

By harmonization we mean to correlate and explicitly identify the relationships among different entities (of the conceptual models) within several SoM. The benefits expected afterwards are to minimize redundant information or models, enable model reuse where possible, detect conflicting models and possibly ease their reconciliation procedure.

- Objective 2: Following the SoSoM1, another objective one might want to address at the SoSoM level is the capitalisation of (the knowledge about) the modelling. By capitalization we mean the backup of models (M) shared or agreed among a set of SoM.

- Objective 3: Suppose now, we are interested in the co-evolution of the modelling within the SoM for addressing some higher goal. This will mean, although SoM are autonomous, there might be the need to agree at some points in their respective life cycle. Such agreements will be necessary to ensure that the two SoM could together, reach some expected states at desired time in the future.

Now, we discuss how a SoSoM and the three kinds (see Section 3.1.2) of models that represent it allow to deal with each of these three objectives.

- **Modelling Harmonization—SoSoM1** Figure 3.4 contains the possible data related to SoSoM1. These data are as follows. — At the right of Figure 3.4, is a process model (PM) that represents a high level view of the tasks carried out in the SoSoM1. This model indicates the tasks to run at the SoSoM level.

  — At the top left of Figure 3.4, are the entities ("System Component" and "System Function") of the conceptual model (AM) that are relevant at the SoSoM level.

  — At the bottom left of Figure 3.4, is a state model (SM) that has 3 states. Each state characterizes a possible state of models within SoM.

  — The mappings are partially performed on SM. Indeed the transitions of the state model are labelled with transitions between the tasks in the process model. It remains to indicate to which entity this state model applies. It typically applies for the two entities.

  Let us now discuss a possible working scenario of SoSoM1. Suppose SoSoM1 comprises 3 SoM namely: three modelling projects that each addresses, the architecture description of a transportation system by plane (SoM1), car (SoM2)

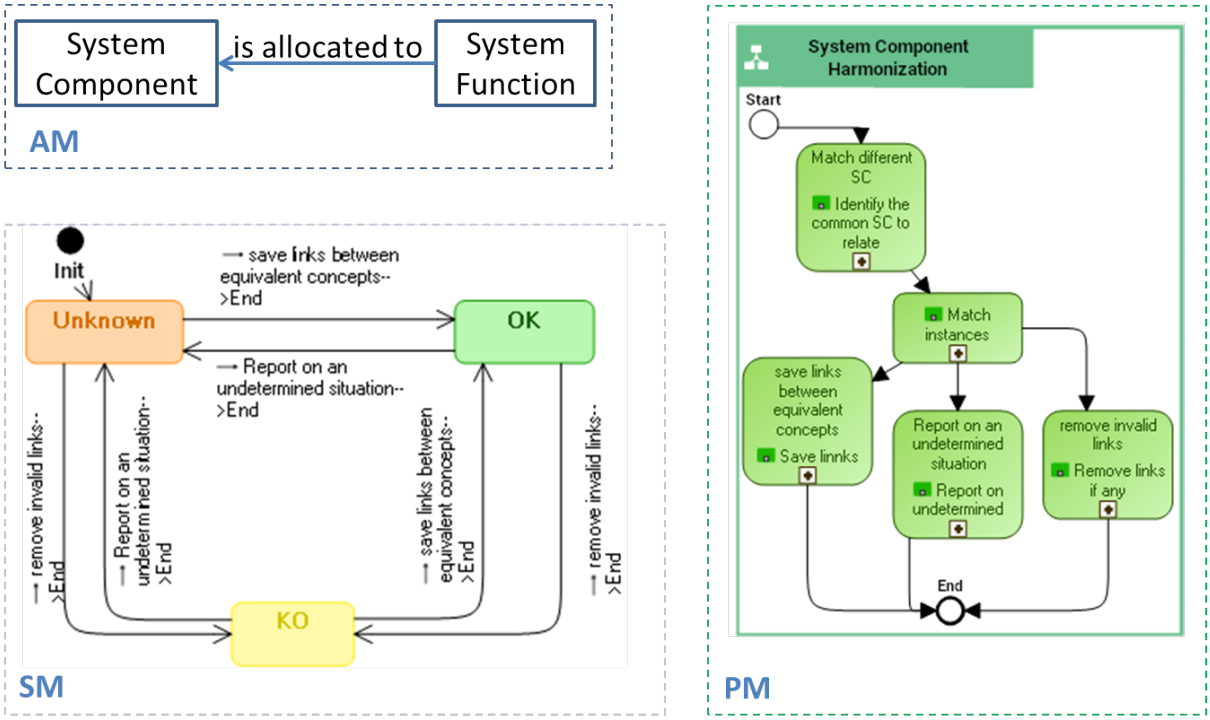


Figure 3.4: At the top left, the bottom left and the right are respectively a structure, state, process models of the SoSoM1

and ship (SoM3) respectively. As indicated by entities of the conceptual model of SoSoM1, the focus is on "System Function" and "System Component". Suppose the 3 SoM also expose these entities (or equivalent) in their conceptual models. As the second task (named "Match instances") of the process model of SoSoM1 indicates, the instances (i.e. models (M)) associated to each entity in the SoM are matched at some point in the process carried out in SoSoM1. Depending on the executed tasks, the states of instances related to "System Component" are possibly updated. If for instance, all the 3 SoM deal with a GPS (Global Positioning System), a link will be created at the SoSoM level to indicate such a dependence.

It follows from this scenario that SoSoM1 could be a first step towards the mastering of the replication of models (M) and model reuse.

It is worth emphasising here that the process model at the SoSoM level (and even the SoM level) does not say how tasks are technically run. Instead, it allows to figure out how tasks are scheduled and then, with the mappings, their effects on states of studied elements. The way to compute the equality and equivalence between concepts of conceptual models has been addressed for instance in [37]. Nonetheless such technical concerns are beyond the scope

of this dissertation. Besides they are specific to the SoSoM's objective.

The SoSoM1 above does not consider the state and process models of its constituent SoM, but only their conceptual models. Depending on the objectives of the SoSoM, such models should be considered; see the following two SoSoM.

- **Modelling Capitalisation—SoSoM2** If the capitalisation process occurs during project development, the state models of SoM would be relevant. For instance, if the state models of SoM address, like with the SoM0, the maturity level of models; they have to be taken into account in the SoSoM2. As models of the SoSoM2, we could have the same conceptual model like the one of SoSoM1. Its state models will correspond to the expected states of entities of the conceptual models. Its process models will correspond to the description of the tasks necessary to deal with the modelling capitalisation process. Finally the mappings must be specified.

- **Modelling Evolution—SoSoM3** In this scenario, the structure, state and process models of the SoSoM will combine the relevant subsets of SoM models.

In this section we characterised the SoM and SoSoM and we gave some application examples. It is now necessary to discuss how the models of these systems might be usefully exploited in practice. We start by presenting in the next section the underlined principles and semantics of the considered models. Then we specify the formalism to specify the requirements or expectations on these models.

Modelling the Modelling Activity and its Expectations

This section elaborates the second step (Represent structure, state and process models), third step (Specify structure-state-process mappings) and the fourth step (Specify expectations) of MODEF(see Figure 1.1). In order to reason with the 3 considered kinds of models, we now introduce their underlined principles and semantics (Section 4.1) considered in this dissertation and their mappings (Section 4.2). Then we present the expectation-specification formalism (Section 4.3) necessary to define system's requirements or specifically the expectations from the system.

4.1 Structure, Process and State models

The definition of the principles of models is especially important in MODEF for automating the analysis of models. This enables to decouple the implementation of the step 5 (Analysis of models) of MODEF from the logic of a particular modelling tool used to draw models. These models are mainly descriptive models. The formal and federated use of these models is discussed in Section 6 under a categorical architecture.

Let us call the *structure (abstract syntax) of a model* of the system, a non-empty finite set of disjoint components. A component is either basic (i.e. without constituent components) or composite (i.e. with a non-empty finite set of constituent components). Since all models are composite structures, suppose a component c is, for the purpose of this section, basically given by

$$G = \langle V, E \rangle \tag{4.1}$$

where $V := \{c_1, \ldots c_n\}$ is the set of internal components of c, $E := \{l_1, \ldots l_m\}$ is the set of directed links/connections between elements of V and c itself (when connection

ports are considered). n and m are positive integers.

From the structure of a model, we aim at obtaining its interpretation in a domain of interest. The interpretation is derived from the structure of a component. Indeed, to model the physical aspect of the system, a component (an element of V) might be interpreted as a physical element and the corresponding E a set of physical, or logical connections, whereas to model the states (and the transitions between them) of the system, a component might be interpreted as a state and the corresponding E the set of transition between its states. The semantics or the formal definition of an interpretation must be defined and perhaps should be algorithmically available in order to automatically use an interpretation in algorithms (e.g. an analysis algorithm).

Note that, although the concrete syntax (graphical symbols used to render the models) is important for the end user, it does not matter in MODEF. Since we aim to decouple MODEF from the logic of a particular tool. It is useful only for the user who will draw and/or visualise the models.

Now, we discuss the formal definitions of the semantics of the three kinds of models considered.

4.1.1 Structure model

The semantics of structure models will depend on the system at stake. For example for the SoM, we interpret the structure models as conceptual models while for a physical system they might be interpreted as computing, physical or human components. Generally, the structure of structure models will be sufficient for the purpose of MODEF. For example, Figure 4.1 is the structure model that describes the Resource Architecture (a combination of physical assets and organization configured to provide a capability) of a supermarket. The structure of this model is a component that has 4 constituent components (*Customer*, *Maintenance Operator*, *Supermarket Director* and *Supermarket Entrance System*) and 3 links.

4.1.2 Process model

The semantics of process models is characterized as generators in the sense of [76]. Roughly, the exploration or execution of a process model should generate a language where the alphabet (Σ) is the finite set of events and the words are the event-traces or the sequences of events. This yields a non-deterministic automaton where accepting states correspond to the end of a possible execution of the process model. The set

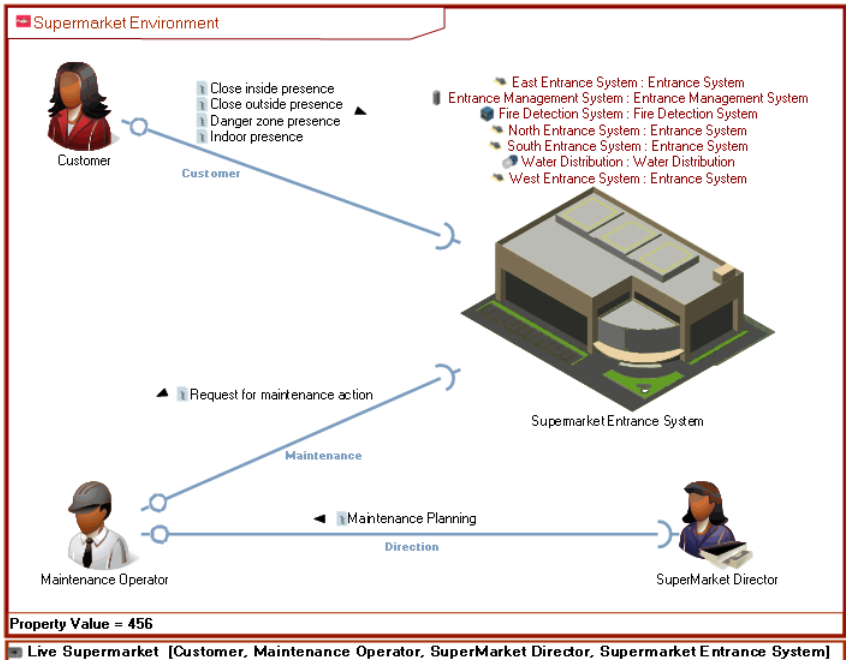


Figure 4.1: A supermarket's environment structure.

of words that bring the automaton from the its initial state to an accepting state is called the language recognized (or "marked") by the automaton.

Such a choice is motivated by the fact that modelling tasks can be indeed considered as discrete-event processes. Also, the class (principal features: discrete, asynchronous and possibly non deterministic) of processes considered in [76] is well-suited because we mainly focus of planning of tasks rather than data processing.

Finally we are interested by words with a finite length i.e., finite execution of processes.

Adapted from [76], such an automaton is formally defined as follows. A generator is a 5-tuple

$$\mathcal{G} = (Q, \Sigma, \delta, q_0, Q_m) \tag{4.2}$$

where Q is the set of *states* q, Σ is the *alphabet* or finite set of output symbols σ, $\delta : \Sigma \times Q \to Q$ the *transition function*; a partial function, $q_0 \in Q$ the initial state and Q_m a subset of Q called *marker states* or final states. $\mathcal{G}$ is equivalent to a directed graph with node set Q and an edge $q \to q'$ labelled σ for each triple (σ, q, q') such that $q' = \delta(\sigma, q)$. This edge or state transition is called an *event*. Events are considered to occur spontaneously, asynchronously and instantaneously. Furthermore an event is recognizable via its label σ by an outside observer. Distinct events at a given node always carry distinct labels.

If Σ^* denote the set of all finite strings s of elements of Σ including the empty

or identity string 1. The extended transition function is given by $\delta : \Sigma^* \times Q \to Q$, $\delta(1, q) = q, q \in Q$ and $\delta(s\sigma, q) = \delta(\sigma, \delta(s, q))$ whenever $q' = \delta(s, q)$ and $\delta(\sigma, q')$ are both defined.

The language *generated* by $\mathcal{G}$ is

$$L(\mathcal{G}) = \{w : w \in \Sigma^* \text{ and } \delta(\sigma, q_0) \text{ is defined}\}. \tag{4.3}$$

It is also the set of all possible finite sequences of events that can occur.

The language *marked* or recognized by $\mathcal{G}$ is

$$L_m(\mathcal{G}) = \{w : w \in L(\mathcal{G}) \text{ and } \delta(w, q_0) \in Q_m)\}. \tag{4.4}$$

Example: At the left of Figure 4.2 is the process model of the SoM0 (see Section 3.2.1) while at right is the graph equivalent to the generator associated to this process model.

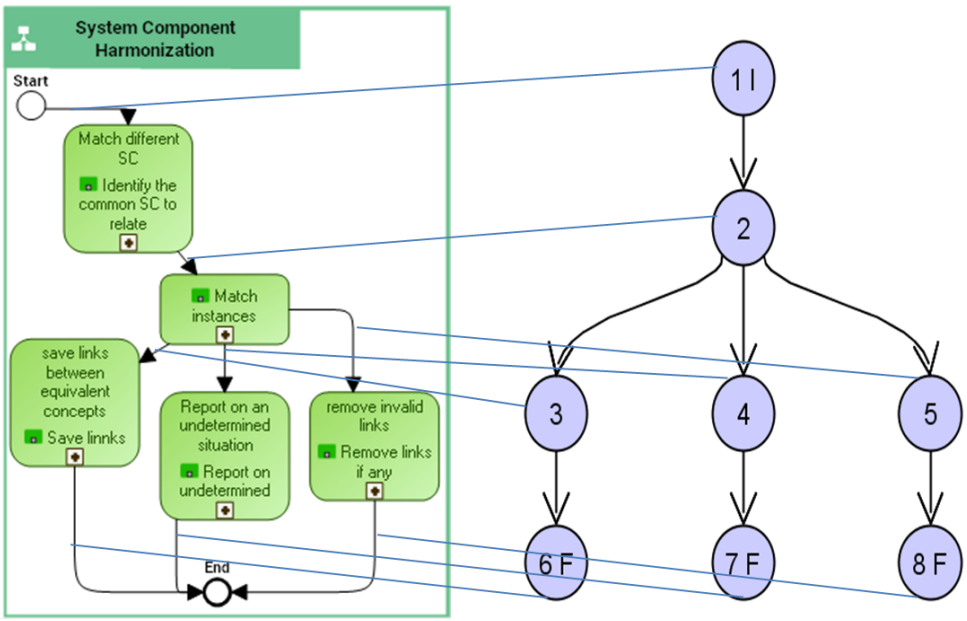


Figure 4.2: A process model at left and the graph equivalent to its associated generator at right

The structure of this process model is a component that has 7 constituent components (named "Start", ... ,"End") and 8 links. To obtain the generator ($\mathcal{G}_0 = (Q, \Sigma, \delta, q_0, Q_m)$) associated to this model, we first need to understand the interpretation of components and links. Therefore, the implementation of the semantics of a process model must explicitly expose the functions necessary to compute the

states and transitions of such an automaton. We present such functions in Section 7. The reason is that since this process model is a BPMN (Business Process Model Notation)-like model, i.e., an application/implementation detail, these functions will be specific to the BPMN syntax. But here, we give the main ingredients to obtain the associated generator.

A component of a BPMN model is a node of a given type (Event, Gateway, Task, etc.). A link between two components is generally called a "sequence flow". In the graph equivalent to the generator, the vertices or states will correspond to the sequence flows and the edges or state transitions the ends of the execution of an executable node. The initial state will correspond to the sequence flow whose the source node is the outermost (in hierarchy structure) "Start" node. The accepting states will correspond to the sequence flows whose the target node is an outermost "End" node.

Therefore, at the right of Figure 4.2, the states i.e. the set Q (the circles) are linked (via blue lines) to the corresponding sequence flows. δ and Σ are straightforwardly obtained from the same figure. The initial and accepting states (q_0 and Q_m respectively) are "1 I" and {"6 F", "7 F", "8 F"} respectively. It is also easy to see that the language recognized by G_0 contains 3 words.

Finally, note that, in the generation or exploration of the graph equivalent to the generator, the graph should not need to be entirely computed. This is due to the mappings (and possibly constraints on processes) that will constrain the exploration of processes.

4.1.3 State model

The semantics of state models is characterized as an hierarchical finite state model (HFSM). The definition of such HFSM is as follows.
If a component c is such that V and E are both empty sets, c is called a *basic state*, otherwise, c is a *composite state*. If c is composite, every element of V is interpreted either as a basic state or a composite state. E is the set of transitions between states. By transitions between states, we do not mean the actual set of events that will effectively trigger those transitions. An HFSM is structurally (syntactically) a composite component. At the semantics level, this composite component is equipped with an *initial state* and a *state-transition-relation* defined in the following:
Current state–The current state of an HFSM c^0, is given by the stack $[c^0, c^1, \ldots c^k]$,

where c^i is a constituent component of c^{i-1}, $i = 1...k$, c^k is a basic component.
Base state-transition relation–Let $\delta : E \times V \times V$, the *base state-transition relation* associated to c. $(l_1, c_1, c_2) \in \delta$ means that there is a link l_1 from c_1 to c_2. The actual label or event that will fire the link or semantically the transition, is obtained via a binary relation $EVENT \subseteq \Sigma \times E$ that associates to a link the trigger event.
Base initial state–Let c_0, $c_0 \in V$ the *base initial state*, a particular state which indicates from which constituent component of c, c is locally initialised. "locally" here means that the notion of hierarchy is not considered.
HFSM–Let c^0 an HFSM and $s([c^0, c^1, \ldots c^k])$ a given state of c^0

- s is the *initial state* of c^0 if and only if c^{i+1} is the *base initial state* of c^i for all $i = 0...k-1$.
- the *state-transition relation* of c^0 consists, given a current state $s([c^0, c^1, \ldots c^k])$, in determining a next state s'.
  Suppose δ_p is the *base state-transition relation* of c^p, $p = 0..k-1$ respectively. s' is given by
  - $s'([c^0, c^{y_1} \ldots, c^{y_j}, c^{m_1} \ldots, c^{m_x}])$ if c^{y_j} is an HFSM and $s''([c^{y_j}, c^{m_1} \ldots, c^{m_x}])$ is the initial state of c^{y_j}, $j > 0$
  - $s'([c^0, c^{y_1} \ldots, c^{y_j}])$ if c^{y_j} is a basic state

  where $c^{y_0} := c^0$, $c^{y_1} := c^1$, ..., $c^{y_{j-1}} := c^{j-1}$ such that whenever $(l_{jy_j}, c^j, c^{y_j}) \in \delta_{j-1}$, $j \in 1..k$, we do not have $(l_{iy_i}, c^i, c^{y_i}) \in \delta_{i-1}$ i.e. the transition l_{iy_i} is not fireable, $i \neq 0$ and $i \in 1..j-1$.

  This means that given a current state $s([c^0, c^1, \ldots c^k])$, the firing of a transition corresponds to the application of one and only one *base state-transition relation* of a component c^i, $i \in 0..k-1$ provided that the *base state-transition relation* of c^j, for all $j = 0..i-1$, $i \neq 0$ are not applicable for c^{j+1}.

Example: Consider the HFSM on Figure 4.3. c^0 is such that $V := \{1, 2, 3\}$ and $E := \{t_1, t_2, t_3, t_4, t_5\}$. Suppose also that $EVENT$ is given by couples (e_i, t_i), $i = 1..5$, $e_i \in \Sigma$. The HFSM c^1 associated to the component c^1 is such that $V := \{5, 6\}$ and $E := \{t_6, t_7\}$. The base initial states of c^0 and c^1 are 2 and 5 respectively. The initial state of c^0 and c^1 are $s_0 := [c^0, c^2]$ and $s_1 := [c^1, c^5]$ respectively. The base state-transition relations of c^0 and c^1 are straightforwardly obtained from Figure 4.3. We are now ready to give the state-transition relation of c^0. For the sake of simplicity and without loss of generality, we assume at a

given moment, only one event in $\{e_{i,i=1..5}\}$, is active i.e., at most one transition of c^0 is fireable. The state-transition relation of c^0 is therefore given by $(s,t,s') \in$
$\{([c^0,c^2],t_2,[c^0,c^3]),([c^0,c^3],t_3,[c^0,c^2]),([c^0,c^3],t_4,[c^0,c^1,c^5]),$
$([c^0,c^1,c^5],t_5,[c^0,c^1,c^5]),([c^0,c^1,c^5],t_6,[c^0,c^1,c^6]),([c^0,c^1,c^5],t_1,[c^0,c^2]),$
$([c^0,c^1,c^6],t_7,[c^0,c^1,c^5]),([c^0,c^1,c^6],t_5,[c^0,c^1,c^5]),([c^0,c^1,c^6],t_1,[c^0,c^2])\}$

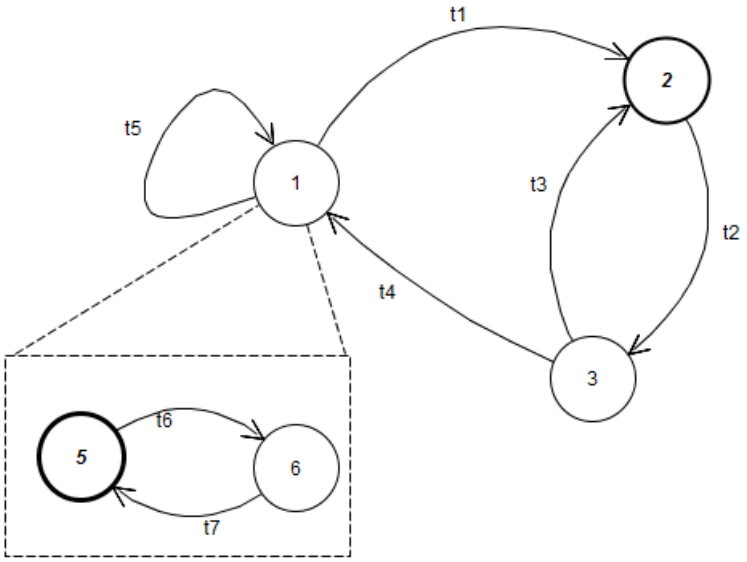


Figure 4.3: An example of an HFSM

The values of elements of E are fully determined after the association of the events from processes on the transitions of the state models via $EVENT$ (see the following Section4.2). In the spirit, this association typically leads us to consider an HFSM as a named/labelled transition system [59]. That is, the transitions of an HFSM are labelled or named with actions or events belonging to the set of the events of processes. An HFSM is deterministic if and only if $\delta' : \Sigma \times E \times V \times V$ is a partial function $\delta' : \Sigma \times E \times V \to V$ such that $(e,l,s_1,s_2) \in \delta'$ if and only if $(l,s_1,s_2) \in \delta$ and $(e,l) \in EVENT$

An HFSM can be compared to (hierarchical) finite-state machines such as Harel statecharts. [47]. It is written in [47]:
"statecharts=state-diagrams+depth+orthogonality+broadcast-communication."
The similarity stems from the fact that, like statecharts, we deal with the depth of states via the composite structure of states. Unlike Statecharts, orthogonality and broadcast-communication are not considered in an HFSM. Indeed, in an HFSM, the transitions are only defined inside a composite component equipped with only one initial state. Orthogonality could be managed with the parallel composition of HFSMs as described in Section 5. Contrary to Statecharts which are a visual

modelling technique for which several semantics exist [38], there is only one semantics for an HFSM.

An HFSM only defines an initial state and a state-transition relation.

4.2 Relations between process and state models

Following the descriptions of the three kinds of models and their exploration semantics, here we aim to map the events from processes P on the transitions of state models of components of structure models. Therefore, by the relations (or mappings) of models, we mean the specification of the expected the effects of process models on state models.

We have assumed that the events or more precisely their labels or values (Σ_P) generated by P are available to an outside observer (see Section 4.1.2). In addition, they correspond to the (spontaneous, asynchronous and instantaneous) transitions between executable nodes during which no change is assumed to occur in the process. Suppose also that to the structure, process and state models, we associate the structure (defined by (4.1)) $G_a = \langle V_a, E_a \rangle$, $G_p = \langle V_p, E_p \rangle$ and $G_s = \langle V_s, E_s \rangle$ respectively.

Let PHY, $TRANS$ and $EVENT$ be the union of all structure components, the union of all sets of E_s and the union of all sets of E_p respectively.

TRIGGER is the set of associations of events on the transitions of state models of structure components. TRIGGER is introduced in Section 3.2 via the associations (or mappings) exemplified. TRIGGER is formally defined by

$$\text{TRIGGER} : PHY \times TRANS \rightarrow 2^{EVENT} \tag{4.5}$$

where $(a, t, e) \in$ TRIGGER means: the transition t of (the state model associated to) the structure component a is possibly triggered by consuming the set e of the events, and conversely, since e is explicitly involved in TRIGGER, it has to be consumed. This latter requirement is rather a business one and directly influences the exploration semantics of models. Besides, note that the mappings allow to preserve the autonomy and specificity of each kind of model. The autonomy is preserved because each kind of model can be developed separately. Whereas the specificity stems from the fact that the meaning of each kind of model is not altered when it (or part thereof) is involved in an interconnection specified via (4.5). By no means does the interconnection influence the development of these

models. Nonetheless, changes in these models would result in necessary changes in TRIGGER.

Every element of the triplet (e, a, t) must be actually defined as a stack (see the definition of a state of an HFSM in Section 4.1.3 and more generally Section 6) in order to have TRIGGER well defined as a function. Composite structures also permit to avoid to systematically flatten all the models before running an exploration.

In this thesis we do not address the same problem as the one addressed in [76]. [76] is concerned with the control of the generator (object to be controlled) by a supervisor (the controller) via a control pattern (the set of all binary assignments to the elements of a subset of Σ which an element is called a *controlled event*; or a specification). Such a problem refers in the literature to the synthesis of a model of a supervisor from the models of object to be controlled or the plant and requirements. See for example [5] on the integration of supervisory control in MBSE. But there are similarities between the approach followed in this thesis with the supervisory control. These similarities are as follows.

The process and state models are rather seen as the models of the master (associated to a generator) instead of a controller and object to be mastered respectively. In this thesis, the master is a reactive and autonomous process for which we require its full specification. The effects (specified via TRIGGER) of this process on the object to be mastered contributes to the specification of the latter. In turn, the object to be mastered will constrain the master's behaviour.

After these models are specified, we want to answer the question: given some expectations on the states of the object to be mastered, how far does the master support the satisfaction of those expectations? We address this question in Section 5. Let us now present the formalism to the specification of the expectations.

4.3 Expectation-specification

The expectation-specification is the formal modelling of system requirements/expectations. It answers the question: how to describe the expectations over the life cycle of the modelling activity?

4.3.1 A/G contracts equipped with a pre-order structure on G

The inspiration from contracts lies on two main facts. On the first hand, they fit well as a basis to model expectations of a system. On the other hand they are easy to describe as rules yet they are supported by formal conceptual frameworks such as [9]. It turns out that they should enable to easily involve stakeholders while allowing to formally deal with expectations in the analysis.

The expectations are grounded on Assume/Guarantee (A/G) contracts. We follow [9] on a meta-theory of contracts.
A contract *con* for a system or a model is defined by $con(A, G)$ where

- A is the assumptions or the "valid environments" for the system or a part of the system.
- G is the guarantees or "the commitments of the component (the system or part of it) itself, when put in interaction with a valid environment.".
- A and G are built on the set of behaviours related to the system over a domain D not explicit in *con*.

Example: A system that realises a real division of x by y and assigns the result to z might conform to a contract $con\_div((x, y \in \mathbb{R} \text{ and } y \neq 0),(z := x/y))$. Meaning: if the system receives as inputs two real numbers x and y such that $y \neq 0$, it guarantees that z will be equal to x/y.

According to the meta-theory of contracts [9], *con* is *consistent*, if there exists a model that effectively implements *con* (satisfies G) for the assumptions A of *con*. *con* is *compatible* if there exists a non empty environment for *con*. See [9, Section VII] for more details on A/G contracts definitions and operations.

Continuing with the example of the real division, con_div is consistent because there exists a system that effectively takes 2 real inputs x and y, $y \neq 0$ and computes $z := x/y$. con_div is compatible because there exists 2 real numbers x and y, and $y \neq 0$.

Note that, a contract $con(A, G)$ for a system (respectively system models here) does not provide any information on how the system is implemented (respectively is modelled) but how it is expected to behave.

An expectation consists in expressing preferences on the states of the system given an assumption (A). Preferences are modelled by equipping a guarantee (G)

with a pre-order (i.e. a relation that is reflexive and transitive) structure. Since in A/G contracts [9], G is a set, we are no longer formally dealing with an A/G contract. Therefore, the term preferences (P) is more adequate here. Whence the extension of A/G contract to an *A/P expectation*.

Formally, an expectation consists of a couple

$$exp(A, P) \tag{4.6}$$

where both A and P are defined from a domain D. A is theoretically a formula of zeroth-order logic where atomic propositions are elements of D. P is a pre-order which the binary relation is defined on D. An element of D is a couple: (*object, state*) meaning the component *object* is in state *state*. Given $m = (o_1, s_1)$, we call s_1 the underlined state of m.

Example: Suppose m_1, m_2, m_3, m_4, m_5, m_6 and m_7 are elements of D. The table below gives two expectations.

Expectations		
Id	**A**	**P**
1	m_1	$m_2 \preccurlyeq m_3, m_2 \preccurlyeq m_4$
2	$m_5 \wedge m_6$	m_7
...	...	...

In practice, the expectations with Ids 1 and 2 stand for the following. When m_1 occurs or is true, the occurrence of the situations (or the propositions) m_4 and m_3, are preferred to m_2. And when $m_5 \wedge m_6$ is true, the occurrence of m_7 is preferred (m_7 should be true). It is clear that, the truth value of elements of D is obtained from the behaviour of the system they relate to. Whenever A is not given for an expectation, we suppose it is a tautology or it corresponds to all possible assumptions with regard to the behaviour of the system.

For the sake of simplicity, we consider that A only consists of atomic propositions in the following.

A basic question which arise is: do operations on contracts apply for expectations? The answer is certainly no, given the structure of P. Indeed the interpretation of the operations on a set (G) are not directly translatable on a relation (P). But we discuss the relevance of these operations for expectations and eventually define the relevant ones. In the following, we start by discussing the compatibility and consistency of expectations or requirements.

Compatibility—A basic way to check the compatibility of an expectation, like

in the spirit of a contract, is to verify that its assumption is valid according to the definitions of the state models it is related to. Statically, one must ensure that A is well-defined as a proposition. However, checking the consistency of some requirements would require to explore the state space (of the behaviour of the system) to check if some states are all reachable together. Indeed, if two situations (or propositions) are not defined from the same state model, the validity of an assumption that involves those situations might yield a reachability problem. For example, for the expectation with Id=2, $A = m_5 \wedge m_6$, compatibility means: is it possible for the system to be simultaneously in the state(s) underlined by m_5 and m_6?

As a result, an expectation *exp*(A,P) is *compatible* if there exists a valid environment (i.e. that makes A true) for *exp*.

Consistency—In the spirit of contracts, we might ask ourselves the question: what does "a model that effectively implements *exp*(A,P) (satisfies P)" mean? i.e., what does the satisfaction of the preferences mean? Before answering this question, it is important to check that P is well-defined.

Statically, one must ensure that P is a pre order and that P is not contradictory. P is contradictory if given $m_2, m_4 \in D$ and $m_2 \neq m_4$ whenever $m_2 \preccurlyeq m_4$ is defined for an expectation exp_1 we also have $m_4 \preccurlyeq m_2$ defined for exp_1 (or another expectation exp_2 such that the assumptions related to both exp_1 and exp_2 might be simultaneously true for the system they relate to).
Using the underlined directed graph of P, one only has to check that it is an acyclic graph or more precisely it does not contain a cycle of length greater than 1. Formally, P is "not contradictory" syntactically means that the pre-order P is antisymmetric i.e a partial order.

However, even if P is statically or syntactically well-defined, it might be the case that it is not possible, according to the behaviour of the system, to leave from the situation m_4 to m_2. This is rather a problem of feasibility of the expectations.

Now suppose P is well-defined, checking its satisfaction is clearly not a boolean question, unless one requires that all preferences are satisfied. But this latter requirement is a too strong assumption. Therefore, with respect to an utility function that allows to quantitatively evaluate the preferences, it would be possible to compute how far preferences are satisfied. This concern is addressed in Section 5.

As a result, an expectation *exp*(A,P) is *consistent* if P is not contradictory and

there exists a model for the assumption A, that effectively implements or achieves P at a given level of satisfaction with respect to a quantitative understanding of P.

In A/G contracts, it is possible to compute a global contract resulting from the combination of a set of contracts, by using, for example, either the conjunction or the composition of contracts [9, Section VII]. Let $con_i = (A_i, G_i)$ a contract and Con a set of contracts.

The conjunction of contracts $con_i \in Con$ noted $\bigwedge_i con_i$ is defined by: $\bigwedge_i con_i = (A, G)$ such that:

$$A = \bigvee_i A_i \text{ and } G = \bigwedge_i G_i \tag{4.7}$$

The composition of contracts con_i noted $\bigotimes_i con_i$ is defined by $\bigotimes_i con_i = (A, G)$ such that

$$A = (\bigwedge_i A_i) \vee \neg(\bigwedge_i G_i) \text{ and } G = \bigwedge_i G_i \tag{4.8}$$

Suppose the requirements of the system are defined in terms of contracts instead of expectations. The resulting guarantee $(\bigwedge_i G_i)$ for the conjunction and composition of all contracts is *de facto* not necessarily equivalent to the guarantee expected from the system. In fact, the conjunction of *non-compatible* guarantees (i.e., that could not together simultaneously be satisfied) does not make sense and should not be defined. This again yields a reachability problem.

Coming back to the expectations, the same problem might arise with the definitions of the conjunction and composition of expectations. Indeed, suppose we attempt to define the conjunction of expectations and we take the resulting preference to be a kind of conjunction of preferences. Following the previous paragraph it clear that such a resulting preference will not make sense.

The operations on expectations should be interesting if expectations were used in a compositional fashion for instance. Since it is not the case in this thesis, we do not deal with operations on expectations in this thesis. Anyway, it would be possible to define some operations on expectations by considering the operations (union, intersection and composition) on binary relations.

The fact that a contract (respectively an expectation) does not provide any information on how the the system is implemented, but how it is (preferably) expected to behave has central consequences.

As a first consequence, the feasibility, i.e. the answer to the question: "does it

exist some system models which satisfy the expectations?" is not easy to answer. On the other side, with the mapping of multiple models and their respective autonomy and possibly constraints on some models, it would be hard to think of realizability i.e. the existence of models and certain mappings that allow to satisfy the expectations. In addition, this could raise the question of completeness of expectations.

It turns out that feasibility is a trade-off problem which could imply either limbering up the expectations or changing the system models and their mappings. Furthermore if we assume we are in presence of autonomous processes, trade-offs are not always possible. In this situation, feasibility would be mainly defined up to process models and their effects on state models. Since satisfiability is sufficient but not necessary to deduce feasibility, undesirable behaviours might help in dealing with such trade-offs once they are allowed.

4.3.2 Related research

Contracts theories have been used for component-based design, layered design and platform design [9]. In particular they have been identified as suitable for open systems for which the context of operation is not fully known in advance. In this thesis, we equip guarantees with a preorder structure making them preferences and we define the notion of Expectation instead of Contract. Additionally, this pre-order is exploited by the analysis procedure, by introducing an utility function that makes the qualitative preferences quantitative (see Section 5.1).

The way the expectations are used in MODEF is different from the traditional use of contracts in system design. In fact, in system design, contracts are generally used to specify the interactions between (heterogeneous) components or different viewpoints [8] [72] [26] and this, essentially for software and cyber-physical systems. We use expectations to map or correlate system expectations on system models. In the spirit, the use of contracts here is close to the first uses in programming where preconditions (assumptions here) and postconditions (preferences here) are defined for programs (system models here) see e.g., [49] [70].

In the field of model checking [22] [6], requirements are generally expressed as temporal properties. In fact, one advantage of temporal properties is that they are both formally and informally understandable in the sense that natural language (informal requirements) could be translated to temporal logic [97]. A/G contracts (then expectations) are also formally and intuitively understandable. Their consistency and compatibility can be verified. We will come back on model checking in

the next Section.

4.4 Conclusion

In this section we presented the principles and exploration semantics of models, then how models are related via a mapping. Finally we present a formalism to specify the requirements or expectations on the models. We now need to analyse those models against the requirements i.e. the expectations. More importantly, the results of the analysis have to be exploited to provide stakeholders with relevant data: we address this concern in the next section.

What is achievable and what can happen with the modelled system?

This section elaborates the fifth (Analysis of models) and sixth (Providing useful feedbacks to the stakeholders) steps of MODEF (see Figure 1.1). Therefore, we present the analysis algorithms (Section 5.1) for the analysis of system models against the expectations. Finally, we discuss the possible exploitations of analysis' outputs (Section 5.2) and present two cases studies (Section 5.3).

5.1 Analysis of system models against expectations

The fourth step of the MODEF aims via its results to provide the means to effectively improve the way the system is operated. Given the models of the system and its requirements and the changes thereof, the stakeholders of the system should be able to permanently figure out the better ways to operate the system. What do we mean by better ways to operate the system? To answer this question, we first clearly state the challenge at stake, then we elaborate on the procedure to solve it and the expected benefits.

5.1.1 General problem

A system (S) and its environment (E) are modelled by structure models (AM) and deterministic state models (SM) for S, and process models (PM) for the behaviours of S and E. S is subject to some expectations (R). A mapping (MG) captures the actions (or the effects) of PM on SM of AM. The models AM, SM, PM, R and MG are characterised and defined in Sections 4.1; 4.3 and 4.2 respectively. Additionally PM might be subject to some constraints (C) for example the cost of the tasks within processes.

So far we have not talked about the "environment" (E) of the system. We introduce the environment of the system to be [2]: "a set of elements and their relevant properties, which elements are not part of the system but a change in any of which can produce a change in the state of the system. Thus a system's environment consists of all variables which can affect its state. External elements which affect irrelevant properties of a system are not part of its environment."
It is also argued in [2] that a system and its environment are relative to an observer, consequently they can be conceptualized in different ways.

Therefore, we split what we have so far called the system into a proper system (henceforth called system) and its environment which together with the system yield the closed system. The environment is autonomous and almost not controllable part.

This choice is consistent with the fact that, while models might be expected to reach some predetermined states, different modelling tasks and their scheduling are possible to reach those states. Therefore, the proper system here becomes the structure models (AM) and their states while its environment is the modelling tasks. As a result, the closed system is the modelling activity abstracted and modelled by AM, SM, PM, MG, R and C.

The global state, henceforth called a point, of the closed system or its state space is basically given by pt(cst, cev). cst is an array of states of concurrent structure components of S. cev is an array of events of concurrent process components of E (and possibly S). Each event in cev is additionally annotated with a chronological ordering and a status. The status is either *proposed* or *accepted*. We come back below on how this status is determined and evolves.

We call the initial point and we note InitialPoint a point from which the closed system is initialised.

We want to generate the future possible points up to a given stop criterion noted StopCriterion.
The general problem is given by Pb(AM, SM, PM, MG, R, C, InitialPoint, StopCriterion). The main question Pb helps to formulate is:

Q1: How to generate the possible future points starting at InitialPoint up to StopCriterion? Which points are, with respect to R and C, admissible and less admissible ones?

5.1.2 General principles for a solution

The answer to the question Q1 basically requires computing the possible behaviours (starting from InitialPoint) of the closed system. Furthermore, by speaking of the admissibility of the behaviours, we basically need to compute the "distance" between behaviours in the state space of the system. This concern raises a second question: Q2: How to define and compute that distance?

The general synthesis procedure to solve the question Q1 is summarized on Figure 5.1. Before we present this procedure, let us discuss the question Q2 whose the answer is necessary to address the question Q1.

The answer to the question Q2 is a means to evaluate these behaviours in order to figure out the preferable ones. Indeed the behaviours that best match expectations (R) and follow constraints (C) have to be preferred than those that do not. In order to exploit the expectations (R) in the exploration procedure, we need to translate the qualitative preferences into quantitative values. Indeed, the preferences are rather relative to the pre-ordering of atomic propositions.
Suppose we have an appreciation (or utility) function u that allows to map qualitative preferences to a domain with numerical values. u is such that, whenever $m_4 \leq m_5$ and $m_5 \leq m_6$ for a given assumption A, then $u(m_4) \leq u(m_5)$ and $u(m_5) \leq u(m_6)$ and $u(m_4) \leq u(m_6)$ i.e., u preserves the preorder structure of preferences. Such an appreciation function is therefore used in the synthesis procedure as the basis to compute an aggregated appreciation of a given point in the state space. Furthermore, since we require the (HFSM of the) system to be deterministic, a transition from a source point to a target point in the state space is evaluated with the aggregated appreciation of the target point. In fact, this target point must be uniquely determined by that source point and the outgoing arrow corresponding to the transition.

Similarly, by using the constraints (C) related on PM, a cost could be associated with an arrow from a point Pt_1 to a point Pt_2 in the state space of the system.

Returning to the question Q1, the aim of an answer thereof is to synthesize the behaviours of the closed system given the input models. The synthesis procedure basically involves the parallel explorations of both PM and SM of AM. These explorations influence mutually via MG (see Section 4.2). Therefore, the co-exploration corresponds to the parallel explorations that influence mutually. Although there is

an interplay between PM and SM, as indicated in Section 4.2, the processes related to PM mainly play the role of a "master" whereas the HFSMs related to SM (of AM) mainly play the role "the object to be mastered" (see Section 4.2).

On the other hand, an answer to the question Q2 will be used to compute the distance between behaviours and to eventually select some acceptable behaviours among the reachable ones. The acceptable behaviours are the ones that do not violate StopCriterion.

The general synthesis procedure depicted on Figure 5.1 works as follows. The rectangles with a white background are inputs and outputs of the procedure and its sub-procedures. The rectangles with a blue background are sub-procedures that transform inputs into outputs. The rectangles with a dashed border are (intermediary) inputs and outputs generated by the sub-procedures. The arrows indicate the dependencies among inputs, outputs and sub-procedures.
In the following, by speaking of the content of a rectangle, we refer to a representation of that content. The meaning of rectangles from the top to the bottom of the

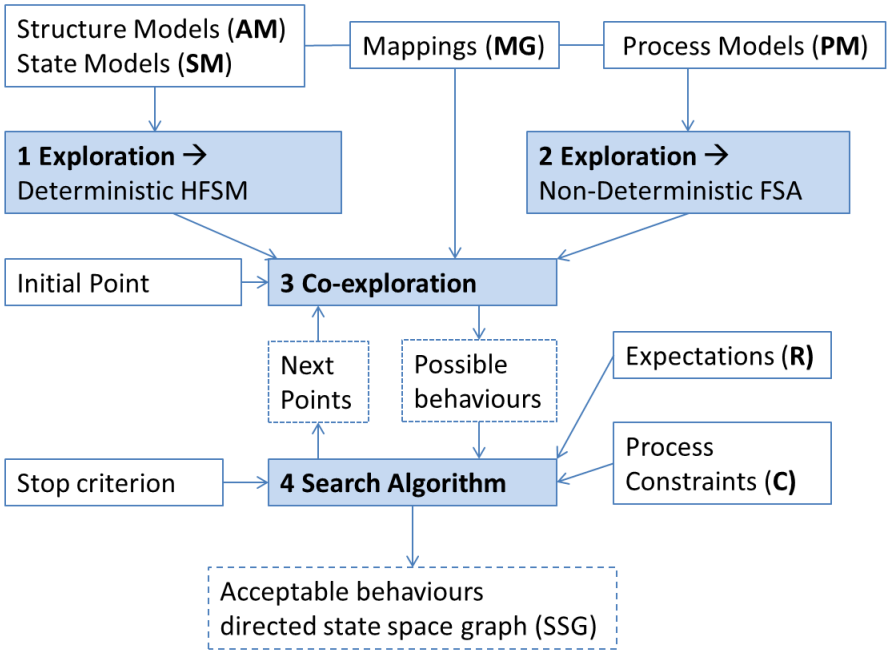


Figure 5.1: General synthesis procedure

Figure 5.1 is:

- The first 3 rectangles are the basic inputs: structure models (AM) and state models (SM), mappings of models (MG) and process models (PM).
- It is necessary to explore the models to compute the behaviours of the system

they represent, whence the upper two blue rectangles (numbered 1 and 2). The first at left consists in the primitives for exploration of SM (of AM) under a deterministic hierarchical finite state model—HFSM (see Section 4.1.3). The second at right consists in the primitives for exploration of the generator (a non-deterministic state automaton) associated to PM (see Section 4.1.2).

- Having the primitives mentioned above and given SM and PM and the mappings (M), it is possible to generate the possible behaviours of the system from an initial state (either at the first iteration using InitialPoint or at other iterations via NextPoints), whence the next blue rectangle (entitled 3 Co-exploration) and its dependencies.

- Among the reachable or possible behaviours, those that do not make StopCriterion true are acceptable. Indeed, the next blue rectangle (entitled 4 Search Algorithm) takes as input the possible behaviours, a stop criterion, R and C. Note that the answer (e.g., the appreciation function u) to the question Q2 is intended to be problem-dependent and has to be used in the Search algorithm to differentiate and compare the research directions in state space of the behaviours. On the other hand, StopCriterion allows to prune some possible behaviours.
  Search Algorithm produces an output (NextPoints) that is the next possible starting points of the procedure Co-exploration. At the same time, NextPoints belong to the acceptable behaviours. Co-exploration can be therefore resumes if the set NextPoints is not empty.

Now we need to instantiate the main sub-procedures: Search Algorithm and Co-exploration. Other primitives for the exploration of SM and PM, are discussed in Sections 4 (their principles and exploration semantics), 5 (their use independently of a modelling tool with the support of a categorical architecture), 6 (their implementation).

5.1.3 Main sub-procedures: Coexploration and a Search Algorithm

Suppose the primitives for the exploration of SM (of AM) and PM are available through the interfaces *smInt* and *pmInt* respectively. In the following, by speaking of *smInt* and *pmInt* we refer to the boxes labelled "1 Exploration → Deterministic HFSM" and "2 Exploration → Non-Deterministic FSA" respectively on Figure 5.1.

As described above (Section 5.1.2), the two main sub-procedures interact in a closed loop in the general synthesis procedure. We named the Search algorithm MinBest to MinWorst–MBMW. MBMW selects on the fly the behaviours (whatever their appreciation) that are at a minimal (against R and C, using an answer to the question Q2) distance (throughout the directed arrows between points) of Initial-Point. We present in the following the algorithms Coexploration (Algorithm 1) and MBMW (Algorithm 2) corresponding to the Co-exploration and Search Algorithm sub-procedures respectively on Figure 5.1.

5.1.3.1 CoExploration

We have already introduced above the inputs MG, $smInt$ and, $pmInt$ of the coExploration algorithm. $CurPoint$ is either the InitialPoint or one of the points in NextPoints both visible on Figure 5.1.

We recall that $smInt$ and $pmInt$ are the interfaces that allow to explore a set of concurrent independent process models and a set of concurrent independent state models respectively. The exploration semantics of those models (or components) are presented in Section 4; but only for one component and not for a set of components. That exploration yields for each component an automaton that is equivalent to a directed graph. The automaton is equivalent, in the case of the process model, to the generator associated to the process model. In the case of the state model, the automaton is equivalent to the corresponding flattened part of the HFSM generated by its exploration. Note however for instance that, although an HFSM is translated to a flattened finite state machine, it is convenient to deal with the composite structure in the exploration since not all states would be necessarily enumerated.

The directed graph (henceforth called the exploration graph) corresponding to the exploration of a set of concurrent independent components is given by the asynchronous composition of automatons underlying the exploration of each component. Formally, given n ($n \in \mathbb{N}_+$, let $n = 2$ for convenience) automatons $G_1(Q_1, \Sigma_1, \delta_1, q_{0_1}, Q_{m_1})$ and $G_2(Q_2, \Sigma_2, \delta_2, q_{0_2}, Q_{m_2})$ as defined by (4.2), the result $G(Q, \Sigma, \delta, q_0, Q_m)$ of the asynchronous composition of G_1 and G_2 is such that $Q = Q_1 \times Q_2$, $\Sigma = (\Sigma_1 \times \Sigma_2) + \Sigma_1 + \Sigma_2$, $Q_m = Q_{m_1} \times Q_{m_2}$, $q_0 = (q_{0_1}, q_{0_2})$, if (q_1, q_2), $(q_1 \in Q_1, q_2 \in Q_2)$ is a node or state in Q then δ is given by:

- $(q_1', q_2') = \delta((q_1, q_2))$ if $q_1' = \delta_1(q_1)$ and $q_2' = \delta_2(q_2)$ are both defined
- $(q_1, q_2') = \delta((q_1, q_2))$ if $\delta_1(q_1)$ is not defined and $q_2' = \delta_2(q_2)$ is defined

- $(q_1', q_2) = \delta((q_1, q_2))$ if $\delta_2(q_2)$ is not defined and $q_1' = \delta_1(q_1)$ is defined

It is easy to extend the composition with $n > 2$. Now we present the principles of the co-exploration semantics before commenting the lines of the CoExploration algorithm.

Given an initial point pt(cst, cev), cev is generated by pmInt, cev is the current state of SM of AM available from smInt. pt is passed through the co-exploration module (the box with a grey background on Figure 5.2). This module works as follows.

The generated events (arrow (1) on Figure 5.2) get the status *proposed*. The status *proposed* means: something might happen in the process.

The events that have the status *proposed* and that are involved in MG are proposed (arrow (2) on Figure 5.2) to smInt to trigger the fireable transitions.

After the (possible) firing of transitions, the events not used or rejected by smInt are reconsidered (arrow (3) on Figure 5.2) and the ones that were used get the status *accepted*. Whenever an event is not involved in MG it gets the status *accepted*.

The status *accepted* means: something happens in the process.

Then, the events that have the status *accepted* are the only valid ones considered (arrow (4) on Figure 5.2) to try to evolve the processes in pmInt.

The events that are not used by smInt and pmInt after arrows (2) and (3) and (1) and (4) respectively on Figure 5.2 maintain their status.

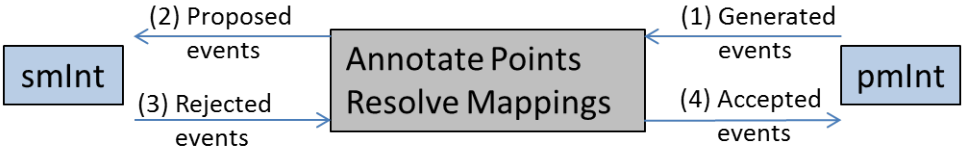


Figure 5.2: The principles of the co-exploration

We now are ready to comment the co-exploration algorithm (Algorithm 1).

Line 1: Given the active states and events in $CurPoint$, MG and smInt are used to compute the fireable transitions from the state cst. These fireable transitions are involved in mappings such that mapped events belong to the set of current events in cev with the status *proposed*.

Lines 2, 3: If no transition is fireable, $CurPoint$, $pmInt$ and M are used for possibly evolving the processes. $evolveProcesses$ uses the *accepted* events in cev to try to generate (using pmInt) new alternative ones i.e. some adjacent nodes to cev in the exploration graph generated by PM.

Algorithm 1 coExploration

Inputs : MG, $CurPoint$, $smInt$, $pmInt$
Output: $PossibleBehaviours$

```
1:  Fireable_Trans ← fireableTrans(smInt, MG, CurPoint)
2:  if Fireable_Trans = ∅ then
3:     PossibleBehaviours ← evolveProcesses(pmInt, MG, CurPoint)
4:  else
5:     PossibleBehaviours ← fireTrans(Fireable_Trans, amInt, CurPoint)
6:     if isDeterministic(CurPoint, PossibleBehaviours) is false then
7:        Report on non-determinism from CurPoint
8:        PossibleBehaviours ← ∅
9:        return PossibleBehaviours
10:    end if
11: end if
12: if PossibleBehaviours = ∅ then
13:    Report on CurPoint //End, Deadlock etc.
14: end if
15: return PossibleBehaviours
```

Lines 4, 5: If some transitions ($Fireable\_Trans$) are fireable, they are used in $fireTrans$ to evolve (via smInt) the active states i.e. by moving on the exploration graph generated by SM of AM.

Lines 6, 7, 8, 9: Whenever new transitions are fired, they should lead the system to a single state otherwise the system is not deterministic. Following a non-deterministic nature of the system computed by $isDeterministic$, an empty set is returned.

Lines 12, 13: If it is not possible to generate new points, $CurPoint$ is either a final point (process end) or deadlock point (process blocked or sink state).

5.1.3.2 A Search algorithm: MBMW

We assume the general synthesis procedure substitutes to MBMW (Algorithm 2) by making CoExploration algorithm and its input parameters the input parameters of MBMW. The additional input parameters of MBMW are:

- *nodeScore* is a function which takes as input a point in the state space and R (the expectations) then computes an aggregated score associated to that point. Recall from the answer of Q2 in Section 5.1.2 that, using the deterministic nature of HFSMs, such a score can be associated to all incoming arrows of that point.

- *edgeCost* a function which takes as input an edge in the state space and C (the

constraints on processes) and computes a cost process to go from the source point to the target point of the edge.

Therefore, the cost or costs of an edge in the state space is given the applications of *nodeScore* and *edgeCost* resulting in a cost vector of $\mathbb{R}^n_+, n \in \mathbb{N}, n > 0$. The value of the first component of this vector is the result of an application of *nodeScore* and the value of its other component(s) is the result of an application of *edgeCost*. Indeed, on the first hand, from an algorithmic point of view, the *nodeScore* can be understood as a cost. On the other hand, the cost obtained from *edgeCost* might be composed of values (time, money, etc.) with different units.

The interest of this vector is to be able to compute a path cost in the state space to eventually obtain the distance sought after with Q2.

The outputs of MBMW are:

- *SSG* is the directed state space graph of the acceptable behaviours.
- *ScoreAndCost* is the map of points' scores and edges' costs from InitialPoint.

MBMW implements an Uniform-Cost Search (UCS) algorithm [80]. It is an algorithm similar to the Dijkstra's shortest path algorithm [28]. It is a special case of the A* algorithm introduced in [48], itself a special case of a branch-and-bound algorithm [71]. The effect of the UCS algorithm here is to always select (based on scores' values and possibly processes' costs) the best (minimal) point(s) at the next iteration in the discovering of SSG. We discuss UCS in Section 5.1.3.3.

We now are ready to comment the main lines of MBMW (Algorithm 2) we add compared to a basic UCS algorithm.

Line 2, 3: These lines are about the initialisation of the map *ScoreAndCost* which associates to each explored point, the smallest path cost necessary to reach it. This cost is $\overrightarrow{0}$ at the initial point and at the initialisation. Since at initialisation InitialPoint has no incoming arrows, the value of *nodeScore* on InitialPoint does not matter at initialisation.

Lines 9, 10, 11: Whenever at a given point, the *stopCriterion* is true, the point is no further expanded towards discovering of SSG or the algorithm has to stop. At this line, the consistency and compatibility of R is also computed (see Section 4.3).

Lines 12: The call of *coExploration* is the means to generate the *CurPoint*'s neighbors i.e. its successors in SSG.

Line 24: A test to check if *Cur* (see Line 23 of Algorithm 1) dominates *Alt* (see Line 17 of Algorithm 1). Such a dominance relation must be defined.

Algorithm 2 MBMW

Inputs : $smInt$, $pmInt$, MG, $coExploration$, $InitialPoint$, R, C, $StopCriterion$, $nodeScore$, $edgeCost$

Outputs: SSG, $ScoreAndCost$

1: Add $InitialPoint$ in SSG
2: $Score \leftarrow nodeScore(InitialPoint, R)$
3: $ScoreAndCost \leftarrow (InitialPoint \mapsto (Score, \overrightarrow{0}))$
4: $add\_withPriority(InitialPoint, ScoreAndCost, ToVisit)$
5: $VisitedPoints \leftarrow \emptyset$
6: **while** $ToVisit \neq \emptyset$ **do**
7: $CurPoint \leftarrow extract\_min(ToVisit, ScoreAndCost)$ //Note that $CurPoint \in NextPoints$
8: $VisitedPoints \leftarrow VisitedPoints + \{CurPoint\}$
9: **if** $satisfy(CurPoint, StopCriterion)$ is true **then**
10: report on CurPoint; **goto** 7:
11: **end if**
12: $PossibleBehaviours \leftarrow coExploration(CurPoint, smInt, pmInt, MG)$
13: **for** Each $NextPoint$ in $PossibleBehaviours$ **do**
14: $Score \leftarrow nodeScore(NextPoint, R)$
15: $EdgeCost \leftarrow edgeCost(CurPoint, NextPoint, C)$
16: $PreviousScoreAndCost \leftarrow ScoreAndCost(CurPoint)$
17: $Alt \leftarrow PreviousScoreAndCost + (Score, EdgeCost)$
18: **if** $NextPoint \notin VisitedPoints + ToVisit$ **then**
19: $ScoreAndCost \leftarrow ScoreAndCost + \{NextPoint \mapsto Alt\}$
20: $add\_withPriority(NextPoint, ScoreAndCost, ToVisit)$
21: **else**
22: **if** $NextPoint \in ToVisit$ **then**
23: $Cur \leftarrow ScoreAndCost(NextPoint)$
24: **if** $Cur > Alt$ **then**
25: $ScoreAndCost \leftarrow ScoreAndCost + \{NextPoint \mapsto Alt\}$
26: $change\_Priority(NextPoint, ScoreAndCost, ToVisit)$
27: **end if**
28: **end if**
29: **end if**
30: $updateSSG(CurPoint, NextPoint, ScoreAndCost, SSG)$
31: **end for**
32: **end while**
33:
34: **return** SSG

Line 30: SSG is updated every time a new point or a new edge is discovered. Indeed, the update either consists in adding a node in SSG, creating a edge or, both.

Now we need to show that at the end of MBMW, i.e. the end of the general procedure regarding: the problem P(AM, SM, BM, MG, R, C, InitialPoint, StopCriterion), the following statement holds:

SSG stores all the minimal paths from InitialPoint to all acceptable points i.e. the points that are reachable and whose predecessors do not make StopCriterion true.

Before we go through the proof which is straightforward, note that a judicious choice of the termination criterion StopCriterion (max-depth, max-score, max-cost, computing time, etc.) ensures that MBMW eventually will stop.

5.1.3.3 Proof and complexity of MBMW

The state space generated by the co-exploration is a directed graph (SS) possibly infinite but gradually discovered. In *SSG*, nodes are points and edges are characterized by the scheduling (or the paths of execution) of events from PM with associated states of SM of AM.
In fact, the procedure MBMW applies the UCS algorihtm on-the-fly while SS is discovered. Note that MBMW has no effect on SS. It allows to rather select a subgraph i.e. *SSG* of SS based on R and C and StopCriterion.

The UCS algorihtm is an optimal, uninformed (or blind) search algorithm [80]. Therefore, the minimality (given completeness) of saved paths (in *SSG*) is guaranteed by UCS which always expands the node with the smallest path cost in the exploration of nodes in *SSG*.

Completeness means: if the maximum total path cost reached for a path in *SSG* is C^* then all acceptable nodes that have a path cost C_x, $C_x \leq C^*$ will be discovered by MBMW after a finite number of iterations, provided that: the branching factor bf of *SSG* is finite and every edge cost in *SSG* is superior than $\varepsilon > 0$.

The branching factor of *SSG* is the (average) out-degree of each node.

ε is the minimum value of an edge cost to avoid the algorithm deadlocks in an infinite loop which implies lack of completeness. The input functions *edgeCost* and *nodeScore* of MBMW and therefore the map *ScoreAndCost* has as codomain positive real vectors should guarantee that minimum. This concludes the proof.

As a result, the complexity (time and space) of MBMW is the complexity of UCS with the input parameter SS. This complexity is given by $O(bf^{1+\lfloor C^*/\varepsilon \rfloor})$ [80].

This complexity of MBMW is, in the worst case, the one necessary to generate SS. It is not possible to do better without introducing an heuristic (thus making the algorithm informed) that systematically guarantees the optimality and completeness of the generated graph (SSG) with respect to the explored region of SS. In this latter case, the worst case complexity remains exponential. Suboptimal solutions would become relevant depending on the application domains.

As argued earlier (see Sections 1, 3) in this dissertation, the SoSoM is built of autonomous and possibly partly independent stakeholders. This makes the evolutions or changes of behaviours of E unpredictable although continuously specified via PM. In turn, the system's behaviour is never definitely fixed.

Insofar we assume a long-term critical system whose evolution is typically over periods of several days to months (see Sections 1, 3) with creative, iterative and almost non-automatable behaviours, we find it better to guarantee optimality on short-term periods. This amounts in repeatedly executing MBMW throughout the life cycle of the system. Moreover, although the SoM and SoSoM are complex, the size of the state space of their behaviours should typically be reasonable compared to that of other kinds of systems, e.g., cyber physical systems.

Henceforth, C^*/ε could be considered here as the value that defines the level and cost of agility for the operation of S. A theoretical understanding of C^* might be: the higher C^* is, the lower agility will be.

5.1.4 Discussion

The problem formulated in Section 5.1.1 and the subsequent proposed solution in Section 5.1.2 could be compared or at least have similarities with model checking [22] [6] and systems synthesis [76] [74].

Model-checking—State space analysis is generally carried out for verifying finite-state concurrent systems. Techniques for verification are mainly: simulation, testing, deductive reasoning and model checking. Model checking is one technique for automatic verification of finite-state concurrent systems. A model-checking process could consist in three main steps [6]. (1) Model the system (S) with a description language and express system's properties, requirements or specifications (R) with a property specification language. (2) Check the validity of P systematically in S. (3) Analyse a violated property or an out of computer memory.
The basic description of S is a state-transition model whereas that of R is a temporal logic formula. Although model checking is automatic, it usually faces the state

explosion problem and the problems of computation cost and computer memory. Advanced techniques such as abstraction, binary decision diagrams, partial order reduction, compositional reasoning and, probabilistic exploration have been developed for addressing such problems even though the memory problem remains [6]. However, model-checking is well-suited when analytic methods are difficult or impossible to apply in practice [97]. It has several advantages compared to the latter, namely, it is fast, executable with partial requirements, it provides counterexamples and does not require proof. Nonetheless, the more there are data variables, the challenging model checking will be.

Systems synthesis—The standard synthesis ("Be Correct" [14]) consists in restricting the actions of the system so that when the environment of the system is known (making the system closed), the system will always satisfy a given property. Depending on the hypothesis on the environment (controllable or not controllable) and on the system (complete or incomplete specification), the synthesis can be reduced to verification or model-checking [95, Chapter 9]. On the one hand, when the specification of the system is fixed independently of its environment the synthesis can be reduced to verification of the closed system. On the other hand, when the specification of the system is fully determined only up to definition of its environment, the synthesis can be reduced to model-checking. However, when neither is the environment fully determined nor is the specification of the system complete, one rather seeks for strategies such that the closed system anyway satisfies a given property [95, Chapter 9].

With respect to the assumptions we made for S and E, which are close to the assumptions made in [14], especially with the involvement of a human operator in the functionality of the system, the operation of the system is not likely to be always optimal. We believe that "Be correct" (i.e. "everything must be ok") in this latter situation is a too strong assumption.
Since some pioneering works [76] [74] on system synthesis, there is an active research on the synthesis of reactive systems (systems that react as the result of the actions of their environment on them); the Reactive Synthesis Competition [54] is an example of that manifestation.

Two usually encountered invariants of synthesis algorithms are: the use of temporal logics as the property specification language, and the search of a wining strategy or a counter strategy. It is also the case in the field of model checking [22] [6]. Furthermore, their application mainly targets the software and cyber-physical systems

while we are mainly concerned with systems where the role of human operators is important. In this thesis, requirements are specified via Assumption/Preferences expectations. We compared A/P expectations and a boolean property based on temporal logics in Section 4.3.2.

Recently, quantitative objectives, i.e. the adoption of a non-binary satisfaction of a specification have been introduced in [13]. Moreover, the proposed approach aims to synthesize a system with respect to a boolean specification complemented with quantitative aspects given by a weighted automata. As argued in [13], the satisfaction of a specification could be evaluated on a scale with various degrees instead of a binary one.

In MBMW the measure of "goodness", i.e., how good an implementation (behaviours of the system here) is with respect to a given specification corresponds to the cost (determined by R and C) of a path here. Indeed, MBMW computes points (from bad to good) that are reachable at a minimal distance of the initial point until the stop criterion becomes true. Unlike [13] and almost all the approaches inspired by the latter, in this thesis, a specification is defined with A/P expectations instead of a temporal property or automata. Furthermore, the quantitative nature of objectives (preferences here), is derived from qualitative objectives and the appreciation function and possibly constraints (cost, time, etc.) on processes. Additionally, this quantitative aspect is taken into account via MBMW (which could be replaced by another search algorithm from operations research) applied on the discoverable state space of the closed system.

We do not deal with quantitative languages [20], weighted automata [31], simulation distances [78] [39]. In addition, we do not consider games (in the sense of Game theory) where the environment is opposed to the system like encountered in most popular synthesis approaches [14]. Our consideration is surely justified by the fact that we do not deal with "a controller" and "an object to be controlled" in the sense of controller synthesis, but instead we deal with "a master" and "an object to be mastered" see Section 4.2 for the explanation. Finally, the general synthesis procedure substituted to MBMW, synthesises the behaviours of the closed system. We could also argue that the closed system is verified against expectations.

As we have seen in the MBMW algorithm, some input parameters apart the input models, need to be configured or set up before a run of MBMW. We discuss an exploitation of MBMW in the next section.

5.2 An exploitation of MBMW

In this section, we discuss some input parameters and the exploitation of SSG necessary to take advantage of MBMW in practice. These parameters are: u, *edgeCost*, *nodeScore*, *StopCriterion*. Some of them will be used in the case studies in Section 5.3.

5.2.1 Setting up input parameters

These input parameters are given as follows.

• *StopCriterion*: It corresponds to a maximal path cost and/or a maximal depth in SSG and/or a maximal number of cycles authorized during the co-exploration.

• *edgeCost*: It gives the total process cost related to the resources consumed by an arrow in SSG. In this thesis, this cost is assumed to be a single business value. But it is not considered in the exploitation of MBMW because we did not especially define the resources allocated to the tasks inside processes. We assumed this business value is the same for all tasks, making therefore this cost irrelevant in the exploitation of MBMW. Indeed, for such a cost to be relevant, aggregation and addition operators related to it must be defined. Nonetheless, we discuss the expected impacts of *edgeCost* on MBMW in Section 8.

• u: This appreciation function maps preferences P of expectations on a numerical domain. In this thesis, u maps each proposition of P to the set $\{0, 1, 2, 3, 4, 5\}$ (or generally $j...j+5$, $j \in \mathbb{Z}$). The elements of this set qualitatively mean in ascending order: the worst (0) to the best (5). This choice of u reduces the diameter of an underlined directed graph of a preference to a maximun of 5. Such a reduction also translates via u, the preorder structure of P into a structure of order. Finally, this amounts to rank atomic propositions used to define P. As a result, the highest level of preferences corresponds to 5 (or $j+5$).

• *nodeScore*: It defines the magnitude of each rank and the aggregation of their corresponding values using u. We therefore define *nodeScore* as follows. Assume for a given R, and a point pt in SSG, there are atomic propositions associated to A (Assumption) that are possibly true (with respect to pt). And let N_0, N_1, N_2, N_3, N_4 and N_5 the corresponding numbers of atomic propositions in P (related to A via an expectation) mapped to 0...5 respectively. Let $POINTS$ be the set of nodes in the graph SSG. Then *nodeScore* is defined as follows.

$$nodeScore : POINTS \times 2^R \rightarrow \mathbb{N}^5 \xrightarrow{aggr} \mathbb{R} \tag{5.1}$$

given by $(pt, req) \mapsto NS(N_0, N_1, N_2, N_3, N_4, N_5) \mapsto s$, where $aggr : \mathbb{N}^5 \rightarrow \mathbb{R}$ is an aggregation function.

s and NS are scores. The former is an aggregated score and the latter a per rank score, where N_j corresponds to the rank j. We recall from Section 5.1.3.3 that, s must be such that $s \geq \varepsilon > 0$ for all function $aggr$.

The importance of $aggr$ and s is as follows. We want to associate to each path in SSG, a cumulative score in order to obtain a priority for each node, which allows to define a total ordering ($\leq$) among the candidate nodes to explore. The lower is the cumulated score of the nodes along a path in SSG, the higher is the priority of the nodes on that path (see the line 7 of the Algorithm 2). Eventually, all nodes where the cumulative score overcomes a given max score are pruned.

Arguably, the question of how $aggr$ is defined is important and must be tailored with respect to the application case. For example, there exists several possibilities to scalarize (max, min, min-max, a mean, etc.) the vector NS. Another possibility is to avoid scalarization and directly deal with vectors or rather a combination of both. Besides, the way (by counting) N_j are obtained might also be questioned since different components of a system could have different importances. But this is not relevant with respect to the input models considered in MODEF, indeed no importance factor is defined for the components.

5.2.2 Exploitation of SSG

The exploitation of SSG aims at extracting from SSG the relevant data related to the behaviour of the closed system. We discuss how we currently exploit this rooted directed graph.

We define the color of a node in this graph as follows. Suppose given the aggregated score $(N_0, N_1, N_2, N_3, N_4, N_5)$, of a node pt in SSG, the color of pt is an integer a defined as follows: if $N_0 \neq 0$ $a := 0$ else if ... else if $N_5 \neq 0$ $a := 5$.

For each node pt in SSG, we additionally associate another color b which is the highest color from the successors of pt in SSG. As a result, we can associate to each node in SSG a couple (a, b). This allows us to tailor SSG as follows. Whenever at a given node pt, b is not greater than a given value, say x, it might not be useful to look throughout the rooted sub-graph that has as root the node pt. This allows to prune SSG to get an exploitable graph.

Example. Suppose given the graph on Figure 5.3. A node in this graph is labelled $a|b$. The rooted sub-graphs represented by the four triangles on Figure 5.3 are such that, at their root node, we have $a \leq b$.

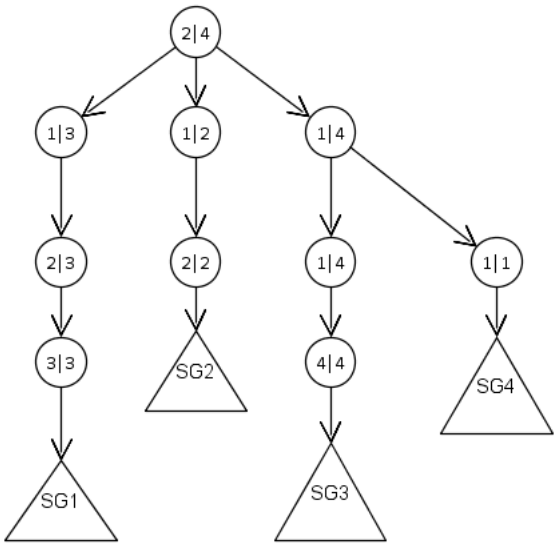


Figure 5.3: Tailoring of SSG

As a result, it is easy to say after the aforementioned calculations, whether from InitialPoint up to the leaf nodes reached in SSG, it will be possible or not to leave a node with a given color. And if it is not possible, the paths going from InitialPoint to the nodes where no improvement is no longer possible are extracted from the tailored SSG.

Just like the information the data (a, b) convey for each node in SSG, other data might be computable (using graph algorithms) to extract relevant information from SSG. For instance, the minimum value of the colors of nodes in a path in SSG.

It follows from the foregoing that, we can provide stakeholders with two kinds of data or results: (R1) either everything would be ok or (R2) something might not be ok during the operation of the system modelled with the input models. For instance, by using the data a of the leaf nodes in SSG or the tailored SSG, if all the colors (i.e. a) are greater than a given value x then provide (R1) else provide (R2).

Apart from the exploitation carried out with SSG, the flattened automatons generated by the exploration of the process and state models are available separately. They might be useful for diagnostics purposes.

5.3 Case Studies

In order to make the principles concrete, we start with a first case study dealing with a common system: a Supermarket Entrance System. Then, we will look at a SoM, which is analogue by the principles but relates to the System of Modelling.

In both cases, we present the general problem Pb(AM, SM, PM, MG, R, C, InitialPoint, StopCriterion) (see Section 5.1.1) and present the exploitation of the analysis. The first case study will be mainly about to demonstrate how the complexity could be effectively mastered with MODEF whereas the second case study will focus on a detailed exploitation of analysis. Indeed, the first case study is not the main kind of systems considered in this thesis (i.e. SoM and SoSoM), but it is useful for demonstration purposes.

For both cases, we define $aggr(N_0, ..., N_5)$ to be equal to $\Sigma_{i=0}^{5} N_i * 10^{5-i} / \Sigma_{i=0}^{5} N_i$. The idea is to give a high priority to the points with good preferences while at the same time being able to control the maximum score authorized along a path.

As mentioned earlier, C is not taken into account. We also associate (see Figure 5.4) the usual colors to the color of the nodes in SSG: green to 5, yellow to 4, orange to 3, red to 2, black to 1 and grey to 0.

Figure 5.4: The colors associated to nodes in SSG

5.3.1 Maintenance of a Supermarket

The first case study is about the maintenance of a Supermarket, where one wants to be sure that it will be possible to well maintain the supermarket during its lifecycle. "Well" in the sense that expectations on the states of the components of the supermarket could be met. For example, it could be expected that during the carrying-out of a maintenance procedure, some states of the supermarket occur rather that other states in some situations.

Following the steps 1, 2 and 3 of MODEF, we come up with the data AM, SM, PM, MG, R where some illustrative examples are as follows.

The structure (an element of AM) of the supermarket's environment is depicted on Figure 5.5. The state and process models (elements of SM and PM) of the supermarket's entrance system are depicted on Figure 5.6 and Figure 5.7 respectively.

Some of the components of these models decompose into sub-components. For example, the supermarket's entrance system decomposes into 4 *Entrance Systems*, a *Fire Detection System* and a *Water Distribution*. The decompositions of the tasks *Execute preventive maintenance actions* and *Execute corrective maintenance actions* in Figure 5.7 are given on Figure 5.8. Some of the mappings (MG) are already

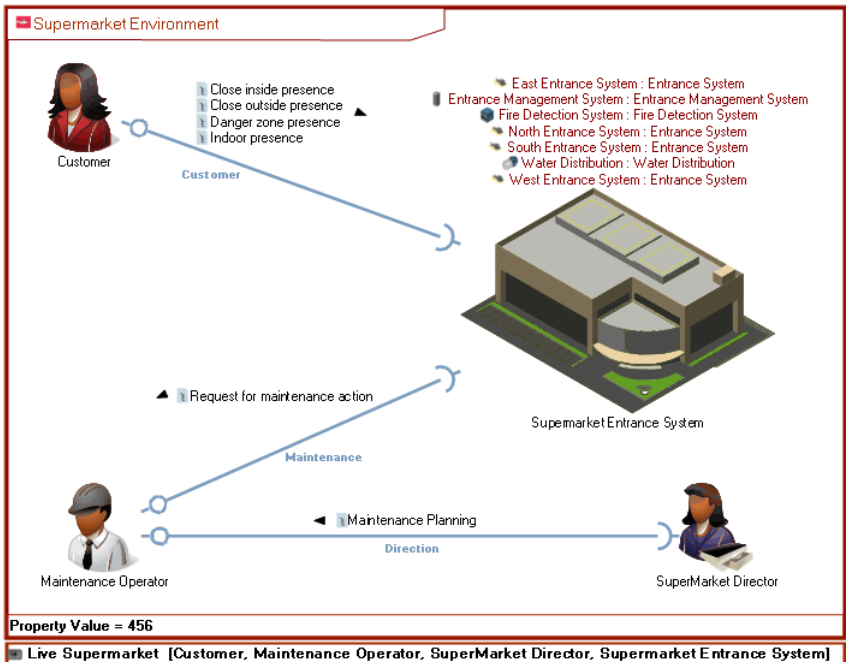


Figure 5.5: The supermarket's environment structure.

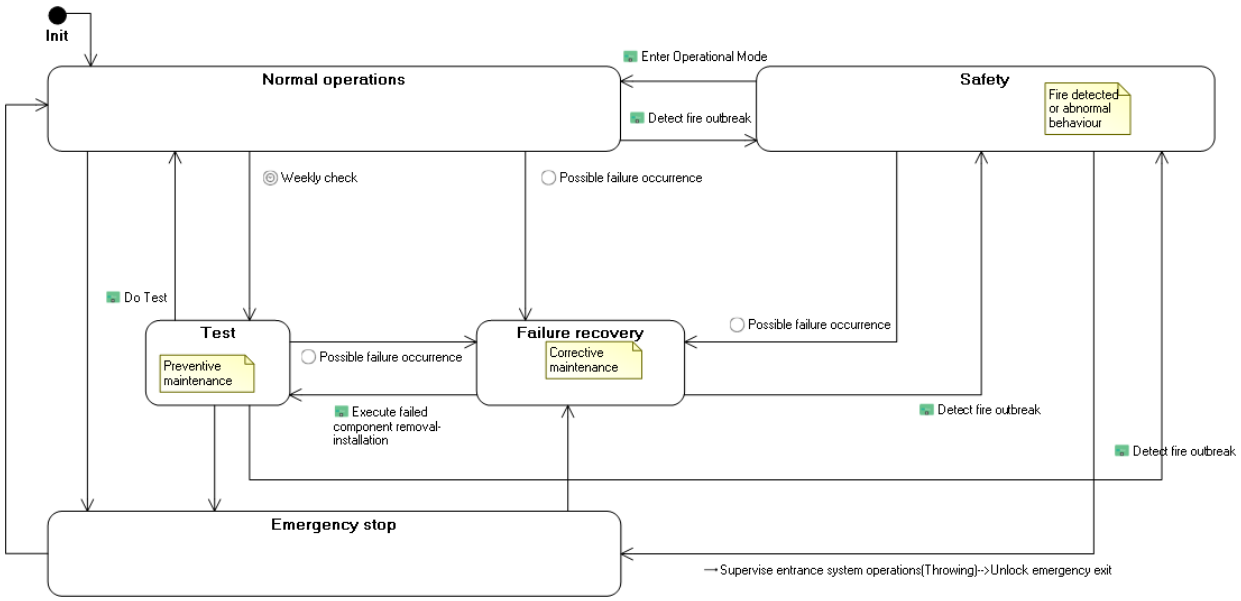


Figure 5.6: The supermarket's entrance system state model.

visible on Figure 5.6 where one can see the elements of PM on the transitions of

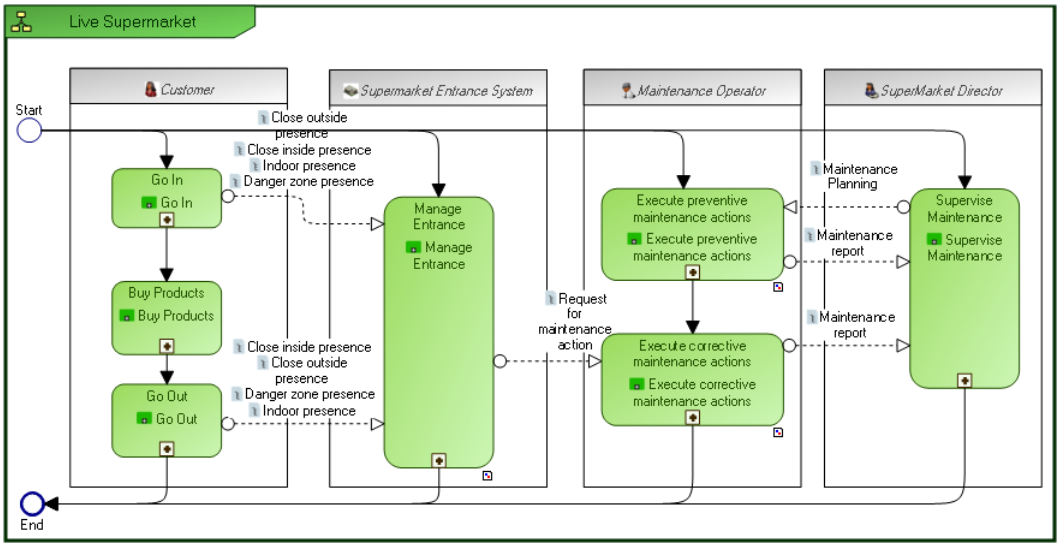


Figure 5.7: The supermarket's entrance system process model.

the depicted state model. For example, the transition between the states *Normal operations* and *Test* is involved in a trigger where the associated event is *Weekly check*. One expectation (an element of R) defined for the supermarket is: When the *Supermarket Entrance System* is in state *Safety*, the system *Emergency Exit* in the state *Unlocked* and *Locked* is ranked 2 and 5 respectively. This means that *Emergency Exit* is a subcomponent of the *Entrance System* and its state model contains the states *Locked* and *Unlocked*.

Therefore given the problem Pb(AM, SM, PM, MG, R, C, InitialPoint, StopCriterion) where the InitialPoint is the couple of initial states of SM of AM and the initial events of PM. A global view of PM is depicted on Figure 5.7.

Given those data, by setting the StopCriterion to max-score equals to 10^2 (roughly equivalent to the fact that, some points with a color less that 3 are not acceptable) and the max process depth equals to 13, and b is such that $b > 4$ one obtains the graph SSG depicted on Figure 5.9. A zoom around the root of this graph is shown on Figure 5.10. It should be noticed that the total number of explored nodes in SS is 742 while SSG contains 162 nodes. We recall that a node in this graph stores the current states and events of the system and additionally the computed metrics. All these data are not displayed for convenience. For instance, some data at the root of this graph are:

Event=[Start–>Supervise Maintenance & Start–>(Start)Manage Entrance & Start–>Execute preventive maintenance actions & Start–>Go In]

State=[Closing time ||Closed ||Closed ||Closed ||Closed ||Safe& Stopped ||Safe & Stopped ||Safe & Stopped ||Safe & Stopped ||Off & Cold ||Off & Cold ||Off & Cold

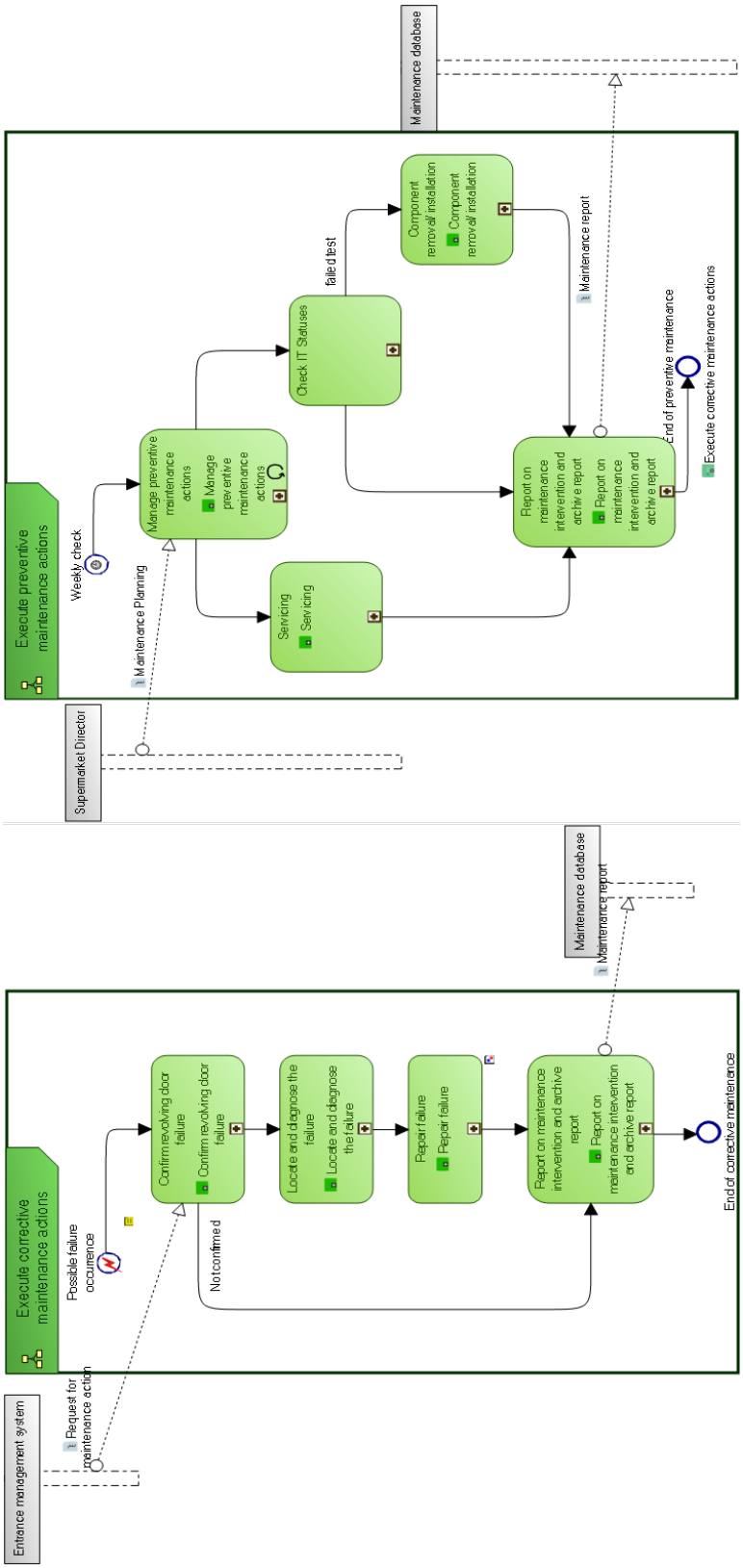


Figure 5.8: At bottom and top the corrective and preventive maintenance actions respectively.

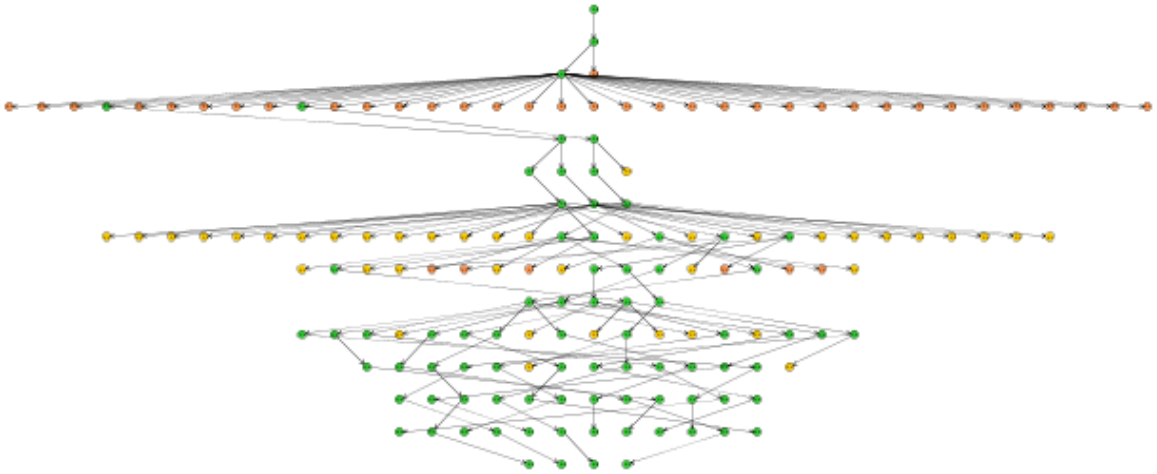


Figure 5.9: The SSG graph for Supermarket with max-score=100, max-process-depth=13 and $b > 4$

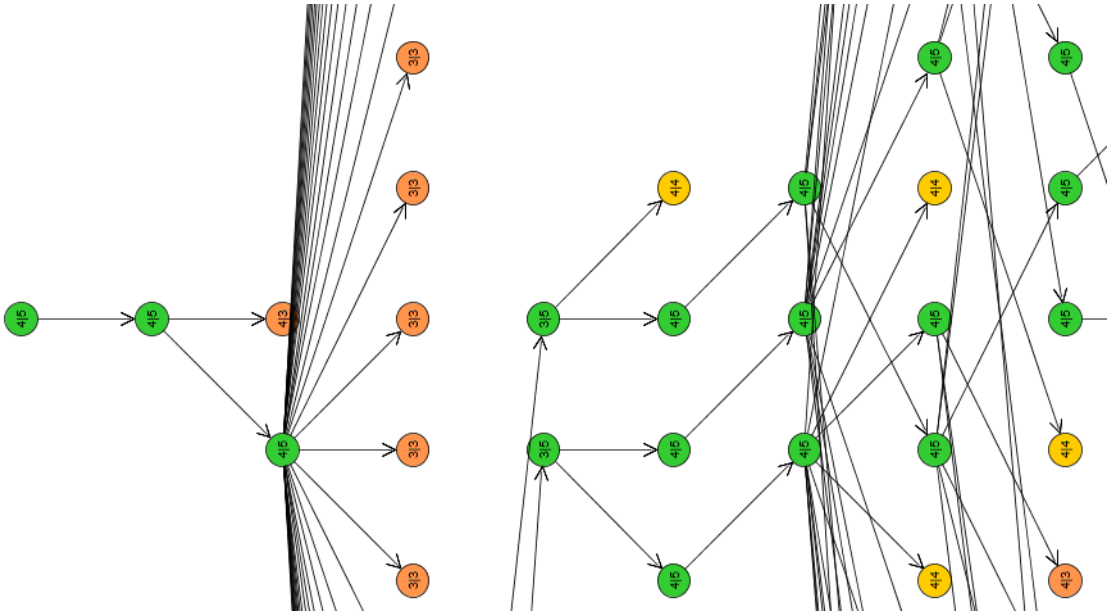


Figure 5.10: Zoom 1

||Off & Cold ||Off & Cold]

The first information that can be provided to the stakeholders is (R2) i.e. something might not be ok during the operation of the system for the input parameters considered. Second, each node that is not green should be subject to study. Indeed, not all nodes that are not green are problematic since, depending on the current operation of the system and its future operation, some behaviours are not going to be executed. The execution of system simply follows a path in SSG. Therefore the nodes provide an understanding of what will happen if some actions are executed. For the nodes that are problematic, the paths from the root are available where critical nodes are highlighted. A critical node is shared by a subset of paths yielding

to the problematic nodes. Such information are useful to either to redesign the processes or review the expectations on the system. On the other hand, MBMW could be resumed from a given point in SSG.

It is clear from this case study that given the size of the system and the different processes necessary to ensure its operation with respect to expectations, MODEF is insightful in providing an understanding of the impacts of such processes *a posteriori* i.e., before their execution.

But, MODEF is more interesting when the processes and expectations might be subject to continuous changes, i.e., when it is difficult to compute the definitive behaviour of the system over a long period.

5.3.2 A SoM: Modelling the functional coverage of a SOI

We continue with the SoM (SoM0) presented in Section 3.2.1 and for which the corresponding models are depicted in Figure 3.3. For the convenience of the reader we display again these models on Figure 5.11. The SoM0 focuses on the modelling of the functional coverage of a SOI whose objective is to ensure that the SOI covers the functional needs.

We additionally add at the bottom left on Figure 5.11, the specification of expectations (R) as done in a modelling tool. This expectation means: regardless of the assumption (A) (therefore considered as a tautology, see Section 4.3), a *System Function* in state *Maturity<30* is less preferred than in state *30<Maturity<60* in turn less preferred than in state *60<Maturity<100*. The rank of preferences is materialized with colors and some qualitative words (Unsatisfactory, Operational etc.) in the tool. For the readability of the figure, the mappings (MG) are indicated with $t_k, k = 1..5$ $((t_k, System\ Function) \mapsto \{\{t_k\}\})$, the one corresponding to t_3 is illustrated with the dashed blue arrows.

Therefore, except the data InitialPoint and StopCriterion, all the other data of the problem Pb(AM, SM, PM, MG, R, C, InitialPoint, StopCriterion) are depicted on Figure 5.11.

Let InitialPoint be (state:=*Maturity<30*, event:=Start).

Although the PM at right of the Figure 5.11 is not complex, its state space is possibly infinite since there are cycles in the scheduling of its sub-processes or tasks. Therefore we set StopCriterion as: max-process-depth:= 7. One can remark from the PM at right of the Figure 5.11 that at least 7 events or six tasks are necessary to reach the end of a possible execution of SoM0. Since PM is not complex, we do

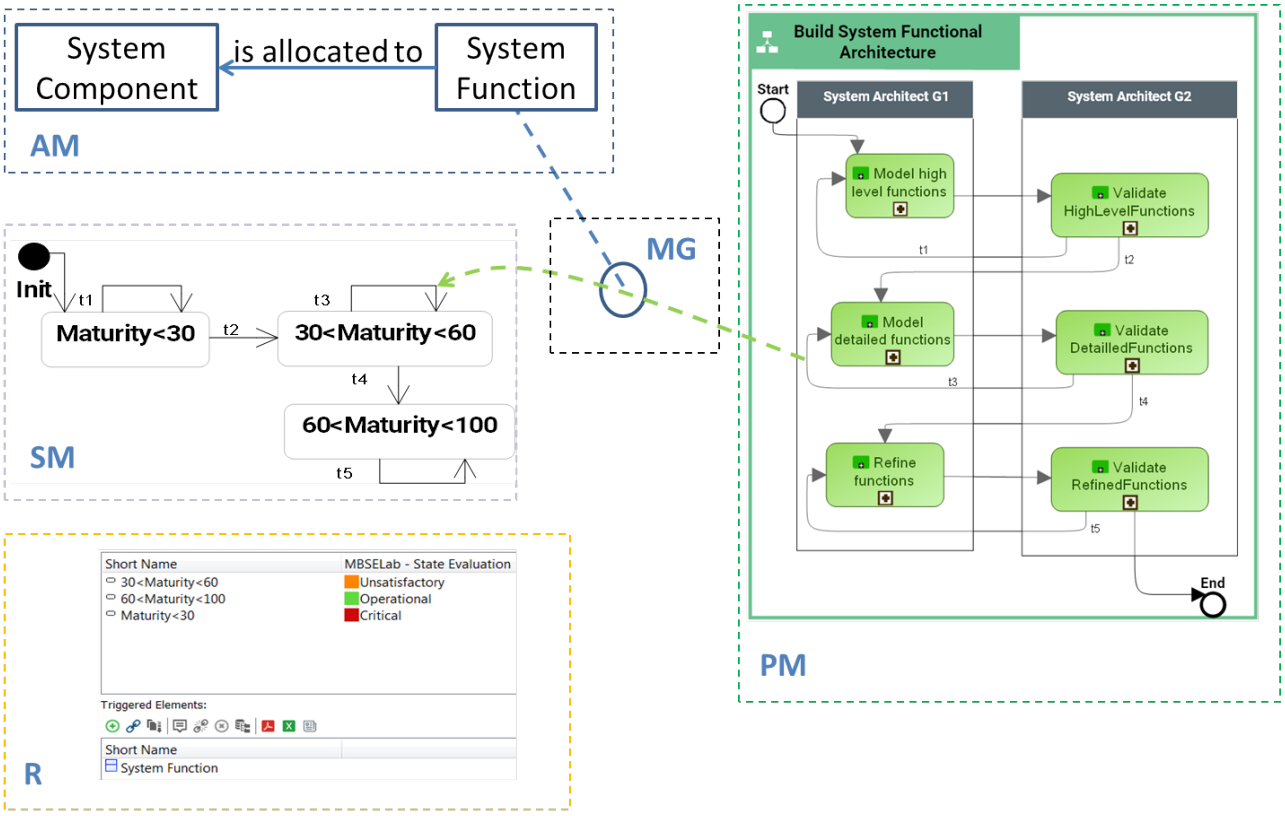


Figure 5.11: The input models for the problem associated to the SoM0

not set a value for max-score. For the exploitation, we also consider $b > 4$.

The total number of nodes enumerated by MBMW is 22 in SS while SSG contains 10 nodes. SSG is depicted on Figure 5.12.

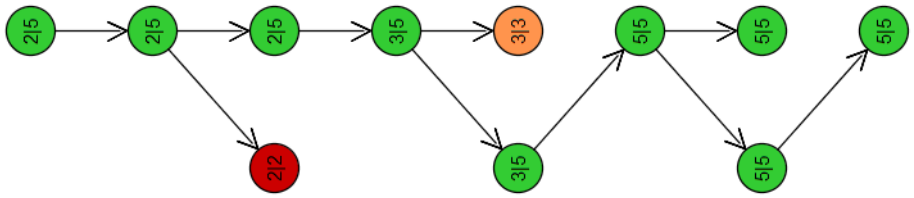


Figure 5.12: The SSG graph for the SoM0 with max-process-depth=7 and $b > 4$

Again the data associated to the nodes are not displayed for the readability of the graph. The states and events associated to leftmost (the root), the red, the orange, and the rightmost nodes on Figure 5.12 are:

(Event=[Start->Model high level functions],State=[Maturity<30]),

(Event=[Validate HighLevelFunctions->Model high level functions] State=[Maturity<30]),

(Event=[Validate DetailledFunctions->Model detailed functions] State=[30<Maturity<60]),

(Event=[Refine functions->Validate RefinedFunctions] State=[60<Maturity<100])

respectively.

It is not hard to apply the MBMW algorithm on this case study to effectively check that the resulting graph SSG is the one depicted on Figure 5.12.

The first information that can be provided to the stakeholders is (R2) i.e. something might not be ok during the operation of the SoM0 for the input parameters considered. Indeed, there are red and orange nodes which correspond to the cases where after the tasks ValidateHigLevelFunctions and ValidateDetailedFunctions, it is the tasks that directly precede them that should be executed again. In practice, such scenarios could typically lead to the need of additional resources if the allocated ones were totally used. Thus, by having the process models of modelling activities to be carried out and their impacts (expectations and mappings) on the states of produced models (M), MODEF makes it possible to anticipate problems that might arise.

5.4 Conclusion

In this section, we presented the analysis and exploitation algorithms. We started by defining the problem that the analysis addresses. Then we presented the main algorithms related to the co-exploration of models and the analysis of the generated state space. We also presented an exploitation of the analysis algorithm and its main output. Finally we presented two cases studies to illustrate the algorithms and demonstrate the interest of MODEF. Further work is discussed in Section 8.

Let us say a word about the results of the exploitation that are perhaps a critical part of MODEF in providing feedback to stakeholders.

Suppose the stakeholders are already capable to build models necessary to run the analysis and exploitation. This is a step to formalise the thing to which models relate and to take advantage of some benefits expected of models such as: understanding, preservation, share and reuse of knowledge about that thing.

On an operational point of view, we saw with the first case study (Section 5.3.1), how MODEF is insightful in providing synthetic and intuitive graph-based data related to the behaviours expected during the maintenance of a Supermarket. With the second case study (Section 5.3.2), we presented how the exploitation allows to get details on a SoM. These details relate to what would be the execution of this SoM. They are a first support to optimize how the system will effectively work in: providing the "path" to follow, taking preventive and corrective actions, or simply modelling new actions. These actions can be again considered in MODEF to obtain

new feedbacks.

It turns out that, if models are accompanied with appropriate analysis and exploitation means, they are going to powerfully support the formal end-to-end operation of the thing to which they relate. They will eventually support the optimization of the operation of that thing. It is clear that, such a statement does apply to a class of systems.

A question not addressed so far is: how are the process, state and structure models produced, obtained and effectively manipulated? In fact, as we saw in Section 2, the way models are considered may hamper the adoption of an approach. The next sections deal with this question.

A federated architecture for plugging and exploiting domain-specific models

This section is concerned with the exploitation of models in practice. The question it aims to answer is: how to reuse models coming from modelling tool for analysis purposes? Is it possible to decouple the reuse of models from the logic of a particular modelling tool? To reach that goal, in the following, we present a federated architecture (FA) that aims to make models (from model-based systems engineering) transparent to the tools used to build or generate them.

FA is intended to promote reuse and exploitation of the semantics of models. One objective is to make models available to any entity that can read the structure of models, and an interpretation of this structure so that models are automatically exploited in practice for engineering concerns such as system analysis e.g., the one presented in Section 5.

FA is federated because, on the first hand, it unifies models on their structure. On the other hand, it enables the definition of a particular semantics via an interpretation of the structure, therefore fulfilling the specificity of models.

Thereby, we argue on the definition of the structure of models within a category-theoretic framework. CT was created to unify and simplify mathematical systems [67]. This gave rise to some universal constructions based on a relational viewpoint. Indeed, any mathematical object (independently of its internal structure) is described in term of relationships with other objects and this description is generally stated by the diagrams (objects, arrows, equivalence between paths (sequences of arrows)). In this perspective, CT is the mathematics of FA. Such a framework enables to abstract away details related to a particular interpretation of the structure of models. At the same time, it enables to specify a formal link (ideally defined as a functor) between the structure of models and various interpretations.

On the other hand, computational and data structures and data format are presented to manipulate the mathematical objects i.e., to take advantage of FA in practice. They also demonstrate the feasibility of FA.

We pursue this section by first stating the problem and challenges it addresses (Section 6.1). After that, the proposed approach and its scope are described (Section 6.2). Then we give the description of the proposed federated architecture (FA) (Section 6.3). Next, the data and computational structures corresponding to the mathematical description of FA are given (Section 6.5).

6.1 Metamodels, data format and interfaces

In the engineering of complex systems, system modelling is often a descriptive activity that is complemented by system specification, verification and simulation activities. Current tools are in general, each specialized to enable a part or all of those activities for some kinds of models.

Unless there exists a translation mechanism that enables the reuse of the models produced by a first tool T1 (e.g., the modelling tool Mega[1] for System architecture), by a second tool T2 (e.g., a model-checking tool : AltaRica[2] tools for safety analysis), it is not always obvious to retrieve or call models from T1 and operate them automatically with T2. T1 and T2 are specialized in system modelling and system analysis respectively with different modelling formalisms. Moreover, it would not be convenient to define (bidirectional) translation mechanisms between a set (ST) of tools that need to communicate models, this because of the iterative nature of the modelling activity. Besides for n tools we would need $n * (n - 1)$ bidirectional translation mechanisms. Furthermore, it could be impossible (since different tools specialize in different domains) or inappropriate (e.g. too expensive) to build a direct translation from T1 to T2.

To remedy this issue, a common solution is to consider (not exclusively):

- a common metamodel
- a common data format
- a common interface

[1] mega.com
[2] https://altarica.labri.fr/wp/

for the models coming from any tool belonging to ST.

For this solution to work automatically, the meaning (independently of its implementation) of data representing the same kinds of models must be exactly the same for any tool in ST.

Generally, metamodels support the graphical representation of models, and description of their semantics. However, when models are used outside of the tool from where they have been built, some assumptions of the metamodels might be missing. For instance, Statecharts [47], a visual modelling technique, have a plethora of existing semantics (implemented within tools) which can be summarized in three main variants [38]. In this situation, in the way the tool implements those semantics, either a missing assumption related a semantics is implicit in the metamodel or it is explicit but rather proper to the modelling tool. In fact, it has also been argued in [86] that even for the same modelling formalism (a mathematical object), multiple concrete implementations (via an abstract syntax and a semantics) may exist. This means that even if one assumes the use of the same modelling formalism, nothing does guarantee the implementation of the same semantics. The resulting gaps would even become worse (e.g., in an attempt to integrate several models) with complex metamodels and the multiplicity of proprietary tools.

As for the data format, it is intrinsically related to the structure of models. It mainly serves as a way to persist the models. But it is relevant only if the underlined structure of models is agreed among ST. Although the data format is useful, it does not influence the semantics of a model.

An interface is of interest if it exposes and provides all the relevant elements necessary to precisely (i.e., explicitly and formally) and automatically deal with models described under it.

6.2 What is an adequate level of abstraction?

It turns out that, whatever the common interface, metamodel and data format which are used among ST, a key element here is the underlined structure of the models, how it could be used to effectively reuse and exploit the models in practice. We believe such a concern would be better addressed at the level of the architecture of models.

We follow the ISO/IEC/IEEE 42010[3] standard on the definition of the architecture of a system. It is defined as the abstract description or the fundamental organization of a system (a model here), embodied in its components, their relationships to each other and to the environment, and the principles governing its design and evolution.

The proposed architecture (FA) is federated in the sense that, while it is based on a specific structure, it aims to enable, via the instantiations and the interpretations of the structure, the plugging and exploitation of domain specific-models. In the spirit, we refer to plug-and-play devices. FA enables to plug models in the sense that it supports the ability to precisely specify them. Once plugged, FA also supports the ability to automatically use models for exploitation (again, for analysis purposes, we are not dealing graphical concerns). Note that, a tool that exploits a model is not necessarily a the same tool used to build that model. It is for instance, a tool tailored for a specific analysis of models.

As a result, in the following, we deal with the fundamental organisation and principles governing the definition of a descriptive model. Towards achieving those goals, FA is devised from a theoretical and practical point of view.

Let us note that we are not primarily concerned with the graphical representations of models, nor the description of a new (modelling or analysis) tool, even less a new modelling language. As reported above, the architecture of a system is not about the effective implementation or definition of the system. Last, we do not deal with (inter-) relations between heterogeneous or multi-domain models. Nevertheless, how a model is considered alone would typically influence such (inter-) relations.

6.3 Description of the proposed architecture FA

In this section, we start by informally motivating our viewpoint on the fundamental organisation of models (Section6.3.1). Then, we argue on the adequacy of category theory to formalise this view point (Section 6.3.2). We also present the necessary background and notation. Finally, the formal description of FA is given (Section 6.4).

[3] http://ieeexplore.ieee.org/document/6129467/

6.3.1 Fundamental organisation of a model

6.3.1.1 Structure

We are mainly interested by descriptive models that look like UML (Unified Modelling Language) and SysML (System Modeling Language) models. The structure of descriptive models is often composed of boxes that are linked with wires. In order to distinguish incoming and outgoing wires, boxes are generally equipped with input and output ports. An incoming wire comes into an input port of a box, while an outgoing wire leaves from an output port of a box. Generally, a port allows a box to interact with its surrounding environment. A box is either basic (a black box) or composite i.e. a structure built of inner constituent box(es). It allows to manage a complex representation of a system by hierarchical decompositions and recompositions. Indeed, the structure of a complex system might be obtained from the interconnection of simpler subsystems. The hierarchical decomposition applies to subsystems until a given level of granularity is reached. This leads to an hierarchy of boxes where one can distinguish composite boxes and non-composite or basic boxes. This explicit composite structure makes it possible to zoom in and out the structure of a box.

6.3.1.2 Structure's interpretations

Since complex systems are generally studied from several views or aspects, namely, physical, electrical, computing, etc., different interpretations of their structure are possible. At the level of models, these interpretations yield domain-specific languages or semantics. As a result, the structure constrains the definition of its interpretation, even if the structure does not make it possible to guess one of its possible interpretations. This means: while the structure of a model is not sufficient to reason with the model, it is nevertheless the source of the definition of an interpretation i.e. a semantics.

In this vein, it is difficult and perhaps impossible to impose or foresee all the possible interpretations of the structure of models. It is rather interesting to provide means that enable a proper exploitation of this structure, that is to say, the use of a structure that is explicitly and formally defined. This would then facilitate a definition of an interpretation of the structure.

6.3.1.3 Structure's usages

Last, but not least, the data of the model that represents the actual system is nothing but an instantiation of the model's structure together with its interpretations.

Example. Suppose the model of a paper is structurally built of 3 main sections. The data of a model of a paper can be given by: a title of each of these 3 sections and an assignment of each section's length. Typically, the meanings of a section's length and a section's title have to be defined as interpretations of the structure i.e., understanding the meaning of a section from some given points of view, the length and the title in this case.

The foregoing gives the main components of FA. They are: (a) the model's structure, (b) the interpretation of the structure (c) the corresponding instances of the structure. The structure and the interpretation can also be called abstract syntax and semantics of the model. Nonetheless, it is more convenient to see the semantics as the target of the interpretation.

The architecture needs to support the ability to define various interpretations of boxes. To that end, it does not model a concrete interpretation of them, however, principles to build such an interpretation will be given. In order to support the semantics of any model, FA will separate the structure and the possible interpretations thereof. The data/instances corresponding to models will simply be the set of usages of the structure and a related semantics.

We are interested in providing a formal description of the model's structure to ease the definition of its interpretations in practice. A data format will be also necessary for serializing the corresponding instances of the structure. The question which arises now is: how to formalise those elements? The ultimate goal is to automate the plugging and exploitation of models. (See Section 6.2, for the precise meaning of "plugging" and "exploitation" of models)

6.3.2 Background and Notation

One might think of graphs to formalise the structure of models. Although graphs are very powerful ways to describe interconnected objects, they lack a basic intrinsic feature to deal with the hierarchical composite structure of vertices (i.e. boxes) we need. To get such a structure, the graph must be equipped with additional data which might complicate the formalisation. Instead of considering a graph, we argue that the notion of "category" is well suited to deal naturally with the formalisation.

A category can be seen as a directed graph where the vertices are called the objects and edges are called the arrows of the category. Additionally, in a category, "new edges" are obtained from: either a new edge (id) from each vertex (v) to itself i.e., an arrow $v \xrightarrow{id} v$; or a pair of consecutive edges i.e., if $v_1 \xrightarrow{a} v_2$ and $v_2 \xrightarrow{b} v_3$ are two edges, in the category there is an arrow $v_1 \xrightarrow{ab} v_3$. In the category, it is required that, if $v_3 \xrightarrow{c} v_4$ is an edge in the graph, then the path a(bc) is equivalent to the path (ab)c i.e., parentheses are removable. Finally it is required that $a \circ id_{v_1} = a = id_{v_2} \circ a$ whenever a is an arrow $v_1 \xrightarrow{a} v_2$.

The major contribution of a category here, is that, although objects can have an internal structure, their relations with other objects are put forward and given by arrows. Furthermore, the category itself can be enriched with an internal structure allowing to combine objects together. Such an internal structure, together with the requirement on paths of consecutive edges in the category, will formally enable the zooming in and out of boxes (seen as objects).

The change of perspective, i.e., understanding, instantiating, interpreting or mapping a source category into a target category, consists of defining an arrow between 2 objects that have the structure of a category. For the mapping to make sense, it is required that the structure of the source object is preserved in the target object. Such a mapping is called a functor. The enrichment of the structure of the objects and the mappings between them give rise to new structures and related concepts studied by Category Theory (CT).

To be able to define the structure of a model as a category, the objects will be the boxes and the arrows will explicit how boxes are linked to bring about composite boxes. The structure of the category considered in this section and its formalisation are inspired and related to works [91] [90] [79], [99] [92] [65]. We will come back to some of them in the following to avoid any confusion.

The categorical concepts and their notations for defining the architecture are mainly brought from [63]. These are only necessary for understanding the underlying mathematics of FA and so its possible canonical extensions (see Section 6.7). Additional background can be found e.g., on basic CT [91], CT in general [67], multicategories [63].

Now, we recall the definition of a multicategory from which the one of a category is straightforwardly obtained. Then we argue on the necessity of a symmetric multicategory (SMuC) that underlies a symmetric monoidal category (SMC) as the effective structure of models.

A **multicategory** C [62, Definition 2.1.1] consists of

- *a class C_0, whose elements are called the **objects** of C*
- *for each $n \in \mathbb{N}$ and $a_1, ..., a_n, a \in C_0$, a class $C(a_1, ..., a_n; a)$ whose elements θ are called **arrows** or **maps** and depicted [...] as $a_1, ..., a_n \xrightarrow{\theta} a$*
- *for each $n, k_1, ..., k_n \in \mathbb{N}$ and $a, a_i, a_i^j \in C_0$, a function (see Fig. 6.1)*

$$C(a_1, ..., a_n; a) \times C(a_1^1, ..., a_1^{k_1}; a_1) \times \cdots \times C(a_n^1, ..., a_n^{k_n}; a_n) \rightarrow C(a_1^1, ..., a_1^{k_1}, ..., a_n^1, ..., a_n^{k_n}; a),$$

*called **composition** and written $(\theta, \theta_1, ..., \theta_n) \longmapsto \theta \circ (\theta_1, ..., \theta_n)$*
*for each $a \in C_0$ an element $1_a \in C(a; a)$, called **identity** on a satisfying*

- ***associativity**:*

$$\theta \circ \left(\theta_1 \circ (\theta_1^1, ..., \theta_1^{k_1}), ..., \theta_n \circ (\theta_n^1, ..., \theta_n^{k_n})\right) = (\theta \circ (\theta_1, ..., \theta_n)) \circ (\theta_1^1, ..., \theta_1^{k_1}, ..., \theta_n^1, ..., \theta_n^{k_n})$$

whenever $\theta, \theta_i, \theta_i^j$ are arrows for which these composites make sense

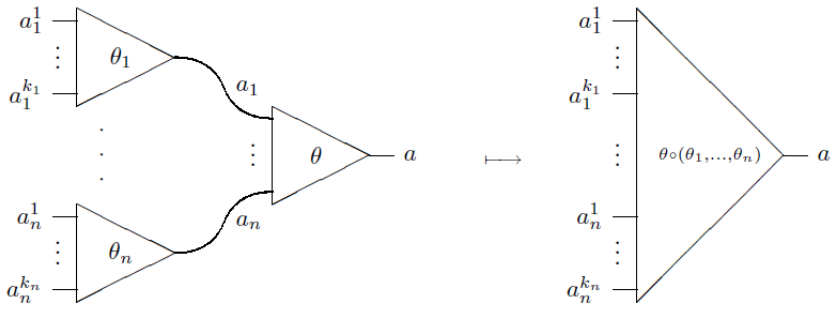


Figure 6.1: *Composition in a multicategory* [62, Figure. 2-B]

- *identity: $\theta \circ (1_{a_1}, ..., 1_{a_n}) = \theta = 1_a \circ (\theta)$ whenever $\theta : a_1, ..., a_n \rightarrow a$ is an arrow.*

A **category** $\mathcal{C}$ is then a multicategory in which every arrow, also called morphism, is unary (n = 1) [62, Example 2.1.2]. There are some classical categories like: **Set**; the category whose objects are sets and whose morphisms are functions, **Mon**; the category whose objects are monoids (a monoid is a tuple $(M, id, *)$ where M is a set, $*$ the multiplication formula and $id \in M$ the identity element such that identity and associativity laws hold) and whose morphisms are homomorphisms of monoids.

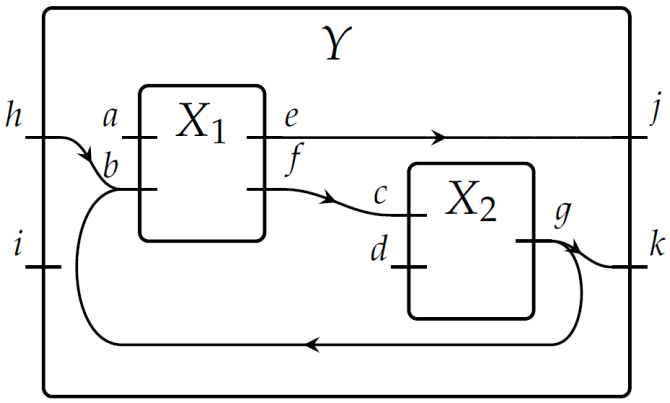


Figure 6.2: A box Y composed of 2 boxes X_1 and X_2

6.4 Components of FA

We start by the informal motivations of the concepts and eventually give the formal definitions.

Consider the box Y on Figure 6.2. It is composed of 2 boxes X_1 and X_2. The input and output ports of a box are respectively at left and right sides of the box. They are identified with alphabetic letters. For instance, the box Y has 2 input ports: h and i and 2 output ports j and k.

We note **Msc** the category that describes the structure of models. The formal definition of **Msc** is given in Section 6.4.1. If we consider the box on Figure 6.2 as the model, then, the objects of **Msc** are Y, X_1 and X_2. There is an arrow $X_1, X_2 \to Y$. **Msc** in this case will be a multicategory. If, instead, we write $X \to Y$, where the "combination" of X_1 and X_2 yields a new object X, then **Msc** in this case will be a category. In the latter case, informally speaking, the combination of two objects is possible via the definition of an internal operator (called the tensor and written $\otimes$) to the category that is associative with right and left units 0 or 1 depending . This operator equips **Msc** with a monoidal structure. (See e.g., [62, definition 1.2.5] for the formal definition of a monoidal category.)
When $\otimes$ is commutative up to isomorphism, (**Msc**,$\otimes$, 0) becomes a symmetrical monoidal category. The commutativity of $\otimes$ allows to indifferently write $X_1 \otimes X_2$ or $X_2 \otimes X_1$.

Finally, we know from [62], that *any symmetric monoidal category is naturally a symmetric multicategory, via the symmetry map* $\sigma \cdot - : a_{\sigma(1)} \otimes \cdots \otimes a_{\sigma(n)} \xrightarrow{\sim} a_1 \otimes \cdots \otimes a_n$. This means that a_i in the sequence $a_1, \ldots, a_n$ are commutative up to σ.

6.4.1 Structure of models

We consider **Msc** to be a symmetric multicategory (SMuC).

We continue with the example of the structure of a model on Figure 6.2.

We basically follow [90] [92] [99] on the definition of wiring diagrams (WD). The diagram on Figure 6.2 looks like a WD but, in fact it is different from a WD: as it will be clearly explained hereinafter.

In this diagram, we authorise: unconnected ports i.e., ports without incoming or outgoing wires (or links); converging wires i.e., wires whose the target port is the same; and diverging wires i.e., wires whose the source port is the same. These authorisations, apart the last, are forbidden in WD. In this diagram it will also be possible to have several links between a source port and a target port. WD could therefore be considered as special cases of this diagram where these authorisations are forbidden.

The primary objective of such authorisations is to encompass as many as possible wiring patterns occurring in practice. For instance, it might be the case that, structurally, while a real component is working, it is not branched on one of its interface with its surrounding environment. See for instance a simple Ptolemy model [98, Figure 9] where these authorisations are graphically visible.

A WD is a symmetric monoidal category (SMC) underlying a symmetric multicategory (SMuC). The relation between the SMC and its underlined SMuC for WD is formally written down in [90] [92] [99]. More interestingly, graphically, a SMuC is more convenient than its SMC. While SMC is more convenient than SMuC in notation in avoiding subscripts. For example, the arrow $X_1 \otimes X_2 \to Y$ in a SMC is depicted on Figure 6.3. One can observe on that figure that the modularity (i.e. the easy distinction of X_1 and X_2 in Y) is no longer available graphically. The SMC will be adopted for defining arrows and the composition in **Msc**. Let **Mscm** the SMC that **Msc** underlies.

We provide in the following the objects, arrows, composition formula and identities of **Msc** and prove that they satisfy the identity and associativity laws.

• Objects of **Msc**, the set $\mathbf{Msc}_0$. An object $a \in \mathbf{Msc}_0$ is a box and consists of a tuple $(in(a), out(a))$ where $in(a)$ and $out(a)$ are the sets of input and output ports of a respectively such that $in(a) \cap out(a) = \emptyset$. For instance, the object Y is given by $(\{h, i\},\{j, k\})$. The objects of **Mscm** are same as those of **Msc**.

• Arrows of **Msc**, the set $\mathbf{Msc}(a_1, \ldots, a_n; a)$. An arrow $a_1, ..., a_n \xrightarrow{\theta} a$ says how a box a is built from boxes $a_1, ..., a_n$. Let $b := a_1 \otimes \cdots \otimes a_n$, the arrow in **Mscm** is

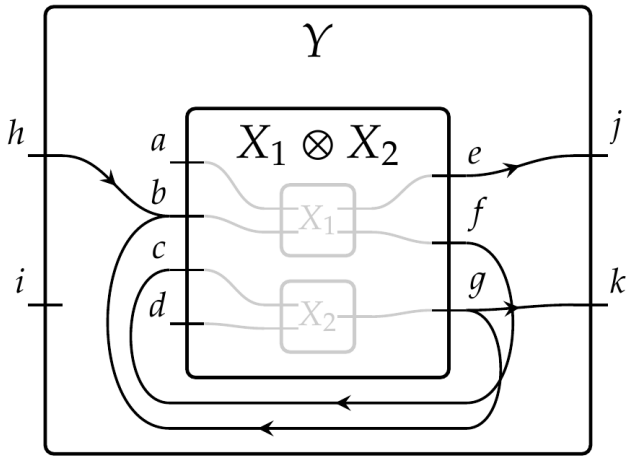


Figure 6.3: A box Y composed of $X_1 \otimes X_2$

given by $b \xrightarrow{\theta} a$. (b and a are called the domain and the codomain respectively of θ.) It consists of a tuple $(\theta^{in}, \theta^{out})$

$$\begin{array}{rrcccc} \theta^{in}: & L^{in} \to & in(b) & \times & (out(b) \sqcup in(a)) \\ \theta^{out}: & L^{out} \to & out(a) & \times & out(b) \end{array} \tag{6.1}$$

where $in(b)$ and $out(b)$ are given by $\coprod_{j=1..n} in(a_j)$ and $\coprod_{j=1..n} out(a_j)$ respectively. $\coprod$ is the disjoint union on sets.
L^{in} and L^{out} are the abstract sets of links coming into an input port and into an output port of one of $a_1, ..., a_n$ and a boxes respectively.

Note that (6.1) forbids a link to go from an output port of a to one of its input ports. Doing this will rather result in a new arrow (e.g. $a \to a'$ that models a kind of self-feedback). This requirement is highly important to avoid ill-defined arrows (composite boxes) in the sense of (6.1). It also forbids a link to go directly from an input port of a to an output one. If such a latter link should be existing, it will suffice to create a basic box with one input port and one output port and linked them to the input port and output ports respectively of a.

Example. The arrow $X_1, X_2 \to Y$ is given by $(\{l_1, l_2, l_3\}, \{l_4, l_5\})$. l_1, l_2, l_3, l_4 and l_5 are associated to $(h, b), (g, b), (f, c)$, (e, j) and (g, k) respectively.
We will eventually define θ^{in} and θ^{out} as matrices where columns and rows will correspond to the source and target of links. L^{in} and L^{out} will be useful to ease the understanding of the composition of arrows.

• Identities of **Msc**, $\mathbf{Msc}(a; a)$. An identity $a \xrightarrow{1_a} a$ consists of θ^{in} and θ^{out} and given by the coproduct inclusion and identity respectively, i.e. links connect identical (input to input and output to output respectively) ports from the domain

(a) to the codomain (a) of 1_a.

• Composition formula of **Msc**. The composition enables the substitution of the constituents of a composite box to get a more detailed one. For each $n, k_1, ..., k_n \in \mathbb{N}$ and boxes $a, a_i, a_i^j \in \mathbf{Msc}_0$, we have a function $\circ$:

$$\mathbf{Msc}(a_1, ..., a_n; a) \times \mathbf{Msc}(a_1^1, ..., a_1^{k_1}; a_1) \times \cdots \times \mathbf{Msc}(a_n^1, ..., a_n^{k_n}; a_n) \rightarrow \mathbf{Msc}(a_1^1, ..., a_1^{k_1}, ..., a_n^1, ..., a_n^{k_n}; a).$$

We must define an arrow $a_1^1, ..., a_1^{k_1}, ..., a_n^1, ..., a_n^{k_n} \xrightarrow{\theta \circ (\theta_1, ..., \theta_n)} a$. This amounts to finding a function $\theta_0 := \theta \circ (\theta_1, ..., \theta_n)$, i.e. θ_0^{in} and θ_0^{out} following (6.1). The data (6.1) of an arrow can be rewritten as follows.
We define two matrices M^{out}, M^{in} as functions

$$\begin{array}{rrcl} M^{out}: & Y^{in} \times (Y^{out} \sqcup Z^{in}) & \rightarrow & 2^{L^{in}} \\ M^{in}: & Z^{out} \times Y^{out} & \rightarrow & 2^{L^{out}} \end{array} \tag{6.2}$$

where $Y^{in} = in(b)$, $Y^{out} = out(b)$, $Z^{in} = in(a)$ and $Z^{out} = out(a)$. 2^X is the powerset.

In the following, we start by giving the semiring on which matrices and their operation will be associated and the connection with boxes, and then we end up with arrow composition. Note that if a matrix M is defined by $M : I \times J \rightarrow K$, where I and J are finite sets with n and m elements, we often write $M(n, m)$ where n and m positive integers are the number of rows and columns of M respectively. A total order on I and J enables to write $M = a_{ij}, a_{ij} \in K, i = 1..n, j = 1..m$. $(K, +, \times, 0, 1)$ a semiring.

Let K the power set of the set of strings of finite lengths, $(K, \cup, \cdot, \emptyset, \{\varepsilon\})$ is a semiring [32] where $\times := \cdot$ is the product induced by the string concatenation operator, $+ := \cup$ is the addition given by union of sets of strings of finite lengths, $0 := \emptyset$ is the zero given by the empty set and $1 := \{\varepsilon\}$ is the unit given by the singleton set containing the empty string.

We note $\emptyset_M$ the matrix with $\emptyset$'s everywhere and $\mathbb{I}_M$ the square matrix with $\{\varepsilon\}$'s on the main diagonal and $\emptyset$'s elsewhere.

In order to multiply two matrices X and Y, additionally to the requirement that the number of columns in X must equal the number of rows in Y, the total orders that labelled these columns and rows must also equal i.e. they must have the same base set and the same total ordering between the elements of this set.

Example: Let us consider $f_A : I_1 \times S \rightarrow K$, and $f_B : S \times I_2 \rightarrow K$ where $I_1 = \{a, b\}$, $S = \{c, d\}$ and $I_2 = \{e, f\}$. Let also consider the total orders on I_1, S

and I_2 are given by $a \leq_{I_1} b$, $c \leq_S d$ and $e \leq_{I_2} f$ respectively. Suppose given
$f_A := \{(a,c) \mapsto \{w_1\}, (a,d) \mapsto \emptyset, (b,c) \mapsto \{w_2, w_3\}, (b,d) \mapsto \emptyset\}$
$f_B := \{(c,e) \mapsto \{w_5\}, (c,f) \mapsto \emptyset, (d,e) \mapsto \emptyset, (d,f) \mapsto \emptyset\}$

The matrices A and B associated to the function f_A and f_B are as follows.

$$A = \begin{array}{c} \\ a \\ b \end{array} \begin{array}{c} \begin{array}{cc} c & d \end{array} \\ \begin{pmatrix} \{w_1\} & \emptyset \\ \{w_2, w_3\} & \emptyset \end{pmatrix} \end{array} \quad B = \begin{array}{c} \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cc} e & f \end{array} \\ \begin{pmatrix} \{w_5\} & \emptyset \\ \emptyset & \emptyset \end{pmatrix} \end{array}$$

The matrix A will mean in practice: there is a link or wire (w_1) which supplies the port a and which comes from the port c. There are also two links (w_2 and w_3) which supply the port b and that come from the port c. The product $A \times B$ given by

$$A \times B = \begin{array}{c} \\ a \\ b \end{array} \begin{array}{c} \begin{array}{cc} e & f \end{array} \\ \begin{pmatrix} \{w_1 w_5\} & \emptyset \\ \{w_2 w_5, w_3 w_5\} & \emptyset \end{pmatrix} \end{array}$$

shows that in practice there is also link ($w_1 w_5$) that supplies the port a and which comes from the port e. Typically, a port will be either an input or output port of a box.

Given (6.2) and the aforementioned notations, we are now ready to define arrow composition.

Let $\mathrm{c} = a_1^1 \otimes \cdots \otimes a_1^{k_1} \otimes \cdots \otimes a_n^1 \otimes \cdots \otimes a_n^{k_n}$ and $\mathrm{b} = a_1 \otimes \cdots \otimes a_n$. Let also $X^{in} = in(c)$, $X^{out} = out(c)$, $Y^{in} = in(b)$, $Y^{out} = out(b)$, and $Z^{in} = in(a)$, $Z^{out} = out(a)$.

The map $\theta \circ (\theta_1, ..., \theta_n)$ (i.e., $c \xrightarrow{\theta_0} a$ in **Mscm**) following the data (6.2) of an arrow via 2 matrices, is: (O^{in}, O^{out}) given by the dashed arrows on the following diagrams.

$$\begin{array}{ccc} X^{in} & \overset{O^{in}}{\dashrightarrow} & Z^{in} \sqcup X^{out} \\ {\scriptstyle N^{in}}\downarrow & & \uparrow{\scriptstyle N'^{out}} \\ Y^{in} \sqcup X^{out} & \xrightarrow[M'^{in}]{} & Z^{in} \sqcup Y^{out} \sqcup X^{out} \end{array} \qquad \begin{array}{ccc} Z^{out} & \overset{O^{out}}{\dashrightarrow} & X^{out} \\ {\scriptstyle M^{out}}\downarrow & \nearrow_{N^{out}} & \\ Y^{out} & & \end{array}$$

An arrow $I \xrightarrow{M} J$ in this diagram defines the matrix $M(\leq_I, \leq_J)$, where $\leq_S$ is a total order on S, (we recall that the elements of S will be the ports of boxes). For convenience, we abuse the notation and identify $\leq_S$ to the sequence of ordered elements of S.

$$\begin{aligned} O^{out} &= M^{out} \times N^{out} \\ O^{in} &= N^{in} \times M'^{in} \times N'^{out} \end{aligned} \tag{6.3}$$

where

N^{in}, N'^{out}, M'^{in} are respectively given by:

$$\begin{array}{c} \\ \leq_{X^{in}} \end{array} \begin{array}{c} \overset{\leq_{Y^{in} \sqcup X^{out}}}{} \\ \begin{pmatrix} N^{in} \end{pmatrix} \end{array} \quad \begin{array}{c} \\ \leq_{Z^{in}} \\ \leq_{Y^{out}} \\ \leq_{X^{out}} \end{array} \begin{array}{c} \overset{\leq_{Z^{in}} \quad \leq_{X^{out}}}{} \\ \begin{pmatrix} \mathbb{I}_M & \emptyset_M \\ \emptyset_M & N^{out} \\ \emptyset_M & \mathbb{I}_M \end{pmatrix} \end{array} \quad \begin{array}{c} \\ \leq_{Y^{in}} \\ \leq_{X^{out}} \end{array} \begin{array}{c} \overset{\leq_{Z^{in} \sqcup Y^{out}} \quad \leq_{X^{out}}}{} \\ \begin{pmatrix} M^{in} & \emptyset_M \\ \emptyset_M & \mathbb{I}_M \end{pmatrix} \end{array}$$

and M^{out}, N^{out} are respectively given by:

$$\begin{array}{c} \\ \leq_{Z^{out}} \end{array} \begin{array}{c} \overset{\leq_{Y^{out}}}{} \\ \begin{pmatrix} M^{out} \end{pmatrix} \end{array} \quad \begin{array}{c} \\ \leq_{Y^{out}} \end{array} \begin{array}{c} \overset{\leq_{X^{out}}}{} \\ \begin{pmatrix} N^{out} \end{pmatrix} \end{array}$$

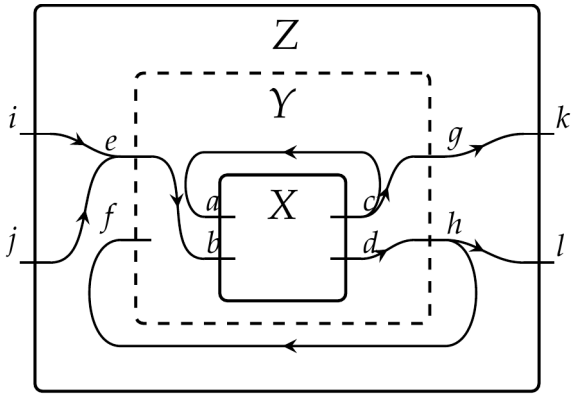


Figure 6.4: A box Z composed of Y, itself composed of X

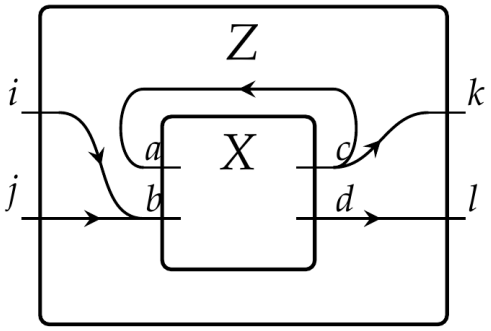


Figure 6.5: A box Z composed of X

Example. Figure 6.4 is the representation of two arrows $X \xrightarrow{\theta_1} Y$ and $Y \xrightarrow{\theta} Z$. The

arrow $X \xrightarrow{\theta\circ\theta_1} Z$ is given by the matrices O^{in} and O^{out} computed as follows. The matrices M^{out}, N^{out}, N^{in}, N'^{out} and M'^{in} are the following.

$$M^{out} = \begin{array}{c} \\ k \\ l \end{array} \begin{array}{c} \begin{array}{cc} g & h \end{array} \\ \begin{pmatrix} \{l_1\} & \emptyset \\ \emptyset & \{l_2\} \end{pmatrix} \end{array} \quad N^{out} = \begin{array}{c} \\ g \\ h \end{array} \begin{array}{c} \begin{array}{cc} c & d \end{array} \\ \begin{pmatrix} \{l_3\} & \emptyset \\ \emptyset & \{l_4\} \end{pmatrix} \end{array} \quad N^{in} = \begin{array}{c} \\ a \\ b \end{array} \begin{array}{c} \begin{array}{cccc} e & f & c & d \end{array} \\ \begin{pmatrix} \emptyset & \emptyset & \{l_5\} & \emptyset \\ \{l_6\} & \emptyset & \emptyset & \emptyset \end{pmatrix} \end{array}$$

$$N'^{out} = \begin{array}{c} \\ i \\ j \\ g \\ h \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccccc} i & j & c & d \end{array} \\ \begin{pmatrix} \{\varepsilon\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \{\varepsilon\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \{l_3\} & \emptyset \\ \emptyset & \emptyset & \emptyset & \{l_4\} \\ \emptyset & \emptyset & \{\varepsilon\} & \emptyset \\ \emptyset & \emptyset & \emptyset & \{\varepsilon\} \end{pmatrix} \end{array} \quad M'^{in} = \begin{array}{c} \\ e \\ f \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccccc} i & j & g & h & c & d \end{array} \\ \begin{pmatrix} \{l_7\} & \{l_8\} & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{l_9\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{\varepsilon\} & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{\varepsilon\} \end{pmatrix} \end{array}$$

It is easy to check that $O^{out} = M^{out} \times N^{out}$ and $O^{in} = N^{in} \times M'^{in\times} N'^{out}$ are given by the following matrices:

$$O^{out} = \begin{array}{c} \\ k \\ l \end{array} \begin{array}{c} \begin{array}{cc} c & d \end{array} \\ \begin{pmatrix} \{l_1 l_3\} & \emptyset \\ \emptyset & \{l_2 l_4\} \end{pmatrix} \end{array} \quad O^{in} = \begin{array}{c} \\ a \\ b \end{array} \begin{array}{c} \begin{array}{cccc} i & j & c & d \end{array} \\ \begin{pmatrix} \emptyset & \emptyset & \{l_5\} & \emptyset \\ \{l_6 l_7\} & \{l_6 l_8\} & \emptyset & \emptyset \end{pmatrix} \end{array}$$

The composition in this example enables to zoom in Z. After the composition, one can remark that the link (on Figure 6.4) from the port h of Y to its port f defined by the arrow $Y \xrightarrow{\theta} Z$, disappears (on Figure 6.5) in the arrow $(X \xrightarrow{\theta\circ\theta_1} Z)$ resulting of the composition. This, because the port f is not connected in $X \xrightarrow{\theta_1} Y$ and the effect of $\times$ in the semiring $(K, \cup, \times, \emptyset, 1)$. In practice, this could mean, although f seems connected, actually it is not.

• Associativity and identity laws of **Msc**. The associativity law follows from the associativity of matrix multiplication. The identity law follows from the fact that the functions of the couple $(1_a^{in}, 1_a^{out}) = 1_a$ are identities of their domain.

We now need to proof that the tensor $\otimes$ in **Mscm** is effectively a monoidal commutative product. Let us define the unit as the empty box $(\emptyset, \emptyset) = 0$ and set the tensor to be the disjoint union ($\sqcup$) on sets (of input and output ports of boxes). It is easy to check that $\otimes$ is effectively a monoidal commutative product.

Finally, **Msc** is the underlying SMuC of the SMC (**Mscm**,$\sqcup$,0).

6.4.2 Structure interpretations

The question this section answers is: how to bridge the gap between a possible semantics (interpretation of the structure) and its implementation in practice? Note that a semantics of the structure might be formally defined differently (denotational, axiomatic, or operational) and implemented in many ways. In this respect, we are not trying to define a particular semantics of models nor its implementation. Instead, principles for formally connecting both the semantics and its implementation, whatever they may be, are given.

Relying on the structure of models, it is possible to expose the specification of an interpretation of boxes such that their semantics can be recovered. Therefore, an actual interpretation could be independently developed, by accommodating the structure of models.

The specification of the interpretation is ideally given by a functor $F : \mathbf{Msc} \to \mathbf{Msin}$, where **Msin** is the category where the structure is interpreted. We say ideally because it may not be always obvious to define a functorial semantics.

Example. (i) Consider again the model as the box depicted on Figure 6.2. A meaning of this model is only known up to the definition of an interpretation of its structure. Suppose this box models a physical structure and we want to study some of its physical properties. This could amount to defining $F : \mathbf{Msc} \to \mathbf{Msin}$ as follows. **Msin** is **Set** with objects sets of real numbers and arrows functions that defined the relationships between the physical properties. Suppose the physical property is the mass. F is therefore defined by $F_0 : \{X_1, X_2, Y\} \to \mathbb{R}$ and the image of $X_1, X_2 \to Y$ is the function $F_0(X_1) \times F_0(X_2) \to F_0(Y)$ given by the sum of masses. Note that another interpretation can be done for another physical property.

(ii) Consider the arrow on Figure 6.6. By defining X, as a two-digit adder which

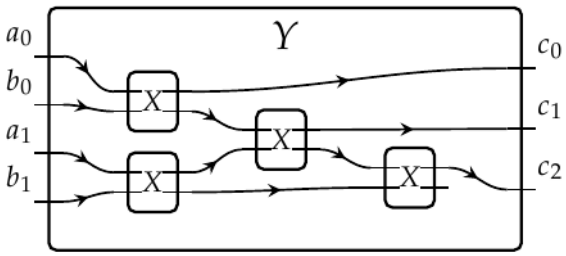


Figure 6.6: $X, X, X, X \to Y$

returns the result and the carry out on the top and bottom output ports of X, we can define Y as two-digit numbers adder ($a_1a_0 + b_1b_0 = c_2c_1c_0$), i.e. a category (**Msin**) with 2 objects and 1 arrow.

In the two examples (i) and (ii) above, the concepts of (mass) and (two-digit adder) and (two-digit numbers adder) respectively must be exposed by F; no matter how they are defined in **Msin** and implemented in practice. However, and more generally, these data must be machine-accessible. As a consequence, we discuss the computational and data structures in Section 6.5.1. Note that functorial semantics of wiring diagrams (i.e. special cases of **Msc**) are defined in [79], [90], [99] and [92] which are concerned with the interconnections of discrete-time processes, databases and plug-and-play circuits, differential equations of open dynamical systems, and mode-dependent networks respectively. In each case, a particular algebra is devised. These algebra could be implemented as definitions of interpretations of the structure of models. For now, we continue to discuss how to obtain the actual models with respect to the structure.

6.4.3 Structure usage or real models data

Like the definition of a structure interpretation, for real models data to be automatically exchangeable, those data must be machine-accessible. As a result, we need a means to specify what is an actual model data with regard to the structure.
We will suppose that **Msc** is representable (see, the formal definition in [62, Definition 3.3.1]), which roughly means: any composite box decomposes uniquely through its constituent boxes. The real models data are defined by a Set-valued functor $I : \mathbf{Msc} \to \mathbf{Set}$. It associates an object $a \in \mathbf{Msc}_0$ to the set of its usage $I_0(a)$ and any arrow $a_1, ..., a_n \xrightarrow{\theta} a \in C(a_1, ..., a_n; a)$ to the function $I_0(a_1) \times ... \times I_0(a_n) \xrightarrow{I(\theta)} I_0(a)$.

Example. Consider the box depicted on Figure 6.6. I for this model is defined by $I_0 : \{X, Y\} \to \mathbf{Set}$ and the image of $X, X, X, X \to Y$ is the function $I_0(X) \times I_0(X) \times I_0(X) \times I_0(X) \times \to I_0(Y)$ given by $x_1, x_2, x_3, x_4 \mapsto y$. Where $x_{i,i=1..4}$ and y are the instances of X and Y respectively. The possible semantics of X and Y have been defined to be a two-digit adder and a two-digit numbers adder respectively via F.

Like the structure's interpretation, we discuss the data structures and format structure in Sections 6.5.1 and 6.5.3.

The summary of this section, that elaborates the informal and formal understanding of the proposed architecture FA, is as follows. For reading models (their data) it is necessary to have at least the specification of $I : \mathbf{Msc} \to \mathbf{Set}$, whereas for exploiting models (their semantics) it is necessary to have at least the specification of $F : \mathbf{Msc} \to \mathbf{Msin}$.

Finally, for plugging and exploiting models through a given automated routine, it is necessary to specify

$$\textbf{Msin} \xleftarrow{F} \textbf{Msc} \xrightarrow{I} \textbf{Set} \tag{6.4}$$

and make the interpretation's definition machine-accessible.

6.5 Computational and data structures, base invariants and data format

The data structures along with some base invariants (described as functions) for implementing the specification **Msin** $\xleftarrow{F}$ **Msc** $\xrightarrow{I}$ **Set** are discussed in Section 6.5.1. Finally, in Section 6.5.3, a possible data format for serializing the data associated to the specification is proposed.

6.5.1 Data structures and base invariants

We recall that a box a in the structure consists in a tuple $(in(a), out(a))$ and a composite box is described by an arrow which consists in a tuple $(\theta^{in}, \theta^{out})$.

Notation. We write $A(p_1{:}T(p_1)$, ...,$p_n{:}T(p_k))$ to define an object or a class (in a programming sense). A has the property p_i of a type $T(p_i)$, $i = 1..k$. We will sometimes omit the type of properties of an object, e.g., by writing $A(p_1, ..., p_n)$. We also write $A[X]()$ to mean that X is a parametric parameter. Let li:List[R] be a list li of elements of type R. We also write li.map(f) where f is a function to say that f applies on each element of li and returns the new list.
We note _ the type that represents any type. Finally let props: String→_ be the function that maps a string to any object.

It is perhaps worth noticing here that this notation is tightly linked to the Scala[4] programming language syntax. Indeed we used Scala to implement the mathematical objects.

One question one might ask here is: what are data structures useful for? It is not clear whether there are well established data structures and basic algorithms to deal with category theory objects in practice see e.g., the summary of the discussions at a NIST (National Institute for Standards and Technology) Computational Category Theory (CCT) Workshop [102].

Therefore, the data structures we present in the following are also an attempt to address this concern.

[4] https://www.scala-lang.org/

We need to define an object corresponding to the representation of **Msc**. We start by defining a generic representation of a symmetric multicategory as follows.

GenericSMultiCategory[Obj,Arr](
dom:Arr→List[Obj], cod:Arr→Obj, id:Obj→Arr,
compose:(Arr, List[Arr])→Arr,
identityOnComposition:Arr→Bool,
associativityOnComposition:(Arr,List[Arr],List[List[Arr]])→Bool,
symmetry:(Arr,Arr)→Bool)

where dom, cod, id, compose, identityOnComposition, and associativityOnComposition are the maps that give the boxes in the domain of an arrow, the box in the codomain of an arrow, for an object the corresponding identity arrow, the compostion of arrows, the identity law, and the associativity law respectively. symmetry is a map that verifies if 2 arrows are equivalent i.e.: they have the same codomain and the domain (a list) of one might be a permutation of the domain of the other.

Note that identityOnComposition and associativityOnComposition are invariable or static in the sense that they do not depend on a particular value of (Obj,Arr). e.g. identityOnComposition is given by **identityOnComposition(f)** $\mapsto$ **compose(f, dom(f).map(id))** == **compose(id(cod(f)),List(f))**.
Both maps should always return true for well-defined calls. Others should be redefined for every instantiations of type parameters (Obj,Arr). Hence, we provide the data structures corresponding to Obj and Arr i.e., objects (boxes) and arrows (how a composite box is built) respectively of **Msc**. We follow the data provided in Section 6.4.1 to describe Obj and Arr. We start by defining the following objects where id is the identifier of any object.
BoxPort(id,props), a port of a box.
Box(id, inPortIDs, outPortIDs,props), a box where inPortIDs and outPortIDs are its set of input and output ports ids.
BoxLink (id, idsrcBox, idsrcPort, posSrcBox, idtgtBox, idtgtPort, posTgtBox,props), a link or wire, where idsrcBox, idsrcPort, posSrcBox and idtgtBox, idtgtPort, posTgtBox are the source box of the link, the position this source box in a domain of an arrow, the source port of the link and the target box of the link id, its position in a domain of an arrow, the target port of the link id respectively.
BoxArrow (id, domBoxesIds, codBoxId, linksIds,
props), an arrow, where domBoxesIds, codBoxId, linksIds are the list of ids of boxes in the domain of the arrow, the set of internal links of the composite box of

id codBoxId respectively.

Now the object corresponding to the representation of **Msc** is : **SMultiCategoryBox := GenericSMultiCategory[Box,BoxArrow]**. The definitions of algorithms corresponding to the maps dom, cod, id and compose are deduced from the data of **Msc** (see Section 6.4.1) and the objects BoxPort, BoxLink plus the parameters Box and BoxArrow.

To specify I and F we need to define an object corresponding to those functor or map of multicategories. A generic definition of the representation of such a map is defined by the object

MultiCategoryMap[Obj,Arr,Obj2,Arr2](
sourceCat:GenericSMultiCategory[Obj,Arr],
targetCat:GenericSMultiCategory[Obj2,Arr2],
fmap0:Obj→Obj2,
fmap1:Arr→ Arr2,
identitiesPreservation:Obj→Bool,
compositionPreservation:(Arr,List[Arr])→Bool)

where sourceCat, targetCat are the source and target multicategories respectively. fmap0, fmap1, map (in a CT sense) objects to objects, arrows to sequences of arrows respectively. identitiesPreservation and compositionPreservation define the identities preservation and the composition preservation respectively. The latter are invariable in the sense that they do not depend on a particular value of (Obj,Arr,Obj2,Arr2). They are given by:

identitiesPreservation(ob) $\mapsto$ **fmap1(sourceCat.id(ob))== targetCat.id(fmap0(ob))** and

compositionPreservation(f,fi) $\mapsto$ **fmap1(sourceCat.compose(f, fi))== targetCat.compose(fmap1(f),fi.map(fmap1))** respectively.

To specify I, we need to define the multicategory **Set**. Similarly to **Msc**, we start by defining the object (in a CT sense) **SetBoxUsage(id,setU)**, a set setU. Then **Set** is given by the object **BoxUsageMultiCategory** := **GenericSMultiCategory[SetBoxUsage, List[SetBoxUsage]→SetBoxUsage]** and the map I : **Msc** $\rightarrow$ **Set** by

MultiCategoryMap[Box,BoxArrow,SetBoxUsage, List[SetBoxUsage] $\rightarrow$ **SetBoxUsage]**

Given **GenericSMultiCategory[Box,BoxArrow]** and

GenericSMultiCategory[SetBoxUsage,

List[SetBoxUsage]→SetBoxUsage] we only need to define the maps fmap0 and and fmap1.

To specify F, we will consider the functor corresponding to the object **MultiCategoryMap[Box,BoxArrow,Box,BoxArrow]**. This MultiCategoryMap maps the boxes and arrows in the structure to boxes and arrows in F**Msc**. The exploitation of models at this stage amounts to making the semantics of models available either as an API or via executable libraries (like in programming with code reuse or web services). It is the reason why we say in Section 6.4.2 that a definition of an interpretation of the structure has to be machine-accessible. An API, a library or a code that implements an interpretation of the structure has to take advantage of MultiCategoryMap, i.e. to define the data F**Msc** or **Msin**.

A key point here, is that the presented data structures encapsulate the data of **Msc**, F and I. They ought to be used to access the actual internal structure of a box and an arrow. Finally, the implementations corresponding to a given interpretation of the structure could take any form provided that the required concepts necessary to operate (the semantics of) models are exposed. The data and computational structures and, invariants presented in this section aim to operationalize the theoretical concepts supporting the proposed architecture. Nonetheless, they are not prescriptive since other choices of data structures might be considered.

6.5.2 Identification of an actual component

A box a, does not refer to an *actual component*. Indeed an actual component is specified via the given of **Msc** $\rightarrow$ **Set**. This means that a can be understood as a *type of component* rather than just a box. When this type of component is used more than one time for building other type components in **Msc**, there are many actual components that are not explicitly named. As a consequence, to specify an actual component by avoiding to flatten types of components, we need an ordered set of types of components (boxes) going from the initial considered type of component to the outermost upper type of component that embeds it. Through this order, when a type of component it at least two times used for the next upper type of component, its position in the domain must be specified. We call such a specification, a *vertical path* on the structure of types of components. And it is defined as follows.

$$\begin{aligned} vpath : VP &\rightarrow Usage \\ h = [(a_0, p_0), \ldots (a_{k-1}, p_{k-1}), (a_k, p_k)] &\mapsto u \end{aligned} \tag{6.5}$$

such that if $a_i^1, ..., a_i^{m_i} \rightarrow a_i$ is an arrow, we have $a_{i-1} \in \{a_i^1, ..., a_i^{m_i}\}$, $i \geq 1$. Where VP, $Usage$, and $a_i^1, ..., a_i^m \rightarrow a_i$ are the set of all possible vertical paths, the set of all actual components, and an arrow in **Msc**.

This identification could be very suitable when analyses are done on the topological properties of models, for instance when exploring an HFSM (see Section 4.1.3).

6.5.3 Data format's structure

The data are representations of models and therefore their structure. The representation is also accompanied by other elements referring to the architecture such as the corresponding boxes and possibly the data referring to an interpretation of the structure of models. The resulting format, defined with the aforementioned object notation is as follows.

dataOfModels(components:Set[Component],other:Props)
Component(ID,ports:Set[Port],links:set[Link], constituents:Set[Constituent], typeComponent:_,other:Props)
Constituent(ID,type:Component,other:Props)
Link(ID,source:Component,sourcePos:Int,sourcePort:Port,target:Component, targetPos:Int,targetPort:Port, typeLink:_,other:Props)
Port(ID,nature:_,other:Props)
Props(String → _)

Data published under this format enable populating the specification (6.4) that characterises FA. In particular, the objects Constituent and Component enable to build **Set** and **Msc** respectively. Whereas the property typeComponent of Component and the property typeLink of LinK enable to build the objects of **Msin**. The data of functors F and I are implicitly embedded in the object Component and Constituent respectively.

Example. The data corresponding to the model whose the structure is the box on Figure 6.2 are given as follows.

dataOfModels0=({X1,X2,Y},nil)
X_1=(1,{a, b, e, f},{},{},nil,nil), X_2=(2,{c, d, g},{},{},nil,nil),
Y=(3,{h, i, j, k},{l_1, l_2, l_3, l_4, l_5},{u1,u2},nil,nil)
a=(a,IN,nil), b=(b,IN,nil), ... k=(k,OUT,nil)
l_1=(l_1,Y,-1,h,X_1,0,b,nil,nil), ... l_5=(l_5,X_1,0,e,Y,-1,j,nil,nil)
u1=(u1,X_1,nil), u2=(u2,X_2,nil)

We discuss the implementation details of FA in Section 7.

6.6 Related research

In this section, we start in Section 6.6.1, by discussing the FMI[5] (Functional Mockup Interface) standard which attempts to really solve the problem stated in Section 6.1. Then in Section 6.6.2, other relevant related works correlated to FA are also discussed.

6.6.1 FMI

FMI is a tool-independent interface that aims to enable the exchange of models and their interoperation (cosimulation). Indeed, FMI describes two modes of operation: FMI for model exchange and FMI for co-simulation. An FMU (Functional Mock-up Unit), a component in FMI, must implement via xml-files and C-code the FMI API. That is, following FMI, a model is described under an FMU, generated by different tools. The kinds of models considered by FMI are simulation models (see, e.g. [12] for more details). Although the co-simulation/operation of models is important almost unavoidable in the engineering of complex systems, it is not the primary goal of this section to deal with the co-operation of models. Nevertheless, the way (like an FMU) governing the definition of a model will influence its reuse.

An FMU separates the description of interface data (via an xml-file) and its functionality via a C-code.
The interface data describes the static elements of the model such as type definitions (Real, Integer etc.); model variables; model structure which is an ordered list of outputs (Outputs) states (Derivatives) and some initial unknown data (InitialUnknowns); units etc.
The C-code and header files define the functions that an FMU must implement. These functions are used to simulate a model described under an FMU. Roughly, the functions allow to initialise an FMU, assign values to an input variable, get the value of an output variable, evolve the state of an FMU, etc. More information about the specification of the FMI standard and the supported tools can be found on the FMI website.

Although the FMI standard provides an interface that FMUs must implement, there exist several semantic gaps (studied in [96] [25]) between different model

[5] https://www.fmi-standard.org/

semantics (e.g, discrete events, dataflow models) or modelling languages and the target interface [97]. The semantics of the target interface has been considered as a timed Mealy machine [96]. Another concern is to what extent the interface can be used to capture the semantics of different kinds of models [97]. It turns out that the specification and definition of the common interface are crucial. Moreover the FMI standard is mainly concerned with the exchange of models and their simulation as black boxes.

We have followed the FMI standard on the separation of concerns (structure and function). But, unlike FMI, we gave a formal definition of FA. The structure of models were considered as composite components defined within a category-theoretic framework. Furthermore, we do not impose a predefined interface for the implementation of the function. Rather, such an implementation has to be derived from an interpretation (or semantics) of the structure of models. The code associated to such implementation must expose the functions whose inputs and outputs are solely correlated to an interpretation of the structure. The data corresponding to the instantiations (i.e. actual serialized models) of the structure are elements that are persisted as a file.

6.6.2 Other works

The mathematical object, a symmetric multicategory (see Section 6.3) corresponding to the definition of the structure of models presented in this section is tightly linked to the "wiring diagrams" structure developed in a series of recent papers by Spivak et al. [79], [90], [99] and [92]. The differences are:

(i) the one presented in this thesis is a more general construction supported by the "wiring" or connection pattern, i.e. almost all ways of connecting sub-components of a composite component are allowed (see Section 6.4.1 on what is not permitted in WD). A WD is basically given by

$$(X^{in} + X^{out}) \to (Y^{in} + X^{out}) \leftarrow (Y^{in} + Y^{out}) \tag{6.6}$$

or precisely

$$(X^{in} + Y^{out}) \to (Y^{in} + X^{out}) \tag{6.7}$$

which can be decomposed into two functions

$$\begin{aligned} \phi^{in} &: \quad X^{in} \to Y^{in} + X^{out} \\ \phi^{out} &: \quad Y^{out} \to X^{out} \end{aligned} \tag{6.8}$$

In this thesis, to obtain (6.1) from (6.8), we basically associate elements of ϕ^{in} and ϕ^{out} to the abstract sets of links or wires L^{in} and L^{out}. To get (6.8) from (6.1) it is sufficient to have θ^{in} and θ^{out} of (6.1) that are injective. The elements of L^{in} and L^{out} can be seen as the labels of wires.

(ii) we are not mainly interested in a particular interpretation (semantics) of the structure. We rather provide data and computational structures and their implementation for dealing with the categorical objects and therefore the architecture in practice.

A theoretical framework where the structures of models are symmetric monoidal and compact closed categories, specified or interpreted with linear logics and verified with proof theories all bound by functors, has been also discussed in [65]. Furthermore, the framework is considered as a possible foundation of systems engineering; since it is adapted to current modelling languages and tools and encompasses existing formalisms such as UML, SysML, etc. We follow this framework on the base category for the structure of models. However, unlike this framework, we focused on the structure of models, means to define and implement it and its interpretations.

Mathematical approaches from Model theory [50], Institution theory [45], and Category theory [67] deal with syntax, structure, algebra and logic of models. But they often remain too theoretical (but not useless), insofar they are not directly applicable/implementable in practice (e.g., what is the right data structure?). Moreover, for applicable ones, although models are formal (based on a specific theory) and built with tools, they might lack an interchangeable and flexible reuse support. On the other hand, it has been pointed out in [29], that it is hard to mathematically classify the numerous modelling languages and techniques and that several MDE (Model-Driven Engineering) approaches are not grounded on formal semantics. Furthermore, formal or semi-formal ones are not necessarily built on an optimal architecture that will ease their reuse and exploitation. So there is a need to bridge the gap between theory and practice. We address this challenge for the architecture of models.

From a more practical yet formal point of view, a box can be compared to the concept of actor [98]. In [98], boxes are encoded following the logic of the modelling tool Ptolemy and the Java programming language. This may hamper the reuse of models. Contrary to [98], we do not associate a particular semantics (i.e. the target of an interpretation of the structure) to a box. In [98], boxes or actors are seen as extended timed state machines since these actors model behavioural models. This semantics is also called the actor interface. The definition of this interface mixes

the structure and the semantics of the box. However, the actors interface can be considered as the target of an interpretation of the structure of boxes since the input and output ports of a box are explicit in this interface. Composite actors are basically considered as a set of actors while a composite box is an arrow in **Msc** in this work. Let us stress that [98] is mainly concerned with an unified description of the behavioural semantics of the composition of several actors resulting in a composite actor.

Let us recall that we do not deal with a particular semantics of models nor a particular implementation of this semantics. Instead, we provide the principles (via 6.4 and the corresponding data structures) governing the definition of a semantics.

Just like a complex system, models do have an architecture and they are complex systems in their own right. The role of architecture is paramount since it affects how systems are built and evolve [24]. In particular, architecture is central in the management of complexity, emergent behavior, function behavior and the so-called "ilities" (extra properties such as flexibility, reliability, scalability, safety etc.) [24].

Finally, note that many successful e-business and software are built on the loosely-coupled architecture so-called SOA–Service Oriented Architecture, which is independent of any implementation, tool or technology. Following the spirit of SOA (see e.g. OSLC[6](Open Services for Lifecyle Collaboration)), we believe the function of models has to be exposed as as functions/services/libraries that are formally correlated to an interpretation of the structure of models.

6.7 Conclusion

In this section, we presented FA from theoretical to practical perspectives. FA is formalised via the specification: $\mathbf{Msin} \xleftarrow{F} \mathbf{Msc} \xrightarrow{I} \mathbf{Set}$ (6.4).The aim was to provide a formal and flexible architecture for models so that they are pluggable (explicitly and formally understood) and exploitable (for system analysis concerns). Categorical settings are useful to abstract away from details, differentiate and correlate the structure of models and their interpretations. On a practical perspective, data and computational structures, invariants, and data format corresponding to mathematical (category-theoretic) objects are presented. Setting up this framework makes it possible to plug and exploit different models independently or transparently to the tool used to build them. A part of this setting up that is not prescriptive, is presented in Section 7. Further work is discussed in Section 8.

[6] https://open-services.net/

Setting up and using MODEF in practice

The foundations of the proposed methodology have been discussed so far. We are now ready to describe how we make things work together in practice. We start by presenting (Section 7.1) the elements that did not exist before MODEF and that we implemented to be able to run MODEF. Then we present (Section 7.2) how they are associated to the different steps of MODEF, i.e., its use. Finally, we discuss (Section 7.3) some qualities associated the setting up and use of MODEF.

7.1 Implementation's building blocks

The main implemented elements are depicted on Figure 7.1. Almost all of the elements related to implementation are not prescriptive in the sense that other implementation choices could be done. But in the following we argue on the appropriateness of the selected elements. In the following we discuss each of these elements in turn.

7.1.1 Reuse models outside the modelling tool

A modelling tool is necessary to build the three considered kind of models. Once models are built, it is necessary to exploit them in the analysis procedure outside the modelling tool. As a consequence, a first step towards their exploitation is to export (or at least make them available) outside the modelling tool. On the other hand, a semantics to incrementally exploring them must be available: either universally or provided by the modelling tool. Here we face a challenge: how to export the relevant (for analysis purposes) data related to models and use their exploration semantics in the analysis code. The second element that comes into play is therefore a way

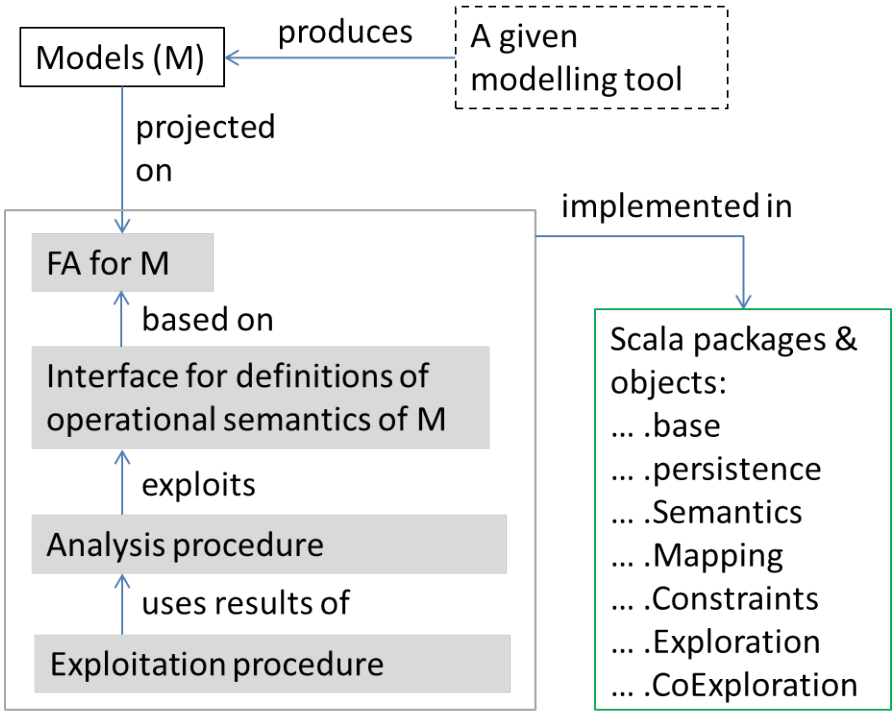


Figure 7.1: Implementation's building blocks

that enables an exploitation of the models outside the modelling tool for analysis purposes.

7.1.2 Availability and exploitation of models

The availability and exploitation of models are addressed under the architecture specified by **Msin** $\overset{F}{\leftarrow}$ **Msc** $\overset{I}{\rightarrow}$ **Set** in Section 6 represented by the box "FA for M" on Figure 7.1. We argued in Section 6 on the benefits expected from such a function-structure-data oriented architecture. Specifically, it supports the ability to implement the functions (related to the semantics of models) necessary to incrementally explore models. This means that such functions could be available as libraries/services/code because their implementation is independent of the logic of a particular tool. The independence also supposes that the data related to the specification of the architecture are all available.

The data structures and base invariants (see Section 6.5.1), corresponding to the specification **Msin** $\overset{F}{\leftarrow}$ **Msc** $\overset{I}{\rightarrow}$ **Set**, have been implemented using the Scala[1] programming language. They are encapsulated in the scala module/package: *modelling.base*. Scala is very useful here since it encompasses the object and functional programming paradigms. The functional aspect is close to the theoretical founda-

[1] https://www.scala-lang.org/

tions' considerations of the specification while the object oriented aspect allows to benefit from its principles such as encapsulation and polymorphism.

The persistence of data related to the specification via a data format (See Section 6.5.3) is realised with JSON[2] (JavaScript Object Notation). JSON which is a strongly and simple data-oriented formalism, has been indeed used to serialize the models projected on the devised architecture. At the code level, the unmarshalling of these data is managed by a module: *modelling.persistence*.

In practice, a tool that needs to explicitly expose models on the specified architecture must first, project the models on the devised specification (6.4), second translate (export) them into the proposed data format. Indeed we have built a connector (a script) from the side of the selected tool (Mega, see below) to achieve this. If the models are not physically exportable, the functions that describe the exploration semantics must be nonetheless (e.g., remotely) accessible and executable.

Thanks to the data associated to **Msin**, F and **Msc**, one can implement the primitives necessary to explore the models. The interface that allows to deal with that implementation is the box labelled "Interface for definitions of operational semantics of M" on Figure 7.1.

We have presented the operational, procedural, computational or functional semantics of state and process models that we consider in Section 4.1. Since we are not primarily interested by a semantics of structure models, in the following, we present the primitives corresponding to the state and process models. At the code level, these primitives are implemented in the *modelling.analysis.Semantics* object. Note that, whether models are physically exportable or not, the description of such functions might be universally available. Note also that we have presented the abstract syntax of the three kinds models in Section 4.

7.1.2.1 State models

For convenience, we recall from Section 4.1.3, that the procedural semantics of state models is defined under an HFSM. An HFSM defines an *initial state*, a *current state* and a *state-transition relation*. At the primitives level, these data translate into two main functions: *initialState* and *nextPossibleStates*. *initialState* is such that given a state model (or an HFSM), it returns its *initial state*. *nextPossibleStates* takes as inputs a state model and a *current state* and returns the next possible states and associated transitions by applying the relevant *state-transition relation*.

[2] http://www.json.org/

Whence the implemented Scala signature of the function *nextPossibleStates*:

$$
\begin{aligned}
&nextPossibleStates(\\
&\quad curState : List[Tuple2[Long, Int]],\\
&\quad smId : Long,\\
&\quad smStructure : SMultiCategoryBox,\\
&\quad struToSem : SMultiCategoryMap[Box, BoxArrow, Box, BoxArrow]\\
&\quad loader : Long => Object => Any) :\\
&\quad Map[Long, List[Tuple2[Long, Int]]]
\end{aligned}
$$

curState is the current state, a given state of the state model with the identifier *smId*. *smStructure* is the object that encapsulates the description of the structure of the model. *struToSem* is the object that functionally determines for each element of the structure of the model, its meaning, function or interpretation. For instance, with *struToSem*, it is possible to know if a box in the structure of a model is simply a "state" or "the initial state". *loader* is an utility function that allows to load any object from its identifier. Recall that **SMultiCategoryBox** and **SMultiCategoryMap** are defined in Section 6.5.1.

7.1.2.2 Process models

We recall from Section 4.1.2, that the procedural semantics of process models is defined under a non-deterministic finite state automaton–FSA. Such an automaton is obtained throughout the exploration of process models. Indeed, the exploration of a process model generates a language where the alphabet is the finite set of possible events from the input model. The words of this language are event-traces or the sequences of events. Accepting words or states are the ones that indicate the end of a branch of execution of the process. Therefore, one needs to implement at least the primitives to compute: (i) given a current state of the generated FSA, if it is accepting or not (ii) given a current state of the generated FSA, the possible next states with associated actions. It is worth noticing here that the notion of event refers to the end of an action, task or process and should not be confound with event in a BPMN models.

Since process models are built under the BPMN notation, one needs to specify for each considered construct, building block or BPMN element (event, task, gateway, etc.) how it behaves, such that one can explore the state space of the process models; i.e. implement the primitives mentioned above. The Business Process Model and

Notation–BPMN[3] is high level modelling notation.

We do not attempt to fully capture the semantics of a BPMN-like model, rather, we focus on a subset of building blocks that are sufficient for the objectives of this thesis. Especially, we mainly focus on the scheduling of tasks i.e. their flow rather than the data processed and shared between those tasks. The syntactic elements of BPMN are available at http://www.omg.org/spec/BPMN/. We exploited their textual semantics to derive the primitives necessary to explore process models. The main implemented primitives are: (a)*nextPossibleEventsByNode*, (b)*isNodeFranchisable*, (c)*usedEventsOfAFranchisableNode*, (d)*initialEventOfAProcess* and (e)*isFinalNode*. (a), (b), (c), (d) and (e) compute respectively: the next possible events of an active node, whether a node is franchisable or not, the events consumed when a node is crossed, the first events of a process, whether a node is final i.e. the process is finished after it or not. For instance, the Scala signature of *nextPossibleEvents* is as follows.

$$
\begin{aligned}
&nextPossibleEvents(\\
&\quad node : List[Tuple2[Long, Int]],\\
&\quad pmStructure : SMultiCategoryBox,\\
&\quad struToSem : SMultiCategoryMap[Box, BoxArrow, Box, BoxArrow],\\
&\quad loader : Long => Object => Any) : Set[Set[SeqFlow]]
\end{aligned}
$$

node is a node (or an internal box) of the process model with the structure encapsulated in *pmStructure*. *struToSem* has the same role as the one mentioned for the state model. For instance, it is possible to know if a box in the structure of a model is a "kind of Gateway", an "Event" etc. *SeqFlow* is an object that represents a transition between 2 nodes or boxes.

Note that, whenever one wants to deal with a model projected on the specified architecture, at the code level, they are mainly accessible via the data structures: **SMultiCategoryBox** and **SMultiCategoryMap**.

Once models are available and exploitable, it is now possible to implement analysis and exploitation algorithms; the next elements that come into play. Note that, other input models/data (Mappings (MG), Expectations (R), etc.) are also easily serialized under a JSON format. At the code level, they are managed with the objects: *modelling.Analysis.Mapping* and *modelling.Analysis.Constraints*.

[3] BPMN–http://www.omg.org/spec/BPMN/

7.1.3 Analysis and exploitation algorithms

The analysis and exploitation algorithms correspond to the boxes "Analysis procedure" and "Exploitation procedure" respectively, on Figure 7.1. In the following we discuss each in turn.

7.1.3.1 Analysis algorithms

The general analysis procedure (Section 5.1.2) is depicted on Figure 5.1. This general procedure involves two main algorithms: CoExploration (1) and MBMW (2). In the coExploration algorithm implementation, the exploration semantics of process and state models are exposed each, via an interface. Such an interface encapsulates, from a high level point of view, the resulting semantics (i.e., resulting from the application of primitives describing the semantics of a model) graph of the exploration of models. This encapsulation is such that the calls to the exploration primitives of process and state models are easily managed in the co-exploration of models. The interface is mainly equipped with the operations on a graph such as the successors of a node. The interface is initialised with the functions (based on primitives available from Section 7.1.2) necessary for an exploration of a model and some initial values. It defines three main functions: $reset, init, next$ which resets all the internal variables to default values; initialises with initial parameters; returns the next possibles points given the current one.

The two interfaces $smInt$ and $pmInt$, that are input parameters of the algorithm CoExploration (1)) are implemented within the object: $modelling.Analysis.Exploration$. Note that $smInt$ and $pmInt$ are such that they can be executed each alone to compute a reachable state space of a model.

MBMW and CoExploration have been implemented together in the same object: $modelling.Analysis.CoExploration$. However, the portions of code corresponding to the different algorithms are identifiable. Indeed, CoExploration is an input parameter of MBMW. As a consequence, in the sequel, we substitute the general analysis procedure to MBMW.

As soon as we can coexplore state models and process models connected by mappings, MBMW is straightforwardly implemented. The input parameters of MBMW are then: AM, SM, PM, MG, Coexploration, InitialPoint, StopCriterion, Expectations (R) and possibly Process constraints (C). MBMW computes a directed graph (SSG) that contains the points reachable at a minimal "distance" from the InitialPoint up to the stop criterion. With this graph, we now need to implement the

exploitation algorithms to provide stakeholders with effective feedbacks or simply, we need to make results understandable by a human.

7.1.3.2 Exploitation algorithms

Since it is possible to explore process and state models separately, the implemented interfaces *smInt* and *pmInt* can be used to generate the flattened graph corresponding to an execution or exploration of one of this model. All generated graphs are saved under a GraphML[4] file. The exploitation algorithms are all implemented in the Scala object: *modelling.analysis.utils*. The main exploitation algorithms and the output data are discussed in Section 5.2. Basically, they involve graph algorithms format to produce new customized graphs.

The originality and relevance of the implementation come from the fact that: i) we do not introduce a new modelling tool or language ii) the definition of the exploration semantics of models is explicitly implemented and decoupled from analysis and exploitation algorithms and typically could be reused as code or library. It follows that some parts of the implementation might be used for other purposes.

7.2 Use of MODEF in practice

The global picture of the structure an usage of MODEF is depicted on Figure 7.2. Figure 7.2 shows the flow of execution of MODEF. At the left of Figure 7.2 are the steps of MODEF. At the right of the different steps are either the results of the activity carried out at that corresponding step, or the tools necessary to carry out the step.

The boxes with the blue background represent the input models of MODEF. The box with a grey background represents the specifications, algorithms and their implementations respectively. It corresponds to the aforementioned building blocks of the implementation.This box allows to almost automatically transform its inputs into its output: the box with a green background. This latter box mainly represents the data that are supplied to stakeholders.

Since we have discussed before almost all of the elements of this usage, it remains to say a word about the selected tool: Mega[5]. MEGA has been the tool used during this thesis to specify models. In fact, it is one modelling tool used at the Airbus

[4]GraphML–http://graphml.graphdrawing.org/
[5]Mega–http://www.mega.com/

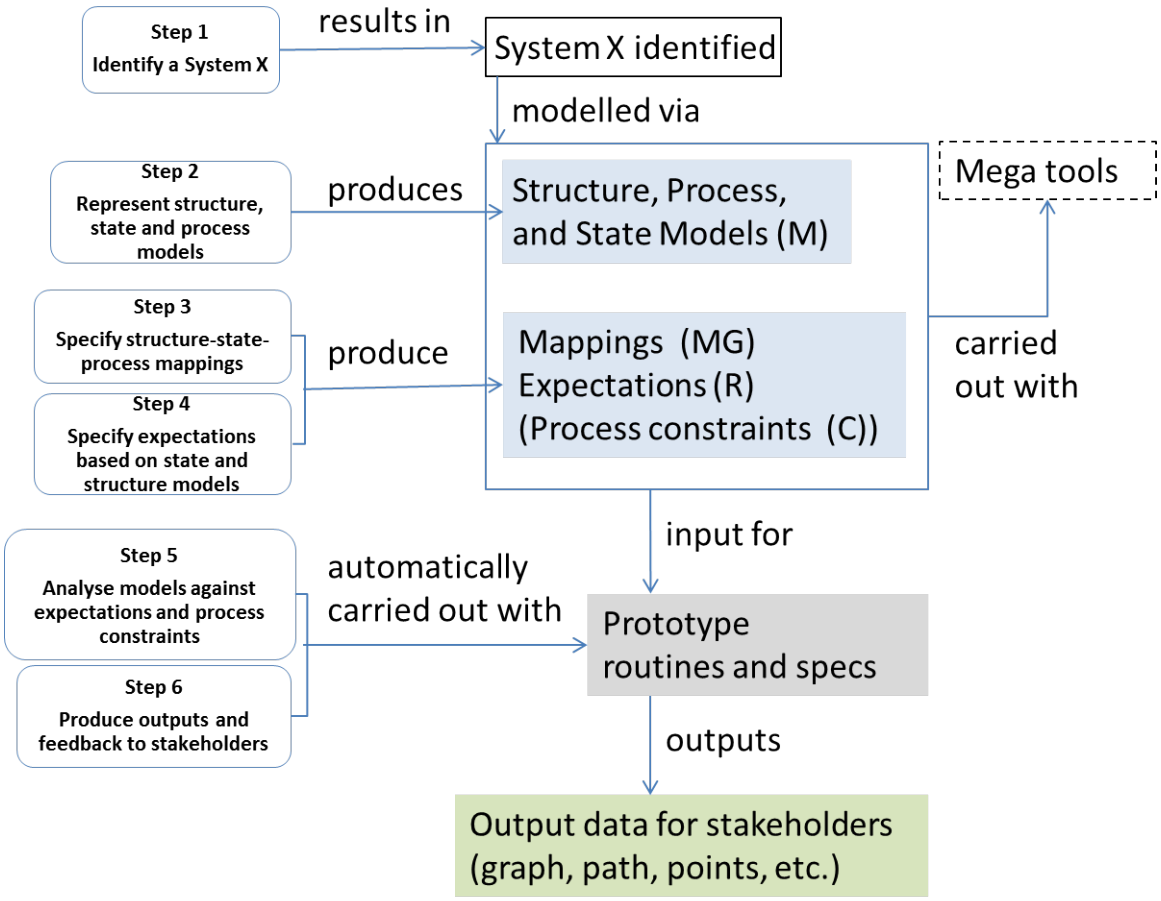


Figure 7.2: Using of MODEF

Defence & Space company. It is also freely available under an academic licence. Mega can support different modelling notations and languages. Indeed the models (M) are built under the standards: Unified Modelling Langage–UML[6] and BPMN. We have also customized the tool to ease the description of Expectations (R) and Mappings (MG). Mega also has a user-friendly graphical interface.

7.3 Algorithm performance and practicality

Despite the theoretical exponential complexity (see Section 5.1.3) of the main algorithm (MBMW) involved in MODEF, we argue on its advantages. Regarding the practicality, we will partly rely on implementation architecture to give an evaluation.

[6]UML–http://www.omg.org/spec/UML/

7.3.1 Algorithm performance

Despite the exponential theoretical complexity of UCS (which could be characterized as the theoretical understanding of the agility in the operation of modelling activities (MA)), significant depths are reachable in the state space in practice thanks to the effect of the cost function. This function allows to prune sub-regions of state space which is exponential in the size of state and process models. Besides, MA being essentially iterative and evolutive (and to a large extent a non-automatable task), a definitive behaviour is not necessarily relevant at a given point in the life cycle. What is necessary is a locally optimal behaviour which UCS allows to compute. Furthermore, we argued in Section 5 that the size of SoM and SoSoM are reasonable compared to other complex systems. Nonetheless, in the worst case, one might be interested in advanced techniques such as those used in model-checking (abstraction, partial order reduction etc.) to combat the state explosion problem.

The efficiency of data structures and implementation of algorithms is also important to discuss. Since we are now dealing with a prototype, we do not discuss it.

7.3.2 Practicality

By practicality, we mean how good, appropriate, MODEF is usable in practice independently of the techniques involved in MODEF.

In fact, the kinds of models considered in this thesis might be generated by several tools and modelling languages supporting the ability to model system architecture. Therefore, we think that an *ad hoc* implementation relative to a single modelling tool or language would not be relevant for the involvement of different stakeholders and application of MODEF. Nonetheless, in order to automatically and semi-formally dealing with models in practice, we require the projection of models on a function-structure-data oriented architecture of models. This architecture enables to explicitly and totally implement the exploration semantics of models, once for all. Therefore, the counterpart of the preservation of the specificities of different stakeholders is the necessity to build a connector that translates (not transforms) models on the proposed federated architecture.

Second, although obtaining the input models of MODEF requires a modelling expertise, the current considered exploitation algorithms produce the outputs that are understandable and exploitable by a non-expert. Besides, other exploitation and

analysis algorithms might be carried out since the architecture of the implementation of MODEF separates concerns.

Finally several applications of MODEF in real *situ* will be necessary to relevantly discuss its usability i.e. its efficiency, and satisfaction against the stakeholders and the objectives MODEF addresses.

A first step towards the demonstration of the usability of MODEF was the application examples and case studies that we presented before. The case study Maintenance of a Supermarket shows that the application of MODEF would not be limited to the Modelling Activity.

Conclusion and perspectives

8.1 Conclusion

In this work, we addressed the mastering of the modelling activity carried out in a model-based systems engineering framework built of autonomous yet collaborating stakeholders. The central question was: how to formally reason on the operation of the modelling activities?

The challenges identified and enumerated were:

(1) How to better understand and use models in the context of concurrent and autonomous development of the modelling activities?

(2) How to analyse and identify the impact of their changes? i.e. what is the current state of models and in which states are they likely to end up?

(3) How to help in mastering their evolution? i.e what is necessary to guarantee that models might reach some expected states?

To tackle these challenges, we have proposed a methodology: MODEF. In this methodology, the modelling activity is firstly characterized as a system (and federation of systems) in its own right. At the level of the architecture of this system, a class of discrete-event processes models, structure models and a class of finite state models are considered respectively to model the tasks carried out in a modelling activity, the conceptual content of the models (M) and their expected life cycles respectively. The effects of the tasks on the life cycles are also modelled via some triggers. We introduced the expectations (or assumption-preference) formalism to formalise the requirements related to the life cycles. An analysis procedure that exploits the co-exploration of process and state models constrained by the triggers was defined to both check how far requirements are achievable and to synthesize the expected behaviours of the system. An exploitation of the results of the analysis procedure enables to figure out what could happen with the modelling tasks and their impact on the whole state of M. We showed on 2 case studies how this exploitation provides insightful data on how the system is operated and how it can behave. Based

on this information, it is possible to take some preventive or corrective actions on how the modelling activity is carried out. As a result, the proof-of-concept of the proposed methodology is demonstrated on the 2 case studies.

To support the effectiveness of our methodology, we devised enablers to formally and modularly deal with models and algorithms involved in the methodology. At the conceptual level, we introduced a federated architecture to exploit models outside of the modelling tools used to produce them. At the practical level, a modular implementation that separates the different concerns (models, analysis and exploitation) is presented and set up. While these enablers are mainly intended to support our methodology, we argued that they might be useful for model-driven engineering from a broader perspective. Finally, the two case studies also demonstrate the applicability of these enablers.

Upstream and downstream of the proposed methodology, we highlighted the similarities, the differences and then the novelties of theoretical and practical elements related to MODEF with respect to the related work and the challenges addressed in this work.

8.2 Perspectives

The perspectives of this work are given for the components of MODEF, taken separately, and MODEF in its entirety. We present each of them in the following.

• **On Expectations**: We argued in Section 4 that it could be interesting to study the operations on Expectations (A/P). Indeed if Expectations were to be exploited in a compositional fashion, operations on expectations such as composition and conjunction would be relevant. Since preferences are based on relations (in a mathematical sense), the operations on relations could be helpful in this regard.

• **On the exploitation of analysis algorithms**: The exploitation of the analysis algorithms in Section 5 currently considers a single cost (related to node score) and a total ordering of the cumulated costs. Another exploitation that deals with constraints (time, etc.) on processes and a partial ordering of cumulated costs will be suitable. Indeed, a total ordering is not always easy to set up and becomes impossible to set up when objectives are conflicting. In this perspective, other aggregation techniques (bipolar evaluation, stochastic ordering, etc.), the Pareto dominance and other comparison criteria that induce a partial order could be helpful.

• **On the proposed federated architecture of models**: One question not addressed in Section 6 is the internal structure's change and evolution of models. It

could be canonically manageable with the structure of models. One could introduce an initial (zero) object (to model a destroyed box) in the structure, yielding an new structure K. Using this structure, one defines again a functor from a category K_t at a time t to the same category $K_{t'}$ at a time $t' > t$. By taking the time as objects in a category T equipped with some order, and defining a functor from T to K, we obtain the change and evolution of the internal structure of models having as structure K[34].

Another direction may consist in using the proposed architecture FA to implement some kinds of descriptive model semantics that are well-known and/or formally written down (see, e.g., [79], [90], [99] and [92] for algebraic ones; or atomic actors in [98]). More importantly, these implementations have to be available universally, in a tool-independent way.

• **On the entirety of MODEF**: Our future work includes the exploitation of the proposed methodology on real modelling projects to empirically discuss its efficiency. Therefore, an optimization of the implementation prototype might be necessary. For instance, although the implementation is modular, some algorithms and data structures are probably not optimally implemented. The exploitation of MODEF will also provide feedbacks to improve its different parts.

References

[1] Joël Abeille. *Vers un couplage des processus de conception de systèmes et de planification de projets: formalisation de connaissances méthodologiques et de connaissances métier*. PhD thesis, 2011.

[2] Russell L Ackoff. Towards a system of systems concepts. *Management science*, 17(11):661–671, 1971.

[3] Michel Aldanondo and Elise Vareilles. Configuration for mass customization: how to extend product configuration towards requirements and process configuration. *Journal of Intelligent Manufacturing*, 19(5):521–535, 2008.

[4] Michel Aldanondo, Elise Vareilles, and Meriem Djefel. Towards an association of product configuration with production planning. *International Journal of Mass Customisation*, 3(4):316–332, 2010.

[5] Jos CM Baeten, Joanna M van de Mortel-Fronczak, and Jacobus E Rooda. Integration of supervisory control synthesis in model-based systems engineering. In *Complex Systems*, pages 39–58. Springer, 2016.

[6] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of model checking*. MIT press, 2008.

[7] Yaneer Bar-Yam. When systems engineering fails-toward complex systems engineering. In *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, volume 2, pages 2021–2028. IEEE, 2003.

[8] Albert Benveniste, Benoît Caillaud, Alberto Ferrari, Leonardo Mangeruca, Roberto Passerone, and Christos Sofronis. Multiple viewpoint contract-based specification and design. In *International Symposium on Formal Methods for Components and Objects*, pages 200–225. Springer, 2007.

[9] Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Sangiovanni-Vincentelli, Werner Damm, Thomas Henzinger, and Kim G Larsen. Contracts for system design. 2012.

[10] Jean Bézivin. Model driven engineering: An emerging technical space. *Lecture Notes in Computer Science*, 4143:36, 2006.

[11] Benjamin S Blanchard. *System engineering management*. John Wiley & Sons, 2004.

[12] Torsten Blochwitz, Martin Otter, Johan Akesson, Martin Arnold, Christoph Clauss, Hilding Elmqvist, Markus Friedrich, Andreas Junghanns, Jakob Mauss, Dietmar Neumerkel, et al. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*, number 076, pages 173–184. Linköping University Electronic Press, 2012.

[13] Roderick Bloem, Krishnendu Chatterjee, Thomas A Henzinger, and Barbara Jobstmann. Better quality in synthesis through quantitative objectives. In *International Conference on Computer Aided Verification*, pages 140–156. Springer, 2009.

[14] Roderick Bloem, Rüdiger Ehlers, Swen Jacobs, and Robert Könighofer. How to handle assumptions in synthesis. *arXiv preprint arXiv:1407.5395*, 2014.

[15] Eric Bonjour. Contributions à l'instrumentation du métier d'architecte système: de l'architecture modulaire du produit à l'organisation du système de conception., 2008.

[16] Tyson R Browning and Steven D Eppinger. Modeling impacts of process architecture on cost and schedule risk in product development. *IEEE transactions on engineering management*, 49(4):428–442, 2002.

[17] Dario Campagna and Andrea Formisano. Product and production process modeling and configuration. *Fundamenta Informaticae*, 124(4):403–425, 2013.

[18] Pascal Cantot and Dominique Luzeaux. *Simulation and Modeling of Systems of Systems*. John Wiley & Sons, 2013.

[19] Handy Charles. Balancing corporate power: A new federalist paper. *Harvard Business Review*, 70(6), 1992.

[20] Krishnendu Chatterjee, Laurent Doyen, and Thomas A Henzinger. Expressiveness and closure properties for quantitative languages. In *Logic In Computer Science, 2009. LICS'09. 24th Annual IEEE Symposium on*, pages 199–208. IEEE, 2009.

[21] Soo-Haeng Cho and Steven Eppinger. Product development process modeling using advanced simulation. 2001.

[22] Edmund M Clarke, Orna Grumberg, and Doron Peled. Model checking. 2000, 2000.

[23] Thierry Coudert. Formalisation et exploitation de connaissances et d'expériences pour l'aide à la décision dans les processus d'ingénierie système. 2014.

[24] Edward Crawley, Olivier De Weck, Christopher Magee, Joel Moses, Warren Seering, Joel Schindall, David Wallace, Daniel Whitney, et al. The influence of architecture in engineering systems (monograph). 2004.

[25] Fabio Cremona, Marten Lohstroh, Stavros Tripakis, Christopher Brooks, and Edward A Lee. Fide: an fmi integrated development environment. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 1759–1766. ACM, 2016.

[26] Werner Damm, Hardi Hungar, Bernhard Josko, Thomas Peikenkamp, and Ingo Stierand. Using contract-based component specifications for virtual integration testing and architecture design. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011.

[27] Frédéric Demoly, Xiu-Tian Yan, Benoît Eynard, Samuel Gomes, and Dimitris Kiritsis. Integrated product relationships management: a model to enable concurrent product design and assembly sequence planning. *Journal of Engineering Design*, 23(7):544–561, 2012.

[28] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[29] Zinovy Diskin and Tom Maibaum. Category theory and model-driven engineering: from formal semantics to design patterns and beyond. *Model-Driven Engineering of Information Systems: Principles, Techniques, and Practice*, page 173, 2014.

[30] Dov Dori. *Object-process methodology: A holistic systems paradigm*. Springer-Verlag Berlin Heidelberg, 2002.

[31] Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theoretical Computer Science*, 380(1-2):69–86, 2007.

[32] Manfred Droste and Werner Kuich. *Semirings and Formal Power Series*, pages 3–28. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[33] Claudia M Eckert, David C Wynn, Jakob F Maier, Albert Albers, Nikola Bursac, Hilario L Xin Chen, P John Clarkson, Kilian Gericke, Bartosz Gladysz, and Daniel Shapiro. On the integration of product and process models in engineering design. *Design Science*, 3, 2017.

[34] Andrée C Ehresmann and J-P Vanbremeersch. Hierarchical evolutive systems: A mathematical model for complex systems. *Bulletin of Mathematical Biology*, 49(1):13–50, 1987.

[35] Dominique Ernadote. An automated objective-driven approach to drive the usage of the naf framework. *Information Systems Technology Panel (IST) Symposium*, 2013. Accessed October 2013.

[36] Dominique Ernadote. An ontology mindset for system engineering. In *Systems Engineering (ISSE), 2015 IEEE International Symposium on*, pages 454–460. IEEE, 2015.

[37] Dominique Ernadote. Ontology reconciliation for system engineering. In *Systems Engineering (ISSE), 2016 IEEE International Symposium on*, pages 1–8. IEEE, 2016.

[38] Rik Eshuis. Reconciling statechart semantics. *Science of Computer Programming*, 74(3):65–99, 2009.

[39] Uli Fahrenberg and Axel Legay. The quantitative linear-time–branching-time spectrum. *Theoretical Computer Science*, 538:54–69, 2014.

[40] Serge Fiorèse and Jean-Pierre Meinadier. Découvrir et comprendre l'ingénierie système. *CEPADUES Editions, ISBN*, 978(36493.005):6, 2012.

[41] John Fitzgerald, Peter Gorm Larsen, and Jim Woodcock. Foundations for model-based engineering of systems of systems. In *Complex Systems Design & Management*, pages 1–19. Springer, 2014.

[42] Kevin Forsberg, Hal Mooz, and Howard Cotterman. *Visualizing project management: models and frameworks for mastering complex systems*. John Wiley & Sons, 2005.

[43] Thomas Friend. Agile Project Success and Failure (The Story of the FBI Sentinel Program), 2017. Last access May 2017.

[44] Jürgen Gausemeier, Tobias Gaukstern, and Christian Tschirner. Systems engineering management based on a discipline-spanning system model. *Procedia Computer Science*, 16:303–312, 2013.

[45] Joseph A Goguen and Rod M Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM (JACM)*, 39(1):95–146, 1992.

[46] Charles Handy. Trust and the virtual organization. *Harvard business review*, 73(3):40, 1995.

[47] David Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.

[48] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[49] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[50] Wilfrid Hodges. *Model theory*, volume 42. Cambridge University Press, 1993.

[51] Thomas O. Holland. *Modeling and Simulation in the Systems Engineering Life Cycle: Core Concepts and Accompanying Lectures*, chapter Model-Based Systems Engineering, pages 299–306. Springer London, London, 2015.

[52] INCOSE. *Systems Engineering Handbook: A Guide for System Life Cycle Process and Activities*. John Wiley & Sons, Inc., 4th edition, 2015.

[53] INCOSE. Systems engineering vision 2025, http://www.incose.org/docs/default-source/aboutse/se-vision-2025.pdf, 2015. Last access August 2015.

[54] Swen Jacobs and Roderick Bloem. The reactive synthesis competition: Syntcomp 2016 and beyond. *arXiv preprint arXiv:1611.07626*, 2016.

[55] MO Jamshidi. System of systems engineering-new challenges for the 21st century. *Aerospace and Electronic Systems Magazine, IEEE*, 23(5):4–19, 2008.

[56] Freddy Kamdem Simo, Dominique Lenne, and Dominique Ernadote. Mastering SoS complexity through a methodical tailoring of modeling: Benefits and new issues. In *Systems Conference (SysCon), 2015 9th Annual IEEE International*, pages 516–520. IEEE, 2015.

[57] Freddy Kamdem Simo, Dominique Lenne, and Dominique Ernadote. Towards Modelling of Modelling in SE. In *2016 IEEE International Symposium on Systems Engineering (ISSE)*, Edinburgh, United Kingdom, oct 2016.

[58] Stephen J Kapurch. *NASA Systems Engineering Handbook*. DIANE Publishing, 2010.

[59] Robert M Keller. Formal verification of parallel programs. *Communications of the ACM*, 19(7):371–384, 1976.

[60] Stuart Kent. Model driven engineering. In *Integrated formal methods*, pages 286–298. Springer, 2002.

[61] Annette J Krygiel. Behind the wizard's curtain. an integration environment for a system of systems. Technical report, DTIC Document, 1999.

[62] Tom Leinster. Higher operads, higher categories. *arXiv preprint math0305049*, 2003.

[63] Tom Leinster. *Higher operads, higher categories*, volume 298. Cambridge University Press, 2004.

[64] John Leonard. Systems engineering fundamentals. Technical report, DTIC Document, 1999.

[65] Dominique Luzeaux. A formal foundation of systems engineering. In *Complex Systems Design& Management*, pages 133–148. Springer, 2015.

[66] Dominique Luzeaux and Jean-René Ruault. *Systèmes de systèmes: concepts et illustrations pratiques*. Hermès science publications-Lavoisier, 2008.

[67] Saunders Mac Lane. *Categories for the working mathematician. 2nd ed.* New York, NY: Springer, 2nd ed edition, 1998.

[68] Mark W Maier. Architecting principles for systems-of-systems. In *INCOSE International Symposium*, volume 6, pages 565–573. Wiley Online Library, 1996.

[69] Andres Felipe Melo. *A state-action model for design process planning.* PhD thesis, Department of Engineering, University of Cambridge, 2002.

[70] Bertrand Meyer. Applying 'design by contract'. *Computer*, 25(10):40–51, 1992.

[71] Dana S Nau, Vipin Kumar, and Laveen Kanal. General branch and bound, and its relation to a* and ao*. *Artificial Intelligence*, 23(1):29–58, 1984.

[72] Pierluigi Nuzzo, Huan Xu, Necmiye Ozay, John B Finn, Alberto L Sangiovanni-Vincentelli, Richard M Murray, Alexandre Donzé, and Sanjit A Seshia. A contract-based methodology for aircraft electric power system design. *IEEE Access*, 2:1–25, 2014.

[73] Brendan Donal O'Donovan. *Modelling and simulation of engineering design processes.* PhD thesis, University of Cambridge, 2004.

[74] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 179–190. ACM, 1989.

[75] G. Pol, C. Merlo, J. Legardeur, and G. Jared. Analysing collaborative practices in design to support project managers. *International Journal of Computer Integrated Manufacturing*, 20(7):654–668, 2007.

[76] Peter J Ramadge and W Murray Wonham. Supervisory control of a class of discrete event processes. *SIAM journal on control and optimization*, 25(1):206–230, 1987.

[77] Donna H Rhodes, Ricardo Valerdi, and Garry J Roedler. Systems engineering leading indicators for assessing program and technical effectiveness. *Systems Engineering*, 12(1):21–35, 2009.

[78] David Romero-Hernández and David de Frutos Escrig. Coinductive definition of distances between processes: beyond bisimulation distances. In *International Conference on Formal Techniques for Distributed Objects, Components, and Systems*, pages 249–265. Springer, 2014.

[79] Dylan Rupel and David I Spivak. The operad of temporal wiring diagrams: formalizing a graphical language for discrete-time processes. *arXiv preprint arXiv:1307.6894*, 2013.

[80] Stuart Russell and Peter Norvig. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27, 1995.

[81] Andrew P Sage and Christopher D Cuppan. On the systems engineering and management of systems of systems and federations of systems. *Information, Knowledge, Systems Management*, 2(4):325–345, 2001.

[82] Andrew P Sage and William B Rouse. *Handbook of systems engineering and management.* John Wiley & Sons, 2009.

[83] Brian Sauser, John Boardman, and Alex Gorod. *System of Systems Management*, pages 191–217. John Wiley & Sons, Inc., 2008.

[84] Douglas C Schmidt. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2):25, 2006.

[85] INCOSE SEBoK. SEBok Guide to the Systems Engineering Body of Knowledge, http://sebokwiki.org, 2016. Last access June 2017.

[86] Sanjit A Seshia, Natasha Sharygina, and Stavros Tripakis. Modeling for verification. *Handbook of Model Checking, EM Clarke, T. Henzinger, and H. Veith, Eds. Springer*, 2014.

[87] Amira Sharon, Olivier L de Weck, and Dov Dori. Project management vs. systems engineering management: A practitioners' view on integrating the project and product domains. *Systems Engineering*, 14(4):427–440, 2011.

[88] Amira Sharon, Valeria Perelman, and Dov Dori. A project-product lifecycle management approach for improved systems engineering practices. In *INCOSE International Symposium*, volume 18, pages 942–957. Wiley Online Library, 2008.

[89] Weiming Shen, Douglas H Norrie, and Jean-Paul Barthès. *Multi-agent systems for concurrent intelligent design and manufacturing*. CRC press, 2003.

[90] David I Spivak. The operad of wiring diagrams: Formalizing a graphical language for databases, recursion, and plug-and-play circuits. *arXiv preprint arXiv:1305.0297*, 2013.

[91] David I Spivak. *Category theory for the sciences*. MIT Press, 2014.

[92] David I Spivak. Nesting of dynamic systems and mode-dependent networks. *arXiv preprint arXiv:1502.07380*, 2015.

[93] Duane Steward and Derrick Tate. Integration of axiomatic design and project planning. In *Proceedings of ICAD2000, First International Conference on Axiomatic Design, MA-June*, pages 21–23, 2000.

[94] CMMI Product Team. Cmmi for development, version 1.2. 2006.

[95] Stavros Tripakis. *L'analyse formelle des systèmes temporisés en pratique*. PhD thesis, Université Joseph-Fourier-Grenoble I, 1998.

[96] Stavros Tripakis. Bridging the semantic gap between heterogeneous modeling formalisms and FMI. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2015 International Conference on*, pages 60–69. IEEE, 2015.

[97] Stavros Tripakis. Compositionality in the science of system design. *Proceedings of the IEEE*, 104(5):960–972, 2016.

[98] Stavros Tripakis, Christos Stergiou, Chris Shaver, and Edward A Lee. A modular formal semantics for ptolemy. *Mathematical Structures in Computer Science*, 23(04):834–881, 2013.

[99] Dmitry Vagner, David I Spivak, and Eugene Lerman. Algebras of open dynamical systems on the operad of wiring diagrams. *Theory and Applications of Categories*, 30(51):1793–1822, 2015.

[100] Elise Vareilles, Thierry Coudert, Michel Aldanondo, Laurent Geneste, and Joel Abeille. System design and project planning: Model and rules to manage their interactions. *Integrated Computer-Aided Engineering*, 22(4):327–342, 2015.

[101] Jon Whittle, John Hutchinson, and Mark Rouncefield. The state of practice in model-driven engineering. *IEEE software*, 31(3):79–85, 2014.

[102] NIST CCT Workshop. Computational Category Theory Workshop at NIST on Sept. 28-29, http://www.appliedcategorytheory.org/?p=10, 2015. Last accessed December 2016.

[103] David C Wynn. *Model-based approaches to support process improvement in complex product development*. PhD thesis, University of Cambridge, 2007.

[104] David C. Wynn and P. John Clarkson. Process models in design and development. *Research in Engineering Design*, Jul 2017.