Research in Engineering Design (2021) 32:3–30
https://doi.org/10.1007/s00163-020-00347-4

ORIGINAL PAPER

Principles for coping with the modelling activity of engineered systems

F. Kamdem Simo[3] · D. Ernadote[2] · D. Lenne[1] · M. Sallak[1]

Received: 22 August 2017 / Revised: 4 October 2020 / Accepted: 7 October 2020 / Published online: 2 November 2020

Abstract
The systems engineering of some systems often involves challenging modelling activity (MA). MA presents challenges, which include understanding the context in which it takes place, understanding and managing its impacts on the life cycles of the models it produces. In this paper, we propose a methodology and its underpinning framework for addressing these challenges and for coping with the operation of MA. The first step in our methodology is to characterize MA as a federation of systems. It then consists in iteratively building a system architecture by modelling the models produced by MA and their expected life cycles, modelling the various tasks that constitute MA, and modelling the effects of MA on these life cycles. It then makes it possible to specify expectations over these life cycles and to analyse models of MA in relation to expectations, to check how far expectations are achievable and to synthesize the acceptable behaviours of the system. Finally, a use of the results of this analysis may provide insightful data on how the system is end-to-end operated and how it might behave. On the basis of this information, informed decisions may be made to act on the logistics of MA. The hypotheses, theoretical foundations, the models, the algorithms and perspectives relating to the proposed methodology and its underpinning framework are all presented and discussed.

Keywords Systems engineering · Modelling activity dynamic

Abbreviations

A/G	Assume/Guarantee
A/P	Assume/Preference
AM	Structure models
C	Constraints on processes
CPM	Critical Path Method
DSM	Design Structure Matrix
EVM	Earned Value Method
HFSM	Hierarchical Finite State Model
M	Models produced by the Modelling Activity
MA	Modelling Activity
MBSE	Model-Based Systems Engineering
MG	Mappings (effects of PM on SM of AM)
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	Expectations
SD	System dynamics
SE	Systems engineering
SEMP	SE Management Plan
SEMS	SE Master Schedule
SM	State models
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

✉ F. Kamdem Simo
frks@protonmail.ch

D. Ernadote
dominique.ernadote@airbus.com

D. Lenne
dlenne@hds.utc.fr

M. Sallak
sallakmo@hds.utc.fr

[1] Alliance Sorbonne Université, Université de Technologie de Compiègne (UTC), Heudiasyc, CNRS Centre de recherche Royallieu CS 60319, 60203 Compiègne, France

[2] Airbus Defence and Space (ADS), Elancourt, France

[3] UTC and ADS, Compiègne and Elancourt, France

TA	Technical activity
TP	Technical processes
TMP	Technical Management Processes

1 Introduction

The engineering of systems will often involve some modelling activity MA. Models generally provide a partial, sometimes incomplete view of the actual modelled thing, and this kind of simplified view is essential when it comes to understand tricky systems. Models also give us a way of preserving and reusing knowledge about the things that they relate to. However, because of the sheer diversity of engineered or studied systems, models are often specific to a particular domain (mechanical, electrical, chemical, hardware, software and systems engineering, purchasing, etc.), and different types of models provide different perspectives on the modelled systems. Combining different models in pursuit of a single objective (such as verification) is not a new challenge, but it remains highly topical.

The dynamic of MA influences and determines the evolution of models. Models arising from modelling activity in industry are often spread over different locations. At the same time, (large engineering) companies will have a number of separate projects and programs running concurrently with different instances of modelling activity that sometimes interact. Models can have useful lifespans ranging from a few days to several months. Thus, the modelling activity itself can be seen as a challenging entity to operate.

We are, therefore, dealing with two distinct levels of complexity, one relating to the engineered systems and the other to the entities and practices that contribute to the modelling of those systems.

As a result, understanding, mastering and engineering systems can be made difficult by the environment of their life-cycle or their tricky nature.

On one hand, the people and other components within these environments are heterogeneous and mature, offering potential opportunities and benefits. On the other hand, these very characteristics can have adverse side effects.

In this context, Systems Engineering aims to create harmony and added value between well-established domains/components (for the resolution of a sub-goal) while targeting the overall system goal.

There is no obvious single approach that can be applied to all types of systems.

If an overview and trend of the dynamics of the MA become difficult to grasp, the following questions may arise.

(1) Existing models: what models are present in a particular location and what do they represent?
(2) Direction of travel: what is the current state of models and how and where are they likely to end up?
(3) Moving towards desired direction and states: what is required for models to reach desired states?

2 Motivation, hypotheses, problem formulation and contributions

Motivation: In seeking to address the three questions above, our immediate concern was the study of MA development and operation. Therefore, the system under study in this paper is MA. By mentioning the system, we refer, unless otherwise specified, to the studied system. MA involves human operators who are called upon to perform tasks that are evolutive and even creative. MA cannot be specified once and for all, but rather, expected behaviour and results need to be continuously reworked and re-specified. This raises the question of whether it is possible to master, in a disciplined way, the dynamics of MA, to contribute to their logistics in an informed manner.

In Kamdem Simo et al. (2015), we reported an attempt to organise some types of modelling activity for system architecture. Modelling activitiy was tailored using a modelling management plan, fed by a Modelling Planning Process (MPP) (Ernadote 2013), itself automated to ease modelling operations. The MPP aligns the MA to the requirements of the projects, ensuring that modelling objectives are defined and prioritized, that they correspond to the various modelling artefacts (project concepts, standards and deliverables), and that the progress of modelling activity can be assessed. The authors concluded that this modelling activity needed to be federated, since the approach proposed did not take account of the autonomous and co-evolving nature of MA. Models have targets and can play a role in different modelling projects in engineering environments.

Consequently, in a subsequent work (Kamdem Simo et al. 2016), we argued that MA can be considered locally as a system and globally as a federation of systems that need to be engineered. Following on from that, in the present paper, we introduce a methodology and its underpinning framework for coping with the operation of MA in systems engineering.

Hypotheses and modelling choices: We consider the environment of the system to be (Ackoff 1971):

“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 Ackoff (1971) that a system and its environment are relative to an observer, consequently they can be conceptualized in different ways.

We are, therefore, concerned with the system proper and with its environment. The system proper and its environment form the closed system. In this paper, we often use the term system to refer to either the system proper or the closed system, what is meant will be clear from the context. We assume that the environment is autonomous and to all intents and purposes not controllable, but that it may always be represented by a model. This assumption reflects the fact that while models can be expected to reach certain predetermined states, those states may be reached via different modelling tasks and different sequences of modelling tasks.

The system proper is structurally represented by structural models (AM). AM are especially useful in abstracting away the main content of models (M) produced by MA. State models (SM) are used to model the life cycles of M and the expected transitions between the states over these life cycles. Process models (PM) are used to model the behaviour of the environment and sometimes the behaviour of components of the system. In particular, PM capture the modelling tasks which cause changes in the state of M. We model the effects of PM on the system proper by the mapping (MG) of events (from the exploration of PM) onto transitions of SM associated with AM. PM might be also subject to some constraints (C) (e.g., time, cost, etc.) related to modelling tasks. The Expectations (R) that specify preferences on the expected states of the system might be defined something like this: given a set of pairs (component of AM, state in SM)—called context—some pairs complementing this context will be more preferable than others. This paper will formalize these different models (AM, PM, SM, MG and R) and discuss their suitability in abstracting, representing, and understanding MA.

General problem to solve: Given the MA, understood and modelled with the data corresponding to AM, SM, PM, MG and R, what are the possible future *points* (foreseeable evolutions) starting from an initial point (InitialPoint) and evolving until a stop criterion (StopCriterion) becomes true? Which points are with respect to R and C, more acceptable / less acceptable? Let us call these questions Q1.

A point is a possible configuration of the state of the closed system. InitialPoint is a given point from which the closed system might need to be initialized. StopCriterion is a means for selecting acceptable points among the reachable points. Therefore, the general problem is given by Pb (AM, SM, PM, MG, R, C, InitialPoint, StopCriterion).

We propose addressing the questions Q1 via a six-step methodology that we call MODEF. MODEF is intended to (1) provide an understanding of the current global state of MA, (2) check whether MA is moving in a satisfactory direction (state path), and (3) help stakeholders build processes to ensure that the models (M) that it produces continue to evolve in appropriate ways.

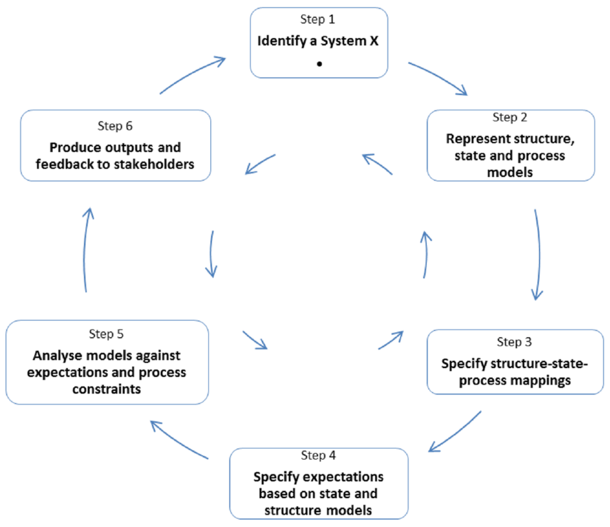

Step 1
Identify a System X
Step 2
Represent structure, state and process models
Step 3
Specify structure-state-process mappings
Step 4
Specify expectations based on state and structure models
Step 5
Analyse models against expectations and process constraints
Step 6
Produce outputs and feedback to stakeholders

Fig. 1 The procedural structure of MODEF

Here, we are not explicitly dealing with the internal practices of MA. The design techniques, methods and tools used in the modelling activity are not explicit, i.e. they are not modelled. The internal practices are black boxes for the proposed methodology. These internal practices of MA are of course relevant and essential for producing M, but we shall argue in this paper why we consider the black-box perspective.

Contributions and organisation:

- From a general systems engineering perspective, the main contribution of this paper is the introduction of MODEF–the procedural structure of which is depicted on Fig. 1, summarized by Act/Identify-Model-Specify-Verify-Inform-Identify—and with its supporting framework with principles, theoretical and practical arguments for understanding, modelling and analysing MA to inform MA's logistics.

On Fig 1, a box depicts a step; the starting and entry step is Step 1 (Identify a System X); an arrow from a step (*s*) to the next one (*t*) denotes: the outcome (data) of *s* is necessary for carrying out *t*; finally, at a given step, it is possible to go back—for instance when new data is available or when a problem is found at the current step—to any previous step, whence the counter clockwise arrows.

The novelties introduced in this work in relation to other work are presented in Sect. 3.

From a narrower perspective, focusing on the different steps of MODEF, the main contributions are the following:

- To our knowledge, it is the first time a modelling of the architecture of MA and expectations relating to MA is based on the models that we have detailed above, that is to say AM, SM, PM, and MG. MA is considered in terms of a *System of Modelling* (**SoM**) and a *System of Systems of Modelling* (**SoSoM**) (Kamdem Simo et al. 2016). We explain the meaning of these terms in Sect. 4 below.
- We introduce an Assume (A)/Preference (P) formalism to support the specification of expectations (or, more generally, expected behaviour) of MA. The stimulus here came from Assume/Guarantee (A/G) contracts (Benveniste et al. 2012). An expectation consists in expressing preferences with respect to the life cycle of the studied elements, given some context or assumption. In particular, these preferences are defined with respect to the states of the studied elements given an assumption (A). In the style of A/G contracts, the G of a contract is replaced by a preference (P) in an expectation. P has a pre-order (a binary relation that is reflexive and transitive) structure. A pre-order structure is generally sufficient to describe preferences among the elements of a given set. Another reason for why we chose to adapt the A/G paradigm is that the verification of consistency and compatibility may be formally defined (like with A/G contracts).
- We introduce a modular analysis procedure that makes it possible to compute the answer to questions Q1. This procedure applies the uniform-cost search (UCS) algorithm (Russell and Norvig 1995) on the state space described by the co-exploration of SM and PM, both being constrained by MG. The novelty of this procedure is that it makes use of R (and potentially C) in UCS to guide the co-exploration throughout the discovered state space and in some cases to prune regions within this space.
- Finally, to make the results of analysis understandable by a human, we provide some operating algorithms for building synthetic data. We want, for example, to be able to determine (1) whether, starting from a given point, no further improvements will be possible (in terms of expectations and process constraints) given the input models; (2) the sequence of points or configurations necessary to reach a target point; (3) critical paths, etc. This kind of synthetic data may help stakeholders to operate the system, to prevent issues and to take corrective actions.

The principles, models and algorithms underlying the six steps of MODEF are illustrated on a running example—introduced in Sect. 4.2.1—used as a synthetic case study. Where necessary, we shall give additional examples.

The remainder of this paper is organised as follows. In Sect. 3, we discuss the relevant related work and position MODEF in relation to it. In Sect. 4, we briefly recall what we mean by SoM and SoSoM then present some application examples that illustrate them. In Sect. 5, we present the principles and semantics of three kinds of models (AM, SM and PM) and their mappings (MG). We also present the formalism of Expectations that underpins the specification of R. In Sect. 6, we present the analysis procedure for exploring and analysing models against the expectations (and potentially constraints of modelling tasks). Then we present an application of the analysis procedure. In Sect. 7, we give a brief answer, with related assumptions, to the three questions put forward in the introduction, then we present some limitations and perspectives of this work.

3 Related work

Here, we discuss various approaches that have sought to manage modelling activity or technical-and-programmatic activity and we try to position MODEF in relation to them.

3.1 Systems engineering and modelling activity

Systems Engineering (SE) has been considered for nearly eighty years (see e.g., (INCOSE 2015; AAAS 2016) for the evolution and definitions of SE) to manage the complexity of the engineered systems and the processes to which they give rise. SE has continued to grow out of two significant disciplines (Leonard 1999), namely “the technical knowledge domain in which the systems engineer operates, and systems engineering management.” The former is generally related to technical activity (requirement engineering, design, etc.), whereas the latter is concerned with the planning and management of the former. It should be noted 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 (SEBoK 2016; Sage and Rouse 2009; Blanchard 2004).

It can also be said that SE is concerned with two distinct systems: on one hand there is the system that is being engineered, developed, or studied, and on the other hand there is the system that enables the engineering of the studied system. These two systems are also called the system to be made (also referred to as the System-Of-Interest (SOI)) and the *system for making* respectively (Fiorèse and Meinadier 2012). Consequently, SE comprises activity of a technical nature and activity of a programmatic nature. Technical activity (TA) seeks to answer the question, “Are we engineering the right SOI and in the right way?” Programmatic activity (PA) seeks to answer the question, “Are we operating the right system for making, and are we doing so in the right way?” It is this programmatic activity that corresponds to SE management. Finally, it should be remarked that PA and TA influence each other mutually.

Depending on the difficulties originating from the SOI itself and the engineering environment, different efforts are made on the two kinds of activity. However, within a context characterized by the two levels of complexity described in Sect. 1, both domains (technical and programmatic) are inexorable and critical: obstacles brought about by the engineering environment that then contributes to additional complexity that must be managed (Bar-Yam 2003) in conjunction with the basic complexity of the SOI.

One approach for dealing efficiently with SE has been the use of (formal) models and modelling as the main support of SE. Such an approach refers to Model-Based SE (MBSE) (INCOSE 2015) and to a large extent model-driven engineering (MDE) (Whittle et al. 2014; Kent 2002). MBSE is not a new discipline different from SE, since (mathematical) models were used many years ago within various fields, including SE. Today's MBSE seems to focus on diagrams with the risk of multiplying less formal and less explanatory models. The efficiency with MBSE comes from the ability of models and modelling to improve communications among the stakeholders; to give rise to better representations and a better understanding of systems, and to facilitate the preservation and reusing of knowledge relating to SE processes. Models and modelling are central in an MBSE approach.

SE is well documented in terms of standards and handbooks (INCOSE 2015; Sage and Rouse 2009; Fiorèse and Meinadier 2012). Some standards that provide an overarching view on SE processes are succinctly described in the following.

ISO/IEC/IEEE 15288 provides a description of processes involved in a SE process. The SE Management Plan encapsulates the artefacts (SE Master Schedule, Work Breakdown Structure) necessary for planning and controlling these processes. Other standards (ANSI/EIA-632, ISO/IEC 15504:2004, IEEE Std 1220-2005, etc.) provide guidance for describing, improving, and assessing these processes. Systems Engineering actors can (and in fact do) therefore leverage those materials, mainly built from past experiences, for new SE developments. It follows from the foregoing that the SE body of knowledge does contribute to the management of SE processes regardless of how they are actually implemented.

Although the materials and standards mentioned above are clearly useful, they often remain document-based, even if some cover formal content. They are generally descriptive in addressing the question *What To Make* but they fail to be prescriptive and to deal with the question *How to*. As a consequence, they are unsatisfactory from the perspective of a model-based management of MA. Indeed, the following problems may arise: reuse, analysis, consistency, traceability. Moreover, the diversity and autonomy of different stakeholders of the MA might prevent an effective use of such documents. Besides, in this latter situation, even with formal artefacts (models or documents), one may encounter the same problems as those encountered with informal artefacts. Nonetheless, formal artefacts offer several advantages (reuse, share, analysis etc.) provided that they are adequately operated. Thereby, if we consider PA interrelated to TA, as a system, SE (or generally design principles) is (are) applicable for its understanding. From this perspective, and provided that *research project* as understood in Reich (2017), is replaced by (or compared with) that system, the expected benefits as the corollary of the *Principle of Reflexive Practice* (Reich 2017) and challenges would apply.

SE has been successfully applied many times over the years, particularly in defence and aerospace, and yet today it is still based on heuristics and informal practices as argued in INCOSE (2014). On the other side, many large engineering projects have failed. For a list of past projects see for example (Bar-Yam 2003), whose conclusion is that the various failures may be attributed to the complexity of the projects themselves; see also the story in Friend (2017) of some recent failed projects. Meanwhile, it is expected that SE will continue to be applied more widely across other industries (INCOSE 2014). For SE to spread successfully, formal and unifying models to support SE processes will need to be created using SE's current body of knowledge.

3.2 Product-and-project approaches in systems engineering

In this section, we start by presenting some relevant model-based product-and-project oriented approaches in systems engineering, and then we attempt to situate MODEF in relation to these various approaches.

The OPM approach In Sharon et al. (2011), the focus is first placed the tandem (project-product dimension) comprising the project (the programmatic aspect) and the product (the technical aspect). The authors compare the methods of project management that are common in SE management together with the Object-Process methodology OPM (Dori 2002) used for project planning and product modelling and design. Among others, these methods include: 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 planning and product design.

Using a simplified unmanned aerial vehicle (UAV) as a system use case, their empirical comparison shows that some methods are relevant with respect to particular product and project factors. Factors include Budget/Schedule measurement/tracking, Stakeholder/Agent tracking, Performance quality, and Product measurement/tracking. Only SD, DSM and OPM methods were found to be able to handle the project-product dimension. SD is a way of correlating factors (schedule, budget) relating to project planning in way that may be plotted. DSM represents the interactions among elements (components, tasks, teams) of both the project and product. The authors finally conclude that OPM is the only suitable for the function-structure-behaviour modelling (i.e. Project-Product Model-Based (Sharon et al. 2008)) of both the project and the product inside an integrated conceptual model.

Apart from OPM, all the methods compared are best derived from models that represent the product and project, since they address a particular, specific concern. The OPM method is required for both TA and PA. The authors claim that OPM is especially well adapted to combining PA and TA within a single model. It follows 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 processes in the project. Another approach that builds on OPM models to support planning and communicating of process and product development is the PROVE-Dev framework (Shaked and Reich 2018).

Coupling of TA and PA In Vareilles et al. (2015), a model and rules are discussed for managing the multi-level interaction between system design processes (typically TA) and project planning processes (typically PA). The rules were integrated into the ATLAS IT platform. In the light of the failures of the A380 Program and Olkiluoto Nuclear Power Plant projects, which were executed within a concurrent engineering environment and based on empirical surveys, the authors argue that there is a vital need to formalize the interactions between the design of a system and its design project. They also highlight the absence of any work formally addressing this need from the perspective of planning and controlling design activity. These interactions have been made explicit in only very few works.

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 development 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 have 3 states, namely 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, and 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 likewise have 3 states: undetermined (UD), verified (OK), unverified (KO). Like for 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, nine synchronisation rules (for S and P) are finally defined to guarantee the consistent evolution of system design and project design. This yields a 27-state state diagram that supports the synchronisation of S and P. In this diagram, a state corresponds to 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 comprising 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 (Abeille 2011; Coudert 2014).

Other works in product (and process) design development (see e.g., Braha and Maimon 2013; Cho and Eppinger 2001; Melo 2002; Browning and Eppinger 2002; O'Donovan 2004; Wynn 2007; Bonjour 2008; Abeille 2011; Karniel and Reich 2011; Gausemeier et al. 2013) have addressed the association, integration or coupling (explicitly or not) of the two systems, i.e. the product and the project (or the development system) using a variety of means and with a variety of aims. Regarding software development projects, see also (Steward and Tate 2000). Here, the functional requirements and design parameters of the product are mapped onto tasks in a Gantt chart project plan following an Axiomatic Design paradigm (Steward and Tate 2000). The authors claim that such an association between tasks and design enables the rapid delivery of product.

A survey on process models and modelling approaches for design and development process (DDP) appears in Wynn and Clarkson (2017). The authors first focus on three features (novelty, complexity and iteration) of DDPs. DDPs are different from business (typically production and manufacturing) processes insofar as they call for creativity; they are iterative by nature. They occur in large-scale concurrent evolving engineering environments. These features also apply to MA. The authors then discuss the different approaches (see (Wynn and Clarkson 2017, Fig. 1)), from model scope (*micro*, *meso* and *macro* levels) and type (*procedural*, *analytic*, *abstract* and *management science/operations research* models) axes (again see Wynn and Clarkson 2017 for more details). Their conclusion is that different models provide different insights depending on how they are used, and models should be used according to specific objectives (i.e. requirements and constraints). Orthogonally to these concerns, tooling issues such as modelling notations and tools should be addressed (Abeille 2011) (Eckert et al. 2017).

A summary of a handful of approaches that associate design and planning appears in Abeille (2011) and 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. Some approaches to the integration of product and process models in engineering design with respect to their purposes (Visualisation, Planning, Execution, Synthesis, Analysis, etc.), their modelling formalisms (Design Structure Matrices, IDEF (Integrated Definition), etc.) and their level of integration (isolated, coupled, integrated) appear in Eckert et al. (2017). The authors remark that few works address the integration of product and process domains. They also remark that approaches appearing theoretically closer to the industrial context would require certain challenges regarding the scope, focus, development and visualisation of models to be overcome.

3.3 Position of MODEF

In this section we position MODEF in relation to the relate work presented above.

3.3.1 Abstracting modelling activity

Not all of the above approaches explicitly deal with the concurrent nature of the engineering environments (comprising autonomous and even independent stakeholders). Although (Vareilles et al. 2015) clearly recognizes the influence of this concurrent nature, the multi-level approach that is presented does not explicitly consider it. Note also that integrated approaches may not be effective in a concurrent context with regard to the capability and autonomy of different stakeholders. This is even more true with engineering activities of Systems of Systems–SoS (Jamshidi 2008). Generally speaking, any traditional approach for SE and management should be reviewed with regard to its ability to meet the requirements of SoS engineering. (Sage and Cuppan 2001) discusses evolutionary principles and implications of the *federalism* concept for SE and management of SoS. This federalism is important, because engineering development alliances have clearly been taking the form of virtual organizations (Sage and Cuppan 2001; Handy 1995). This encourages autonomy and loosely coupled systems but requires well-defined interfaces between autonomous systems. The authors argue that the components of those system may be: “locally managed and optimized independently.” They also argue this kinds of federations of engineering development projects should be considered as Complex Adaptive Systems, and they discuss the need for federated organisations to be thoroughly understood. They point out the importance of modelling, simulations and analyses.

Unlike the approaches developed in Vareilles et al. (2015) and Sharon et al. (2011), we do not focus on a particular methodology within the TA. We make no stipulations regarding the content (paradigms, methods, languages, tools, etc.) necessary for TA to function, because we consider that such stipulations do not take into account the diversity , heterogeneity of domain-specific approaches in MBSE. For instance, if we were to stipulate that the System-of-Interest should be also modelled with a function structure behaviour approach (Sharon et al. 2008, 2011), this might be restrictive, since TA is generally characterized by several model-based domain-specific practices. Instead, we abstract away details and concentrate on the relevant content of models (M) produced by MA.

The approach in Vareilles et al. (2015) (and even (Sharon et al. 2008, 2011)) imposes a structure for TA and PA. In doing so, it also creates what would appear to be an interesting dynamic for TA and PA. However, this diminishes the flexibility of the approach. We believe a bijective link between the system S and the project P is too strong an assumption and might not always be relevant given the structure of S and its granularity. On the PA side, we only consider the sequencing of MA and its impacts on the state of M. The link between PA and TA is therefore modelled via the mappings.

On the other hand, the verification and validation attributes (Vareilles et al. 2015) are interesting, since they emanate from quantitative data that provide insights into the states of the elements that they relate to. In our proposed methodology, we believe such insights could be considered either as constraints for processes or as a useful addition for corroborating the states of M in practice. In the approach (Vareilles et al. 2015), these attributes are used for both TA and PA.

3.3.2 Modelling modelling activity

One similarity that our methodology has with (Sharon et al. 2011) 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 as 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, while in our proposed methodology, the processes bring about changes of states of M.

The methodology developed in this paper 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.) that are studied and compared in Sharon et al. (2011). Those techniques and factors should either be derived from the models and their analysis or considered as constraints for models. On the other hand, specific approaches focusing on quantitative insights could help in determining the constraints of MA, insofar as 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. Processes give procedural insights that may be useful in anticipating bad states or the deviation of thresholds. More interestingly, it may be possible to specify the required actions for applying remedies and adjustments afterwards.

3.3.3 Analysis with models

Looking at the above approaches, with two exceptions, to a lesser extent, we were unable to find any explicit formal analysis of the models concerned with respect to certain formalized expected properties. The first exception is (Vareilles et al. 2015), where the analysis derives from the synchronisation rules, and the second exception is in the OPM approach (Sharon et al. 2008) (Sharon et al. 2011), where the simulations of OPM models are intended to detect various problems (product and project parameters feasibility, deviations and impacts) and take appropriate actions.

The UCS algorithm that we use (see Sect. 6) to guide the exploration of models could possibly be replaced by another OR (Operations Research) or AI (Artificial Intelligence) algorithm, and the exploration algorithm (see Sect. 6) could even be customized.

3.3.4 Implementing methods

The methodology and its underpinning framework presented in this paper are not intended for use with a specific modelling tool or modelling language. Instead, we detail concepts and algorithms needed to facilitate implementation. The modelling tool provides a way of building models that illustrate the methodology and that may be of great practical use.

Referring to the classification of process models proposed by Wynn and Clarkson (2017) in the wider context of design and development, some models involved in MODEF belong to the following *model scope dimension* and *model type dimension* respectively: in between the *meso* and *macro* levels and a mix of *abstract*, *procedural*, *analytic*, (and possibly *MS/OR* models), respectively. Thus, after the analysis carried out within MODEF, it might feed approaches with other model scope dimension and model type dimension. These latter approaches might in turn influence the specification of input models of MODEF. For instance, the detailed information relative to MA may be useful to understand how it quantitatively

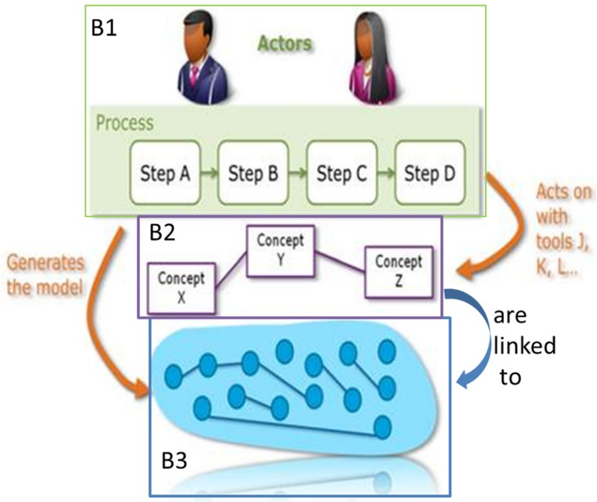


Fig. 2 Some components (and their relations) of the SoM

affects the state—in SM—of models (M). The following pattern summarizes the position of MODEF in that classification: *Macro-level model-based approaches* ↔ *MODEF* ↔ *Micro-level model-based approaches*", where $a \rightarrow b$ or $b \leftarrow a$ reads: the output of a might contribute to the input of b.

4 Abstractions of modelling activity

4.1 System of modelling (SoM) and system and systems of modelling (SoSoM)

The SoM whose some components and their relations are depicted on Fig. 2, is to a large extent a set of stakeholders and their practices. Broadly speaking it is a system of people, methods, tools, processes, standards and models (M) in interaction, and it may be seen as a system that reflects the engineering/modelling of a System-Of-Interest (SOI). On Fig. 2, we have three main blocks, a blue arrow and two orange arrows.

- At the top of Fig. 2, we have people and a sequencing of tasks (Step A, then Step B, etc.) they carry out. This green block (B1) represents the sequencing of MA.
- The purple block B2 contains three boxes corresponding to conceptual models that represent the conceptual contents of M.
- The blue block B3 contains the actual models (M) that represent the SOI that is being modelled.
- The left-side and right-side orange arrows indicate that the tasks in B1 generate models in B3 and act on conceptual models with different tools respectively.

– The blue arrow labelled “are linked to” on Fig. 2 indicates the connection between M and the conceptual models. Such a connection, its definition and implementation, applications are partly reported in Ernadote (2013, 2015, 2016); Kamdem Simo et al. (2015). Those conceptual models are useful as a means of involving different stakeholders in the modelling process (Ernadote 2015). For instance, stakeholders who are unfamiliar with specific metamodel concepts are involved via domain-specific concepts related to their viewpoints. The two kinds of concepts are therefore related via the combination of metamodels and ontologies.

In investigating the SoM, we focus on the sequencing of modelling tasks, the conceptual models that encapsulates the main content of M. To understand the impacts of modelling tasks on M, we link the conceptual models to state models that characterise the expected (or lifecyle) states of M, and we map events relating to the execution of tasks onto transitions of state models, so as to indicate the effects that these tasks have on states. We therefore examine the SoM from three different architectural perspectives.

Actually, several modelling projects are run in parallel. Indeed, many engineering programs involve several organisations located on different sites. Regardless of the geographical distribution, the engineering projects often address concerns that overlap. At the same time, the SoMs are generally autonomous.The question is how best to understand these SoMs and how they change over time.

The answer to this question is to be found in the SoSoM, which is constituted by different SoMs that encompass different projects and programs. The SoSoM is a way of understanding the evolving relationships between SoMs. It addresses the interactions or commonalities between SoMs. The added-value of the SoSoM emerges from these interactions. With the autonomy and the proper operational capabilities of individual SoMs, it is difficult to impose an integrated approach. Instead, as we argue in Kamdem Simo et al. (2016), following (Sage and Cuppan 2001; Handy 1995; Charles 1992), the SoSoM may be characterized as a federation. We recall that a Federation has in the past been considered as a type of System of Systems (Krygiel 1999; Sage and Cuppan 2001).

The views that we look at when analyzing the SoSoM are the same as when analyzing the SoM: that is to say the structure, state and MA processes linking the different SoMs that are its components. 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: the states of some elements of the structure view change under the effects of processes.

This means that the SoM and SoSoM are abstracted from the same perspectives but have different concerns.

4.2 Examples of application

We start by presenting elements that may in practice constitute a SoM, and we then discuss the problems that may need to be addressed at the level of the SoSoM comprising more than one SoM.

4.2.1 Running example: 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 purposes such as verification, testing, or integration. Suppose we are only interested in the functional architecture designed to ensure that the modelled SOI covers the functional needs. What characteristics does a modelling project need to possess for it to be defined as a SoM, say *SoM0*?

The following elements need to be identified:

– the sequencing of modelling tasks required to achieve some objectives
– the conceptual models that abstract away the main contents of models (M)
– the expected states of M and transitions between them corresponding to elements of the conceptual models
– the effects (mappings) of modelling tasks on these states.

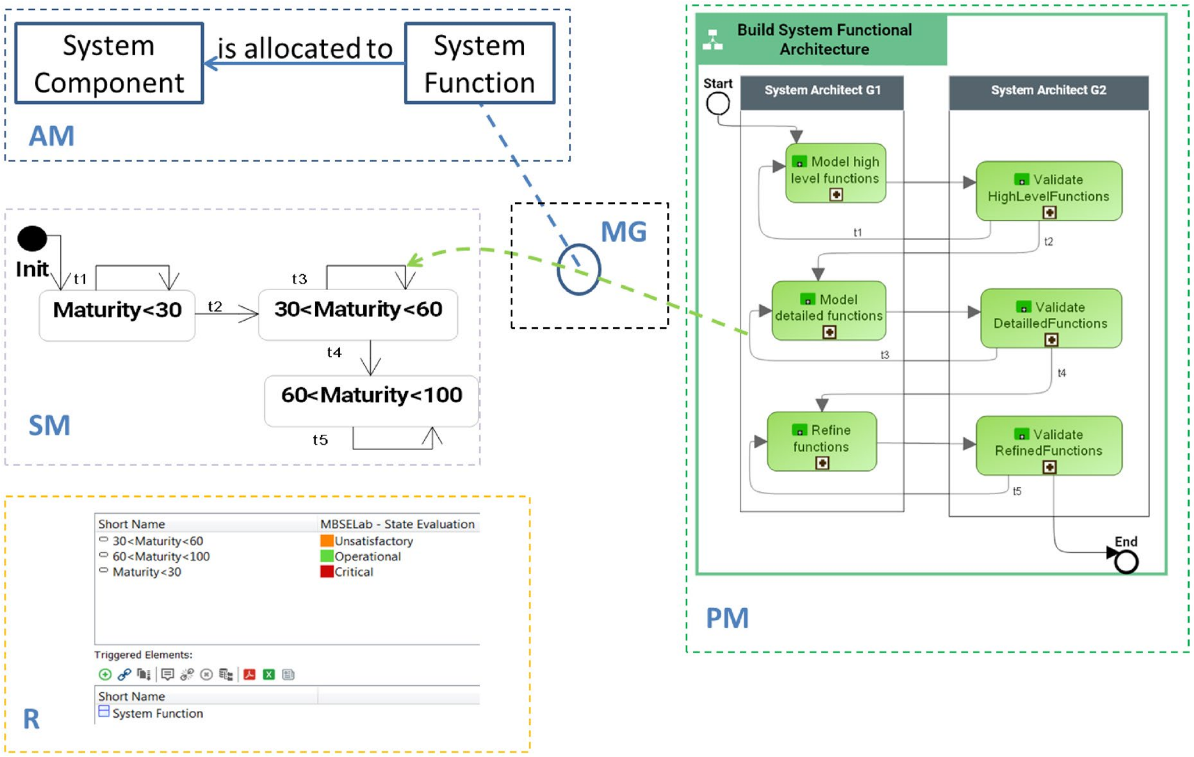
Fig. 3 is an illustration of these elements. This figure contains 5 boxes (PM, AM, SM, MG and R), with dashed borders, interpreted as follows. Note here that the formal specifications of the data related to these boxes are given in the following sections.

The box (PM)—corresponding to B1 according to Fig. 2—represents a high level view of the tasks carried out in SoM0. This process model contains six tasks beginning with the the task named “Model high level functions” and ending with the task named “Validate RefinedFunctions”.

The box (AM)—corresponding to B2 according to Fig. 2—represents the components or entities (named “System Component” and “System Function”) and their relation taken into account in SoM0.

The box (SM)—not depicted in Fig. 2—that has 3 states. The initial state is named “Maturity < 30”. Typically, a state here describes the maturity level of something that should be known after a mapping.

The box (MG) is a mapping. In Fig 3, there is actually 5 mappings. A mapping is an element of a function TRIGGER defined in Sect. 5.2. A mapping is a triplet $(a, l, e) \in$ TRIGGER. a is an element of AM, l is a transition in SM and e is a set of sequences flow. For convenience, in Fig 3, the symbols $t_i, i = 1..5$ are used to carry the 5 mappings and replace l and e when associated in a mapping. a is “System



Short Name	MBSELab - State Evaluation
30<Maturity<60	🟧 Unsatisfactory
60<Maturity<100	🟩 Operational
Maturity<30	🟥 Critical

Short Name
System Function

Fig. 3 Running example: The input models for the problem associated to the SoM0

Function" for all the five mappings. Therefore, the box MG roughly means: the sequence flow labelled "t3" in PM, triggers (dashed green arrow) the transition labelled "t3" in SM, which is associated (dashed blue line) to the component or entity "System Function" in AM. In practice, that implies the state of a model's component abstracted away by the entity "System Function" remains "$30 < \text{Maturity} < 60$" if it was already in that state and if the sequence flow labelled "t3" in PM is taken in the process. As a result, the task "Model detailed functions" should be carried out again. Additionally, the mapping "t3" does not allow to evolve from the state "$30 < \text{Maturity} < 60$" to the state "$60 < \text{Maturity} < 100$", while the mapping carried by "t4" does.

The box (R) corresponding to an A/P expectation will be discussed later (Sect. 5.3). We may say here that a preference (Satisfactory, Operational and Critical) is defined on the possible states associated with the entity "System Function".

Other SoMs, which could for example be a modelling project regarding a different viewpoint of the SOI, might be defined in the same way.

4.2.2 Examples of problems addressed at a SoSoM level

A SoSoM exists to encapsulate the relationships between at least two SoMs. The local view of the SoSoM is characterized by the different SoMs, whereas the global view will depend on the objectives at the SoSoM level. Suppose that these objectives are as follows:

Objective 1: We want to harmonize the models (M) produced by several SoMs. By harmonization we mean to correlate and explicitly identify the relationships among different entities (of the conceptual models) within several SoMs.

Objective 2: Suppose now, we are interested in the co-evolution of the modelling within the SoMs in order to achieve some higher goal. For this reason, although the SoMs are autonomous, there might be a need for agreements between them at some points in their respective life cycles. These agreements may ensure that different SoMs are together able to reach some expected states at some desired time in the future.

For each of these two objectives, we now present a SoSoM and the three main kinds of models that represent it.

Modelling harmonization—SoSoM1 Process models (PM) will abstract the tasks carried out at the SoSoM level. Conceptual models (AM) will correspond to the conceptual models that are relevant at the SoSoM level. State models (SM) will abstract the possible states at the SoSoM level (e.g., Unknown, Ok, Ko) of M within SoM.

Suppose now that the SoSoM1 comprises $n > 1$ SoM working on different modelling projects. The fact that conceptual models are explicitly given by the various SoMs represents a first step in reducing the number of redundant models and enabling model reuse where possible. Using processes (PM), and the expected states (SM) at the SoSoM level, it is possible to compute the attainable points (see modelling choices in Sect. 2) at the SoSoM level. This may have benefits including managing model replication and reconciliation procedures.

It is worth emphasising here that the process model at the SoSoM level (and even at the SoM level) does not indicate by what technical means tasks are executed, but it is instead concerned with how tasks are sequenced and, using the mappings, the effects that different tasks have on the states of elements.

SoSoM1 is not concerned with the state and process models of its constituent SoMs, but only with their conceptual models. However, depending on the objectives of the SoSoM, it might be interesting to consider the state and process models. The following SoSoM SoSoM2 provides an illustration of this.

Modelling evolution—SoSoM2 In this case, where Objective 2 is relevant to the SoSoM, the structure, state and process models of the SoSoM will combine the relevant subsets of the SoM models.

So far in this section, we have characterized the SoM and the SoSoM and we have given some application examples. We now need to discuss how we can make use of the models in these systems in practice.

5 Modelling modelling activity and expectations

This section elaborates the second step (Represent structure, state and process models) (Sect. 5.1), third step (Specify structure-state-process mappings) (Sect. 5.2) and the fourth step (Specify expectations) of MODEF (see Fig. 1) (Sect. 5.3). We start by introducing the principles and semantics underlying these kinds of models, and then we present the expectation-specification formalism.

5.1 Structure, process and state models

When seeking to automate the analysis of models using MODEF it is especially important to define the principles underlying the models. This enables us to decouple the implementation of step 5 (Analysis of models) of MODEF from the logic of a particular modelling tool used to design models. These models are mainly descriptive models.

Let the *structure (abstract syntax) of a model* of the system correspond to 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, we may suppose for the purposes of this section that a component c is given by

$$G = \langle V, E \rangle \tag{1}$$

where $V := \{c_1, \dots c_n\}$ is the set of internal components of c, and $E := \{l_1, \dots l_m\}$ 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.

Given the structure of a model, our aim is to obtain an interpretation of the structure in a domain of interest. The interpretation is derived from the structure of a component. When modelling 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. However, when modelling 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 transitions between its states.

Note that although the concrete syntax (graphical symbols used to render the models) is important for the end user, it has no importance in MODEF, because our aim is to decouple MODEF from the logic of a particular tool. It is useful only for the user designing and/or visualizing the models.

We now discuss the formal definitions of the semantics of the three kinds of models considered.

5.1.1 Structure model

The semantics of structure models will depend on the system at hand. For example, for the SoM, the structure models are interpreted as conceptual models, while for a physical system, they might be interpreted as computing, physical or human components. Generally speaking, the structure of structure models will be sufficient for the purposes of MODEF.

Running example See the box AM in Fig. 3.

5.1.2 Process model

The semantics of process models may be seen as generators in the sense given to the term by Ramadge and Wonham (1987). Roughly speaking, 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 possible terminations when the process is run. The set of words that bring the automaton from its initial state to an accepting state is referred to as the language recognized (or "marked") by the automaton.

The reasoning behind this is that modelling tasks may be considered as discrete-event processes. Also, the class (principal features: discrete, asynchronous and possibly non deterministic) of processes considered in Ramadge and Wonham (1987) is particularly well-suited to our needs, because our main focus is the sequencing of tasks rather than data processing.

We are interested in words of finite length, corresponding to executions of processes that terminate.

Adapting (Ramadge and Wonham 1987), we formally define our automaton as follows. A generator is a 5-tuple

$$\mathcal{G} = (Q, \Sigma, \delta, q_0, Q_m), \tag{2}$$

where Q is the set of *states* q, Σ is the *alphabet* or finite set of output symbols σ, $\delta : \Sigma \times Q \rightarrow Q$ 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 $q \rightarrow q'$ 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 may be recognized by an outside observer via its label σ by an outside observer. Distinct events at a given node always have distinct labels.

Running example In the box PM in Fig. 3, the sequence flows (the arrows labelled t_i in that process) and tasks (the green boxes) will typically correspond to events (i.e. Σ) and states (i.e. Q) respectively. See also Sect. 6.

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 $\delta : \Sigma^* \times Q \rightarrow Q, \delta(1, q) = q, q \in Q$ 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

$$L(\mathcal{G}) = \{w : w \in \Sigma^* \text{ and } \delta(\sigma, q_0) \text{ is defined}\}. \tag{3}$$

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

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

$$L_m(\mathcal{G}) = \{w : w \in L(\mathcal{G}) \text{ and } \delta(w, q_0) \in Q_m)\}. \tag{4}$$

5.1.3 State model

The semantics of state models is characterized as an Hierarchical Finite State Model (HFSM). By building on (1), the definition of an 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 a *constituent component* of c and interpreted either as a basic state or a composite state. 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 $[c^0, c^1, \dots c^k]$, where c^i is a constituent component of c^{i-1}, $i = 1...k$, c^k is a basic state.

Base state-transition relation–Let $\delta_p : E \times V \times V$, the *base state-transition relation* associated to every composite component c^p. $(l_1, c_1, c_2) \in \delta_p$ 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 $s([c^0, c^1, \dots c^k])$ 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 $s([c^0, c^1, \dots c^k])$, in determining a next state s'.
  s' is given by
  - $s'([c^0, c^{y_1} \dots, c^{y_j}, c^{m_1} \dots, c^{m_x}])$ if c^{y_j} is an HFSM and $s''([c^{y_j}, c^{m_1} \dots, c^{m_x}])$ is the initial state of c^{y_j}, $j > 0$;
  - $s'([c^0, c^{y_1} \dots, c^{y_j}])$ 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 $(l_{iy_i}, c^i, c^{y_i}) \in \delta_{i-1}$, $i \neq 0$ and $i \in 1..j-1$.

  This means that given a current state $s([c^0, c^1, \dots c^k])$, 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 defined. In other words, upper components in the constituent hierarchy have priority when a transition needs to be triggered.

An HFSM is deterministic if and only if given any base state-transition relation δ_p it exists $\delta'_p : \Sigma \times V \rightarrow E \times V$, a partial function such that $(e, s_1, l, s_2) \in \delta'_p$ if and only if $(l, s_1, s_2) \in \delta_p$ and $(e, l) \in EVENT$.

Example Consider the HFSM c^0 whose structure is depicted on Fig. 4. c^0 is a composite component with constituents: c^1, c^2 and c^3. c^1 is a composite state whose structure is depicted on Fig. 4 by the rectangle with dotted

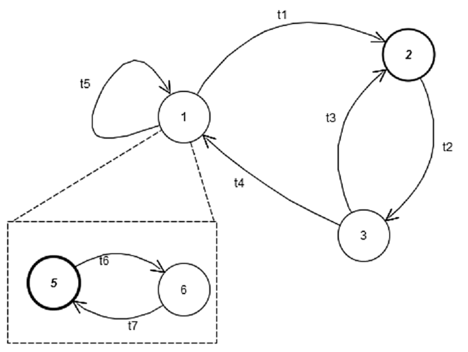


Fig. 4 The structural part of an HFSM c^0

border. c^1 is equally an HFSM. c^2 and c^3 are basic states. For convenience, the HFSM c^0 will be deterministic. Let *EVENT*, given by couples (e_i, t_i), $i = 1..5$, $e_i \in \Sigma$. Σ is the set of events that may cause transitions in c^0. The base initial states of c^0 and c^1 are c^2 and c^5, respectively. The initial states of c^0 and c^1 are $[c^0, c^2]$ and $[c^1, c^5]$ respectively. The (base) state-transition relation of c^1 is given by $(s, t, s') \in \{([c^1, c^5], t_6, [c^1, c^6]), ([c^1, c^6], t_7, [c^1, c^5])\}$.

The state-transition relation of c^0 is given by $(s, t, s') \in \{([c^0, c^2], t_2, [c^0, c^3]), ([c^0, c^3], t_3, [c^0, c^2]), ([c^0, c^3], t_4, [c^0, c^1, c^5]), ([c^0, c^1, c^5], t_5, [c^0, c^1, c^5]), ([c^0, c^1, c^5], t_6, [c^0, c^1, c^6]), ([c^0, c^1, c^5], t_1, [c^0, c^2]), ([c^0, c^1, c^6], t_7, [c^0, c^1, c^5]), ([c^0, c^1, c^6], t_5, [c^0, c^1, c^5]), ([c^0, c^1, c^6], t_1, [c^0, c^2])\}$.

The values of elements of E are fully determined after the events from processes have been mapped or linked to transitions of the state models via *EVENT* (see the following Sect. 5.2). In virtue of this, linking an HFSM may be considered as a named/labelled transition system (Keller 1976). That is, the transitions of an HFSM are labelled or named with actions or events belonging to the set of the events within processes.

HFSM have similarities with (hierarchical) finite-state machines such as Harel statecharts (Harel 1987). In Harel (1987) we read the following:

"statecharts=state-diagrams+depth+orthogonality+broadcast-communication."

Just like with statecharts, we are dealing with the depth of states via the composite structure of states. However, in an HFSM, unlike in statecharts, orthogonality and broadcast-communication are not considered. In an HFSM, the transitions are allowed only inside a composite component c and between its constituent components (deeper constituent components i.e. constituent components of constituent components etc. are not considered as constituent components of c). Orthogonality may be managed with the parallel composition of HFSM as described in Sect. 6. In contrast to statecharts, which are a visual modelling techniques for which several semantics exist (Eshuis 2009), there is only one semantics for an HFSM, where history is not considered by the state-transition relation. Therefore, an HFSM must be specified accordingly. An HFSM only defines an initial state and a state-transition relation.

Running example Consider the HFSM c^α whose structure (the box SM) is depicted on Fig. 3. c^α is a composite component with constituents: c^1, c^3 and c^2 labelled "Maturity < 30", " $60 <$ Maturity < 100 " and " $30 <$ Maturity < 60 ", respectively.

Let *EVENT*, given by couples (e_i, t_i), $i = 1..5$, $e_i \in \Sigma$. Σ is the set of events that may cause transitions in c^α. The base initial states of c^α is c^1. The initial state of c^α is $[c^\alpha, c^1]$. The state-transition relation of c^α is given by $(s, t, s') \in \{([c^\alpha, c^1], t_1, [c^\alpha, c^1]), ([c^\alpha, c^1], t_2, [c^\alpha, c^2]), ([c^\alpha, c^2], t_3, [c^\alpha, c^2]), ([c^\alpha, c^2], t_4, [c^\alpha, c^3]), ([c^\alpha, c^3], t_5, [c^\alpha, c^3])\}$

5.2 Relations between process and state models

Having described the three kinds of models and their exploration semantics, our aim is now to map the events from processes P onto the transitions of state models of components of structure models. In other words, these mappings (or relations) between models encapsulate the expected effects of process models on state models.

We assume that the events generated by P—or, more precisely, their labels or values (Σ_P)—are available to an outside observer (see Sect. 5.1.2). Let $G_a = \langle V_a, E_a \rangle$, $G_p = \langle V_p, E_p \rangle$ and $G_s = \langle V_s, E_s \rangle$ be structures (defined by (1)) corresponding to the structure, process and state models, respectively.

Let *PHY*, *TRANS* and *EVT* 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. The elements of this set are introduced in the running example (Sect. 4.2.1). Indeed, the information of the box MG in Fig. 3, corresponds to an element of the set. TRIGGER is formally defined by

$$\text{TRIGGER} : PHY \times TRANS \rightarrow 2^{EVT} \qquad (5)$$

where $(a, t, e) \in$ TRIGGER is to be interpreted as follows: 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 a semantics one and directly influences the exploration semantics of models.

Running example See the box MG in Fig. 3 and the explanation in Sect. 4.2.1.

The mappings mean that the autonomy and specificity of the different kinds of model are preserved. Autonomy is preserved because each kind of model may be developed separately, while specificity emanates from the fact that the meaning of each kind model is not altered once the model or a part of the model is involved in an interconnection specified via (5). The interconnection has strictly no influence on the development of these models, but rather, a change in the models will result in the necessary changes in TRIGGER and the re-execution of the other following steps of MODEF.

Every element of the triplet (e, a, t) must actually be defined as a stack (see the definition of a state of an HFSM in Sect. 5.1.3) in order that TRIGGER is well-defined as a function. Composite structures also avoid the need for all the models to be systematically flattened before (and while) an exploration is run.

The problem addressed in this paper is not the same as the one addressed in Ramadge and Wonham (1987). Ramadge and Wonham (1987) 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 (these elements are referred to as *controlled events* or specifications). Ramadge and Wonham were addressing a problem that the literature sometimes refers to as supervisory control, in other words, the synthesis of a model of a supervisor from the model of the object to be controlled or the plant and the requirements. See for example (Baeten et al. 2016) on the integration of supervisory control in MBSE. However, even though the approach we present in this paper is different from supervisory control, there are nevertheless similarities.

In our approach, the process and state models are seen as models of the *master* (linked to a generator), rather than a controller and an object to be managed or mastered separately.

The master is a reactive autonomous process for which a complete specification is required. A partial (i.e., without all the labels of transitions) specification of the object to be managed is also required. To obtain a full specification of the object to be managed, TRIGGER must be defined. TRIGGER specifies the expected effects of the master on the object to be managed (which in turn will place constraints on how the master may evolve).

5.3 Expectation-specification

The expectation-specification is the formal modelling of the expected behaviours of the system. It encapsulates expectations over the MA life cycle.

5.3.1 A/P expectations derived from A/G contracts with a pre-order structure on G

There are two good reasons for looking at the problem through the lens of contracts. First, contracts provide a suitable basis for modelling expectations within a system. Second, contracts are easy to describe as rules, yet they are supported by formal conceptual frameworks such as (Benveniste et al. 2012). We are seeking a means for easily involving stakeholders while at the same time allowing expectations to be formally dealt with in the analysis.

Expectations are grounded on Assume/Guarantee (A/G) contracts. We are following in the footsteps of Benveniste et al. (2012) in regard to a meta-theory of contracts.

A contract *con* for a system or a model is defined by *con*(A, G) where

- A corresponds to the assumptions or the "valid environments" for the system or a part of the system.
- G corresponds to 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* $((x, y \in \mathbb{R}$ 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 (Benveniste et al. 2012), *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 (Benveniste et al. 2012, 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). Rather, it provides information on how the system 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 binary relation that is reflexive and transitive) structure. Since A/G contracts (Benveniste et al. 2012) are not defined with a particular structure for G, we are no longer formally dealing with an A/G contract. For this reason, the term preferences (P) is more suitable here.

Formally, an A/P expectation consists of a couple

$$exp(A, P) \tag{6}$$

where both A and P are defined from a domain D. A is theoretically a formula of propositional logic (or zeroth-order logic) where atomic propositions are elements of D. Whenever A is not given for an expectation, we suppose that it is a tautology or that it corresponds to all possible assumptions with regard to the behaviour of the system. P is a pre-order. 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 underlying state of m.

Running example The box R in Fig 3 is actually an expectation. It specifies the following.

There is no explicit assumption i.e., A is not given. The preferences are as follows. P:= ("System Function", "Maturity < 30") ≼ ("System Function", "30 < Maturity < 60") ≼ ("System Function", "30 < Maturity < 60") Such a pre-order (≼) is derived from the column title *MBSELab State Evaluation* in Fig 3 where "Critical" is less preferred that "Unsatisfactory" itself less preferred than "Operational".

Another 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, occurrences of the situations (or the propositions) m_4 and m_3, are more preferred than m_2. And when $m_5 \wedge m_6$ is true, occurrences of m_7 are 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. For the sake of simplicity, we will assume below that A only consists of atomic propositions.

A basic question that arises is whether operations on contracts also apply to 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). We rather consider properties (compatibility and consistency from contracts) which, defined for expectations, shall determine whether an expectation is well defined and feasible.

Compatibility A basic way of checking the compatibility of an expectation, just like a contract, is to verify that its assumption is valid according to the definitions of the state models that it is related to. Statically, A needs to be well-defined as a proposition. However, checking the consistency of some expectations would mean exploring the state space (of the behaviour of the system) to check whether certain states may all be reached together. If two situations (or propositions) are not defined from the same state model, the validity of an assumption that involves these situations might yield a reachability problem. For example, for the expectation with Id=2, $A = m_5 \wedge m_6$, compatibility means the possibility of the system being 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. an environment that makes A true) for *exp*.

Consistency Thinking in terms of contracts, we might wonder what is meant by "a model that effectively implements *exp*(A,P) (satisfies *P*)" mean, that is to say what satisfying the preferences might entail. Before addressing this question it is important to check that P is well defined.

Statically, we need to 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, it will suffice to check that it is an acyclic graph. Formally, P "is not contradictory" syntactically means that the pre-order P is antisymmetric i.e a partial order.

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

If P is well defined, checking that it is satisfied is clearly not a Boolean question, unless there were a requirement that all preferences are satisfied, which would be an excessively heavy requirement. Regarding a utility function for quantitatively evaluating the preferences, it would be possible to compute how far preferences are satisfied. This concern is addressed in Sect. 6.

Consequently, 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.

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

As a first consequence, feasibility (that is to say, whether system models exist that satisfy expectations) is not easy to determine. Where multiple models are mapped, given the autonomy of different models and the fact that some models may be subject to constraints, realizability (whether there are models and mappings that satisfy expectations) might also be an issue. This in turn raises the question of the completeness of expectations.

It turns out that feasibility is a trade-off problem which could imply either modifying the expectations or changing the system models and their mappings. Furthermore, on the assumption that we have autonomous processes, trade-offs are not always possible. In this situation, feasibility would be mainly defined in relation to process models and their effects on state models. Since satisfiability is sufficient but not necessary to deduce feasibility, all (less or more) acceptable behaviours might help in dealing with such trade-offs once they are allowed.

5.4 Conclusion and discussion

This section looked at the principles and exploration semantics of models, described how models are related via a mapping, and finally we presented a formalism for specifying the expectations on models. We now need to analyse these models against the expectations. More importantly, the results of the analysis have to be utilized in providing stakeholders with relevant data: we address this concern in the next section.

Contracts theories have been used for component-based design, layered design and platform design (Benveniste et al. 2012). 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 paper, we replace guarantees by preferences and we define and consider the notion of Expectation instead of Contract. Additionally, this pre-order structure of preferences is utilized by the analysis procedure, by introducing a utility function that makes the qualitative preferences quantitative (see Section 6.1).

The way that expectations are used in MODEF is different from the traditional use of contracts in system design. In system design, contracts are generally used to specify the interactions between (heterogeneous) components or different viewpoints (Benveniste et al. 2007; Nuzzo et al. 2014; Damm et al. 2011), mainly in software and cyber-physical systems. In MODEF expectations are used to specify the expected behaviour of the system in relation to the system models. This use of contracts is close to how they were first used in programming, where *preconditions* (in our case assumptions) and *postconditions* (in our case preferences) are defined for *programs* (in our case system models)—see e.g., (Hoare 1969; Meyer 1992).

In the area of model checking (Clarke et al. 2000; Baier et al. 2008), property specifications are generally expressed as temporal properties. One advantage of temporal properties is that they are both formally and informally understandable in the sense that natural language (informal properties) may be translated into temporal logic (Tripakis 2016). A/G contracts (and consequently expectations) are also formally and intuitively understandable. Their consistency and compatibility may be verified.

6 What is achievable and what may 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 Fig. 1). We present the analysis algorithms (Sect. 6.1) for analyzing system models in regard to expectations. We then discuss possible uses of outputs from the analysis (Sect. 6.2) and present an application on our running example (Sect. 6.3).

6.1 Analysis of system models against expectations

The fifth step of the MODEF is intended to provide the means to effectively improve the way the system is operated. Given the models of the system and expectations and any changes that occur therein, the stakeholders of the system should be able to permanently identify better ways of operating the system. What do we mean by better ways of operating the system? To answer this question, we will first formulate the problem clearly, and we will then look at the procedure for solving it and the expected benefits.

6.1.1 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) encapsulates the actions (or the effects) of PM on SM of AM. The different types of models AM, SM, PM, R and MG are characterised and defined in Sects. 5.1; 5.3 and 5.2 respectively. Additionally PM might be subject to some constraints (C) for example the cost of the tasks within processes.

The global state 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.

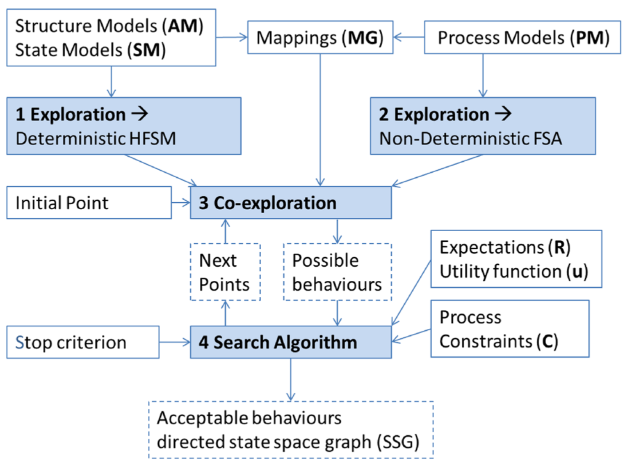


Fig. 5 General synthesis procedure structure

We recall from Sect. 2 that the general problem is given by Pb(AM, SM, PM, MG, R, C, InitialPoint, StopCriterion) relating to our question Q1: How to generate the possible future points starting at InitialPoint up to StopCriterion? Which points are, with respect to R and C, more acceptable and less acceptable ones?

6.1.2 General principles for a solution

The answer to the question Q1 basically requires synthesising the possible behaviours (starting from InitialPoint) of the closed system. Furthermore, by speaking of the acceptability of the behaviours, we basically need to compute the “distance” between behaviours in the state space of the system. This raises a second question Q2: How may this distance be defined and computed?

We wish to synthesize the future possible points from InitialPoint up to StopCriterion. The general synthesis procedure for solving question Q1 is summarized in Fig. 5. Before we present this procedure, let us discuss question Q2, since the answer to Q2 is necessary for addressing Q1.

The answer to question Q2 will consist of a means of evaluating behaviours in order to identify their degree of acceptability. Behaviours that more match expectations (R) and respect constraints (C) are to be more acceptable than those that do less or not. If expectations (R) are to be utilized in the exploration procedure, qualitative preferences need to be translated into quantitative values. Preferences are relative to the pre-ordering of atomic propositions.

Let us suppose that we have an appreciation (or utility) function u for mapping qualitative preferences onto 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 pre-order structure of preferences. We will, therefore, use an appreciation function of this kind in the synthesis procedure as the basis for computing an aggregated appreciation of a given point in the state space. 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. This target point must be uniquely determined by the source point and the outgoing arrow corresponding to the transition.

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

The synthesis procedure basically involves parallel explorations of the PM and the SM of AM. These explorations impact each other via the mapping MG (see Sect. 5.2). Although there is an interplay between PM and SM, as indicated in Sect. 5.2, a PM process will generally function as a “master”, and an HFSM relating to the SM of AM as an “object to be managed” (see Sect. 5.2).

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

The general synthesis procedure structure is depicted on Fig. 5. The structural explanation of Fig. 5 is a follows. The (4) rectangles with a blue background are (sub-)procedures. The (10) rectangles with a white background are inputs and outputs of the 4 (sub-)procedures. The (3) rectangles with a dashed border are outputs generated by the (sub-)procedures. Two of those outputs, namely the rectangles entitled ’Next Points’ and ’Possibles behaviours’, respectively, are also inputs of 2 (sub-)procedures—the two rectangles entitled: **3 Co-exploration** and **4 Search Algorithm** respectively. The source of an arrow in Fig. 5 is required for (utilizing or obtaining) the target of the same arrow.

The meanings of rectangles from the top to the bottom of the Fig. 5 are as follows.

- The first 3 rectangles are the basic inputs: structure models (AM) and state models (SM), mappings of models (MG) and process models (PM).
- The models need to be explored in order to compute the behaviours of the system they represent, whence the upper two blue rectangles (numbered 1 and 2). The first at left represents the primitives for exploring the SM (of AM) under a deterministic hierarchical finite state model—HFSM (see Section 5.1.3). The second at right represents

the primitives for exploring the generator (a non-deterministic state automaton) associated to PM (see Section 5.1.2).

- With the primitives mentioned above, and given SM and PM and the mappings (MG), the possible behaviours of the system may be generated from an initial state (either at the first iteration using InitialPoint or at other iterations via NextPoints), whence the blue rectangle (entitled 3 Co-exploration) and its dependencies.
- Among the reachable or possible behaviours, those that do not make StopCriterion true are acceptable. The 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 will be used in the Search algorithm to differentiate and compare the research directions in state space of the behaviours. StopCriterion allows some possible behaviours to be to pruned.

  Search Algorithm produces an output (NextPoints) corresponding to the next possible starting points of the procedure Co-exploration. Co-exploration may therefore be resumed 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 Sect. 5 (their principles and exploration semantics).

6.1.3 Main sub-procedures: coexploration and a search algorithm

We will suppose that the primitives for exploring SM (of AM) and PM are available through the interfaces *smInt* and *pmInt* respectively. Below, *smInt* and *pmInt* refer to the boxes labelled "1 Exploration → Deterministic HFSM" and "2 Exploration → Non-Deterministic FSA" respectively in Fig. 5.

As described above (Sect. 6.1.2), the two main sub-procedures interact in a closed loop in the general synthesis procedure. We have named the Search Algorithm *MinBest to MinWorst* (MBMW). MBMW selects the behaviours, whatever their appreciation, that are closest to InitialPoint (in regard to R and C, using an answer to question Q2). We now present the algorithms CoExploration (Algorithm 1) and MBMW (Algorithm 2) corresponding to the Co-exploration and Search Algorithm sub-procedures respectively in Fig. 5.

Algorithm 1 coExploration

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

CoExploration We have already introduced the inputs *MG*, *smInt*, and *pmInt* used by the CoExploration algorithm. *CurPoint* is either the InitialPoint or one of the points in NextPoints, both of which feature in Fig. 5.

We recall that *smInt* and *pmInt* are the interfaces for exploring, respectively, a set of concurrent independent process models and a set of concurrent independent state models. The exploration semantics of these models (or components) are presented in Section 5; but only for one component and not for a set of components. For each component the exploration yields an automaton that is equivalent to a directed graph. In the case of the process model the automaton is equivalent to the corresponding generator. In the case of the state model the automaton is equivalent to the corresponding flattened part of the HFSM generated by the exploration. We can remark 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 are necessarily enumerated.

The directed graph (that we will from now on call 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 (2), the result $G(Q, \Sigma, \delta, q_0, Q_m)$ of the asynchronous composition of G_1 and G_2 is such that $Q = Q_1 \times Q_2$, $\Sigma = (\Sigma_1 \times \Sigma_2) + \Sigma_1 + \Sigma_2, Q_m = Q_{m_1} \times Q_{m_2}, q_0 = (q_{0_1}, q_{0_2})$,

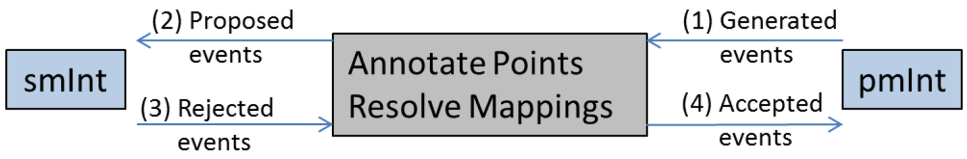


Fig. 6 The principles of the co-exploration

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

The composition may easily be extended with $n > 2$. We now present the principles of the co-exploration semantics, before detailing the lines of the CoExploration algorithm.

Given an initial point pt(*cst*, *cev*), *cev* is generated by pmInt, and corresponds to the current state of SM of AM available from smInt. pt is passed through the co-exploration module (the grey box in Fig. 6). This module works as follows.

The generated events (arrow (1) in Fig. 6) are given the status *proposed*. The status *proposed* means that the event *might occur* in the process.

Events that have the status *proposed* and that are involved in MG are presented as inputs to *smInt* (arrow (2) in Fig. 6) to trigger any fireable transitions.

After the (possible) firing of transitions, the events not used or rejected by smInt are reconsidered (arrow (3) in Fig. 6), while the events that were used are given the status *accepted*. Whenever an event is not involved in MG it gets the status *accepted*.

The status *accepted* means that an event *does occur* in the process.

The events that have the status *accepted* are the only valid events considered (arrow (4) in Fig. 6) in seeking to refine the processes in pmInt.

The events that are not used by smInt (arrows (2) and (3) in Fig. 6) and pmInt (arrows (1) and (4)) retain their status.

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.

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).

A Search algorithm: MBMW We assume the general synthesis procedure may be replaced by MBMW (Algorithm 2) by including among MBMW's parameters the CoExploration algorithm and its input parameters. The additional input parameters of MBMW are:

- *nodeScore* is a function that has as its input a point in the state space and R (the expectations), and then computes an aggregated score corresponding to that point. It will be recalled from the answer to Q2 in Sect. 6.1.2 that this score may apply to all incoming arrows at that point, in view of the deterministic nature of HFSM.
- *edgeCost* is a function that has as its 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.

The cost or costs of an edge in the state space are therefore obtained by applying *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 given by *nodeScore*, and the value or values of its other component(s) are given by *edgeCost*. From an algorithmic point of view, the value obtained from *nodeScore* may 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 that it may be used to compute a path cost in the state space that may potentially provide the distance that Q2 seeks to identify.

The outputs of MBMW are:

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

MBMW implements a Uniform-Cost Search (UCS) algorithm (Russell and Norvig 1995). It is an algorithm similar to Dijkstra's shortest path algorithm (Dijkstra 1959), and is a special case of the A* algorithm introduced in Hart et al. (1968), itself a special case of a branch-and-bound algorithm (Nau et al. 1984). The effect of the UCS algorithm here is to always select (based on the value of scores and possibly on the costs of processes) the best (minimal) point(s) at the next iteration in exploring SSG. We discuss UCS in Section 6.1.3.

We now are ready to discuss the lines of the MBMW (Algorithm 2) that are additional with respect 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 $\vec{0}$ 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 not expanded further in the exploration of SSG. At this line, the consistency and compatibility of R are also computed (see Sect. 5.3).

Lines 12: Calling *CoExploration* is the means by which *CurPoint*'s neighbors i.e. its successors in SSG, are generated.

Line 24: A test to check if *Cur* (see Line 23 of Algorithm 2) dominates *Alt* (see Line 17 of Algorithm 2). Such a dominance relation must be defined.

Line 30: SSG is updated every time a new point or a new edge is discovered. An update involves adding a node in SSG, creating a edge or, both.

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: $ScoreAndCost \leftarrow (InitialPoint \mapsto (Score, \vec{0}))$
 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: return $SSG$
```

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.) should ensure that MBMW will eventually stop.

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 sequencing (or the paths of execution) of events from PM with associated states of SM of AM.

The procedure MBMW applies the UCS algorithm on-the-fly while SS is discovered. It allows to select a subgraph (*SSG*) of SS based on R and C and StopCriterion.

The UCS algorihtm is an optimal, uninformed (or blind) search algorithm (Russell and Norvig 1995). This concludes the proof.

The complexity (time and space) of MBMW is the complexity of UCS with the input parameter SS. This complexity is given by $O(bf^{1+\lfloor C^*/\epsilon \rfloor})$ (Russell and Norvig 1995). The branching factor (*bf*) is the (average) out-degree of each node in *SSG*. ϵ is the minimum value of an edge cost to avoid the algorithm deadlocks in an infinite loop which implies lack of completeness. The map *ScoreAndCost* (built on the input functions *edgeCost* and *nodeScore* of MBMW) has as codomain positive real vectors that should guarantee that minimum. C^* is the maximum total path-cost reached for a path in SSG.

This complexity might be seen to corroborate some theoretical results from the literature. It has previously been shown that the design process problem (the design process being a non-deterministic, evolutionary process that seeks to go from an initial set of presumed specifications and an empty product to the realized product in the ideal case,[1] with all specifications satisfied) is NP[2]-hard (Braha and Maimon 2013, Chapter 6). This informally means the problem is at least as hard as NP-Complete problems. Subsequently, the system design problem (system design being a process that translates customers' needs into a buildable system design) was shown to be a NP-complete problem (Chapman et al. 2001). This informally implies that currently there is no known fast procedure for providing a solution to the system design problem. See e.g., (Michael and David 1979) for details on NP-completeness.

As argued earlier this paper, the SoSoM comprises autonomous and possibly partly independent stakeholders. This makes changes in behaviours within E unpredictable although they are continuously specified via PM. In turn, the system's behaviour is never definitely fixed.

If we are dealing with a critical system that typically evolves over periods of between several days and several months (see Sections 1) and includes creative, iterative behaviours that it is difficult to automate, we will probably find it better to seek to guarantee optimality only over short periods. This will involve repeatedly executing MBMW throughout the life cycle of the system. Although the SoM and SoSoM are complex, the state space of their behaviours should typically be smaller than that of other kinds of systems, e.g., cyber-physical systems.

[1] See (Braha and Maimon 2013, Chapter 6) for the other terminal states.

[2] Non-deterministic Polynomial-time.

6.2 Using MBMW

In this section, we discuss some input parameters and how SSG may be used to obtain useful results from MBMW in practice. These parameters are: *u*, *edgeCost*, *nodeScore*, *StopCriterion*. Some of them will be used in the running example in Sect. 6.3.

6.2.1 Setting up input parameters

Input parameters are as follows.

- *StopCriterion*: This corresponds to a maximum path cost and/or a maximum depth in SSG and/or a maximum number of cycles authorized during the co-exploration.
- *edgeCost*: This gives the total process cost related to the resources consumed by an arrow in SSG. In this paper this cost is assumed to be a single business value. But it is not considered in the operation of MBMW because we did not specifically define the resources allocated to the tasks inside processes. We have assumed that this business value is the same for all tasks, thus making this cost irrelevant in the operation of MBMW. For such a cost to be relevant, aggregation and addition operators related to it would need to be defined. Nonetheless, we discuss the expected impacts of *edgeCost* on MBMW in Section 7.
- *u*: This appreciation function maps preferences P for expectations onto a numerical domain. In this paper, *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 in this set indicate, 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 maximum of 5. Such a reduction also translates via *u*, the preorder structure of P into a structure of total order.
- *nodeScore*: It defines the magnitude of each rank and the aggregation of 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 linked 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.

$$nodeScore : POINTS \times 2^R \rightarrow \mathbb{N}^5 \xrightarrow{aggr} \mathbb{R} \tag{7}$$

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. s is an aggregated score and the NS a per rank score, where N_j corresponds to the rank j. We recall from Sect. 6.1.3 that, s must be such that $s \geq \epsilon > 0$ for all function *aggr*.

The importance of *aggr* and *s* is as follows. We want every path in SSG to be given a cumulative score enabling us to obtain a priority for each node so as to define a total ordering ($\leq$) among the candidate nodes to explore. The lower the cumulative score of the nodes along a path in SSG, the higher the priority of the nodes on that path (see line 7 in the Algorithm 2). Any nodes where the cumulative score exceeds a given maximum score may be pruned.

Arguably, the question of how *aggr* is defined is important, and definitions need to be tailored for particular applications. For example, there are a number of ways of scalarizing the vector *NS* (max, min, min-max, a mean, etc.). Another possibility would be to avoid scalarization and to deal directly with vectors. Different strategies may also be combined. The way that N_j are obtained (by counting) might also be questioned, since different components of a system may not all be of equal importance. However, this is not relevant with respect to the input models considered in MODEF, where no importance factor is defined for the components.

6.2.2 Utilization of SSG

SSG is a rooted directed graph that is utilized by extracting data from it that are relevant to the behaviour of the closed system.

We first define what is meant by the *color* of a node in this graph. 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* whose value is as follows: if $N_0 \neq 0$ $a := 0$ else if ... else if $N_5 \neq 0$ $a := 5$.

In addition, each node *pt* in SSG is linked to a second color *b*, whose value is the highest color among the successors of *pt* in SSG. Each node in SSG thus has a corresponding couple (*a*, *b*), that may be used as follows to refine SSG. Whenever, at a given node *pt*, *b* does not exceed a given value, say *x*, we may not consider it useful to investigate the sub-graph that has *pt* as its root, and so SSG is pruned to yield a usable graph.

Example. Take the graph in Fig. 7. The nodes in this graph are labelled with *a|b*. The rooted sub-graphs represented by the four triangles in Fig. 7 are such that, at their root node, we have $a \leq b$.

From InitialPoint up to the leaf nodes it is straightforward to determine whether or not a node with a given color will be investigated.

For instance, in Fig. 7 there is a node with the color *b* equal to 1. Normally, any node requires investigation if the modelling tasks take a path that leads to it. Since an execution simply follows a path in SSG, the nodes provide an understanding of what will happen if some tasks are executed. As a result, knowing which nodes may be considered problematic and the path(s) leading to them is useful, enabling us either to redesign the processes or to review the expectations on the system. It will be remarked that MBMW may be resumed from a given point in SSG.

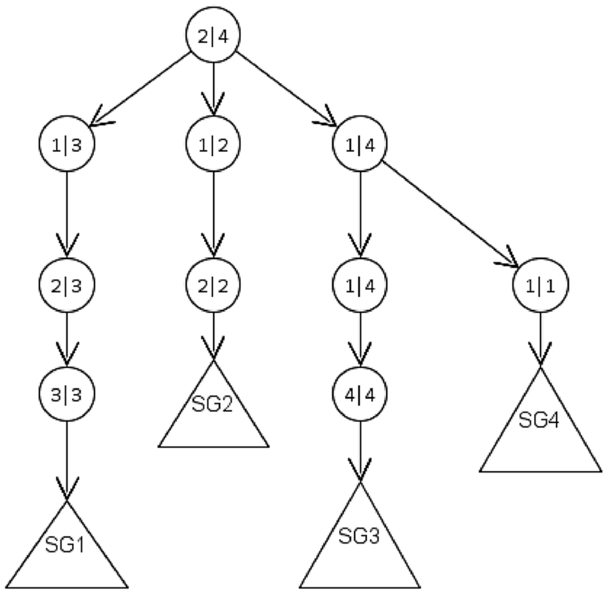


Fig. 7 Tailoring of SSG

There might be other data analogous to the information conveyed by (*a*, *b*) for each node in SSG, computable using graph algorithms, and providing further relevant information from SSG. An example might be the smallest value among the colors of nodes in a path.

During the operation of the system modelled with the input models, stakeholders can be provided with two kinds of information: either (R1) *everything is ok*, or (R2) *something might not be ok*. For instance, using the data *a* of the leaf nodes in SSG or the refined SSG, it might be decided to signal (R1) if every *a* is greater than a given value *x*, otherwise to signal (R2). Returning to Fig 7, let $x = 2$. In this case the information is R2 because of the two leaf nodes ($a = 2$, $a = 1$).

It follows from the foregoing that MODEF is insightful in providing an understanding of the impacts of processes before they are executed. However, 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. Where there is definitive behaviour, state space explosion might become a serious issue.

In addition to the utilization of SSG described above, the flattened automatons generated by the exploration of the process and state models are available separately. These might be useful for diagnostic purposes.

Fig. 8 The colors associated to nodes in SSG

6.3 Running example

We shall summarize the general problem Pb(AM, SM, PM, MG, R, C, InitialPoint, StopCriterion) (see Sect. 6.1.1) related to SoM0 and show how the analysis may be utilized.

For this case study, 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 Fig. 8) 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.

6.3.1 Problem, analysis and information

We continue with the SoM (SoM0) presented in Sect. 4.2.1 and for which the corresponding models are depicted in Fig. 3. The SoM0 focuses on the modelling of the functional coverage of a SOI whose objective is to ensure that the SOI meets functional requirements.

The box (R) in Fig. 3 is the representation of the specification of expectations (R) as in a modelling tool. This expectation means: regardless of the assumption (A) (therefore considered as a tautology, see Sect. 5.3), a *System Function* in state *Maturity* < 30 is less preferred than in state 30 < *Maturity* < 60, which in turn is less preferred than in state 60 < *Maturity* < 100. The rank of preferences is materialized with colors and some qualitative words (Operational, Unsatisfactory, etc.) in the tool. For the readability of the figure, the mappings (MG) are indicated with $t_k, k = 1..5$ $((t_k, System\ Function) \mapsto \{\{t_k\}\})$, and that corresponding to t_3 is illustrated with the dashed blue and green arrows.

Therefore, apart from the data InitialPoint and StopCriterion, all the other data of the problem Pb(AM, SM, PM, MG, R, C, InitialPoint, StopCriterion) are depicted in Fig. 3.

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

Although the PM on the right of Fig. 3 is not complex, its state space is possibly infinite since there are cycles in the sequencing of its sub-processes or tasks. Therefore we set StopCriterion as: max-process-depth$:=$ 7. It will be remarked from the PM at right of Fig. 3 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 not set a value for max-score. For the execution we also set $b > 4$.

SSG is depicted in Fig. 9. In this figure the color of each node is given by the value of b, which is given by the execution algorithm. The value of a is given by R (in Fig. 3). For instance, the state "Maturity < 30" is evaluated as "Critical", and therefore for the InitialPoint we have $a = 2$.

Again the data corresponding to the nodes are not displayed, for the readability of the graph. The states and events associated to the leftmost (the root), the red, the orange, and the rightmost nodes in Fig. 9 are:

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

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

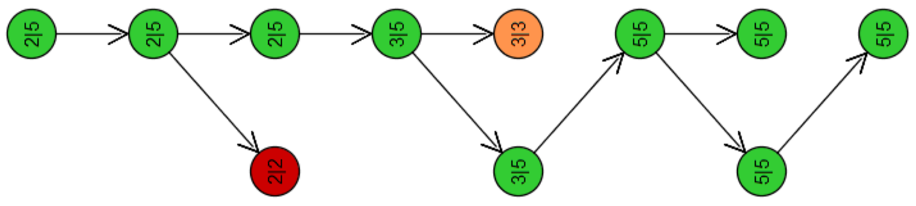
(Event = [Validate DetailledFunctions–>Model detailed functions], State = [30 < Maturity < 60]),

(Event = [Refine functions–>Validate RefinedFunctions], State = [60 < Maturity < 100]) respectively.

The red and orange nodes come from the fact that MBMW computes the shortest path to reach each of them. In Fig. 9 we can see that after the end of the task "Validate HighLevelFunctions" two transitions are possible with respect to MG: t_1 or t_2. If t_1 (resp. t_2) occurs, the system will stay at the state "Maturity < 30" (resp. "30 < Maturity < 60 "). This is why after the node successor of the root node, two paths are possible. The path leading to the red node (resp. the other green node) is related to the case where t_1 (resp. t_2) occurs. Similar logic applies to the orange node.

The first piece of information that can be signalled to the stakeholders is (R2) i.e. something might not be ok during the operation of the SoM0. There are red and orange nodes that correspond to the cases where, after the tasks "Validate HigLevelFunctions" and "ValidateDetailedFunctions", the tasks that directly precede them need to be executed again. In practice, such scenarios could typically require the allocation of additional resources if those allocated previously

Fig. 9 The SSG graph for the SoM0 with max-process-depth=7 and $b > 4$



have been used up. Thus, by having the process models of modelling activity that is to be carried out and its impacts (expectations and mappings) on the states of produced models (M), MODEF makes it possible to anticipate problems that might arise.

6.4 Conclusion and discussion

6.4.1 On the analysis

The problem formulated in Sect. 6.1.1 and the subsequent proposed solution in Sect. 6.1.2 have similarities with model checking (Clarke et al. 2000; Baier et al. 2008) and systems synthesis (Ramadge and Wonham 1987; Pnueli and Rosner 1989).

Model checking—State space analysis is generally carried out to verify finite-state concurrent systems. The techniques generally used for verification are 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 comprise three main steps (Baier et al. 2008). (1) Model the system (S) with a description language and express the properties or specifications (R) of the system using 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 (Baier et al. 2008). However, model checking is well-suited when analytical methods are difficult or impossible to apply in practice (Tripakis 2016). Model checking has several advantages, namely, it is fast, executable with partial specifications, it provides counter-examples and does not require proof. Nonetheless, the more data variables there are, the more challenging model checking becomes.

Systems synthesis—The standard synthesis ("Be Correct" (Bloem et al. 2014)) 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 may be reduced to verification or model checking (Tripakis 1998, Chapter 9). When the specification of the system is fixed independently of its environment the synthesis may be reduced to verification of the closed system. When, on the other hand, the specification of the system is fully determined only up to the definition of its environment, the synthesis may be reduced to model checking. However, in cases where the environment is not fully determined and the specification of the system is incomplete, it is rather a question of seeking strategies to ensure that the closed system satisfies a given property (Tripakis 1998, Chapter 9).

With respect to the assumptions we made for S and E, which are close to the assumptions made in Bloem et al. (2014), especially when the system features human operators, the operation of the system is not likely always to be optimal. We believe that "Be correct" (i.e. "everything must be ok") in this latter situation is too strong a requirement.

Since some pioneering works (Ramadge and Wonham 1987; Pnueli and Rosner 1989) on system synthesis, there has been active research on the synthesis of reactive systems (systems that react as the result of the actions of their environment on them). Reactive Synthesis Competition (Jacobs and Bloem 2016) is an example of work to emerge from this trend.

Two invariants of synthesis algorithms are, first, the use of temporal logics as the property specification language and, second, the search for a wining strategy or a counter strategy. This is also the case in model checking (Clarke et al. 2000; Baier et al. 2008). Synthesis algorithms are more usually applied to software and cyber-physical systems, while we are mainly concerned with systems where the role of human operators is important. In this paper, expectations are specified via Assume/Preference formalism.

Recently, quantitative objectives, i.e. the adoption of a non-binary satisfaction of a specification were introduced in Bloem et al. (2009). There, the authors' approach involved synthesizing a system with respect to a boolean specification complemented with quantitative aspects given by weighted automata. As argued in Bloem et al. (2009), the satisfaction of a specification could be evaluated on a scale of varying degree rather than via a binary indicator.

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. MBMW computes points (from bad to good) that are reachable at a minimal distance from the initial point until the stop criterion becomes true. In the present paper, unlike (Bloem et al. 2009) and almost all the approaches that emerged subsequently on the basis of their work, a specification is defined using A/P expectations instead of a temporal property or automata. The quantitative nature of objectives (preferences here) is derived from qualitative objectives and the appreciation function, and possibly also from constraints (cost, time, etc.) on processes. 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 (Chatterjee et al. 2009), weighted automata (Droste and Gastin 2007), or simulation distances (Fahrenberg and Legay 2014; Romero-Hernández and de Frutos Escrig 2014). Neither do we consider games (in the sense of Game theory) where the environment and the system are adversarial, as in most popular synthesis approaches (Bloem et al. 2014). We believe that our reasoning is justified by the fact that we are not dealing with "a controller" and "an object to be controlled" in the sense of controller synthesis, but rather with "a master" and "an object to be managed or mastered"; see Sect. 5.2 for the explanation. Finally, the general synthesis procedure synthesises the behaviours of the closed system. We could also argue that the closed system is verified with respect to expectations and possibly constraints on processes.

6.4.2 Utilizing the results of analysis

An essential part of MODEF is the way feedback is provided to stakeholders.

Assuming that stakeholders are already capable of building the models necessary to run the analysis and to generate results, the feedback step formalizes what it is the models relate to and helps stakeholders to understand, preserve, and share and reuse knowledge about the modelled entities. From an operational point of view we show in Sects. 6.2.2 and 6.3.1 how MODEF may be insightful in providing synthetic and intuitive graph-based data relating to the behaviours of the system. These data may help stakeholders to optimize how the system functions, know the "path" to follow, take preventive and corrective actions, or simply model new actions. These new actions may in turn be processed via MODEF and generate further feedback.

If models are accompanied by an appropriate analysis and appropriate tools for utilizing them, they can greatly support the formal end-to-end operation of the system they represent. They can be considered as inputs or references to a further optimization.

7 Concluding remarks

In this work, we addressed the modelling activity carried out in a context of model-based systems engineering framework and environment characterized in Section 1. The challenges identified and enumerated were:

(1) How to better understand and use models in this context: what models are present in a particular location and what do they represent?
(2) How to analyse and identify the impact of changes in models: what is the current state of models and in which states are they likely to end up?
(3) How best to manage the way that models evolve: what is required for models to reach desired states?

To tackle those challenges, we have proposed MODEF and its underpinning framework. We have then considered with rationale some hypotheses and assumptions and the derived principles and models on which the proposed framework and MODEF stand. Thus, the modelling activity is first characterized as a system (and a federation of systems) in its own right. At the level of the architecture of this system, we described a class of discrete-event processes models for modelling the tasks carried out in a modelling activity; a class of conceptual models for modelling the conceptual content of models (M) generated and operated by modelling activity; and a class of finite state models for modelling the expected life cycles of the same models (M). The effects of the tasks on life cycles are also modelled via some triggers (a function). We introduced the assumption/preference expectations formalism (a pre-order structure based on propositions in propositional logic) to formalise the expectations related to the life cycles. In order to check how far expectations (and possibly process constraints) are achievable and to synthesize the expected behaviours of the system, we defined an analysis procedure based on a co-exploration of process and state models and constrained by triggers. Results generated by the analysis procedure provide information about what might occur in the modelling tasks, and about their potential impact on the whole state of M. We showed how these results can be utilized in a way that provides insightful data on how the system is operated and how it can behave. Based on this information, it is possible to take informed actions impacting the way that modelling activity proceeds.

AM and SM i.e. models of models (M) produced by modelling activity (MA) answer the challenge (1). The analysis algorithm answers the challenge (2). The processing and use of results (data) produced by the analysis algorithm partly answer the challenge (3). The other part to answer (3) being a decision-making issue, although analysis and its use underpin decisions, they cannot necessarily trigger or prevent decisions!

Some limitations and perspectives are as follows.

Constraints and analysis In our use of the analysis algorithm, described in Sect. 6, we currently consider a single cost (related to node score) and a total ordering of the cumulated costs. Another use 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. Besides, as argued in the paper, the UCS algorithm may be replaced by another one—e.g. an algorithm from AI or OR.

A/P Expectations Independently of MODEF and just like the operations (see (Benveniste et al. 2012, Section VII)) on A/G contracts, it could be interesting to study operations that could be applied on A/P expectations. Since preferences (P) are based on relations (in a mathematical sense), operations on relations could be helpful in this regard.

Modelling choices We made some modelling choices that allow us to understand certain relevant aspects of Modelling Activity. One might argue that those modelling choices do not represent some aspects of Modelling Activity, what would probably be true, but we argued in Section 1 that "Models generally provide a partial, sometimes incomplete view of the actual modelled thing ...". Nevertheless, these modelling choices have allowed us to further build ways to address the 3 identified challenges. Just like the UCS algorithm which may be replaced, equivalent modelling choices could be considered. A goal is to get new insights—not obtainable by a human within reasonable amount of time—from models by running the analysis routine.

A class of systems We said in Section 2 that MA is evolutive and even creative, and furthermore, MA may not be specified once for all. These features (evolutive, creative and iterative) are not specific to MA. They may characterize a class of systems. Systems that *do not* belong to such a class[3] include for instance working cyber-physical systems such as aircraft and autonomous vehicles, because neither does the pilot or the human operator exhibit creative and evolutive behaviour that alters the system design, nor is the system continuously reworked, re-specified. Even recurring maintenance operations are not intended to change the design of a car or an aircraft. Note: a system *aircraft* is different from a system *engineering-of-an-aircraft*.

Thus, the principles of MODEF and its underpinning framework could be applicable for other systems included in such a class.

Models reuse for analysis Although the practical reuse of models (AM, SM, and PM) for analysis concerns is out of the scope of this paper, a Federated Architecture (FA) and means (data structures and base algorithms) for its implementation are specified in Kamdem Simo (2017). Those means are useful for projecting a class of models coming from a modelling tool to data structures independent of it. This may foster the implementation of methods.

Experimentation Several experiments of MODEF are necessary to make new observations in situ and examine its overall validity. In particular, such experiments will provide data on MODEF's components efficiency and accuracy, and thus feedbacks for improvements and for considering new hypotheses.

[3] except if they may be modelled under the classes of finite state machine and discrete event processes. See *Hypotheses* in Section 2.

Acknowledgements This work was carried out and funded in the framework of the Labex MS2T. It is supported by the French Government, through the program "Investments for the future" managed by the National Agency for Research (Reference ANR-11-IDEX-0004-02). This work is co-funded by the Airbus Defence and Space company. The first author would like to thank all those who contributed to the quality of this paper.

References

Abeille J (2011) Vers un couplage des processus de conception de systèmes et de planification de projets: formalisation de connaissances méthodologiques et de connaissances métier. PhD thesis

Ackoff RL (1971) Towards a system of systems concepts. Manag Sci 17(11):661–671

American Association for the Advancement of Science (2016) The Rise of Systems Engineering in China (Science/AAAS, Washington, DC, 2016). Science/AAAS Custom Publishing Office, https://www.sciencemag.org/sites/default/files/custom-publishing/documents/ALSSEsupplement_Finalonline.pdf. Accessed June 2018

Baeten JC, van de Mortel-Fronczak JM, Rooda JE (2016) Integration of supervisory control synthesis in model-based systems engineering. Complex systems. Springer, Berlin, pp 39–58

Baier C, Katoen JP, Larsen KG (2008) Principles of model checking. MIT Press, London

Bar-Yam Y (2003) When systems engineering fails-toward complex systems engineering. In: Systems, Man and Cybernetics, 2003. IEEE International Conference on, IEEE, vol 2, pp 2021–2028

Benveniste A, Caillaud B, Ferrari A, Mangeruca L, Passerone R, Sofronis C (2007) Multiple viewpoint contract-based specification and design. International symposium on formal methods for components and objects. Springer, Berlin, pp 200–225

Benveniste A, Caillaud B, Nickovic D, Passerone R, Raclet JB, Reinkemeier P, Sangiovanni-Vincentelli A, Damm W, Henzinger T, Larsen KG (2012) Contracts for system design. Research Report, INRIA. https://hal.inria.fr/hal-00757488/document. Accessed Apr 2015

Blanchard BS (2004) System engineering management. John Wiley and Sons, New Jersey

Bloem R, Chatterjee K, Henzinger TA, Jobstmann B (2009) Better quality in synthesis through quantitative objectives. International Conference on Computer Aided Verification. Springer, Berlin, pp 140–156

Bloem R, Ehlers R, Jacobs S, Könighofer R (2014) How to handle assumptions in synthesis. https://arxiv.org/pdf/1407.5395. Accessed July 2015

Bonjour E (2008) Contributions à l'instrumentation du métier d'architecte système: de l'architecture modulaire du produit à l'organisation du système de conception. https://tel.archives-ouvertes.fr/tel-00348034/document. Accessed Dec 2014

Braha D, Maimon O (2013) A mathematical theory of design: foundations, algorithms and applications, vol 17. Springer Science and Business Media, Berlin

Browning TR, Eppinger SD (2002) Modeling impacts of process architecture on cost and schedule risk in product development. IEEE Trans Eng Manag 49(4):428–442

Chapman WL, Rozenblit J, Bahill AT (2001) System design is an np-complete problem. Syst Eng 4(3):222–229

Charles H (1992) Balancing corporate power: a new federalist paper. Harvard Bus Rev 70:6

Chatterjee K, Doyen L, Henzinger TA (2009) Expressiveness and closure properties for quantitative languages. In: Logic In Computer Science, 2009. LICS'09. 24th Annual IEEE Symposium on, IEEE, pp 199–208

Cho SH, Eppinger S (2001) Product development process modeling using advanced simulation. http://web.mit.edu/eppinger/www/pdf/Cho_DTM2001.pdf. Accessed Apr 2015

Clarke EM, Grumberg O, Peled D (2000) Model checking. The MIT Press

Coudert T (2014) Formalisation et exploitation de connaissances et d'expériences pour l'aide à la décision dans les processus d'ingénierie système. https://oatao.univ-toulouse.fr/12182/1/Coudert_12182.pdf. Accessed Dec 2014

Damm W, Hungar H, Josko B, Peikenkamp T, Stierand I (2011) Using contract-based component specifications for virtual integration testing and architecture design. In: Design, Automation and Test in Europe Conference and Exhibition (DATE), 2011, IEEE, pp 1–6

Dijkstra EW (1959) A note on two problems in connexion with graphs. Numerische Mathematik 1(1):269–271

Dori D (2002) Object-process methodology: a holistic systems paradigm. Springer-Verlag, Berlin

Droste M, Gastin P (2007) Weighted automata and weighted logics. Theor Comput Sci 380(1–2):69–86

Eckert CM, Wynn DC, Maier JF, Albers A, Bursac N, Chen HLX, Clarkson PJ, Gericke K, Gladysz B, Shapiro D (2017) On the integration of product and process models in engineering design. Des Sci 3:E3. https://doi.org/10.1017/dsj.2017.2

Ernadote D (2013) An automated objective-driven approach to drive the usage of the naf framework. Information Systems Technology Panel (IST) Symposium Accessed October 2013

Ernadote D (2015) An ontology mindset for system engineering. In: Systems Engineering (ISSE), 2015 IEEE International Symposium on, IEEE, pp 454–460

Ernadote D (2016) Ontology reconciliation for system engineering. In: Systems Engineering (ISSE), 2016 IEEE International Symposium on, IEEE, pp 1–8

Eshuis R (2009) Reconciling statechart semantics. Sci Comput Programm 74(3):65–99

Fahrenberg U, Legay A (2014) The quantitative linear-time-branching-time spectrum. Theor Comput Sci 538:54–69

Fiorèse S, Meinadier JP (2012) Découvrir et comprendre l'ingénierie système. CEPADUES Editions. ISBN 978.2.36493.005.6

Friend T (2017) Agile Project Success and Failure (The Story of the FBI Sentinel Program). https://resources.sei.cmu.edu/asset_files/Presentation/2017_017_001_495733.pdf Accessed May 2017

Gausemeier J, Gaukstern T, Tschirner C (2013) Systems engineering management based on a discipline-spanning system model. Proced Comput Sci 16:303–312

Handy C (1995) Trust and the virtual organization. Harvard Bus Rev 73(3):40

Harel D (1987) Statecharts: a visual formalism for complex systems. Sci Comput Programm 8(3):231–274

Hart PE, Nilsson NJ, Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Syst Sci Cybern 4(2):100–107

Hoare CAR (1969) An axiomatic basis for computer programming. Commun ACM 12(10):576–580

INCOSE (2014) Systems engineering vision 2025, http://www.incose.org/docs/default-source/aboutse/se-vision-2025.pdf. Accessed Aug 2015

INCOSE (2015) Systems engineering handbook: a guide for system life cycle process and activities, 4th edn. John Wiley and Sons, Inc, New Jersey

Jacobs S, Bloem R (2016) The reactive synthesis competition: Syntcomp 2016 and beyond. https://arxiv.org/pdf/1611.07626. Accessed Dec 2016

Jamshidi M (2008) System of systems engineering-new challenges for the 21st century. Aerosp Electron Syst Magaz IEEE 23(5):4–19

Kamdem Simo F, Lenne D, Ernadote D, (2016) Towards modelling of modelling in SE. In: (2016) IEEE International Symposium on Systems Engineering (ISSE). Edinburgh, United Kingdom

Kamdem Simo F (2017) Model-based federation of systems of modelling. Systems and Control [cs.SY]. Université de Technologie Compiègne (UTC), 2017. English. NNT : 2017COMP2374. https://tel.archives-ouvertes.fr/tel-01948889/

Kamdem Simo F, Lenne D, Ernadote D (2015) Mastering SoS complexity through a methodical tailoring of modeling: Benefits and new issues. In: Systems Conference (SysCon), 2015 9th Annual IEEE International, IEEE, pp 516–520

Karniel A, Reich Y (2011) Managing the dynamics of new product development processes: a new product lifecycle management paradigm. Springer Science and Business Media, Berlin

Keller RM (1976) Formal verification of parallel programs. Commun ACM 19(7):371–384

Kent S (2002) Model driven engineering. Integrated formal methods. Springer, Berlin, pp 286–298

Krygiel AJ (1999) Behind the wizard's curtain. an integration environment for a system of systems. Technical report, DTIC Document

Leonard J (1999) Systems engineering fundamentals. Technical report, DTIC Document

Melo AF (2002) A state-action model for design process planning. PhD thesis, Department of Engineering, University of Cambridge

Meyer B (1992) Applying 'design by contract'. Computer 25(10):40–51

Michael RG, David SJ (1979) Computers and intractability: a guide to the theory of np-completeness. WH Free Co, San Fransico, pp 90–91

Nau DS, Kumar V, Kanal L (1984) General branch and bound, and its relation to a* and ao*. Artif Intell 23(1):29–58

Nuzzo P, Xu H, Ozay N, Finn JB, Sangiovanni-Vincentelli AL, Murray RM, Donzé A, Seshia SA (2014) A contract-based methodology for aircraft electric power system design. IEEE Access 2:1–25

O'Donovan BD (2004) Modelling and simulation of engineering design processes. PhD thesis, University of Cambridge

Pnueli A, Rosner R (1989) On the synthesis of a reactive module. In: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, ACM, pp 179–190

Ramadge PJ, Wonham WM (1987) Supervisory control of a class of discrete event processes. SIAM J Control Optim 25(1):206–230

Reich Y (2017) The principle of reflexive practice. Des Sci 3:E4. https://doi.org/10.1017/dsj.2017.3

Romero-Hernández D, de Frutos Escrig D (2014) Coinductive definition of distances between processes: beyond bisimulation distances. International Conference on formal techniques for distributed objects, components, and systems. Springer, Berlin, pp 249–265

Russell S, Norvig P (1995) Artificial intelligence. Prentice-Hall, Egnlewood Cliffs, p 27

Sage AP, Cuppan CD (2001) On the systems engineering and management of systems of systems and federations of systems. Informa Knowl Syst Manag 2(4):325–345

Sage AP, Rouse WB (2009) Handbook of systems engineering and management. John Wiley and Sons, New Jersey

SEBoK (2016) SEBok Guide to the Systems Engineering Body of Knowledge, http://sebokwiki.org. Accessed June 2017

Shaked F, Reich Y (2018) A Framework for Development Process Design and its use for Establishing Intellectual Property Governance: Introduction of the PROVE framework using a case study. In: IEEE 13th Annual Conference on System of Systems Engineering (SoSE), pp 22–28

Sharon A, Perelman V, Dori D (2008) A project-product lifecycle management approach for improved systems engineering practices. INCOSE Int Sympos 18:942–957

Sharon A, de Weck OL, Dori D (2011) Project management vs. systems engineering management: a practitioners' view on integrating the project and product domains. Syst Eng 14(4):427–440

Steward D, Tate D (2000) Integration of axiomatic design and project planning. In: Proceedings of ICAD2000, First International Conference on Axiomatic Design, MA-June, pp 21–23

Tripakis S (1998) L'analyse formelle des systèmes temporisés en pratique. PhD thesis, Université Joseph-Fourier-Grenoble I

Tripakis S (2016) Compositionality in the science of system design. Proceed IEEE 104(5):960–972

Vareilles E, Coudert T, Aldanondo M, Geneste L, Abeille J (2015) System design and project planning: model and rules to manage their interactions. Integr Comput-Aided Eng 22(4):327–342

Whittle J, Hutchinson J, Rouncefield M (2014) The state of practice in model-driven engineering. IEEE Softw 31(3):79–85

Wynn DC (2007) Model-based approaches to support process improvement in complex product development. PhD thesis, University of Cambridge

Wynn DC, Clarkson PJ (2017) Process models in design and development. Res Eng Des. https://doi.org/10.1007/s00163-017-0262-7