Requirements and Architecture within Modelling Context

This White Paper is an AFIS collective work
directed by the MBSE technical Committee

Released in December 2016

ISBN 978-2-900969-00-7

ORIGIN OF THE WORK

This White Paper “Requirements and architecture within modelling context” has been produced by the Model-Based Systems Engineering technical committee of AFIS (French INCOSE chapter).
That committee deals with:

- Improvement of knowledge and skills around use of models in systems engineering field during all system life cycle stages
- Share of industrial feedback on approach and issues to setup and deploy MBSE methodology in their company
- Development of specific topics concerned by MBSE (SysML, requirements within models, simulation as early validation means…) and development of small projects to address some of those topics in coordination with other technical committees.

THANKS

This White Paper from AFIS (INCOSE French chapter) collection on “Requirements and architecture within modelling context” is the product of a collaborative work of AFIS.

Raphaël Faudou wrote successive versions by integrating contributions from Gauthier Fanmuy, Jean Duprez, Jean-Denis Piques, Xavier Dorel and himself; in this way, he was able to ensure the editorial and pedagogical consistency of the whole.

Proofreading was carried out by a small group (thank you to David Lesens, Jean-Michel Bruel, Isabelle Amaury, Emmanuel Laurain, Guilhem Chevalier, Jean-Charles Chaudemar, Frederic Risy), then by the MBSE Technical Committee of AFIS.

The following people have participated in its development:

Isabelle AMAURY, MBDA-SYSTEMS

Jean-Michel BRUEL, IUT Blagnac, IRIT

Jean-Charles CHAUDEMAR, ISAE-SUPAERO

Guilhem CHEVALIER, AIRBUS

Xavier DOREL, SCHNEIDER ELECTRIC

Jean DUPREZ, AIRBUS

Gauthier FANMUY, DASSAULT SYSTEMES

Raphaël FAUDOU, SAMARES ENGINEERING

Emmanuel LAURAIN, RENAULT

David LESENS, AIRBUS SAFRAN LAUNCHERS

Jean-Denis PIQUES, VALEO

Frederic RISY, AIRBUS DEFENSE & SPACE

The main authors and moderator hereby thanks all these contributors, as well as the many others.

INTRODUCTION

Purpose

The purpose of this document is to provide industrial feedback and challenges on the way Model-Based Systems Engineering can be used to define system requirements and system architecture with traceability to system requirements.
Document lists different modeling strategies and associated benefits and efforts ranging from the use of models for illustration up to the use of models for specification with full support of traceability within models.

Scope

As stated by ISO/IEC/IEEE 24765:2010 (System and software engineering –- Vocabulary), 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, ISO/IEEE/IEC 15288:2015(E) standard defines system lifecycle processes amongst which technical processes that deal with system requirements and architecture definition:

- Stakeholders Needs and Requirements definition
- System Requirements definition
- Architecture definition
- Design definition
- System analysis

Note: appendix 5.1 recalls purpose of each of those processes as stated in ISO/IEC/IEEE 15288:2015.

This paper concerns use of models in industry in the scope of technical processes presented above 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.
Main discussion concerns the gradual use of modeling activity to support requirement engineering and architecture definition.

Document organization

Chapter 2 provides some useful definitions about system engineering concerning concepts that will be widely used in the whole document. It will allow alignment of readers on vocabulary if needed.

Chapter 3 is the core chapter of this white paper: it gives a set of industrial feedbacks 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.

Chapter 4 is a set of common agreements shared between all contributors of this white paper regarding models to support requirement engineering and architecture definition.

Chapter 5 suggests a conclusion and provides some references on external documents available on the web.

Chapter 6 is a set of appendices that give details on some elements discussed in this document.

TABLE OF CONTENTS

TABLES OF ILLUSTRATIONS

1 USEFUL 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.

MBSE (Model-Based Systems Engineering)
Please refer to “MBSE introduction” presentation available here: <link to provide>. It provides all related definitions and explains potential benefits.

System Architecture
As stated in ISO/IEC/IEEE 42010:2011, 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”.

The architecture can be represented according to several viewpoints and at several abstraction or granularity levels. Viewpoints commonly used are the following:

- The **operational architecture viewpoint**, that describes the system organization in terms of black box functions or use cases and associated operational scenarios. It represents services, sub-services and interactions between themselves provided in response to desired capabilities from stakeholders in operational environment/context.
- The **logical architecture viewpoint** of a system is composed of a set of related technical concepts and principles that support the logical operation of the system. It includes a functional architecture viewpoint, a behavioral architecture viewpoint, a temporal architecture viewpoint and a partition of functions into system components (building blocks) excluding implementation or technological issues [completed from SeBoK [2] V1.4]
    - The **functional architecture viewpoint** describes the inside system organization in terms of “white box” functions, their interfaces (inputs/outputs) and their decomposition in sub-functions down to a level where function transformation can be entirely allocated to a building block. Generally decomposition shows distribution of data/energy flows from inputs of parent function to its outputs and between sub functions.
    - The **behavioral architecture viewpoint** describes system functions behavior in terms of both data/energy flows and control flows. Control flows describe execution logics (sequence, decision, synchronization…) of functions and conditions of activation/deactivation of functions (events, end of another function…).
    - The **temporal architecture viewpoint** describes the frequency of execution of functions and defines the synchronous or asynchronous aspects of functions (created from SeBoK).
- The **physical architecture** viewpoint describes the arrangement of physical elements (system elements and physical interfaces), which provides the design solution of system of interest, and is intended to satisfy logical architecture elements and system requirements. It is implementable through technologies

These viewpoints are intended to organize and structure the system specification, not to describe implementation discipline solutions, such as mechanics, software or hardware.

Requirement (source: ISO 29148:2011)
A requirement is a statement that translates or expresses a need and its associated constraints and conditions.

Constraint vs. functional requirement
Requirements can be classified into several categories and there exist several classifications. We can mention standard ISO/IEC/IEEE 29148:2011 "Requirement Engineering" section 9.4.2.3 (ISO 2011) for a suggestion (see appendix 7.4) 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:

- Functional requirements defining what the system shall do,
- Non-functional requirements and constraints, referring to qualities the system shall have.

A constraint is a limitation or implied requirement that constrains the design solution or implementation of the systems engineering process and is not changeable by the enterprise. [IEEE 1220-2005 IEEE Standard for the Application and Management of the Systems Engineering Process.3.1.5]

Decomposition and allocation
The “System life cycle processes” as described in the ISO/IEC 15288 standard 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
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 the following:

1 A unique identifier for each item under traceability,
2 Relationships to link items, with a well-defined semantics (e.g. “refine”, “derive”, “verify”,...).

2 SYSTEMS ENGINEERING APPROACHES: FROM REQUIREMENTS EXPRESSED IN NATURAL LANGUAGE TO REQUIREMENTS FORMALIZED WITH MODELS

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 way of working before trying to change their interfaces with external stakeholders.

Note: models can be used for different purposes and this is very important to clearly define the goal according to the context (stakeholders culture):

- Use of models for direct formalization of requirements: in that case, models will be used for validation and they shall reflect stakeholders needs through simple concepts easy to learn like “messages” exchanged in system of interest and its operational context (sequence diagrams)
- Use of models as intermediate means in order to support maturation of requirements that will remain expressed in natural language. For instance, there can be simulation on models used to evaluate some ranges of values for specific performance properties. Such models can even be used as “rationale” of requirements and avoid multiplication of “useless” requirements.
- Use of models as pivotal format toward a “user friendly” representation of requirements: 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, goals remain the same: ensure good requirement engineering.
Next chapter recalls good practices and key properties that requirements shall fulfill (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.

2.1 Requirement engineering goals: Correctness, Consistency, Completeness

Requirements Based Engineering (RBE) consist 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 (see figure below).
Traceability is established:

- During system definition, from System Design Requirements (= requirements identified during the System Design stage), System Requirements and Stakeholders Requirements
- During integration, verification, validation level: from requirements to integration, verification and validation plans.

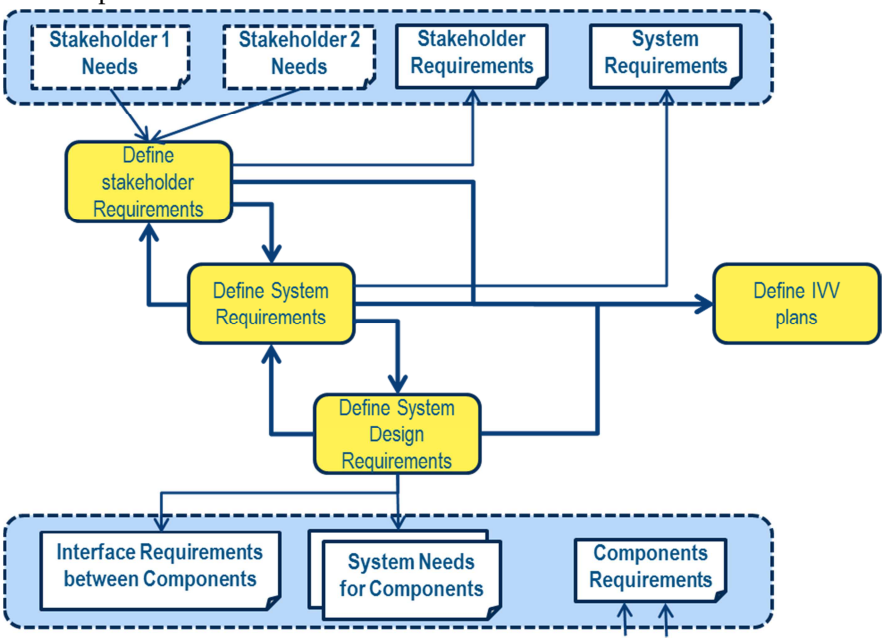


Figure 1: RBE process, adapted from [1]

Several goals are expected

- Goal 1: Obtain well-formed requirements: SMART (Specific, Measurable, Achievable, Realistic, Traceable)
- Goal 2: Ensure completeness and consistency of stakeholders Needs
- Goal 3: Ensure completeness and consistency of the System design against the stakeholders Needs

2.1.1 Goal 1 - Correctness: Obtain well-formed requirements (SMART)

Requirements statements should have the following characteristics (based on ISO/IEC 29148 – Source INCOSE Guide to writing Requirements, INCOSE SE Handbook), detailed in annexes (5.2Requirement correctness: Obtain well-formed requirements (SMART))

- Necessary
- Implementation Independent
- Unambiguous
- Complete
- Singular
- Feasible
- Verifiable

[1] *Bonnes pratiques en Ingénierie des Exigences, collection AFIS, Éditions CÉPADUÈS, ISBN 978.2.36493.020.9*

- Correct
- Traceable

2.1.2 Goal 2: Ensure completeness and consistency

A set of stakeholders' requirements should have the following characteristics to ensure that the needs and constraints are complete and consistent. This includes:

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

2.1.3 Goal 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 ISO/IEC 29148 – Source INCOSE Guide to writing Requirements).

- Complete:
  - Address complete set of expectations and Constraints
- 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.
  - Are aligned and consistent with architecture elements
- Feasible (affordable).
  - Can be satisfied by a solution that is obtainable/feasible within life cycle constraints (e.g., cost, schedule, technical, legal, regulatory).
- Bounded.
  - Maintains the identified scope for the intended solution without increasing beyond what is needed to satisfy user needs

2.1.4 Attributes of Requirements statements

In addition to the characteristics listed in previous chapters, 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 table below gives a list of possible attributes.

Typical scenarios based on Requirements analysis are (non- exhaustive list):

- Requirements Analysis: Identifier, maturity, version, risk, V&V method
- Call for proposal: flexibility and priority of Requirements
- Suppliers responses analysis: degree of compliance
- Justification of design choices

We find generally the following attributes (detailed in annexes - 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 ant that most of defined attributes are never filled or with a lot of mistakes. So there are two simple suggestions for improvement:

- Assess usage and usefulness of each attribute on each project in order to be sure that attributes will bring value.
- And 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.

2.2 Requirements defined in natural language and illustrated with diagrams

2.2.1 Description

This approach is the “traditional” approach found in almost all companies. Requirements are written in natural language and some diagrams are inserted in the specification document.
When diagrams or drawings are used, the goal is purely illustrative, and most of the time motivated by the need to provide visual support for requirements that are hard to explain and understand in their textual format. Those diagrams are used as an introduction to a category of functional requirements, to give the context.

Concerning edition of those diagrams, it exists a wide range of tools. Most used are Microsoft Visio and PowerPoint because they provide “easy to use” editors with large libraries of icons that can cover many symbols and then represent many concepts.
Those diagrams have limited semantics and it is not always obvious for readers to clearly understand their semantics without explanation. So, generally system designers provide semantics through a legend that explains what arrows and boxes mean.

Sometimes those drawings are done with standard modeling languages like OMG UML or SysML. In that case there is generally no need to provide a legend because the notation is standardized and thus considered as known by readers.

So the question becomes: in that case (requirements expressed in natural language and just illustrated with diagrams), shall we use MBSE tools with semantics or continue using drawing tools with “free semantics”?

2.2.2 Captured benefits of using diagrams

Benefits:
By using graphical modeling languages, diagram notation is less ambiguous than with drawings as it relies on a language that is semi or fully formal and notation is easier to follow if the modeling language is standardized.
But on the other hand those diagrams are not prescriptive (specifying). So there is little or no consequence if they remain ambiguous because there will be requirements to provide expected precision.

Drawbacks: with modeling languages, notation and tooling are more binding than “free” notation of drawings, what leads to more efforts (learn the language, complexity of the tool). And the advantage of getting a model to support consistency is not decisive in that context because those diagrams are not used to ensure consistency.

Conclusion:
Finally, in that context, using modeling languages has limited returns, as this approach requires efforts to produce diagrams that cannot give credit for specification. Benefits remain limited to high-level communication.

2.3 Usage of models to mature and verify requirements defined in natural language

2.3.1 Description

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.
Compared to the approach that uses models in a purely descriptive way, using a model and not only diagrams provides first level of consistency by construction (model is a kind of “database” with all

elements connected) and allows completing consistency through usage of model queries (for instance check that all system ports are connected to ports of subsystems).
Industry recognizes that use of models allows to mature requirements (reach finalized requirements) faster than with natural language approach and this approach is now more and more adopted.

With this approach, it is important that system designers follow a modeling method adjusted according to the engineering objectives and risks identified on the project concerning specification quality. If there is no modeling method, designers are left alone in front of the modeling language diagrams, concepts and tool capabilities, and they do not know which concepts to use at which stage and when to stop in their refinement through models. It leads to heterogeneous model hard to review and to maintain.

If model produced does not contain traceability links with system requirements, any change in system requirements cannot be traced to the model, and conversely architecture alternatives or modifications cannot be easily assessed with regards to applicable requirement baseline. Therefore, the contribution of models to impact analysis cannot be fully operational, and relies on manual analysis.
That is why there is a variant from this approach with addition of links between architecture model and system requirements. Benefits are more important as it becomes now possible to get impact analysis on model when upstream requirements change (by following links).

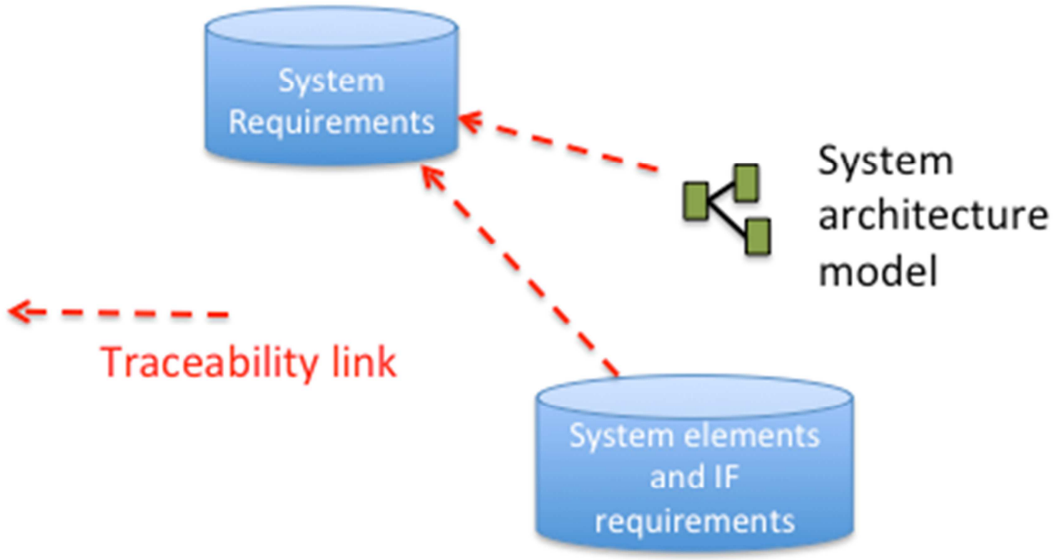


Figure 2: model is traced to system requirements

3 main usages of this approach are:

- Stakeholder requirements definition
- System requirements definition
- System architecture definition, mainly around Product Breakdown Structure (PBS) and verification of consistency between system element interfaces.

Next paragraphs provide some benefits captured in industry of using models in those 3 usages.

2.3.2 Benefits on using models for Stakeholders Requirements Definition

As mentioned in 5.1.1, the objective of the "Stakeholder Requirements Definition" process is to define the requirements in terms of precise expectations of users and other stakeholders concerning the future system, in the targeted operational environment.
The key steps of this phase are at least the following:

- Identify all the stakeholders involved,
- Identify the mission and scope of the system,
- Define the boundaries of the system and external actors involved,
- Describe interactions of the system with external actors,
- Identify and describe the operational use cases,
- Identify the user level operating modes,
- Formalize the related stakeholders requirements and link them to the operational use cases.

At this stage, all the analyses are performed from the system stakeholders external point of view, the system being considered as a black box. Main focus is put on understanding and trying to answer to most of stakeholders' expectations (avoid missing important expectations).

Globally, main captured benefits of using models can be summarized with the following points:

- Graphical and standard notation helps reflecting expectations in a concise and less ambiguous way: useful for both system team and stakeholders.
- Use of different types of diagrams helps separating concerns and looking at the problem from different view points
  - Useful as systematic approach: avoid missing important requirements
  - Useful as complexity breaker: each diagram deals with a limited set of concepts
- Diagrams help pointing out missing or unclear requirements by highlighting constructs not yet completed (port, message…)
  - Useful to provide local synthesis and local consistency
  - Useful to measure progress in capture
  - Useful to support reviews: graphical notes focusing to the point
- Model brings a centralized definition for requirements. For instance, if there exist different documents that define power supply characteristics, using a model will help building a unique definition in one place by aggregating all characteristics coming from the various documents. There can be several diagrams representing each displaying a subset of the properties, but the model itself contains the whole definition.
- Modeling approach brings guidance in the sequence of engineering activities and in the level of details through use of different diagrams at different abstraction levels
  - Useful to know where to start and when to stop

At a more practical viewpoint, here are some following modeling techniques that have proved to be useful in industry in order to understand stakeholder expectations:

- Context diagrams: With block descriptions, those diagrams represent the direct environment of the system and give initial information about the system boundaries and the interactions between the system and external systems and users. It is a synthetic view of environment/context that can be used as support for discussion with stakeholders.

- Context use cases and scenarios: Use cases represent the main services expected by the system users (people or other systems). They are detailed with scenarios that precisely describe sequences of interactions between system and its environment and show when the main events and actions occur. Those sequences can be easily discussed with end users because they translate operational scenarios in a simple representation with a small set of concepts easy to teach: lifeline (for each system and actor) and messages exchanged between lifelines during time for the given scenario.
- User level state machine: A state machine can be derived from scenarios by aggregating steps (messages flowing in and out of system) into *states* and transitions. It is an interesting way to factorize the expected behavior of the system into a unique representation. This diagram is less easy to explain to end-users because it aggregates several scenarios and is thus the result of some "consolidation" that was not mentioned by end users. But it is very useful for system team to prepare verifications about global consistency.
- Data (dictionary) model: Modeling is indirectly a way to capture a precise glossary of all useful concepts and associated relationships for a given system of interest. With blocks or classes, properties and relationships like "association", "composition" and "inheritance", a modeling domain provides immediate access to a vocabulary in a formal way and reduces ambiguities and interpretations. It can replace a very large number of requirements in a very synthetic approach, what is useful to limit documentation size and reading efforts.
- Traceability relationships: The modeling workbench generally provides built-in mechanisms to define traceability relationships between modeling artifacts and also external pieces of information, either by explicit links or by parenting or annotations. These traceability links are determinant for consistency and completeness analysis purpose. As mentioned previously, without traceability links, model remains a "one shot" means to mature requirements and becomes obsolete when requirements change. So traceability is a fundamental engineering activity if there is intent to maintain model after first baseline.

2.3.3 Benefits of using models for system requirements definition

At system level, focus is mainly put on consolidated definition and consistency: there is a switch from capture of a set of usages (operational scenarios) to the building of a centralized system definition.

Globally, a modeling approach is useful to build *a concise and consistent system definition* by construction (model constructs) and by method (aggregation of usages + links/allocations between model elements).
From this definition it is possible to derive system requirements and efforts to derive those requirements are generally reduced compared to the writing of requirements by derivation from stakeholder requirements.

At a practical view point, modeling technics presented in the previous chapter about stakeholder needs and requirements definition are still useful and can be reused to describe what the system shall do as a black box with more details. As an example (not exhaustive) there can be:

- External interface diagrams: they can be used with a logical or physical view:
    - With a logical view, those diagrams give details on interaction flows between the *actors (roles of external systems or humans in interaction with system of interest)* and the

system (always seen as a black box). State machine diagrams can complete them if the interaction follows some protocol. It is also possible to precise data types through blocks or classes in order to improve precision and consistency between all definitions; and then possible to get a detailed description of each system external interface

  - With a physical view, those diagrams show connectivity: the way connected systems exchange data or energy with the system of interest through its physical ports (sensor, network, actuator, power plug, discrete, button…). Here again, blocks can be used for instance to describe ports more precisely than just a name and a comment.

- System scenarios: scenarios identified in definition of stakeholder requirements can be refined to identify main services or functions the system shall perform. Those scenarios that are mainly focused on system behavior can be represented by flow diagrams like EFFBD or activity with data, energy or any item flowing and control flows. Here the main focus is put on consistency and an activity or EFFBD can show a lot of different scenarios on same diagram (thanks to control flow concepts with decision, fork, wait conditions on event…). Those diagrams are less easy to understand by end users but they ease verification of consistency and reviews by system team because they are more synthetic than a lot of sequence diagrams.
- System level state machine: A state machine can be derived from use case analyses by aggregating all the steps (*states*) and main transitions identified in the system scenarios. Comparatively to those established at operational level this state machine is refined from a designer point of view, capturing additional states and defining precisely how system level functions are triggered. The system state machine can become the central element of the system model, allowing the simulation of its behavior for validation purpose. Alternatively behavior can also be represented by flow diagrams like EFFBD or activity.

2.3.4 Benefits of using models to support breakdown activity

A first aspect of the system architecture definition concerns the definition and management of its parts (System elements) and associated interfaces. Models are largely used to support this breakdown activity. The system breakdown is often modeled in hierarchical trees with at least the associated Product Breakdown Structure (PBS) that uses block, block properties and block decomposition as main concepts.

During design, development and testing phases, the PBS is often linked to design teams' organization and very often associated to more functional concerns. There is generally an activity of functional breakdown to decompose functional requirements down to the level where sub functions can be clearly understood and allocated to the PBS.
Here again, blocks and block decomposition are used as main concepts, completed with block ports and connectors between ports to show functional flow (data or energy). Allocation is often done through tables that can be formalized in tools like Microsoft Excel or directly inside model when modeling language contains "table" concept or "allocation" relationships (SysML provides both).
Models can also be used to allocate required system performance over several requirements and/or architectural elements.

Using model makes it easier to get the global picture and to drive optimizations of functional interfaces or verification of consistent allocation of functions on PBS.

PBS can also be linked to the Work Breakdown Structure (WBS) and the associated workflow to follow development progress.

2.3.5 Benefits of using models to verify that parts can be connected consistently

Globally, at any stage of the development life cycle, system's parts definition must remain consistent to ensure that those parts can be integrated when implemented. This supposes that all system's parts interfaces are well defined, that they can be assembled (compliance of data/energy/else provided and consumed) and that interaction protocols are clearly depicted and managed.
Modeling of interfaces (structure and behavior) brings a lot of support to verify those properties.

Firstly, the availability of modeling syntax and semantics helps system designer in building consistent models by construction: a block diagram will allow connecting blocks through connectors and eventually ports (interaction points), which is a recognized interface pattern. Tool will forbid connecting two blocks without ports or will allow creating rules to check those cases. Without model, there is no support for such verification (Visio or PowerPoint will never warn system designer that a box is connected directly to another box without intermediate smaller boxes).

Secondly, because of quite clear semantics (formal or semi-formal modeling language), modeling strongly reduces ambiguities. If there is different possible ways to formalize one interface it means that interface requirements are not completely defined and there shall be another iteration concerning definition. So modeling forces system designers raising issues about ambiguities or missing details and misunderstanding or insufficiently detailed definition can be detected early. Associated with a clear interfaces' definition process, modeling approach can very efficiently support consistency verification and, for example, allow automatic checking of sets of pre-defined rules.

Last, modeling helps in reuse. Generally, interfaces are described in tabular forms, based on clear templates that details sets of properties to be defined at each step of the design process: functional data (ranges, units, meaning, description, …) in first stages, and at the end all information linked to the platform and networks formatting (label formatting on buses, communication protocol used, discrete types or wiring data). Such description of system interfaces can be capitalized and reused for other purposes (for example, it can strongly ease management and integration of behavioral models in complex simulation platforms) or other programs.

2.3.6 Benefits of using models in other related processes

In addition to the support of previous engineering activities, model can also help in:

- Re-use & change management: thanks to traceability, impact identification and modification cascading in design definition is easier to perform with models than with documents because links are more direct.

- Continuity from functional analysis to product architecture.
    - Improves traceability between customer expectations and system design.
    - V&V activities: identification of functions to be tested toward components under test.

- Safety analysis:

2.3.7 Change in efforts and challenges

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.

Note that approach requires maintaining links between requirements and the system architecture model in addition of “traditional” links between system requirements and system elements requirements. Those efforts are necessary to maintain model when requirements change.

It is important to notice a challenge concerning traceability of model with system requirements: they 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 in models put back in the requirement database?

2.4 Usage of models to formalize architecture definition and system specification

2.4.1 Description

With this approach, the goal is to support system architecture definition from system requirements down to system element requirements with full traceability done through models.

In terms of system engineering activities, it means that system requirements are formalized and decomposed/refined through a model and associated diagrams. Model contains system elements, their interfaces and associated requirements traced to system requirements.
As the goal is to cover all requirements, model is completed with textual requirements when modeling language or its extension does not allow easy formalization of some system requirements or some system element requirements. Those textual complements are traced to their source.

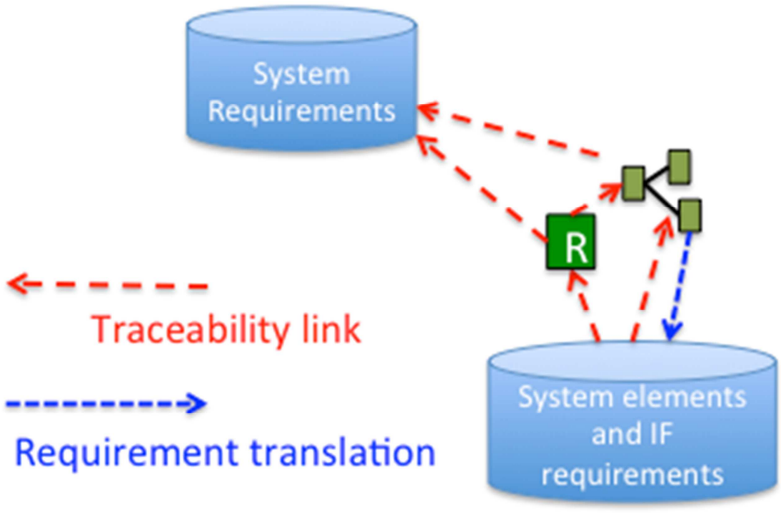


Figure 3: models are completed (extended, annotated) with textual requirements (R)

2.4.2 Industrial feedback

With this approach specification can be generated from model because model contains specification tree and traceability between system requirements and system element requirements, with their attributes (that will be used for generation).
But there is first the need to know precisely what part of the model is specification (not all model elements are requirements) and this is quite a big challenge. In fact, when formalizing requirements, there is generally introduction of “context” and “glue” elements around requirements and system designers are not aware of that. So sometimes formalization is a real translation, but most of time this is a refinement with addition of extra information. This is a very difficult point to address that can be considered as a “blocking” point for some projects.

Another point concerns the reverse transformation from modeling language to natural language. Indeed there are several situations when we want to provide a specification expressed in natural language to other teams (quality, sub-contractors…). In those cases there is the need to translate formalized requirements into natural language.

Note: it is always possible to use “comments” or “documentation” associated to the different model elements in order to add textual requirements and then retrieve those requirements through a script or a query applied to the model. But it would mean duplicating in text some information already available as a model element: tedious and error prone…

2.4.2.1 Benefits

Benefits are really high:

- Fast maturation of requirements,
- Impact analysis on requirement change eased thanks to use of models and navigation between diagrams (instead of navigating in large documents)
- Specification document can be generated (now that all data are in the model)

2.4.2.2 Challenges

But there are many challenges:

1. Ability to identify non textual requirements and traceability links in models
2. Translate requirements and their traceability links available in models to textual representation.

Next paragraph focuses on those challenges.

2.4.3 Challenges and questions about use of models for specification

This chapter explains what formalizing requirements through models means in practice.

Focus is put on 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 modeling tool shall allow supporting many activities concerning requirement engineering and management.

Whatever format of requirement (text, table, graphical…) and its formalization (natural language, modeling language...) requirement shall be engineered in a way that can be certified. Next paragraph recalls what requirement engineering good practices still have to be applied for formal requirements. Then chapter lists challenges and questions that arise when using models in the goal of building reference for specification.

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? it possible to ensure that those developed requirements will be at the right level?

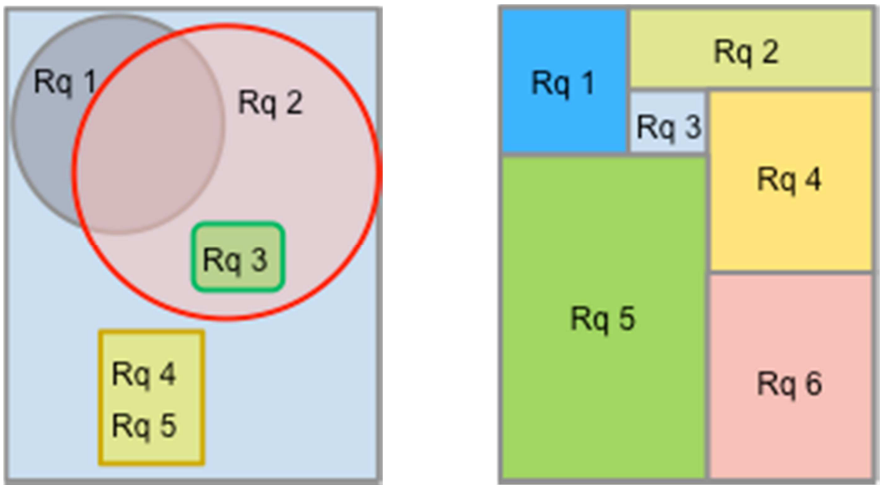


Figure 4: from overlap of requirements to a consistent set of requirements

2.4.3.1 Building complete and consistent logical architecture

When defining system level architecture, SeBoK 1.4 recommends starting with logical architecture expressed with 3 viewpoints (a lot of other viewpoints exist that will complete that first definition): functional architecture viewpoint, behavioral architecture viewpoint and temporal architecture viewpoint.

There exist a lot of modeling technics to support one or part of those viewpoints: Functional decomposition, FFBD, EFFBD, Integrated Definition for Functional Modeling (IDEF), N2 charts, operational scenarios… But challenge is here to ensure that all used techniques can be applied on same model (else we will have to deal with model transformations…) and can be combined consistently (without overlap).

Formalization of functions and scenarios of functions

SeBok suggests starting by analyzing functional, operational and interface requirements in order to deduce system level functions and their external interfaces in terms of data/energy/… inputs and outputs.

Some modeling languages provide functional blocks as native concept but others do not. What concept shall be used with SysML?
Globally we find three main options in industry:

- Blocks

- Activities
- Operations

Blocks seem natural candidates because they support input and output ports and can be decomposed. But what about control flow between functions? How to express function activation? How to define a scenario of functions as recommended by SeBok 1.4? For those concerns, “Activity” concept is better suited. Operations fit well with the notion of function (they have input and output parameters) and their behavior can be specified through the “behavior” attribute that can reference an activity element and associated activity diagram. But the decomposition of functions is harder to achieve with operations than with blocks and activities because an operation does not contain operations. So mapping is a little bit less natural than with activities.

Traceability links of functions with system requirements:

In order to reach completeness there is a need to ensure that all functional requirements have been analyzed and taken into account in logical architecture. Traceability links are used for that purpose.
OMG SysML provides different semantics for traceability relationships in addition to « trace » (that has poor semantics) and suggests using specialized relationships like « Refine », « satisfy » or « verify » for test.
When functions are just a representation of functional requirements, what kind of traceability semantics shall we use for that purpose? Shall we use « Refine », “Satisfy”, else?
“Satisfy” seems to be adapted for solutions and in that case functions represent the problem space. So “Refine” seems a better approach for this situation
But functions can also represent design decisions (choices between alternate solutions): in that case, « satisfy » seems to be the good semantics, and it is recommended to provide rationale for that.

Linking functional and behavioral elements with temporal architecture

Good SE practices (including SE Handbook) 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 view (state machine, state flow, Petri net…) are different techniques that can support those concepts.
Challenge is here to links functions with phases, modes and states.
If different modeling languages are used with different semantics, it will become hard to define links with clear semantics between functions and modes/states.

Now let us take the assumption that there is same modeling language that can cover both functions and states concepts. What would be the link?
Here there are mainly three ways of creating this link:

- Reference functions in states through “doActivity”, “onEntry”, “onExit” or any “onEvent”. This applies for functions represented by activities.
- Use allocation relationships between functions and states. This applies for any representation of functions (activities, operations or blocks).
- Addition of assertions (pre conditions, post conditions) in function behavior to mention modes/states.

2.4.3.2 Trade-offs concerning system architectures

Is it possible to formalize different architectures in 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. Can even use inheritance for 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. There is a dedicated white paper on MBSE applied to product line engineering.

2.4.3.3 Creation of new requirements in the model

Function decomposition brings new requirements associated to sub functions. 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 this means that all new requirements are exported and put back in the requirement database. Requirement hierarchy and Function hierarchy are strongly related. This is the "zig zag pattern" presented by Tim Weilkins in sysmod approach: http://model-based-systems-engineering.com/2012/03/26/the-sysmod-zigzag-pattern/:
- 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.

Addition of metadata on requirements in models

If requirements are kept in model, do the modeling language and the modeling tool provide capabilities to add all requirement attributes coming from RBE good practices and presented in previous paragraphs? Is it possible to apply cascading of attribute values along requirement hierarchies to ensure some efficiency? to consider that hierarchies provide implicit "reference" on parent requirement attributes?

Is it possible to identify requirements formalized in model? to tag them or to find rules to detect them?

Is it possible to extract identified requirements in models? What is the minimum of needed to extract around model element to get its definition? is it possible to automatically translate in textual format a requirement expressed as model element?

Is it possible to extract a diagram to represent a requirement instead of model elements? How to ensure verification and validation of a diagram? Does it make sense to define tests for a diagram? What does it mean if a diagram is a container of graphical elements linked to semantic elements?

Appendix 5.7 provides some answers concerning identification of requirements in models through results of advanced research.

3 COMMON AGREEMENTS ON MODELING AND MODELS

3.1 Efficient modeling requires goal and strategy

The document showed that modeling can be used in different gradual approaches and that benefits are different according to the way modeling is used. For instance, in chapter 3.2 there was a case when models are used as an illustration of some textual requirements. In that situation, requirement engineering activity was not modified and modeling is just used as a complement added to the reading of textual requirements. It is important to get in mind that benefit will be low in that situation and that it is not worth investing a lot in elaborated modeling languages and tools. More, it forces to maintain consistency between requirements and models which can be seen as extra work (especially when the same statements are expressed simultaneously through textual descriptions and diagrams)
If a company uses model without clear objectives about the engineering activities it will support, there is great chance that modeling activities end with frustration: frustration on returns that will remain low if modeling activities did not replace or reduce some engineering activities; frustration also on modeling use with low recognition from management or rest of the team if there was a lot of efforts done with modeling and no clear benefits to align on this investment.

When the goal is clearly defined, aligned on reducing efforts on some engineering activities, this is important to define a strategy to reach this goal. Practically it means defining a modeling method that addresses 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 to organize model, ease review and collaborative editing

Try to see in your organization which engineering activities can be improved and if modeling might help: then try to identify some savings and setup a modeling method in order to reach that target savings.

3.2 Which modeling strategy? Short-term or long-term?

There is a big question that should be addressed very early in the project when a company intends to use models: “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 we want to use model for long term we will have to maintain it and as for any reference-engineering artifact, it is important to put model in configuration (baseline) and to ensure traceability with upstream requirements.

Why is traceability so important for models? We saw in chapter 2.3.1 that if the model produced is not traced with upstream requirements (stakeholder requirements for operational model, system requirements for system architecture model), this model will be useful only for current baseline of requirements. As soon as there are new or changed requirements, it becomes very hard to detect changes on the model, and at some point in time model becomes 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.

3.3 Using model to formalize a large part of specification is a big investment

We saw in chapter 2.4 that this strategy is not easy because it requires that model becomes fully consistent, complete and at right level of details and can be traced upward to system requirements and stakeholder requirements, backward to system element requirements.
This is not so complex when dealing only with a few requirements like interface requirements, 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 and method that can be used as-is for any project. Some challenges are still in "research" status (identify requirements from model, ensure traceability through model elements) with limited methodology and tooling support.

Targeting formalization of whole system specification with model is a strategy that requires a large investment on knowledge in the modeling language and its 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 got through accelerated maturation of system definition and design[2], and all the analysis efforts and time saved during maintenance of the system thanks to the easy navigation through models.

3.4 One master repository for specification at each refinement level

We recommend identifying one and only one master repository for specification at each level of refinement. It can be for instance "PowerPoint" documents at marketing level and then "DOORS" database at system level and finally "UML models" at software level.

This possible use of different master repositories across levels helps adapting tooling with regards to the skills and culture of people in charge of defining specifications at this level. Configuration management will ensure consistency across refinement levels: the baseline will provide links between data from the different repositories used as reference for each refinement level.
Note: using different repositories for different levels allows adapting to different teams (skills, culture) with more flexibility, what is very important for MBSE deployment, but might generate complexity in tooling in order to create the right links between dispatched data for the baseline.

But concerning one refinement level there shall be a unique master repository for specification. Experience shows that if there are two repositories at 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

[2] Model benefits are not all mentioned there but are detailed in « learning MBSE » published by INCOSE through AFIS (french chapter).

modifications done concurrently in both repositories, root cause of errors, or one repository will not be maintained.
This is particularly true for model and document. If document remains 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 to do same modifications on two repositories: document and model.

So if there were a lot of efforts to build a model that contains most 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 (generated) from the model.

3.5 Align SE practices before deploying MBSE

As already expressed in *3.1 Efficient modeling requires goal and strategy*, there is no unique strategy to adopt Model Based Systems Engineering but 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.

Indeed, when trying to use models we sometimes discover that we do not understand same concepts behind same model elements. Everybody will do the mapping with his/her 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 amongst team members.

What about training? That is one good solution to align systems engineers on same definition for model elements, and same mapping to systems engineering “standard” vocabulary. 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. Or training might take this situation into account and teach both systems engineering and use of models to support it. In that case, 5 days are a minimum and would rather be called “introduction to “MBSE” …

4 CONCLUSION

Current difficulties
Requirement Based Engineering through documents has reached limits on complex and critical systems that are developed today: writing, reviewing and maintaining documents of 500 to 1000 pages is no more a (satisfying) option because it is hard to find errors, ambiguities and inconsistencies.
It is now necessary to have more formal and structural way to organize the requirements and to link them with the design concepts.

MBSE brings solution
The models provides a way to insure consistency between the different objects (requirements, architecture, …) and MBSE provides a way to verify the quality and consistency of the models (using static controls and using simulation).
MBSE helps analyzing and validating concepts early in the development cycle and during maintenance and MBSE helps detecting issues with requirements. So MBSE can be considered as a good means to accelerate maturity of requirements

Road Blocks to apply
There is no unique strategy to adopt Model Based Systems Engineering: you can decide between short and long term, but have to know the consequences. If long term strategy is chosen, models will become reference engineering artifacts and shall be managed in configuration and traced to requirements.
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.
Efficient use of models has impacts on configuration management, on organization, and there is still a lot to do in that area, especially in tooling support and connections between teams and disciplines.

INCOSE vision
Finally, have a look on *2020-2025 vision*[3] compiled from INCOSE and you will have confirmation that MBSE is not seen as a small trend but rather as a core knowledge to enable the development of future systems that will continue growing in complexity.

[3] http://www.incose.org/AboutSE/sevision

5 APPENDIX

5.1 ISO 15288:2015 technical processes considered in the paper

5.1.1 Stakeholder Needs and Requirements definition

> The purpose of the Stakeholder Needs and Requirements Definition process is to define the stakeholder requirements for a system that can provide the capabilities needed by users and other stakeholders in a defined environment.
>
> It identifies stakeholders, or stakeholder classes, involved with the system throughout its life cycle, and their needs. It analyzes and transforms these needs into a common set of stakeholder requirements that express the intended interaction the system will have with its operational environment and that are the reference against which each resulting operational capability is validated. The stakeholder requirements are defined considering the context of the system-of-interest with the interoperating systems and enabling systems.

Note: Requirement definition is generally performed through some formalization according to requirement writing best practices, compiled and shared at INCOSE website:
http://www.incose.org/ProductsPublications/techpublications/GuideRequirements
When there are conflicts or feasibility issues, modifications are negotiated.

Note: many stakeholders' requirements (in particular regarding the licensing, operation and maintenance of the system) will be refined once downstream design choices are done or after external events occur (such as incidents or accidents that occurred in similar systems, or regulatory changes) and cannot be fully elicited upfront. The third purpose of the process is to ensure that the stakeholders are informed as necessary all along the development process, which could lead to new baselines (new or modified stakeholder requirements).

5.1.2 System Requirements definition

> The purpose of the System Requirements Definition process is to transform the stakeholder, user-oriented view of desired capabilities into a technical view of a solution that meets the operational needs of the user.
>
> This process creates a set of measurable system requirements that specify, from the supplier's perspective, what characteristics, attributes, and functional and performance requirements the system is to possess, in order to satisfy stakeholder requirements. As far as constraints permit, the requirements should not imply any specific implementation.

Requirements and architecture within modelling context - page 31

5.1.3 Architecture definition

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.

Iteration of the Architecture Definition process with the Business or Mission Analysis process, System Requirements Definition process, Design Definition process, and Stakeholder Needs and Requirements Definition process is often employed so that there is a negotiated understanding of the problem to be solved and a satisfactory solution is identified. The results of the Architecture Definition process are widely used across the life cycle processes. Architecture definition may be applied at many levels of abstraction, highlighting the relevant detail that is necessary for the decisions at that level.

Note: for details about architecture description please consider reading ISO/IEC/IEEE 42010:2011 - *Systems and software engineering standard - Architecture description.*

5.1.4 Design definition

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.

5.1.5 System analysis

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.

The System Analysis process applies to the development of inputs needed for any technical assessment. It can provide confidence in the utility 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. It includes mathematical analysis, modeling, simulation, experimentation, and other techniques to analyze technical performance, system behavior, feasibility, affordability, critical quality characteristics, technical risks, life cycle costs, and to perform sensitivity analysis of the potential range of values for parameters across all life cycle stages. It is used for a wide range of analytical needs concerning operational concepts, determination of requirement values, resolution of requirements conflicts, assessment of alternative architectures or system elements, and evaluation of engineering strategies (integration, verification, validation, and maintenance). Formality and rigor of the analysis will depend on the criticality of the information need or work product supported, the amount of information/data available, the size of the project, and the schedule for the results.

5.2 Requirement correctness: Obtain well-formed requirements (SMART)

- 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

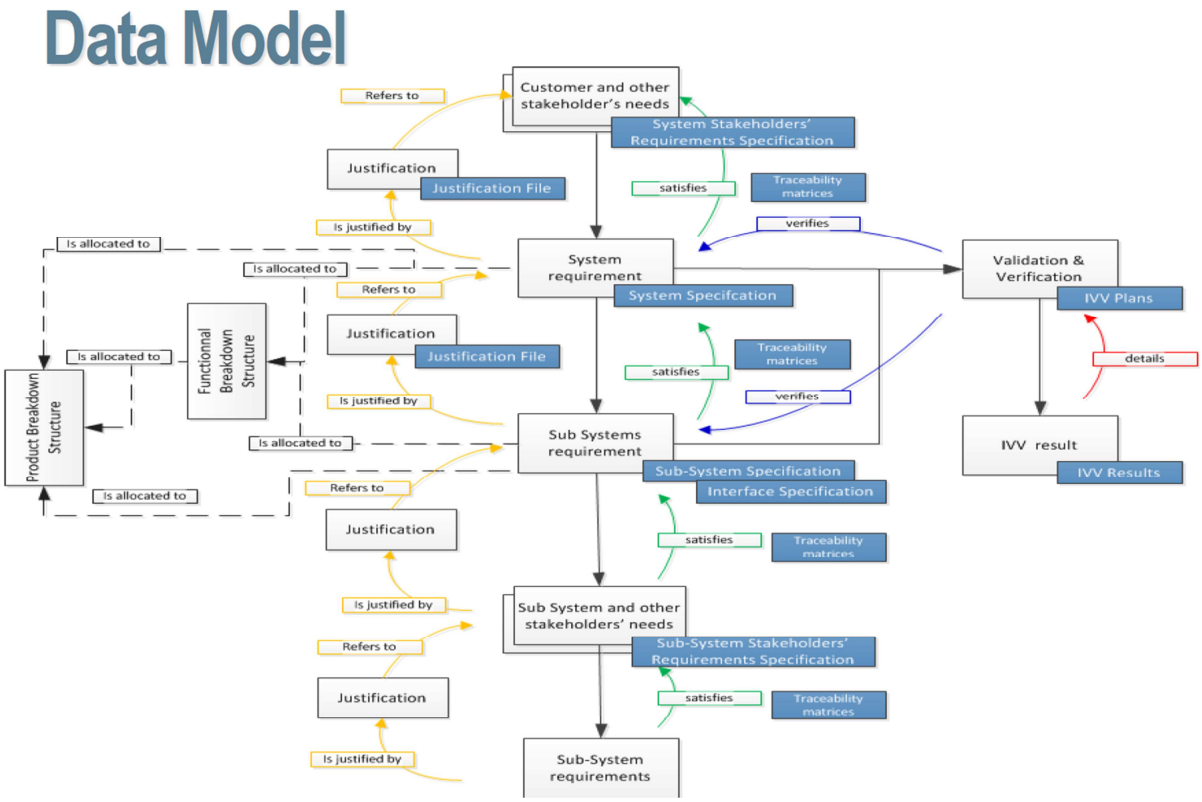
- Implementation Independent
    - Address what is necessary and sufficient in the system
    - Avoids placing unnecessary constraints on the architectural design.
    - States what is required, not how the requirement should be met.
- Unambiguous.
    - Stated in such a way so that it can be interpreted 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.

5.3 Good practices about requirement attributes

Attribute	Typical Scenarios	Description
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 (Ex: Draft, Verified, Accepted, Rejected, etc.)
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 (ex: final users, maintenance team, etc.) But also Key Design Drivers

Negotiability, Flexibility	Qualify requirements	Assessment of the possibilities to negotiate the requirement. (ex: 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 (ex: 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	Allows identifying Verification and Validation methods (ex: Inspection / Analysis / Demonstration / Test) when elaborating the requirement. Allows to agree with stakeholders on V&V methods Can be separated in Verification, Validation
Compliance	Traceability	Allows evaluating an answer to a call for tender: verification of the answer's 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

5.4 Example of traceability process data model



5.5 Suggestion of classification for requirement types

ISO/IEC/IEEE 29148:2011 "Requirement Engineering" section 9.4.2.3 (ISO 2011) suggests one possible classification with the following table that lists categories and associated semantics.

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.

5.6 Examples of system elements and physical interfaces

Extract from INCOSE SE Handbook V4.0:

TABLE 4.1 Examples of system elements and physical interfaces

Element	Product system	Service system	Enterprise system
System element	Hardware parts (mechanics, electronics, electrical, plastic, chemical, etc.)	Processes, databases, procedures, etc.	Corporate, direction, division, department, project, technical team, leader, etc.
	Operator roles	Operator roles	IT components
	Software pieces	Software applications	
Physical interface	Hardware parts, protocols, procedures, etc.	Protocols, documents, etc.	Protocols, procedures, documents, etc.

5.7 Two examples of advanced research to identify requirement in models

5.7.1 PBR theory (written by its author: P. Micouin)

Between 2006 and 2008, Patrice Micouin has developed a PBR theory[4], which is now fully integrated in a complete Model Based Systems Engineering method called PMM[5] (PMM stands for Property Model

[4] P.Micouin, Toward a property based requirements theory: System requirements structured as a semilattice, Systems Engineering, vol. XI n°3, p235-245, 2008

[5] P. Micouin: Model Based Systems Engineering: Fundamentals and Methods, Wiley & ISTE (2014).

Methodology). The PBR theory is based on the theory of properties[6] due to the Canadian epistemologist Mario A. Bunge.

PBR definition: the basic form of a PBR is as follows:

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

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.

Assumption definition: Assumptions are specific PBRs, limited to input properties since they are outside the system's developer control and only presumed.

PBR conjunction: relying on Bunge's properties algebra, PBRs can be combined thanks to conjunction operator "∧" in order to build composite PBRs.

PBR comparison: More, a partial order relationship "≤" allows comparing two PBRs.

Thus, the expression "PBR1 ∧ PBR2" is the conjunction of PBR1 and PBR2, and is itself a PBR. Moreover, the statement "PBR1 ≤ PBR2" means PBR1 is less constraining than PBR2.

Specification model

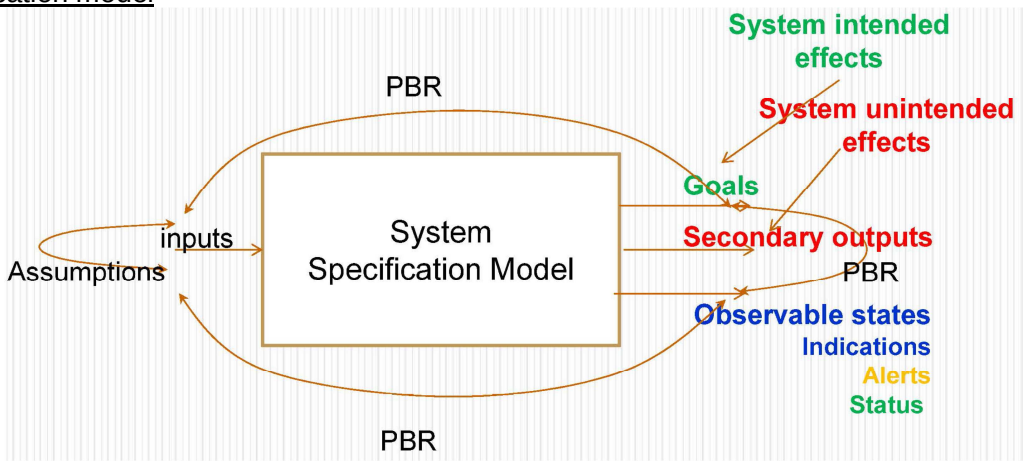


Figure 5: specification model in PBR theory

[6] M. Bunge, Treatise on Basic Philosophy, vol 3, Ontology I: The furniture of the World, D. Reidel Publishing Compagny (1977).

According to PMM, a system specification model is a formal model of a represented system that includes

(1) system requirements (PBRs),
(2) system interface requirements (inputs and outputs definition) and
(3) system assumptions (PBRs).

System specification process: with PMM, it starts with the definition of goals, that is, the intended effects that are identified and modelled as outputs of the system specification model. PMM proposes a goal-oriented requirements development approach. The next step is to elicit the occurrence conditions of the identified goals. The process of elicitation enables analysts to identify and model the expected inputs and additional outputs, such as observable states, secondary (undesired) outputs, secondary (undesired) inputs and system failures.

Then, we formalize the result of the elicitation as one or several PBRs. PBRs are predicates that link goals, secondary outputs, observable states and inputs in order to specify the actualization conditions of system properties.

System requirement validation: The translation of specification models into simulation models thanks to adequate languages is a solution to make sure that specification models and interfaces are consistent. In addition, simulation of a specification model linked with its corresponding equation design model provides analysts with various advantages.

First, it is an assistance to guarantee the completeness and correctness of the considered system specification model for a given set of validation scenarios.

Simulation also provides capabilities for validating building block specification models regarding its including system specification model.

Design and PBR derivation: For each candidate structural design model (PMM concept), the third system development activity consists in the derivation of the system requirements {PBRs} into building-block requirements $\{PBR_{1,}, PBR_n\}$. To be valid, for a given system structural design model (A) and for a set of assumptions on the environment (EA), the conjunction of derived building-blocks PBR_1, …, PBR_n must be more constraining than the system PBRs.

Derivation: when $A \wedge EA \Rightarrow PBR \leq PBR_1 \wedge ... \wedge PBR_n$

This validity condition of requirement derivation leads to the “prime contractor” theorem:

Prime contractor theorem: A sufficient condition for a system to comply with its PBRs is that its building blocks comply with the PBRs validly derived from the system PBRs, provided the design choices and assumptions made about the environment driving the derivation remain valid.

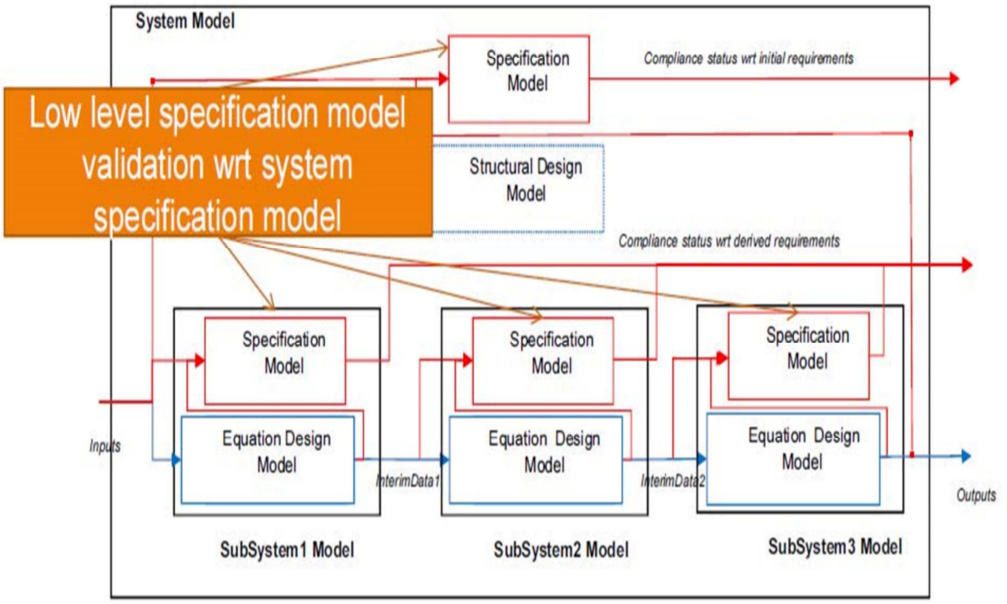


Figure 6: low level specification model validation wrt system specification model

Derivation validation: with PBRs and PMM, simulation is the main method to validate the derivation of system PBRs into a set of derived building block PBRs for validation scenarios.

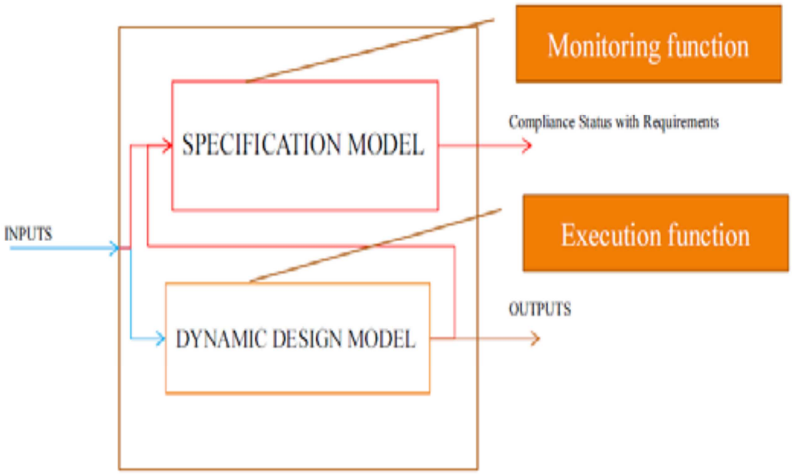


Figure 7: monitoring function and execution function

Design verification: Simulation also provides capabilities for verifying design models. While the simulation is running, for all submitted simulation scenarios, the specification models monitor the interacting design models so as to check whether any requirement is violated. If there is no violation that is detected during the verification of a building block design model, then the building block design model is verified.

5.7.2 Requirement Tracker Artefacts

Yves Bernard (Airbus) has extended PBR theory with the RTA (Requirement Tracker Artifact) approach[7]. 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 is possible to interpret any instance of those metaclasses as a requirement. Those elements (so-called "RTA") 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.

In his paper Yves BERNARD gives an example of application of this approach to the UML and give a list of UML metaclasses matching the PBR pattern[8]. Here is a sample UML model used he defined for illustration:

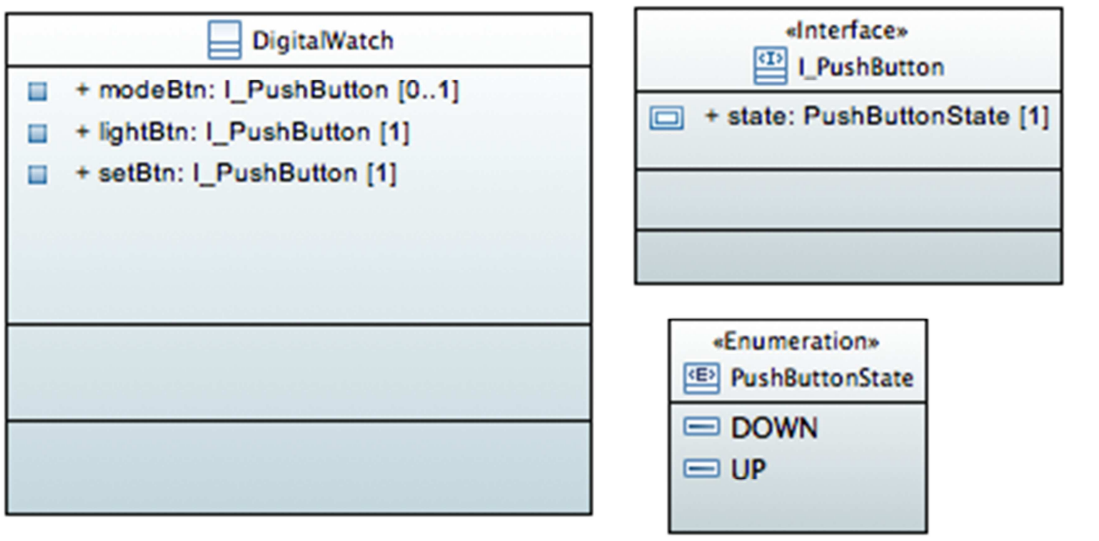


Figure 8: sample model - structure

[7] Yves Bernard: Requirements management within a full model-based engineering approach - nov 2011

[8] Note that, based on the development method used this list might require some adjustments

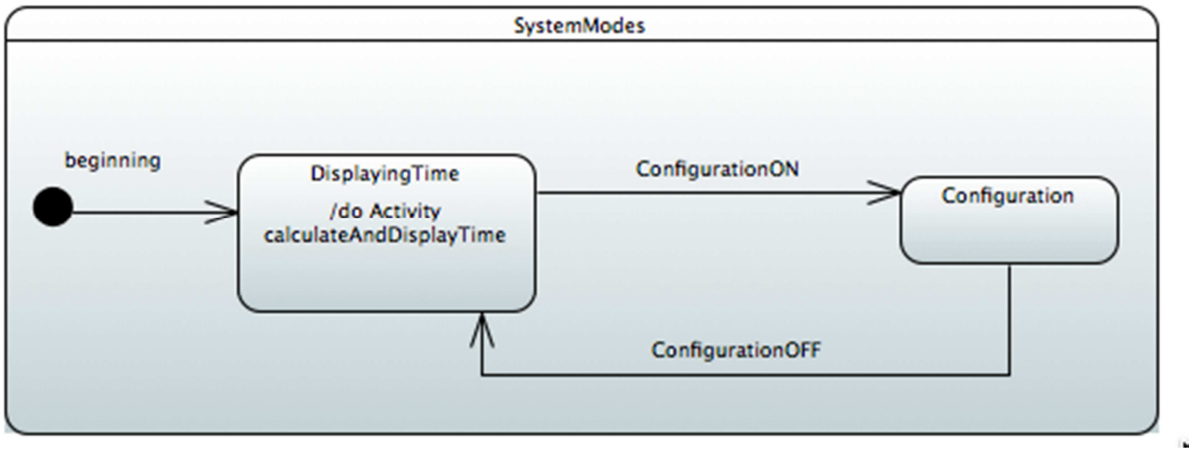


Figure 9: sample model - modes

Here is a possible translation into text of requirements identified directly from this sample model (Req ID is simply the model element name). This translation is based on a native UML interpretation but could be easily customized to be adapted to a specific domain:

Req ID	Req Text
Model_modelBtn	An item of type “DigitalWatch” shall have one optional “modelBtn” port(s) of the “I_PushButton” type
Model_setBtn	An item of type “DigitalWatch” shall have one and only one “setBtn” port(s) of the “I_PushButton” type
Model_lightBtn	An item of type “DigitalWatch” shall have one and only one “lightBtn” port(s) of the “I_PushButton” type
Model_SystemModes	By invocation, and item of type “DigitalWatch”
Model_MainRegion	The behavior described by the “SystemModes” state machine defines only one automation
Model_Transition0	If the “SystemModes” state machine is in “DisplayTime” state, it shall exit this state and enter the “Configuration” State on reception of event “ConfigurationON”
Model_Transition1	If the “SystemModes” state machine is in “Configuration” state, it can exit this state and enter the “DisplayTime ” State on reception of event “ConfigurationOFF”
Model_DisplayTime	The “SystemModes” state machine defines the “DisplayTime” sub-state(s).
Model_Configuration	The “SystemModes” state machine defines the “Configuration” sub-state(s).
Model_displayTime	By invocation, an item of type “calculateAndDisplayTime” shall behave as specified by its “displayTime” activity definition

Model_beginning	The “SystemModes” state machine starts in the “DisplayTime” (sub) state
Model_state	A provider conforming to the “I_PushButton” interface contract shall provide access to one and only one “state” reference(s) to item(s) of the “PushButtonState” type

With this approach, it becomes possible to use models as a specification and to preserve traceability analysis capabilities without creating manually textual requirements paraphrasing the model.

The main challenges of this approach are:

- To define the exact set of metaclasses that have to be used for RTA identification. This set is valid only in the context of the methodology on which it has been defined. If the methodology changes, this set could be impacted.
- To define the transformation able to generate human readable text equivalents for RTAs. Note however that this translation is not mandatory since RTAs already have the concrete notation (either graphical or not) used for defining them in the model.

www.afis.fr

ISBN 978-2-900969-00-7