An industrial feedback on model-based requirements engineering in systems engineering context

Raphaël Faudou[1] and Jean-Michel Bruel[2]

Abstract— **In this paper, we synthesize a study aiming at providing industrial feedback, challenges and advanced research on the way Model-Based Systems Engineering can be used to define system requirements as well as system architecture with traceability to system requirements, which are considered as key success factors for the concerned industries.**

I. Introduction

Requirements Engineering (RE) is a set of activities that capture and transform requirements during all the stages of a system life cycle: elicitation of requirements, identification, analysis and negotiation, definition, validation and management (classification, storage, documentation, change management). Requirements address the "What" and not the "How" and should therefore remain independent of any implementation.

In most Software Engineering (SW) domains, requirements are often considered as either the result of an initial step (in classical development), or being defined at each iterative introduction of "stories" (in agile developments). But in both case they are considered as definitive, terminal artifacts.

In System Engineering (SE) it can also be the case in some very specific domains, but it is more likely that requirements at one level will produce other requirements at lower level (sub-systems), down to a level where we find requirements for software or hardware element that can be implemented, reused or purchased.

Thanks to the progress of Model-Driven Engineering techniques (methods and tools) in the past years, SE is slowly but surely moving from a document-centric activity towards a model-centric activity and the INCOSE (INternational COuncil of Systems ENgineering) expects MBSE to become a common practice in the future, as explained in the document "Vision 2025".http://www.incose.org/AboutSE/sevision. So, as Requirements Engineering being is a crucial part in the development of a system, there are high expectations in the field of Model-Driven Requirements Engineering [17].

In this paper we provide some feedback from industry on that matter that we believe interesting, especially for the SW community. It has to be noted that we do not claim to talk about all RE in general, but We only focus in this paper on the specific case when the goal is to formalize, refine, decompose, allocate and derive system requirements into system element requirements through models.

[1]R. Faudou is CEO of Samares Engineering, Blagnac, France raphael.faudou at samares-engineering.com

[2]J.-M. Bruel is Professor at University of Toulouse, France bruel at irit.fr

A. Context of the study

This study was lead by Raphaël Faudou for the French chapter of INCOSE [14], called AFIS, gathering people from academia, industry, or consulting with the goal of writing a report on current trends and challenges in Model-Based Requirements Engineering. Several brainstorming workshops have been organized during the last year. We would like to give a special thanks to the other contributors who helped gathering feedback or participated to the document reviews: Jean-Denis Piques, Gautier Fanmuy, Jean Duprez, Stéphanie Cheutin, Isabelle Amaury, Xavier Dorel, Franois Candauthil, Thuy Nguyen, Frederic Risy, Emmanuel Laurain, Jean-Charles Chaudemar and David Lesens.. . .

B. Technical processes

As stated by [4]: "System Engineering is an Interdisciplinary approach governing the total technical and managerial effort required to transform a set of customer needs, expectations, and constraints into a solution and to support that solution throughout its life." Amongst technical efforts, [3] defines system lifecycle processes and lists some technical processes that deal more specifically with system requirements and architecture definition. We detail the technical processes specifically dedicated to Requirements in the following.

1) Stakeholders Needs and Requirements definition: The first purpose of this process is to identify the stakeholders or stakeholder classes concerned by the system throughout its life cycle, and to collect their needs and expectations. Most of the time, there are conflicting needs and feasibility issues. Thus, the second purpose of the process is to analyze and transform these needs into a common set of stakeholder requirements with removal of conflicts, some trust in feasibility (first analysis) and acceptation (validation) of the stakeholders (compliance with initial needs or negotiated deviation). The main outcomes are the following: Stakeholders Assumptions made regarding the system context; Stakeholders Requirements and Rationales; Requirements; Concepts models (concept of production, deployment, operations, support, disposal, . . .); Measures of Effectiveness related to Stakeholder Needs; Traceability of Stakeholder Requirements and Requirements to stakeholders and their needs, to missions.

2) System Requirements definition: The purpose of this process is to derive the stakeholders requirements that express desired capabilities into a consistent system definition including system boundaries/interfaces and functions that meet all stakeholders requirements. This process creates a set of measurable system requirements that specify, from the suppliers perspective, what characteristics, attributes, and functional and performance requirements the system is to possess, in order to satisfy stakeholders requirements. As far as constraints permit, the requirements should not influence any specific implementation. The main outcomes of this process are the following: System Requirements in response to stakeholders requirements; System models (e.g., System Functions, System context); System interfaces (functional and physical aspects); Expected performance and associated verification means; Traceability of System requirements and Requirements to stakeholders Requirements and Concept.

3) Architecture definition: As stated in [3]: "The purpose of the Architecture Definition process is to generate system architecture alternatives, to select one or more alternative(s) that frame stakeholder concerns and meet system requirements, and to express this in a set of consistent views." The main outcomes of this process are the following: Architecture definition strategy, System architecture description, including functional architecture and physical architecture, System architecture rationale, Documentation tree, Preliminary interface definition, Preliminary TPM needs and data, Architecture traceability, Architecture definition record.

4) Design definition: As stated in [3]: "The purpose of the Design Definition process is to provide sufficient detailed data and information about the system and its elements to enable the implementation consistent with architectural entities as defined in models and views of the system architecture." The main outcomes of this process are the following: Design definition strategy, System design description, System design rationale, Interface definition, TPM needs and data, Design traceability, System element descriptions, Design definition record.

5) System analysis: As stated in [3]: "The purpose of the System Analysis process is to provide a rigorous basis of data and information for technical understanding to aid decision-making across the life cycle." This process applies to the development of inputs needed for any technical assessment. It can provide confidence in the usefulness and integrity of system requirements, architecture, and design. System analysis covers a wide range of differing analytic functions, levels of complexity, and levels of rigor. The main outcomes of this process are: Justifications about assumptions, choices, decisions on systems engineering activities; System assessment issues related to operational concepts, risk assessment and specialty engineering.

We are not describing in detail the two important processes that are Verification and Validation, but they are taken into account in the remaining analysis presented in the following.

C. Scope and organization

This paper concerns the use of models in industry in the scope of technical processes presented in previous paragraph, with special focus on *stakeholder needs* and *requirements definition*, *system requirements definition* and *architecture definition*. Models are obviously also used to support *Design definition* and *System analysis* but this document does not insist much on those activities. The remaining of the paper is as follow.

In section II we provide some useful definitions about system engineering concerning concepts that will be widely used in the following paragraphs. It will allow alignment of readers on vocabulary if needed.

In section III we summarize the industrial feedback collected during the study about usages of engineering models to support system requirements definition and architecture definition processes. Those usages are presented according to gradual approaches in the use of models and for each approach we have tried to associate benefits and efforts. We list a set of technical challenges revealed by those feedbacks.

In section IV we focus on advanced research in identification of requirements as model elements.

In section V we provide a set of common agreements shared between all contributors of the study regarding models to support requirement engineering and architecture definition.

We conclude in section VI.

II. Context and definitions

Before diving into issues and methodological axes for solutions, it is important to clearly define the scope of work and recall or refine relevant terminology.

MDE: (Model-Based Systems Engineering) is an "approach to engineering that uses models as an integral part of the technical baseline that includes the requirements, analysis, design, implementation, and verification of a capability, system, and/or product throughout the acquisition life cycle." [18]

System Architecture: is defined by "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution." [6].

Requirement: is a statement that identifies a system, product, or process characteristic or constraint, which is unambiguous, clear, unique, consistent, stand-alone (not grouped), and verifiable, and is deemed necessary for stakeholder acceptability [8];

Constraint vs. functional requirement: Requirements can be classified into several categories and there exist several classifications. We can mention [5] for a suggestion but classification is to adapt to each company. There also exist specialized classifications according to industrial domains like ARP4754a for aerospace and ECSS for space industry. As minimal distinction we find: (i) Functional requirements defining what the system shall do, and (ii) Non-functional requirements and constraints, referring to qualities the system shall have.

Decomposition and allocation: The "System life cycle processes" as described in the [3] formalize the decomposition of a system into a set of interacting system elements, each of them being then implemented for integration into the system. Requirements on the whole system ("System requirements") are refined into requirements allocated to system elements ("Specified requirements") in order that the implementation of requirements related to system elements may be delegated to another party through an agreement. Decomposition and allocation of requirements have to be performed consistently with architectural and/or design definition.

Traceability: is a technical mean in development process used primarily to ensure the continuity and completeness in the refinement of the need (specification and requirement set) in the solution. Therefore, traceability is not limited to requirements but to any item or piece of information that have to be managed to guarantee compliance of results to expectations. The minimal prerequisites from a basic traceability mechanism are: (i) a unique identifier for each item under traceability, and (ii) relationships to link items, with a well-defined semantics (e.g., "refine", "derive", "verify", ...).

III. Industrial feedback

We asked several industrial partners their feedback on the way they moved or plan to move from requirements expressed in natural language to requirements formalized with models. Indeed many organizations consider nowadays the opportunity to switch from purely textual requirements to a model-based approach, where models tend to complete or even replace the use of free text and act as the new contractual baseline. In such a paradigm, model exchanges would replace the traditional textual specifications flows between stakeholders. However such a change requires in particular that all stakeholders share a common understanding of the various models involved in the process and that they agree on new data exchange modalities at organizational boundaries. This is far from obvious and strongly depends on background, experience and culture that are naturally different according to companies and teams. This is why in practice, today, organizations tend to evolve "progressively" from textual requirements to model-based approaches, trying to change their internal ways of working before trying to change their interfaces with external stakeholders.

A. The goal of modeling requirements

Models can be used at different stages and for different purposes and it is very important to clearly define goal and context (e.g., stakeholders culture) of the modeling activity if we want to get returns.

Models can be used for clarification, structuring and first formalization of stakeholders needs. In that case, models shall reflect stakeholders needs through simple, easy to learn, concepts like "messages" exchanged between the system of interest and its operational context over time (e.g., sequence diagrams in SysML). Representation shall be simple enough so that stakeholders can validate it as conforming to their original expression of the needs.

Models can be used as intermediate means in order to support maturation (or refinement) of requirements expressed in natural language in order to detect issues with completeness, correctness or consistency. For instance, there can be simulation models used to check some ranges of values for particular performance properties. Such models can even be used as "rationale to demonstrate interest of requirements and avoid multiplication of "useless requirements.

Models can also be used as pivotal format toward a "user friendly" representation of finalized requirements to support validation by stakeholders. For example it can be a dynamic mockup based on model execution/simulation in order to validate some innovative functional requirements quite complex to understand on their textual format. Note that in that case, to provide full credit for validation in certification context, there is some extra work to be done in order to demonstrate good translation between initial textual requirements, model pivotal format and final mockup that is "validated by stakeholders (also called "early validation). Whatever the approach, the goal remains the same: to ensure good requirement engineering.

In the following, we recall good engineering practices and key properties that requirements shall fulfill (whether textual or formalized through models). Then we suggest a graduation of approaches that reflect the various rates at which models are integrated within the specification activities and present challenges and questions about the use of models for requirements specification.

B. Key goals of RE: Correctness, Consistency, Completeness

Requirements Engineering consists in establishing and maintaining requirements from users needs to system or design requirements up to elementary system elements. For each system requirement activities are done in parallel of architecture activities (e.g., Figure 1).

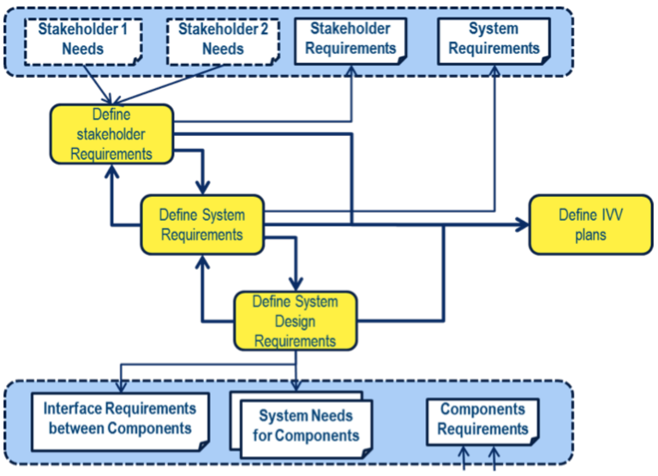

Stakeholder 1 Needs
Stakeholder 2 Needs
Stakeholder Requirements
System Requirements
Define stakeholder Requirements
Define System Requirements
Define IVV plans
Define System Design Requirements
Interface Requirements between Components
System Needs for Components
Components Requirements

Fig. 1. RE process (adapted from [7]

Traceability is established during system definition as well as during integration, verification and validation. Several goals are expected and we detail them in the following.

1) Correctness: Obtain well-formed requirements, also known as SMART (Specific, Measurable, Achievable, Realistic, Traceable). The detailed list of characteristics are detailed in appendices.

2) Ensure completeness and consistency: A set of stakeholders requirements should have the following characteristics to ensure that the needs and constraints are complete:

- Address complete set of Needs and Constraints from stakeholders, external systems and enabling systems.
- Address complete set of missions or business processes.

A set of stakeholders requirements should have the following characteristics to ensure that the needs and constraints are consistent:

- Does not have individual requirements that are contradictory.
- Requirements are not duplicated.
- The same term is used for the same item in all requirements.

3) Ensure completeness and consistency of the System design against the stakeholders expectations: A set of System and Design Requirements should have the following characteristics to ensure that the set of requirements collectively provides for a feasible solution that meets the stakeholders expectations and constraints (based on [5]):

- Address complete set of expectations and Constraints
- Does not have individual requirements that are contradictory.
- Requirements are not duplicated.
- The same term is used for the same item in all requirements.
- Are aligned and consistent with architecture elements
- Can be satisfied by a solution that is obtainable/feasible within life cycle constraints (e.g., cost, schedule, technical, legal, regulatory).
- Maintains the identified scope for the intended solution without increasing beyond what is needed to satisfy user needs

C. Different approaches

In terms of requirement modeling, we can describe several approaches.

1) Requirements defined in natural language and illustrated with drawings/diagrams: This approach is the “traditional” one found in almost all companies. Requirements are written in natural language and some diagrams are inserted in the specification document. The goal is mostly illustrative. Main tools used to build diagrams are Microsoft Visio/PowerPoint because they provide easy to use editors with large libraries of icons. When so, the diagrams have limited semantics and it is not always obvious for readers to clearly understand their semantics without explanation. When modeling tools are used to describe requirements using standard languages like SysML, there is no need to provide a legend because the notation is standardized and thus considered as known by readers. Finally, in that context, using modeling languages has limited returns, as this approach requires efforts to produce diagrams that cannot be used for specification, verification and validation activities. Benefits remain limited to high-level communication.

2) Use of models to mature and verify requirements defined in natural language: In this approach some requirements are formalized through a model and associated diagrams (graphical views of the model). In comparison to previous approach, the modeling activity is here motivated by requirement maturation and verification. The benefits of the use of models can vary according to the type of requirements covered (), but the overall conclusion is that, compared to the “traditional” use of documents, use of models requires a different distribution of efforts. Use of models requires more efforts at the very beginning (learn the modeling language, learn associated tooling, ask more questions to define the right model) and less after because there is less iteration to reach requirement maturity. Savings are expected in this reduction of iterations and faster maturation in the first cycles.

3) Use of models to formalize requirements initially expressed in natural language: This approach consists in translating requirements in a formalized language that will ease verification activities (consistency, correctness) and will help building a functional architecture correct by construction. We can not treat that aspect in details in that paper due to space limitations.

It is important to notice that this approach considers formalized requirements model as an engineering artifact. It means maintaining that model in configuration and maintaining links with initial textual requirements. Those efforts are necessary to maintain requirement model and other links to architecture when textual requirements change. There can be some tooling challenge here as initial textual requirements are generally stored in a database. This challenge is partially covered today, as most commercial modeling tools provide gateway with most famous requirement database solutions. But there is still a methodological challenge: should new requirements identified after refinement through the model be put back in the requirement database? Does that mean “duplication” in two different repositories?

D. Challenges and questions about use of models for requirements specification

Let us examine what formalizing requirements through models means in practice. As mentioned in the introduction, we focus in this paper on the special case when the goal is to formalize, refine, decompose, allocate and derive system requirements into system element requirements through models. It means that modeling language and tools shall allow supporting many activities concerning requirement engineering and management. Whatever the format of the requirements (text, table, graphical, ...) and whatever their formalization (natural language, modeling language, ...), requirements shall be engineered in a way that can be formally traced. In some critical industries such as avionics, this will be called certification. In the following, we recall the RE good practices that particularly apply for requirement models. The question we try to answer here is: starting from system-level requirements, is it possible to ensure that our architecture definition model will provide a set of complete and consistent requirements with no overlapping? And is it possible to ensure that those developed requirements will be at the right level of abstraction?

1) Building complete and consistent logical architecture: When defining system architecture, SeBoK [9] recommends starting with logical architecture expressed with 3 view points: functional architecture, behavioral architecture and temporal architecture. There exist a lot of modeling techniques to support one or part of those view points: Functional decomposition, FFBD, EFFBD, Integrated Definition for Functional Modeling, N2 charts, operational scenarios, ... But the challenge is here:

Challenge: To ensure that all used techniques can be applied on the same model (else we will have to deal with model transformations) and can be combined consistently (without overlap).

2) Formalization of functions and scenarios of functions: SeBoK [9] suggests starting by analyzing operational scenarios and interface constraints in order to deduce system level functions and their external interfaces in terms of data/energy/... inputs and outputs. Some modeling languages provide "function" or "functional block" as native concept but others do not. For example, in SYSML there is no such concept, hence in this kind of notation:

Challenge: How can functions be modeled so that they can be traced to operational scenarios, not just at syntactic level, but also with conformance between them?

While out of the scope of this paper, let us mention that, for example in SYSML, we can find three main options in industry: Blocks, Activities or Operations with implications on each of these approaches.

3) Linking functional and behavioral elements with temporal architecture: Good SE practices (including SE Handbook [8]) recommend defining mission phases, system operational states and modes in order to precise conditions for function execution. And here again decomposition shall be possible in order to represent those conditions with different abstraction levels. States views (state machine, state flow, Petri net, ...) are different techniques that can support those concepts.

Challenge: How can functions be linked to phases, modes and states?

If different modeling languages are used with different semantics to express both modes and functions, it will become hard to define links with clear semantics between functions and modes/states. Even if we consider the hypothesis of using a unique modeling language that can cover both functions and states concepts, what would be the link? For example in SYSML, there is at least three ways of creating these links.

4) Trade-offs concerning system architectures: Most of the time, the work of an engineer is to examine several potential solutions of a given problem, and hopefully chose the best one.

Challenge: Is it possible to formalize different architectures in the same model in order to get visual support for comparison? In that case, how to interpret traceability links? Should the links be put on all solutions or should they be put only on preferred architecture? Should there be only one architecture for a given model?

Different alternatives exist in industry:

- Define alternate solutions through different models put in version control and configuration. Comparison can then be performed between models (requires modeling tool comparison capability) or through assessment of each model and comparison of those assessments.
- Some provide alternate solutions in same model and use inheritance to show how those solutions relate to each other. Inheritance can be used to show different solutions expressed through blocks or to show different functions expressed as activities. One can even use inheritance between Use Cases to capture abstract system functionalities.
- In case there are many solutions to consider, inheritance is not enough and it is better to consider variability outside of system model. Variability models, defined orthogonally to the system specification, can be used to support that challenge (cf. a dedicated white paper on MBSE applied to Product Line engineering []).

5) Creation of new requirements in the model: Function decomposition brings new requirements associated to sub functions.

Challenge: How to express those requirements? Is it needed to extract them from model and put them in the requirement database? Can they remain in the model with ability to identify them as requirements?

Some industrial experts argue that all requirements shall be managed in the "requirements database. In that case it means that all new requirements are exported back into the requirement database. Requirement hierarchy and Function hierarchy are strongly related. This is Tim Weilkins' "zig zag pattern" in the SYSMOD approach[1]. Another option is to consider that decomposition of requirements that follows functional decomposition can be kept as part of the model. Only requirements derived at lower level will be exported to the requirement database. It is simpler with fewer requirements to manage in the requirement database but it provides less control on function decomposition.

IV. REQUIREMENTS AS MODEL ELEMENTS

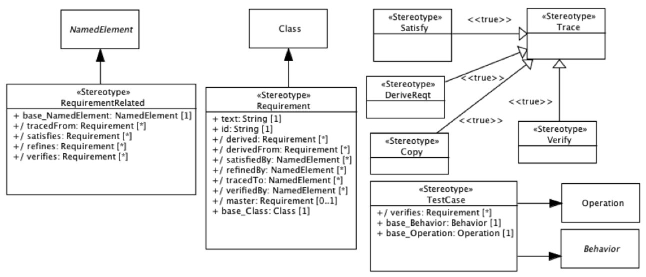
In this section we mention examples of dedicated efforts towards the modeling of requirements or their identification in the models. The objective is to characterize model elements that can be considered as requirements.

The goal of this section is not to be an exhaustive state of the art, but to illustrate some proposed approaches, from the less formal to the more formal.

[1]See `http://model-based-systems-engineering.com/2012/03/26/the-sysmod-zigzag-pattern/`.

A. SysML

SYSML is a de facto standard for modeling systems and as such has introduced (in addition to the UML concepts it has inherited) the dedicated concept of Requirement with a specific *Requirement Diagram* and *Requirement Table* among the 9 types of diagram available.



NamedElement
Class
«Stereotype» Satisfy
<<true>>
«Stereotype» Trace
«Stereotype» RequirementRelated
+ base_NamedElement: NamedElement [1]
+/ tracedFrom: Requirement [*]
+/ satisfies: Requirement [*]
+/ refines: Requirement [*]
+/ verifies: Requirement [*]
«Stereotype» Requirement
+ text: String [1]
+ id: String [1]
+/ derived: Requirement [*]
+/ derivedFrom: Requirement [*]
+/ satisfiedBy: NamedElement [*]
+/ refinedBy: NamedElement [*]
+/ tracedTo: NamedElement [*]
+/ verifiedBy: NamedElement [*]
+/ master: Requirement [0..1]
+ base_Class: Class [1]
«Stereotype» DeriveReqt
<<true>>
<<true>>
<<true>>
«Stereotype» Copy
«Stereotype» Verify
«Stereotype» TestCase
+/ verifies: Requirement [*]
+ base_Behavior: Behavior [1]
+ base_Operation: Operation [1]
Operation
Behavior

Fig. 2. Requirements concepts (metamodel) in SysML

A requirement in SYSML as: "...specifies a capability or condition that must (or should) be satisfied. A requirement may specify a function that a system must perform or a performance condition a system must achieve" [13].

The most interesting mechanisms are the relationships that can be defined between the requirements or between the requirements and other model elements, such as Containment, <<deriveRqt>>, <<refine>>, <<satisfy>> or <<verify>>.

Despite all the existing mechanisms and their detailed definition in the standard, no formal semantics is attached to those traceability links which might imply additional efforts in terms of semantics (e.g., ReqCycle [1] at tooling level or Figure 3 at methodological level).

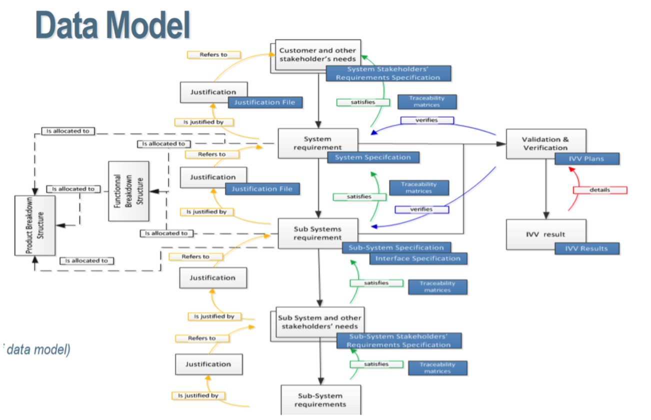


Fig. 3. Example of traceability process data model (from Dassault Systems)

It is out of the scope of this paper to be more precise on SYSML, but for more details on the way requirements are specified in SYSML, see [20], [10].

B. Property-Based Requirements

Patrice Micouin has developed the Property-Based Requirement (PBR) theory, which is now fully integrated in a complete MBSE method called PMM (Property Model Methodology) [16]. In this approach a requirement is expressed by a formula like:

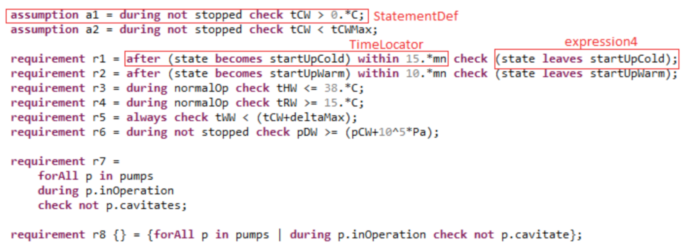
$$PBR : [When\ C \rightarrow] val(O.P) \in D$$

This formula has to be read and understood as the following statement: "when the condition C is true, the property P of the object O is actual and its value shall belong to a domain D" where C is a relevant condition for the system or its environment and where the domain D is a finite or infinite set such as $0, 1$ or R^n (possibly linked to a frame and a physical unit). The concept of PBR can be implemented directly as an assertion (boolean function) in various simulation languages. According to PMM, a system specification model is a formal model of a represented system that includes: (i) system requirements (PBRs), (ii) system interface requirements (inputs and outputs definition) and (iii) system assumptions (PBRs).

Airbus has extended PBR theory with the RTA (Requirement Tracker Artifact) approach[11]. This paper shows how the PBR definition given by Micouin can be used as a pattern for finding matching elements among UML native metaclasses of a modeling language. Then, based on the Micouin theory it is possible to interpret any instance of those metaclasses as a requirement. Those RTAs are used for "tracking" requirement information within the model. Also, based on their semantics it possible to identify what kind of relationship among the selected metaclasses can be interpreted as traceability links and how. The analysis of the modeling language for selecting the metaclasses is essential. For some of them the decision is obvious but for some other it may depends on the way the modeling language is used and by the way, may depend on the modeling methodology that will need to be precisely specified. Once the list of RTA metaclasses is defined, it is relatively easy to write queries that can be used for parsing a model and identifying requirements. In addition the identification of the requirements and their relationship, it is also possible to extract the specifications thereof and to translate them in "human readable" text. By interpreting the RTA metaclasses according to the UML semantics it is possible to design a set of text boilerplates that can be filled according to the specific context of each RTA instance.

C. EDF R&D

EDF R&D has defined a formal notation to help ensuring that digital systems and complex cyber-physical systems comply with their behavioral requirements. FORM-L (Formal Requirements Modelling Language) has been specified in the framework of the ITEA2 MODRIO project. This language aims to fill some gaps that exist in systems engineering and especially those which are related to simulation of requirements and assumption of a system. FORM-L provides a formal way for modeling properties (requirements and assumptions) of a given system, that means that this language provides formal syntax and semantics that can be used as input to a dedicated simulation tool [19]. Based on an extension to MODELICA, a language used to model the behavior of physical systems they have defined FORM-L, a textual notation where a requirement or an assumption is a particular case of the more general notion of *property* (see illustration in Figure 4). In its most general form, the specification of a property addresses four questions: (i) WHAT needs to be satisfied, (ii) WHEN in time that needs to be satisfied, (iii) WHERE in the system that needs to be satisfied, and (iv) HOW WELL that needs to be satisfied (given the fact that real life systems can and will fail). This language is also intended to support the coordination of the many teams and disciplines involved in the design and operation of a large and complex system.



```
assumption a1 = during not stopped check tCW > 0.*C;
assumption a2 = during not stopped check tCW < tCWMax;

requirement r1 = after (state becomes startUpCold) within 15.*mn check (state leaves startUpCold);
requirement r2 = after (state becomes startUpWarm) within 10.*mn check (state leaves startUpWarm);
requirement r3 = during normalOp check tHW <= 38.*C;
requirement r4 = during normalOp check tRW >= 15.*C;
requirement r5 = always check tWW < (tCW+deltaMax);
requirement r6 = during not stopped check pDW >= (pCW+10^5*Pa);

requirement r7 =
    forAll p in pumps
    during p.inOperation
    check not p.cavitates;

requirement r8 {} = {forAll p in pumps | during p.inOperation check not p.cavitate};
```

Fig. 4. Example of FORM-L requirements (from [12])

V. Lesson learned and recommendations

We provide in this section a list of issues that have been commonly agreed by the participants of the study as the important ones for a more systematic use of models in requirements engineering.

A. *Align SE practices before deploying MBSE*

When trying to use models we sometimes discover that we do not have the same understanding behind the same model element. Everyone does the mapping with his/her own culture and vocabulary and they might slightly differ between colleagues. For instance, take a survey about what a use case means for systems engineers: some will explain that it captures the mission purpose while others will translate it into "system function" or "operational scenario". It is not possible to use modeling language properly and take advantage of its unified notation if systems engineering concepts are not shared among team members. A good solution to align systems engineers on same definition for model elements is training. Quite often the training sessions start with assumptions on knowledge about systems engineering and they reveal to be false for several attendees. It is better to align people on systems engineering vocabulary and processes before starting using models. One option is also that the training might take this situation into account and teach both systems engineering and use of models to support it at the same time. Finally, while there is no unique strategy to adopt MBSE, one of the requisites is to know systems engineering good practices and align on definitions and concepts: this is the foundation layer for a system team before building and sharing models.

B. *Efficient modeling requires goal and strategy*

Modeling can be used in different gradual approaches and benefits are different according to the way modeling is involved. For instance, when models are used as a simple illustration of some textual requirements, RE activity remains unchanged. Modeling in that case can be considered as a complement to the textual requirements. It is important to understand that benefits, in that situation, will be low and that it is not worth investing a lot in elaborated modeling languages and tools. Moreover the effort to maintain consistency between requirements and models will be often seen as a useless extra effort. If a company uses models without clear objectives about the benefits and the engineering activities they will support, there is great chance that modeling activities end with frustrations on benefits that will remain low as well as on low recognition from management (or rest of the team). Especially if there was a lot of efforts done with modeling and no clear benefits to align on this investment. When the goal is instead clearly defined and aligned on reducing efforts on some engineering activities, it is important to define a strategy to reach this goal. Practically it means defining a modeling method that addresses at least the following elements:

- Which concepts to use and when (which stage)
- Which diagrams to produce and with which abstraction level
- Modeling refinement stop criteria
- Modeling structure for model organization, ease review and collaborative editing

The challenge can then summarize as: (i) to find which engineering activities can be improved and if modeling might help; (ii) then try to identify some savings; and (iii) setup a modeling method in order to reach that target savings.

C. *Which modeling strategy (short-term or long-term)?*

An important question that should be addressed very early when a company intends to use models is "Do we intend to use models as a means to improve the first baseline only (use model in one shot and then throw it away)? Or do we want to use model as a key artifact of engineering on the whole life of product development and maintenance?" If models are used for long term, they will need to be maintained. Hence, like for any reference-engineering artifact, it is important to place the models in configuration (baseline) and to ensure traceability with upstream requirements. Models' traceability is important because if the produced models are not traced with upstream requirements (stakeholder requirements for operational model, system requirements for system architecture model, . . .), they will be useful only for current baseline of requirements. As soon as there are new or modified requirements, it becomes very hard to detect changes on the model, and at some point in time some models become obsolete and cannot be used to support maturation of requirements or verification of architecture. So, it is important to get this consequence in mind very early in the project: If we choose long term modeling, it is highly recommended to trace model with upstream requirements so that impacts can be analyzed quite easily after some change in upstream requirements.

D. Using model to formalize a large part of the specification is a big investment

This strategy is not easy because it requires that models become fully consistent, complete and at the right level of abstraction. It also implies that they can be traced upward to system requirements and stakeholder requirements, or backward to system element requirements. This is not so complex when dealing only with a few requirements like interface ones, but it becomes a hard task when we want to be able to generate most of the system specification from modeling elements. There are a lot of challenges to address and there is no "magical" modeling language or method that can be used as-is for any project. Some challenges are still actively researched. Let us mention:

- identify requirements from model,
- ensure traceability through model elements
- provide methodology and tooling support

Targeting formalization of whole system specification with models is a strategy that requires a large investment on knowledge in the modeling languages and their customization/specialization in order to cover all needed concepts in the engineering domain considered in project. Special attention shall also be given to verification activities and rules because model becomes the reference. But those efforts should be balanced with the good returns that are generally obtained through accelerated maturation of system definition and design, and all the analysis efforts and time saved during maintenance of the system thanks to the easy navigation through models.

E. One master repository for specification at each refinement level

There is no reason to try to adopt only one notation to model requirements as well as there is no need to have a completely centralized repository of models. Nevertheless, we recommend identifying one and only one master repository for system definition at each level of refinement. It can be for instance *PowerPoint-like* documents at business/mission level, *DOORS-like* database at stakeholder requirements definition level, and *SysML-like* models at system level and architecture level (definition of system elements) down to hardware parts and software applications. Different master repositories across levels help adapting tooling with regards to the skills and culture of people in charge of defining specifications at that particular level. Different techniques can ensure consistency across refinement levels (configuration management, use of dedicated pivot language, use of globalization frameworks, . . .). The baseline will provide links between data from the different repositories used as reference for each refinement level. In addition, using different repositories for different levels allows adapting to different teams (skills, culture) with more flexibility, which is very important for MBSE deployment. It could be argued that it might generate complexity in tooling in order to create the right links between dispatched data for the baseline, but progress from the MDE community in this matter has demonstrated that this can be handled [2].

In the general consensus from the participants of the study, there shall be a unique master repository for specification for each refinement level. It can of course be the same one for different levels. But experience has shown that if there are two or more repositories at the same level without priority in reference, there is a high risk of confusion for systems engineers and at some point in time there will be different modifications done concurrently in both repositories, root cause of errors, or one repository will not be maintained. This is particularly true for models and documents. If documents remain the reference for specification (for instance at system level), when project has to deliver a new release of specification and deadline is about to be reached, team will focus on document because it is the contractual deliverable. Consequence is that model will become obsolete within months because nobody will accept or dare to do the same modifications on two repositories: document and model. So if there is a lot of efforts to build a model that contains most of the specification information, and especially if goal is to use model to support formalization of specification, it is very important to decide that the model becomes the reference and use the document as an output (automatically generated) from the model.

VI. Conclusion

We focused on this paper on the way requirements-related concepts, (such as stakeholders needs, requirements definition and system requirements definition) are addressed through the use of models to support systems engineering of complex systems. We have illustrated the important role models can play and we have raised a number of challenges for a more efficient use of models to address requirements engineering concerns and we provided some recommendations from lessons learned.

Appendix

A. Attributes of Requirements statements

Individual requirement statements may have a number of attributes attached to them. The aim of these attributes is to provide complementary information to Requirements statements to enable the analysis of Requirements. The list below gives an overview of good practices about requirement attributes:

- Identifier
- Title
- Statement
- Version
- Stakeholder, submitter
- Maturity
- Concerned Products
- Targeted Release
- Priority, Importance, Weight
- Negotiability, Flexibility
- Cost impact
- Risk
- Verification Method
- Validation Method
- Compliance
- Change Request
- Created by
- Last modified by
- Created on
- Last modified on

Note: from industrial feedback, it is important to notice that management of attributes is very costly and that most of defined attributes are never filled or with a lot of mistakes. So there are two simple suggestions for improvement: (i) Assess usage and usefulness of each attribute on each project in order to be sure that attributes will bring value. (ii) When attributes have been confirmed for their usefulness, it is a good practice to guide engineers on the phase or activities when they are supposed to fill/update the attribute.

B. Requirement correctness

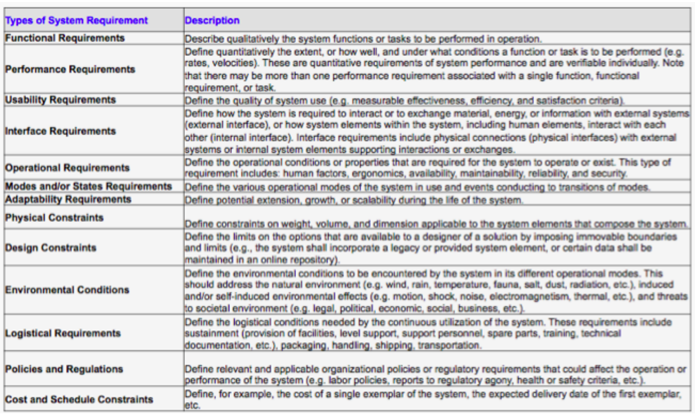
Is it important that requirements are well-formed, whatever notation is used to model them. It might even be a good benefit that the modeling notation itself enforce this well-formedness (for example by constraining the syntax, or by providing validation rules).

The following important requirements' criteria are taken from SMART [15].

- Necessary
  - Defines an essential capability, characteristic, constraint, and/or quality factor.
  - If removed or deleted, a deficiency will exist, which cannot be fulfilled by other capabilities of the product or process.
  - Is currently applicable and has not been made obsolete by the passage of time.
  - Requirements with planned expiration dates or applicability dates are clearly identified
- Implementation Independent
  - Address what is necessary and sufficient in the system
  - Avoids placing unnecessary constraints on the architectural design.
  - States what is required, not how the requirement should be met.
- Unambiguous
  - Stated in such a way so that it can be interpreted in only one way.
  - Stated simply and is easy to understand
- Complete
  - Needs no further amplification: it is measurable and sufficiently describes the capability and characteristics to meet the stakeholder's need.
  - Contains no To Be Defined (TBD), To Be Specified (TBS), or To Be Resolved (TBR) clauses.
  - Resolution of the TBx designations may be iterative and there is an acceptable timeframe for TBx items, determined by risks and dependencies
- Singular
  - Includes only one requirement with no use of conjunctions.
- Feasible
  - Is technically achievable, does not require major technology advances, and fits within system constraints (e.g., cost, schedule, technical, legal, regulatory) with acceptable risk.
- Verifiable
  - Has the means to prove that the system satisfies the specified requirement.
  - Evidence may be collected that proves that the system can satisfy the specified requirement. Verifiability is enhanced when the requirement is measurable.
- Correct
  - Has to be met or possessed by a system to solve a stakeholder problem or to achieve a stakeholder objective.
  - Is qualified by measurable conditions and bounded by constraints.
  - Defines the performance of the system when used by a specific stakeholder or the corresponding capability of the system, but not a capability of the user, operator, or other stakeholder.
- Traceable
  - Is upwards traceable to specific documented stakeholder statement(s) of need, higher tier requirement, or other source (e.g., a trade or design study).
  - Is downwards traceable to the specific requirements in the lower tier requirements specification or other system definition artifacts.
  - All parent-child relationships for the requirement are identified in tracing such that the requirement traces to its source and implementation.

C. Suggestion of classification for requirement types

Figure 5, taken from [5, section 9.4.2.3], provides a list of categories and associated semantics when it comes to requirement types.



Types of System Requirement	Description
Functional Requirements	Describe qualitatively the system functions or tasks to be performed in operation.
Performance Requirements	Define quantitatively the extent, or how well, and under what conditions a function or task is to be performed (e.g. rates, velocities). These are quantitative requirements of system performance and are verifiable individually. Note that there may be more than one performance requirement associated with a single function, functional requirement, or task.
Usability Requirements	Define the quality of system use (e.g. measurable effectiveness, efficiency, and satisfaction criteria).
Interface Requirements	Define how the system is required to interact or to exchange material, energy, or information with external systems (external interface), or how system elements within the system, including human elements, interact with each other (internal interface). Interface requirements include physical connections (physical interfaces) with external systems or internal system elements supporting interactions or exchanges.
Operational Requirements	Define the operational conditions or properties that are required for the system to operate or exist. This type of requirement includes: human factors, ergonomics, availability, maintainability, reliability, and security.
Modes and/or States Requirements	Define the various operational modes of the system in use and events conducting to transitions of modes.
Adaptability Requirements	Define potential extension, growth, or scalability during the life of the system.
Physical Constraints	Define constraints on weight, volume, and dimension applicable to the system elements that compose the system.
Design Constraints	Define the limits on the options that are available to a designer of a solution by imposing immovable boundaries and limits (e.g., the system shall incorporate a legacy or provided system element, or certain data shall be maintained in an online repository).
Environmental Conditions	Define the environmental conditions to be encountered by the system in its different operational modes. This should address the natural environment (e.g. wind, rain, temperature, fauna, salt, dust, radiation, etc.), induced and/or self-induced environmental effects (e.g. motion, shock, noise, electromagnetism, thermal, etc.), and threats to societal environment (e.g. legal, political, economic, social, business, etc.).
Logistical Requirements	Define the logistical conditions needed by the continuous utilization of the system. These requirements include sustainment (provision of facilities, level support, support personnel, spare parts, training, technical documentation, etc.), packaging, handling, shipping, transportation.
Policies and Regulations	Define relevant and applicable organizational policies or regulatory requirements that could affect the operation or performance of the system (e.g. labor policies, reports to regulatory agony, health or safety criteria, etc.).
Cost and Schedule Constraints	Define, for example, the cost of a single exemplar of the system, the expected delivery date of the first exemplar, etc.

Fig. 5. Types of systems requirements (from [5]

D. Good practices about requirement attributes

Table I provides a set of good practices about requirement attributes.

TABLE I

GOOD PRACTICES ABOUT REQUIREMENT ATTRIBUTES

Identifier	Identify a requirement	Allows to uniquely identifying each requirement, in order to reference it in documents or other tools, or for traceability.
Title	Identify a requirement	Short text that summarizes the detailed description of the requirement.
Statement	Specify a requirement	Statement of the requirement, short and precise, without any justification nor additional details. Could be textual, graphical...
Version	Lifecycle	Current version of the requirement.
Stakeholder, Submitter	Traceability	Originator of the requirement, responsible for the requirements.
Maturity	Lifecycle	Current state of the requirement in its lifecycle.
Concerned Products	Define the scope	Allows defining which elements of a product line are concerned by a requirement. Do not confuse with allocation of system requirements to the components of this system. Can also be made through traceability links to a description of product families.
Targeted Release	Define the scope	In case of incremental delivery, this attribute allows defining which increment will take this requirement into account. Can also be made through traceability links.
Priority, Importance, Weight	Qualify requirements	Assessment of the requirement business value for stakeholders (e.g., final users, maintenance team, etc.). But also Key Design Drivers.
Negotiability, Flexibility	Qualify requirements	Assessment of the possibilities to negotiate the requirement. (e.g., regulatory control and security are not negotiable).
Cost Impact	Qualify requirements	Assessment of the impact of the requirement on the final product cost or development cost (e.g., High, Medium, Low). Note: Usually made for a set of requirements.
Risk	Risk Management	According to the performed risk analysis (security, performance, processes, etc.), attributes like 'Risk for Safety', 'Risk for process'... may be filled. Risk analysis is usually a separate process. The risks attributes can also be managed through links.
V&V Method	Perform V&V	Allow identifying Verification and Validation methods (e.g., Inspection / Analysis / Demonstration / Test) when elaborating the requirement.
Compliance	Traceability	Allows evaluating an answer to a call for tender: verification of the answers compliance to the requirements of the emitter of the call.
Change Request	Traceability	Reference of the change request at the origin of the requirement creation or modification. Note: should be implemented as a link to a change management system.
Created by	Lifecycle	Creator of the requirement. Do not confuse with the stakeholder.
Last Modified by	Lifecycle	Author of the latest requirement modification. Do not confuse with the stakeholder.
Created on	Lifecycle	Date of the requirement creation.
Last Modified on	Lifecycle	Date of the latest requirement modification.

ACKNOWLEDGMENT

The author would like to thank their colleagues from AFIS for their contributions and for the valuable discussions.

REFERENCES

[1] ReqCycle. Available at `https://www.polarsys.org/projects/polarsys.reqcycle`.

[2] The GEMOC Initiative – On the Globalization of Modeling Languages.

[3] ISO/IEC/IEEE 15288:2015. Systems and software engineering – System life cycle processes.

[4] ISO/IEC/IEEE 24765:2010. Systems and software engineering – Vocabulary.

[5] ISO/IEC/IEEE 29148:2011. Systems and software engineering – Life cycle processes – Requirements engineering.

[6] ISO/IEC/IEEE 42010:2011. Systems and software engineering – Architecture description.

[7] Collectif AFIS. *Guide Bonnes Pratiques en Ingénierie des Exigences*. 2012.

[8] Collectif AFIS. *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. 2015.

[9] SEBoK authors. Guide to the Systems Engineering Body of Knowledge (SEBoK), version 1.6. Available at `http://sebokwiki.org/w/index.php?title=MediaWiki:Cite_text&oldid=52160`, 2016. Online; accessed 14 Jun 2016 16:40:21 UTC.

[10] Nicolas Belloir, Jean-Michel Bruel, and Raphaël Faudou. Modlisation des exigences en UML/SysML. *Génie Logiciel*, pages 6–12, 2014.

[11] Yves Bernard. Requirements Management Within a Full Model-based Engineering Approach. *Syst. Eng.*, 15(2):119–139, June 2012.

[12] Alexandre Le Borgne, Nicolas Belloir, Jean-Michel Bruel, and Thuy Nguyen. Formal Requirements Engineering for Smart Industries: Toward a Model-Based Graphical Language, 2016.

[13] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML: Systems Modeling Language*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[14] INCOSE. SE Vision 2025. Available at: `http://www.incose.org/docs/default-source/aboutse/se-vision-2025.pdf?sfvrsn=4`, 2016.

[15] Mike Mannion and Barry Keepence. SMART Requirements. *SIGSOFT Softw. Eng. Notes*, 20(2):42–47, April 1995.

[16] Patrice Micouin. *Model Based Systems Engineering: Fundamentals and Methods*. Wiley & ISTE, 2014.

[17] Gunter Mussbacher, Joao Araujo, Ana Moreira, and Pablo Sanchez. IEEE International Model-Driven Requirements Engineering Workshop (MoDRE). In *Model-Driven Requirements Engineering Workshop (MoDRE), 2015 IEEE International*, pages c1–c2, Aug 2015.

[18] NDIA. Final Report, Model-Based Engineering Subcommittee, 2011.

[19] Thuy Nguyen. Modelling and Simulation of the Dynamics of Complex Socio-Cyber-Physical Systems and Large Scale Systems of Systems all along their Lifetime. technical report, EDF, 2016.

[20] Pascal Roques. Modeling Requirements with SysML. Requirements Engineering Magazine, available at `https://re-magazine.ireb.org/issues/2015-2-bridging-the-impossible/modeling-requirements-with-sysml/`, 2015.