Deliverable D2.1.a

Language specification v1

March 7, 2016

1 Introduction

The Formose ANR project (ANR-14-CE28-0009) aims to design a formally-grounded, model-based requirements engineering (RE) method for critical complex systems, supported by an open-source environment. The project has been launched on November 17, 2014. The main partners are: ClearSy, LACL, Institut Mines-Telecom, OpenFlexo, and THALES.

One of the main issues in the domain of RE for critical complex systems is to take into account the high complexity of such systems, the need of a better integration of RE with verification and validation techniques to ensure a better quality of requirements, and more generally the need of method guidance and tool support during the process of elaborating high quality requirements models.

The aim of Work Package 2.1 (WP2.1) is to elicit a set of concepts for RE and then to define its abstract syntax as a meta-model. All this work will be inspired by the case studies from WP1.1, as well as from the academic state of the art and from the state of practice in the technical fields known to the partners. Two deliverables are planned in WP2.1:

D2.1.a	Language specification v1	Document & software	T0+6
D2.1.b	Language specification v2	Document & software	T0+12

This document corresponds to the first deliverable: D2.1.a.

Our aim is to define a requirements modeling language integrating basic concepts of existing languages, such as KAOS or Tropos/*i**, and adding new ones to take into account the specific characteristics of critical complex systems: their abstract architecture will be considered by allowing requirements to be defined at different abstraction layers and verifying their consistency; the language will allow to specify not only non-functional requirements related to safety and performance but also specific requirements related to the presence of different operational modes and reconfigurations in such systems. The language will be multi-views (natural language, graphical notations, formal notations) to be understandable by all the stakeholders. For verification purpose, we will adopt existing and complementary formal methods, supported by efficient tools.

In this first deliverable, we focus on the state of the art. Section 2 is an introduction to requirements engineering. Then, in Section 3, the main formal concepts and languages that will be needed are introduced. Section 4 shows how requirements engineering is currently involved in formal methods. Sections 5 and 6 show the main practices of RE in industry. Finally, section 7 concludes the document with the main lessons learnt.

Based on this state of the art and on the case studies from WP1.1, Deliverable D2.1.b will focus on the definition of the new requirements engineering language.

2 Requirements Engineering

2.1 Aim and Objectives of Requirements Engineering

Many studies [HBR02, Sta95] have shown that one of the main reasons of failure in system development project comes from a weakness during the requirements engineering phase. On the website *Forum On Risks To The Public In Computers And Related Systems* [1], there are many examples illustrating poor requirements analysis (e.g. one of the most well-known is LAS (London Ambulance Service) [Fin96]). An average of 40% of projects fail or do not meet all the expected requirements because of a weakness in requirements engineering. In 1976, Bell and Tayer have pointed out that the non-conformance of system functions with respect to user requirements, the inconsistency, incompleteness, and ambiguities of requirements documentation, are mainly involved in the decrease of software quality [TB76]. These authors conclude that "the requirements for a system do not arise naturally; instead they need to be engineered and have continuing review and revision".

The requirements engineering community tries to improve this important stage in system development. It aims at defining methods, techniques, and tools in order to elicit, specify, negotiate, and validate the requirements of the system to build. Consequently, requirements engineering is an fundamental activity for developing systems of quality. It guarantees that the system to be built will meet user requirements. For [SS97], requirements engineering first consists in collecting requirements and allows them to be expressed. Each requirement should be correctly derived, allocated, followed, satisfied, verified, and justified.

[1] `http://catless.ncl.ac.uk/Risks`

2.2 Definition: Concept of Requirement

There are many definitions in the literature, the oldest comes from Ross et Schoman [RJ77] in 1977: *requirements definition is a careful assessment of the needs that a system is to fulfill. It must say why a system is needed, based on current and foreseen conditions, which may be internal operations or an external force. It must say what system features will serve and satisfy this context. And it must say how the system is to be constructed.*

Another definition is proposed by the norm IEEE 610.12-1990:

1. *A condition or capability needed by a user to solve a problem or achieve an objective;*
2. *A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents;*
3. *A documented representation of such a condition or capability.*

According to this definition, requirements define both the user needs and objectives and the system properties induced by the requirements and organizational constraints.

In the context of information systems, requirements engineering is defined by [Rol03] as an activity which transforms a fuzzy idea into a precise specification of stakeholder's needs, that shape the system to be built, and therefore, defines the intended link between a system and its environment.

2.3 Requirements Taxonomy

In the literature [Som07], there exist many kinds of requirements.

User Requirements.

Sommerville defines them as follows [Som04]. User requirements are statements expressed in natural language associated to diagrams, which describe the services that the system must provide, and the constraints under which it must perform. These requirements aim at capturing and at formalizing the needs of clients. The use of a natural language is described by Sommerville as a necessary condition for making the communication with stakeholders easier.

System Requirements.

A system requirement represents a system functionality or capability which allows it to solve or to fulfill an objective of the client. According to [Som04], such requirements enumerate the functionalities, services and operational constraints of the system. The system requirements document (also called functional specification document) must be precise, since it defines what will be implemented. It is included in the contract between the client and the developers. These requirements are derived from the user requirements, and represent an expression of the proposed solution. The IEEE guideline [IEE98] for the building of system requirement specification (SyRS) describes the characteristics required for this level: *A well formed requirement is a statement of system functionality (a capability) that can be validated, must be met or possessed by a system to solve a customer problem or to achieve a customer objective, and is qualified by measurable conditions and bounded by constraints.*

System requirements can be decomposed into two main categories: functional requirement and non-functional requirement. According to the IEEE glossary for software engineering terminology [JM90], a functional requirement specifies a function that the system or a system component must be able to execute. A non-functional requirement represents a constraint or an expected behavior which is applied on the system. Such a constraint can refer to the emerging properties of the software in development. According to Sommerville, non-functional requirements are constraints on the services and functions proposed by the system. They include temporal constraints, and constraints on the development process and standards. In general, non-functional requirement apply on the whole system, and not on particular functions or individual services.

Software (or Hardware) Requirements.

Software (or hardware) requirements are defined at the design stage. They are derived from the above-mentioned requirements.

2.4 Requirements Engineering Process

The different activities during requirements engineering are well defined by Sommerville et Sawyer [SS97]: *requirements engineering involves activities for discovering, documenting and maintaining a set of requirements for a system. Requirements engineering (RE) activities are often divided into five categories: requirement elicitation, requirement analysis, requirement specification, requirement validation and requirement management.*

Requirement Elicitation.

Requirement elicitation, which is the first stage of the process, is very important. It consists in collecting, discovering, capturing the requirements from different sources, and by involving all the stakeholders.

Requirement Analysis.

This activity consists in analyzing the requirements and in solving the possible conflicts, often by using negotiation.

Requirement Specification.

This stage consists in describing and structuring the requirements by using different kind of modelling languages (informal, semi-formal and formal).

Requirement Validation.

This activity aims at verifying the requirements quality in terms of consistency and completeness, as well as their adequacy with respect to the stakeholders wishes. In case of conflicts, a negotiation is initiated in order to reconciliate all the stakeholders [Bou14].

Requirement Management.

This activity consists in following the requirements evolution; it identifies, controls and keeps a trace of the subsequent modifications on requirements.

2.5 Requirements Engineering Approaches

Conventional Approaches.

In conventional approaches like SADT, MERISE, UML, etc., requirements engineering is integrated in the analysis phase, and produces, thanks to a modeling process, a conceptual schema which describes an abstract representation of data and transactions. The main objective is to describe what the system must do, i.e. its functionalities. As indicated in Fig. 1, requirements engineering based on conceptual modeling is focused on the question WHAT, to which we answer with a acquisition, modeling, validation cycle. For the acquisition of domain knowledge, the task mainly consists of stakeholder interviews and specifications documentation. This knowledge is then modeled in order to produce a conceptual schema. The latter is also used to validate requirements thanks to prototyping.

These approaches are limited, because they represent a unique solution, and therefore avoids reasoning between different alternatives. Consequently, user requirements are poorly taken into account.

For applying these approaches, we suppose that requirements are stable and will not evolve in the future. Today, because of the growing complexity of systems and the impact of new technologies, user requirements evolve faster, and new approaches must be applied.

Goal-oriented Approaches.

Conventional approaches *tend to focus on the WHAT rather than the WHY. Having answers to these WHY questions are important not only to help develop successful systems in the first instance, but also to facilitate the development of cooperations with other systems, as well as the ongoing evolution of these systems* [Yu97]. Goal-oriented approaches are focused on the WHY and WHAT questions. Fig. 2 illustrates such an approach.

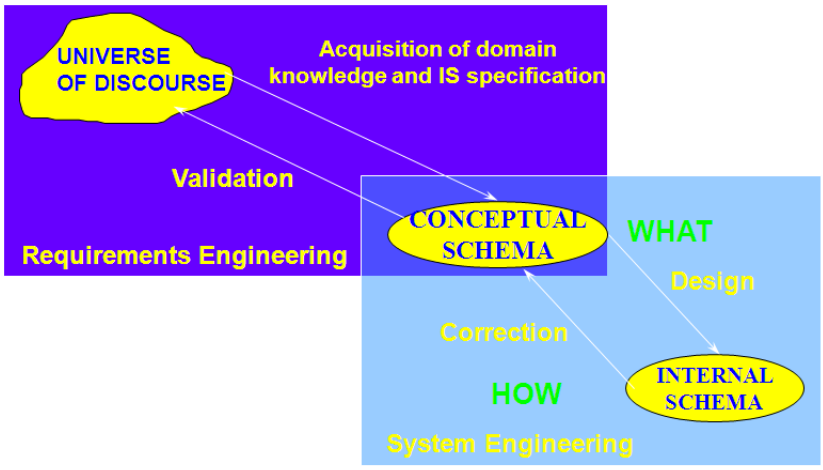

Figure 1: Requirements engineering in conventional approaches

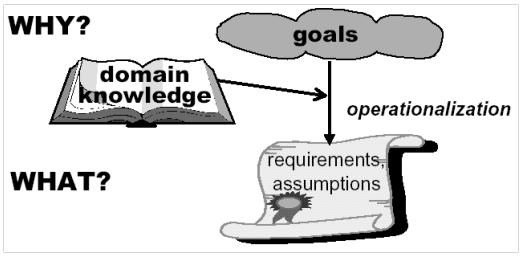

Figure 2: Goal-oriented approaches

A goal is *a prescriptive statement of intent the system should satisfy through cooperation of its agents* [L01]. A Goal-Oriented Requirement Engineering (GORE) approach is concerned with the elicitation of the goals to be achieved. Paradigms using the goal concept have been proposed by several approaches: KAOS [L01], I* [Yu97], CREWS [RSB98], GBRAM [Ant96]. The KAOS approach allows to express goals and their operationalization into specifications of services and constraints (WHAT issues), and the assignment of responsibilities to agents such as humans, devices and software pieces available or to be developed (WHO issues).

The KAOS method provides four complementary sub-models that describe the system and its environment: a goal model, a responsibility model, an operation model and an object model. The main concept of the goal model is the concept of goal. A goal is *a prescriptive statement of intent the system should satisfy through cooperation of its agents* [L01]. A goal model is an AND/OR graph where higher-level goals can be refined into lower-level sub-goals, and then, recursively, into low-level sub-goals that lead to the satisfaction of requirements of the system-to-be. The refinement relationship between a high level goal and its sub-goals is an AND/OR meta-relationship. When a goal is AND-refined into sub-goals, all of them must be satisfied for the parent goal to be satisfied. When a goal is OR-refined, the satisfaction of one of them is sufficient for the satisfaction of the parent goal. A goal that cannot be refined further is assignable to an agent. An agent is an active system component having to play some role in goal satisfaction. A goal assignable to an agent is a requisite. A requisite that is placed under responsibility of an agent of the system-to-be is a requirement, whereas a requisite that is placed under responsibility of an agent in the environment of the system is an expectation.

Scenario-based Approaches.

Users have often difficulties to directly express the expected objectives of the system. The idea behind of scenario-based approaches is that it is easier to discover objectives from scenarios expressed in natural language. A scenario [Rol03] is *a sequence of interactions between the system to be and its environment described in a restricted context*.

Scenarios are described by using several notations: informal, semi-formal or formal. Informal scenarios are described in natural language [AR97, Eri95, Hol90], with videos [SWC94, WHP98], etc. Semi-formal scenarios use structured notations like arrays [APT94], scripts [GR92], UML use case diagrams [2]. Formal scenarios are described by languages based on regular grammar [Gli95] or state diagrams [Har87]. They can be used to simulate the system behavior and evaluate user interactions.

Some scenario-based approaches focus on the functionalities description of the the system to be [Fir94, Gli95, AR97, APT94, GR92, SDV96]. They are restricted to functional requirements. There exist methods that combine goals and scenarios [Pot95, LRB97, Kai00, AP98, WHP98, LW98, Rol03].

Limitations and Perspectives.

Despite the many contributions of the approaches presented above, there are still some issues that are not fully addressed. Our past experience has allowed us to identify some topics that are relevant to the domain of critical systems. Requirements engineering in the field of critical systems such as transportation systems requires approaches capable of taking into account the significant variability of the requirements. This issue is presented in more detail in [SLG09].

The notion of AND/OR refinement relation proposed by the goal based approaches is not sufficient to take into account non-functional requirements and their relationship to functional requirements [GSL14]. Additional concepts, such as the notion of contribution defined in the NFR Framework, are required to take into account all the aspects related to non-functional.

Finally, thanks to the AND/OR refinement link, traceability links can be established between the high-level goals and the requirements that are resulting. However, this is not enough to define traceability to the final system. It would be useful to have explicit relationships such as those proposed in the SysML language [3].

References

[Ant96] A.I. Anton. Goal based requirements analysis. In *Proceedings of International Conference on Requirements Engineering (ICRE96)*, pages 136-144, 1996.

[AP98] A.I. Anton and C. Potts. The use of goals to surface requirements for evolving systems. In *International Conference on Software Engineering (ICSE '98)*, Kyoto, Japan, pp. 157-166, 1998.

[APT94] A.I. Anton, C. Potts and K. Takahashi. Inquiry-based requirements analysis. In *In IEEE Software*, volume 11(2), pages pp.21–32. IEEE, 1994.

[AR97] C. Ben Achour and C. Rolland. Guiding the construction of textual use case specifications. *Data and Knowledge Engineering Journal*, 1997.

[Bou14] I. Boukhari. Intégration et exploitation de besoins en entreprise étendue fondées sur la sémantique. Ph.D. thesis, January 14, 2014.

[Eri95] T. Erickson. Notes on design practice: Stories and prototypes as catalysts for communication. In Ed J.M. Carroll, editor, *Scenario-Based Design: Envisioning Work and Technology in System Development*, 1995.

[Fin96] A. Finkelstein and J. Dowell. A comedy of errors – the London Ambulance Service Case Study. In *Proceedings of the 8th International Workshop on Software specifications & Design*, Los Alamitos, 4-6 Jan 1996, pp. 2-4, IEEE Computer Society Press.

[Fir94] D.G. Firesmith. Modelling the Dynamic Behaviour of Systems, Mechanisms, and Classes with Scenarios. In *Software DevCon 94*, pages 73-82, SIGS Publications, 1994.

[Gli95] M. Glinz. An integrated formal model of scenarios based on statecharts. In *Lecture Notes in Computer Science*.

[2] http://www.uml.org/
[3] http://www.omg.org/spec/SysML/1.2

[GSL14] C. Gnaho, F. Semmak and R. Laleau. Modelling the impact of Non-functional Requirements on Functional Requirements. In *Advances in Conceptual Modeling, ER Workshop RIGIM*, LNCS, Volume 8697, 2014.

[Har87] D. Harel. Statecharts: a visual formalism for complex systems. In *Sci. Computer Program*, volume 8.

[HBR02] T. Hall, S. Beechham and A. Rainer. Requirements Problems in Twelve Companies – An Empirical Analysis. In *Proceedings of the 6th International Conference on Empirical Assessment and Evaluation in Software Engineering (EASE 2002)*.

[Hol90] C. H. Holbrook. A scenario - based methodology for conducting requirements elicitation. In *ACM SIGSOFT, Software Engineering Notes*, volume 15.

[IEE98] IEEE.

[JM90] F. Jay and R. Mayer. IEEE standard glossary of software engineering terminology. IEEE Std 610.12-1990, 1990.

[Kai00] H. Kaindl. A design process based on a model combining scenarios with goals and functions. In *IEEE Trans. on Systems, Man and Cybernetic*, Vol. 30 No. 5, 537-551, 2000.

[LRB97] J.C.S. Leite, G. Rossi, F. Balaguer, A. Maiorana, G. Kaplan, G. Hadad and A. Oliveros. Enhancing a requirements baseline with scenarios. In *Third IEEE International Symposium On Requirements Engineering RE'97*, Antapolis, Maryland, IEEE Computer Society Press, pp. 44-53, 1997.

[Pot95] C. Potts. Using schematic scenarios to understand user needs. In *Proc. DIS'95 - ACM Symposium on Designing interactive Systems: Processes, Practices and Techniques*, University of Michigan, 1995.

[GR92] A.Golberg and K.S. Rubin. Object behavior analysis. In *Communications of the ACM*, volume 35.

[RJ77] D.T. Ross and K.E. Schoman Jr. Structured analysis for requirements definition. In *IEEE Transactions on Software Engineering*, pp. 6-15, 1977.

[Rol03] C. Rolland. L'ingénierie des besoins : L'approche l'écritoire. In *Journal Techniques de l'Ingénieur*, 2003.

[RSB98] C. Rolland, C. Souveyet and C. Ben Achour. Guiding Goal Modeling Using Scenarios. In *IEEE Trans. on Sofware Engineering*, 2.4(12), pp. 1055-1071, 1998.

[SLG09] F. Semmak, R. Laleau and C. Gnaho. Supporting variability in goal-based requirements. In *Proceedings of the IEEE International Conference on Research Challenges in Information Science*, 2009.

[SDV96] S. Some, R. Dssouli and J. Vaucher. Toward an Automation of Requirements Engineering using Scenarios. In *Journal of Computing and Information, Special issue: ICCI'96, 8th International Conference of Computing and Information, Waterloo, Canada*, 2(1) pp. 1110-1132, 1996.

[Som04] I. Sommerville. Software engineering (7th edition). Wiley, 2004.

[Som07] I. Sommerville. Software engineering (8th edition). Pearson Addison Wesley, 2007.

[SS97] I. Sommerville and P. Sawyer. Requirements engineering. Worldwide Series in Computer Science, Wiley, 1997.

[Sta95] The Standish Group. The Standish Group, Chaos. Standish Group Internal Report. 1995.

[SWC94] S. M. Stevens, D.P. Wood and M. G. Christel. A multimedia approach to requirements capture and modelling. In *Proceedings. ICRE 94*. Colorado Springs, 1994.

[TB76] T.A. Thayer T.E. Bell. Software requirements: Are they really a problem ? In *Proc. ICSE-2: 2nd International Conference on Software Enginering*, pages 61–68, San Francisco, 1976.

[L01] A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proc. Of the 5th Int. Symposium on Requirements Engineering*, Toronto, 2001.

[LW98] A. van Lamsweerde and L. Willemet. Inferring declarative requirements specifications from operational scenarios. In *IEEE Transactions on Software Engineering, Special Issue on Scenario Management*, Vol. 24, No. 12, pp. 1089-1114 , 1998.

[WHP98] Klaus Weidenhaupt, Peter Haumer and Klaus Pohl. Requirements elicitation and validation with real world scenes. In *IEEE Transactions on Software Engineering*, volume 24.

[Yu97] E. Yu. Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering. In *Proc. 3rd IEEE International Symposium on Requirements engineering*, pages 226-235, ACM Press, 1997.

3 Formal Languages and Tools

This section presents the formal languages/tools we plan to use in the context of the FORMOSE project, mainly the Event-B language together with its ProB animator/model checker and the UPPAAL model checker. We have adopted these language/tool for the following reasons:

1. the refinement technique of the Event-B method permits to deal with the complexity of a system by gradually introducing its components;
2. the Event-B language is supported by tools (Rodin and AtelierB) that assist the user in the building of complex projects. Different provers are associated with these tools to demonstrate the correctness of the development;
3. ProB will be useful for two purposes: the validation of B specifications against a set of requirements by animating them; the rapid detection of some errors before the proof phase by model checking the B model;
4. UPPAAL will be used to deal with timed aspects of the systems we will consider in the project.

3.1 Event-B method

Event-B is the successor of the B method [Abr05] permitting to model discrete systems using mathematical notations. The complexity of a system is mastered thanks to the refinement concept that allows to gradually introduce the different parts that constitute the system starting from an abstract model to a more concrete one. An Event-B specification is made of two elements: *context* and *machine*. A context describes the static part of an Event-B specification; it consists of constants and sets (user-defined types) together with axioms that specify their properties:

CONTEXT *Cont*
Sets *S*
Constants C
Axioms *A*
END

The dynamic part of an Event-B specification is included in a machine that defines variables V and a set of events V. The possible values that the variables hold are restricted using an invariant, denoted Inv, written using a first-order predicate on the state variables:

MACHINE *Name*
SEES *Cont*
Variables *V*
Invariants *Inv*
Events *E*

Each event has the following form:

ANY *X*
WHEN G
THEN *Act*
END

This event can be executed if it is enabled, i.e. all the conditions G, named guards, prior to its execution hold. Among all enabled events, only one is executed. In this case, substitutions Act, called actions, are applied over variables. In this paper, we restrict ourselves to the *becomes equal* substitution, denoted by $(x := e)$.

The execution of each event should maintain the invariant. To this aim, proof obligations are generated. For each event, we have to establish that:

$$\forall S, C, X.\ (A \wedge G \wedge Inv \Rightarrow [Act]Inv)$$

where $[Act]Inv$ gives the weakest constraint on the *before* state such that the execution of Act leads to an *after* state satisfying Inv.

We have also to prove the feasibility of each event by discharging the following proof obligation:

$$\forall S, C, X.\ (A \wedge G \wedge Inv \Rightarrow \exists X.G)$$

Refinement is a process of enriching or modifying a model in order to augment the functionality being modeled, or/and explain how some purposes are achieved. Both Event-B elements *context* and *machine* can be refined. A context can be extended by defining new sets S_r and/or constants C_r together with new axioms A_r. A machine is refined by adding new variables and/or replacing existing variables by new ones V_r that are typed with a additional invariant Inv_r. New events can also be introduced to implicitly refine a **skip** event. In this paper, the refined events have the same form:

ANY X_r
WHEN G_r
THEN Act_r
END

To prove that a refinement is correct, we have to establish the following two proof obligations:

– *guard refinement*: the guard of the refined event should be stronger than the guard of the abstract one:

$$\forall(S, C, S_r, C_r, V, V_r, X, X_r).$$
$$(A \wedge A_r \wedge Inv \wedge Inv_r \Rightarrow (G_r \Rightarrow G))$$

– *Simulation*: the effect of the refined action should be stronger than the effect of the abstract one:

$$\forall(S, C, S_r, C_r, V, V_r, X, X_r).$$
$$(A \wedge A_r \wedge Inv \wedge Inv_r \wedge [Act_r]Inv_r \Rightarrow [Act]Inv)$$

3.2 The ProB model checker

ProB is an animator and explicit model checker, originally developed for the verification and validation of software development based on the B language. Developed at the University of Düsseldorf starting from 2003, ProB[4] implements a model checking technique to check LTL and CTL properties against a B specification.

The core of ProB is written in a logical programming language called Prolog. The probability of the state space explosion problem is reduced by a high-level specification such as the B-Method, and Symmetry Reduction techniques [LB03]. A mixed depth-first, breadth-first search strategy of ProB can encounter programming mistakes where a depth-first search might fail and a breadth-first search might be too exhaustive [Leu08].

The purpose of ProB is to be a comprehensive tool in the area of formal verification methods. ProB has an easy to use user interface and offers guided animation of the state space. Its main functionalities can be summarized up as follow:

1. ProB can find a sequence of operations that, starting from a valid initial state of the machine, moves the machine into a state that violates its invariant,
2. giving a valid state, ProB can exhibit the operation that make the invariant violated,
3. ProB allows the animation of the B/EventB specification to permit the user play different scenarios from a given starting state that satisfies the invariant. Through a graphical user interface implemented in Tcl/Tk, the animator provides the user with: (i) the current state, (2) the history of the operation executions that has led to the current state and (3) a list of all the enabled operations, along with proper argument instantiations. In this way, the user does not have to guess the right values for the operation arguments.
4. ProB supports the model checking of the LTL [Pnu77] (linear temporal logic) and CTL [CE81] (Computational Tree Logic) assertions.

3.3 The UPPAAL tool

Developed jointly by Uppsala University and Aalborg University, UPPAAL [BDL+06] is an integrated tool environment for modeling, simulation and verification of real-time systems, based on constraint-solving and on-the-fly techniques.

The description language of UPPAAL is a non-deterministic guarded command language with data types. It is suitable for systems modeled as a collection of non-deterministic processes with finite control structure and real-valued clocks, communicating through channels and (or) shared variables. Typical application areas include real time controllers and communication protocols in particular, those where timing aspects are critical.

The simulator and the model-checker are designed for interactive and automated analysis of system behavior by manipulating and solving constraints that represent the state space of a system description. The model checker is used mainly to check invariant and reachability properties by exploring the state space of a system, i.e. reachability analysis. The simulator permits the examination of possible dynamic executions of a system during early modeling (or design) stages and thus provides an inexpensive mean of fault detection prior to verification by the model-checker which covers the exhaustive dynamic behavior of the system.

[4] https://github.com/bendisposto/probparsers

Figure 3 gives an example of an UPPAAL model composed of two processes A and B. Each of these processes is composed of four control nodes. The model uses two clocks x and y, one integer variable n and one channel a. Each control nodes may be decorated with an invariant representing a constraints on the clock values in order for control to remain in a particular node. Similarly, three labels types may be associated with each edge: a guard, expressing a condition on the values of the clocks and the integer variables that must be fulfilled in order for the edge to be triggered; a synchronization action which is executed when the edge is taken and finally a number of clock resets and assignments to integer variables. For instance, the edge from A_0 to A_1 is taken when the value of the clock y is greater than 3. In that case, the process sends a message through the channel a and resets the clock y.

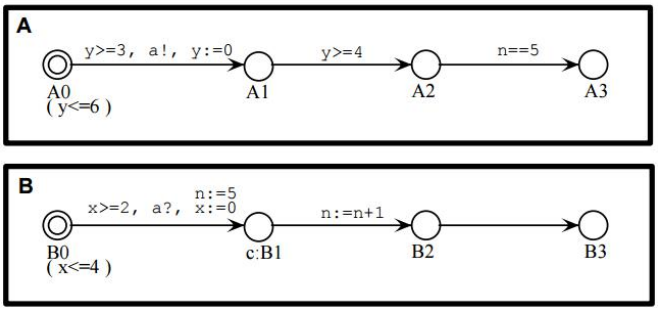

Figure 3: An example UPPAAL model

In UPPAAL, properties are expressed using a simplified version of TCTL which consists of path and state formulae. State formulae denote properties that are evaluated on each individual state whereas path formulae quantify over paths or traces of the model. Path formulae can be classified into *reachability*, *safety* and *liveness*.

References

[Abr05] Jean-Raymond Abrial. *The B-book - assigning programs to meanings*. Cambridge University Press, 2005.

[BDL+06] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. UPPAAL 4.0. In *Third International Conference on the Quantitative Evaluation of Systems (QEST 2006), 11-14 September 2006, Riverside, California, USA*, pages 125–126, 2006.

[CE81] Edmund-M. Clarke and E-A. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logic of Programs*, pages 52–71, 1981.

[LB03] Michael Leuschel and Michael J. Butler. Prob: A model checker for B. In *FME 2003: Formal Methods, International Symposium of Formal Methods Europe, Pisa, Italy, September 8-14, 2003, Proceedings*, pages 855–874, 2003.

[Leu08] Michael Leuschel. The high road to formal validation:. In *Abstract State Machines, B and Z, First International Conference, ABZ 2008, London, UK, September 16-18, 2008. Proceedings*, pages 4–23, 2008.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57, 1977.

4 Requirements Engineering and Formal Methods

In their seminal paper, Ross and Schoman stated that requirements definition is a careful assessment of the needs that a system has to fulfill. It must say why a system is needed, based on current or foreseen conditions, which may be internal operations or an external market. It must say what system features will serve and satisfy this context. And it must say how the system is to be built [RS77]. Since then, requirements engineering has been an active research area and it is widely accepted that it is a crucial phase in software and system development [vL00].

There are different kinds of RE methods [NE00] but goal-oriented RE methods, such as KAOS [DvLF93], Tropos/*i** [Yu97, BPG+04], GSN [KW04] or those developed in CRI-Paris 1[5] [NEKZ04], seem to be the most promising for complex systems engineering. Goals are objectives that the system under construction must achieve and refer to intended properties to be ensured. They are mostly expressed using natural language, but some approaches propose to use formal languages. In particular KAOS defines behavioral goals [vL09, LVL02b, Let01] using Linear Temporal Logic (LTL) [Pnu77]. Goals are formulated at different levels of abstraction from high-level, strategic concerns to low-level technical concerns, with different kinds of refinement relationships between the levels. This feature could be very relevant for our project, as complex systems have complex architectures with multiple components. Nevertheless, the abstract architecture of complex systems is never considered, as existing work [vL03, JKPS10] deal with the intertwining of requirements and design or even implementation architecture.

In KAOS, lowest-level goals, called requirements, are then realized by operations, which correspond to the specification of state transitions. They are defined by descriptive pre- and post-conditions that characterize state transitions corresponding to the execution of operations, together with prescriptive pre-, trigger- and post-conditions that constrain the application of operations for each goal to be satisfied by the operation [vL09, LVL02b, Let01, LvL02a]. In other approaches, the concepts of commitments [CDGM10] or tasks [Yu09] play an analogous role of specifying goal realization by individual actions upon the system.

Another major concept is the one of agent, which stands for an active component of the system. In most approaches, agents are in charge of realizing goals, as in KAOS and Albert & Albert II [DDBP94, DB97]. Tropos and *i** also consider that agents aim for goals that can be different from the goals they are responsible for. This consideration helps determining why each goal appears in a model. It is also a way for checking that, when responsibilities are assigned to agents, they are not in contradiction with their interests.

A recent development at Onera [CBC11, CBC13] combines these various notions of goals and agents in a single core modeling language called KHI. This language is equipped with an original semantics based upon a multi-agent-temporal logic. The latter allows to assess complex problems such as the "assignment problem" which asks whether a coalition of agents (components or sub-systems for instance) is able to satisfy a *temporal* property. This approach seems to be particularly promising to address the intertwining between requirements and architectural structure in RE for complex systems.

As seen above, many propositions have been equipped with a formal semantics. The most expressive formalizations are based upon temporal logics, such as LTL for KAOS or the novel Updatable Strategy Logic (USL) for Onera. These formalizations pave the way for various verifications and assessments that can thus be done very early in the development process, such as consistency checking, refinement correctness, conflict detection, obstacle generation, etc. [vL09] or assignment feasibility [CBC11]. In principle, model-checkers are natural candidates to perform these kinds of processing but, as of today, no formal chain from RE models to model-checkers is available in practice. Besides, the formalization of domain models is usually not considered *per se* in the aforementioned formalizations. More importantly, the use of formally-expressive temporal logics is needed for certain types of goals and certain temporal properties are hard to write: this makes it hard for engineers to formalize goals (the use of Dwyer's patterns [DAC99] as in KAOS is welcome but insufficient in our opinion: some extra language support would likely be beneficial) and this also impedes automated verification (cf. the use of a *first-order* temporal logic with past operators in KAOS). Finally, many "non-functional" requirements require other forms of logics, such as *timed* temporal logic: up to now, these approaches have not really been considered in the RE framework.

Another way for giving formal semantics to requirements models is to combine semi-formal requirements models with formal notations to be able to verify different types of properties. In [JHLR10], the authors present an approach to devise a traceability relation between a requirements model, expressed using the formalism WR-

[5] http://crinfo.univ-paris1.fr

SPM [GGJZ00], and its corresponding EventB formal representation. Based on a previous work [LGG+10] that dealt with the combination of Problem Frame [Jac00] and EventB, this approach defines three kinds of traceability links by putting in relation the different elements of a WRSPM model and the EventB concepts, which is interesting as both formalisms offer the possibility to model complex systems using a refinement mechanism. As models evolve over time, *evolution trace* helps stakeholders verify that changes to the requirements reflect their intentions. *Explicit trace* permits to explicitly link each non-formal requirement to at least one formal statement. These traces are annotated with a justification that explains why the formal statement corresponds to the non-formal requirement. *Implicit trace* permits to link the proof obligations, which ensure the correctness of an EventB model, to the various invariants. Using these three traceability relations, it becomes possible not only to point out the inconsistent requirements according to the falsifiable EventB proof obligations but also to make both specifications evolve conjointly. Nevertheless, this approach is only illustrated through a case study, no formal traceability rules are defined. Recently, a tool supporting this approach has been developed [HJL14].

In [JDB09], an operational formal semantics for SysML activity diagrams is presented. However as the transformation rules are defined in a single direction, it is not possible to benefit from the analysis of the formal specification in order to improve or correct the corresponding SysML diagrams.

Finally, Mitsubishi Electric has investigated the translation of SysML diagrams into the B language [Bou02]. To this aim, translation rules have been defined to map SysML concepts into B. This work considers mainly block diagrams and state machines for which abstract B machines and implementation components are derived. As previous approaches, the translation rules are defined in one direction, no traceability is defined.

In [PEML10], authors illustrate an approach to decompose abstract requirements into more concrete ones and their association with the different components of the related system. The behavior of these components is modeled using SysML activity diagrams that are then mapped into UPPAAL for property verification purpose. A similar but more advanced approach is also proposed in [GB11] in the aerospace context where a system is represented by a set of axioms on which deduction can be carried out. Another approach consisting of a Petri net-based formal semantics of LTL temporal requirements, represented by SysML diagrams, is defined in [LdOFV07]. A recent work at LACL consists in devising an approach to check the consistency of a SysML/KAOS goals model by mapping it into an EventB specification [MGL11]. However, the translation rules are uni-directional and no traceability links are defined.

Regarding these coupling work, the present project makes this state of the art move forward by proposing a complete approach that will deal with the different aspects of requirements engineering including the definition of a new description language to cover the most relevant concepts of the RE domain, its translation into formal notations for properties verification purpose. Let us stress that the translation will be achieved in both directions in order to be able to correct/change the RE model according to the results of the formal verification phase.

According to [EV10], even though software modeling has brought recent progress, still only one of three software projects is successful. Among the proposed solutions to get better figures, better support for processes and process modeling seems to be recognized as a major step [CKO92, Bar08]. Within the domain of requirements engineering, many studies conclude that process modeling is a very effective tool [CAG09]. Roughly, the process of requirements engineering can be seen as a workflow of activities directed towards discovering, specifying, classifying, and validating a set of requirements that the system under development must satisfy. The use of process engineering for supporting requirements engineering is not a recent strategy but there is still plenty of room for improvement. For example, a methodology for using *Rational Unified Process* (RUP) to carry out requirements engineering has been proposed in [LSZ+09]. Besides the use of processes, some recently proposed requirements engineering process models reinforce the use of process modeling in terms of defining the requirements engineering methodologies [PSSD10, PSR10]. More generally, often requirements engineering approaches come with basic method relying on generic process model such as the one proposed in [HJD11]. Such basic workflows can serve as a base for a method for requirements engineering but to support tooled enactment and full traceability as required by standards for critical systems we need more detailed process description. The COSMOD-RE method [JKPS10] goes one step beyond proposing more detailed activities and their precise dependencies. But, as more and more sophisticated process modeling languages supporting enactment are defined and tools supporting them are released, we think that there is need to define more precisely requirements engineering methods. This is even more true in the domain of critical system where a high degree of quality must be certified.

In this context, the objective of the Formose project is to define a customizable method including a require-

ments modeling language. The two main innovations, we intend to carry on, are: (1) defining in our method a process modeling to support an enactment by a tool and (2) offering a high level of integration between the various modeling tools and the provers to offer traceability and consistency checks.

References

[Bar08] J. Barjis. The importance of business process modeling in software systems design. *Sci. Comput. Program.*, 71:73–87, March 2008.

[Bou02] E. Bousse. Requirement management led by formal verification. Master's thesis, University of Rennes, France, 202.

[BPG+04] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, pages 203–236, 2004.

[CAG09] E.C.S. Cardoso, J.P.A. Almeida, and G. Guizzardi. Requirements engineering based on business process models: A case study. In *Enterprise Distributed Object Computing Conference Workshops, 2009. EDOCW 2009. 13th*, pages 320 –327, Septembre 2009.

[CBC11] C. Chareton, J. Brunel, and D. Chemouil. A Formal Treatment of Agents, Goals and Operations Using Alternating-Time Temporal Logic. In *14th Brazilian Symposium, SBMF 2011*, pages 188–203, 2011.

[CBC13] C. Chareton, J. Brunel, and D. Chemouil. Towards an Updatable Strategy Logic. In *1st Int. Workshop on Strategic Reasoning (SR 2013)*, volume 112, pages 91–98, 2013.

[CDGM10] A.K. Chopra, F. Dalpiaz, P. Giorgini, and J. Mylopoulos. Reasoning about agents and protocols via goals and commitments. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 457–464, 2010.

[CKO92] B. Curtis, M. I. Kellner, and J. Over. Process modeling. *Commun. ACM*, 35:75–90, September 1992.

[DAC99] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *ICSE*, pages 411–420, 1999.

[DB97] P. Du Bois. The Albert II reference manual. Technical report, Technical report, University of Namur (Belgium), 1997. Available at `http://www.info.fundp.ac.be/~phe/albert.html`, 1997.

[DDBP94] E. Dubois, P. Du Bois, and M. Petit. Albert: An agent-oriented language for building and eliciting requirements for real-time systems. In *HICSS (4)*, pages 713–722, 1994.

[DvLF93] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Sci. Comput. Program.*, 20(1-2):3–50, 1993.

[EV10] J. L. Eveleens and C. Verhoef. The rise and fall of the chaos report figures. *IEEE Software*, 27:30–36, 2010.

[GB11] H. Graves and Y. Bijan. Using formal methods with SysML in aerospace design and engineering. *Ann. Math. Artif. Intell.*, 63(1):53–102, 2011.

[GGJZ00] C. A. Gunter, E. L. Gunter, M. Jackson, and P. Zave. A reference model for requirements and specifications. *IEEE Software*, 17(3):37–43, 2000.

[HJD11] M. E. C. Hull, K. Jackson, and J. Dick, editors. *Requirements Engineering, Third Edition*. Springer, 2011.

[HJL14] Stefan Hallerstede, Michael Jastram, and Lukas Ladenberger. A method and tool for tracing requirements into specifications. *Sci. Comput. Program.*, 82:2–21, 2014.

[Jac00] M. A. Jackson. *Problem Frames - Analysing and Structuring Software Development Problems*. Pearson Education, 2000.

[JDB09] Yosr Jarraya, Mourad Debbabi, and Jamal Bentahar. On the meaning of sysml activity diagrams. In *16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, ECBS 2009, San Francisco, California, USA, 14-16 April 2009*, pages 95–105. IEEE Computer Society, 2009.

[JHLR10] M. Jastram, S. Hallerstede, M. Leuschel, and A. G. Russo. An approach of requirements tracing in formal refinement. In *Verified Software: Theories, Tools, Experiments, VSTTE 2010*, volume 6217 of *Lecture Notes in Computer Science*, pages 97–111. Springer, 2010.

[JKPS10] M. Jarke, R. Klamma, K. Pohl, and E. Sikora. Requirements engineering in complex domains. In *Graph Transformations and Model-Driven Engineering*, pages 602–620, 2010.

[KW04] T. P. Kelly and R. A. Weaver. The goal structuring notation - a safety argument notation. In *Proceedings of the Dependable Systems and Networks 2004 Workshop on Assurance Cases*, 2004.

[LdOFV07] M. Vinicius Linhares, R. Silva de Oliveira, J-M. Farines, and F. Vernadat. Introducing the modeling and verification process in SysML. In *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, Patras, Greece*, pages 344–351. IEEE, 2007.

[Let01] E. Letier. *Reasoning about agents in goal-oriented requirements engineering*. PhD thesis, Université Catholique de Louvain, 2001.

[LGG+10] F. Loesch, R. Gmehlich, K. Grau, C. Jones, and M. Mazzara. Report on pilot deployment in automotive sector. Technical report, D7, DEPLOY Project, 2010.

[LSZ+09] P. Liegl, R. Schuster, M. Zapletal, C. Huemer, H. Werthner, M. Aigner, M. Bernauer, B. Klinger, M. Mayr, R. Mizani, and M. Windisch. A methodology for process based requirements engineering. *IEEE International Conference on Requirements Engineering*, pages 193–202, 2009.

[LvL02a] E. Letier and A. van Lamsweerde. Agent-based tactics for goal-oriented requirements elaboration. In *24th International Conference on Software Engineering*, ICSE, pages 83–93, New York, NY, USA, 2002.

[LVL02b] E. Letier and A. Van Lamsweerde. Deriving operational software specifications from system goals. In *10th ACM SIGSOFT symposium on Foundations of software engineering*, page 128. ACM, 2002.

[MGL11] A. Matoussi, F. Gervais, and R. Laleau. A goal-based approach to guide the design of an abstract Event-B specification. In *16th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2011), Las Vegas, USA, 27-29 April*, 2011.

[NE00] B. Nuseibeh and S. M. Easterbrook. Requirements engineering: a roadmap. In *22nd International Conference on Software Engineering, Future of Software Engineering Track, June 4-11. ACM*, pages 35–46, 2000.

[NEKZ04] S. Nurcan, A. Etien, R. Kaabi, and I. Zoukar. A strategy driven business process modeling approac. *Special issue of the Business Process Management Journal on "Goal-oriented Business Process Modeling"*, 2004.

[PEML10] J-F. Pètin, D. Evrot, G. Morel, and P. Lamy. Combining SysML and formal models for safety requirement verification. In *International Conference on Software and Systems Engineering and their Applications*, 2010.

[Pnu77] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[PSR10] D. Pandey, U. Suman, and A.K. Ramani. An effective requirement engineering process model for software development and requirements management. *Advances in Recent Technologies in Communication and Computing, International Conference on*, 0:287–291, 2010.

[PSSD10] V. Pavanasam, C. Subramaniam, T. Srinivasan, and J. Kumar Jain D. Membrane computing model for software requirement engineering. *Computer and Network Technology, Inter. Conf. on*, 0:487–491, 2010.

[RS77] D. T. Ross and K. E. Schoman, Jr. Structured analysis for requirements definition. *IEEE Trans. Software Eng.*, 3(1):6–15, 1977.

[vL00] A. van Lamsweerde. Requirements engineering in the year 00: a research perspective. In *ICSE 2000, Proceedings of the 22nd International Conference on on Software Engineering, June 4-11, 2000, Limerick Ireland. ACM*, pages 5–19, 2000.

[vL03] A. van Lamsweerde. From system goals to software architecture. In M. Bernardo and P. Inverardi, editors, *SFM*, volume 2804 of *Lecture Notes in Computer Science*, pages 25–43. Springer, 2003.

[vL09] A. van Lamsweerde. *Requirements engineering, From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[Yu97] E. S. K. Yu. Towards modeling and reasoning support for early-phase requirements engineering. In *3rd IEEE International Symposium on Requirements Engineering (RE'97), January 5-8, 1997, Annapolis, MD, USA*, pages 226–235. IEEE Computer Society, 1997.

[Yu09] E. S. K. Yu. Social modeling and i*. In *Conceptual Modeling: Foundations and Applications*, pages 99–121, 2009.

5 ClearSy Requirements Engineering Process

5.1 Requirements level

Requirements may exists on several level in the development cycle of a project. They are initially derived from the highe level expression of the customer need. The figure 4 on page 17 illustrates these different levels and the different links relying upon requirements:

- between "customer requirements" and "specificaiton"
- between "specification" and "conception"
- between "specification" and "validation" (qualification tests)
- between "conception" and "development"
- between "conception" and "verification" (unitary tests)

Requirements ca also come from standards and other legal documents. In this case, such requirements are considered as in the same level than customer's ones or above them.

5.2 Tools

5.2.1 Requirements review

The requirements review takes place at the very beginning of the project. Participants to this review are :

- customer technical manager
- customer project manager
- industrial technical manager

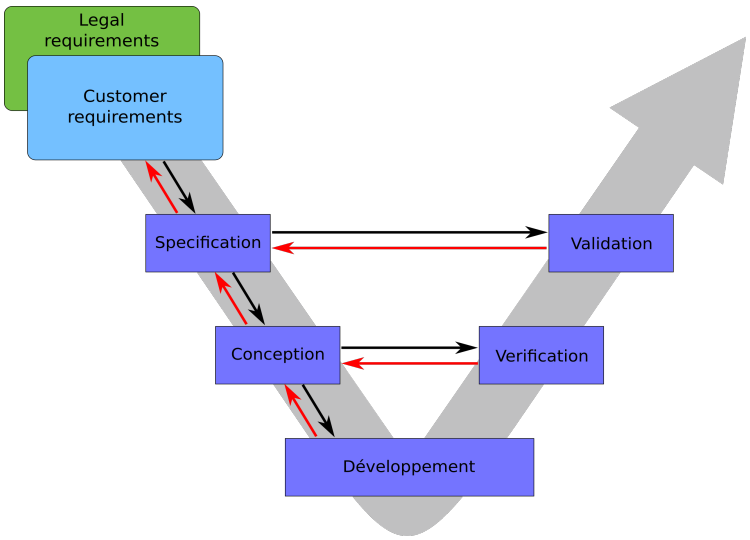

Figure 4: Requirements in project development cycle

– industrial project manager

The aim of such a review is to clarify the needs and requirements of the customer. It's also the occasion for the industrial to point out some problems or some facts that the customer hadn't identified, or some conflicts in the requirements. For the customer, it's the moment for developing some of the ideas underlying its requirements.

5.2.2 Traceability matrix

The main tool allowing to deal with requirements is the traceability matrix. This document is a table which links one requirement level to another and which gives the coverage. Each requirement of one level may be related to one or several requirements of the other level. This document serves to show the completeness of the coverage of the requirements of the higher level by the one of the lower level.

In the general case, completeness has to be shown for:

– the coverage of "legal and customer requirements" by "specification requirements"

– the coverage of "specification requirements" by "conception requirements"

– the coverage of "specification requirements" by "validation test cases"

– the coverage of "conception requirements" by "development requirements"

– the coverage of "conception requirements" by "verification test cases"

6 Thales Requirement Engineering Process

To master complexity and assure the performance of mission critical systems demands the right processes supported by the right tools. Chorus is the Thales reference system that provides a common and efficient way of working across the globe. A common approach enables work to be shared across distributed teams, and then seamlessly brought together into complex solutions. By using best practices, including the requirements of the Capability Maturity Model Integrated (CMMI), Chorus contributes to risk reduction and achievement of cost and schedule performance. Orchestra is an integrated suite of engineering tools for system and software engineering. The Orchestra Engineering Desk provides a portal to access project engineering data and the tools required to perform assigned tasks. Data is created once and accessed by teams wherever they reside. Orchestra's core is a model-driven approach to engineering, which supports solution verification at each level of elaboration. By supporting work flow management, Orchestra aims to increase the efficiency and process compliance. Orchestra is declined in three segments: Bronze, Silver, and Gold. Depending on the complexity of the project, one has to choose one of the segment (as presented in Figure 5).

– The Bronze segment is used for quick, cheap, and efficient developments based on everyday work tools.

– The Silver segment is used for medium complexity projects. It combines a full-functionality workbench while still allowing a simple development.

– The Gold segment is used for the development of complex systems by ensuring a global end-to-end traceability on large scale collaborations.

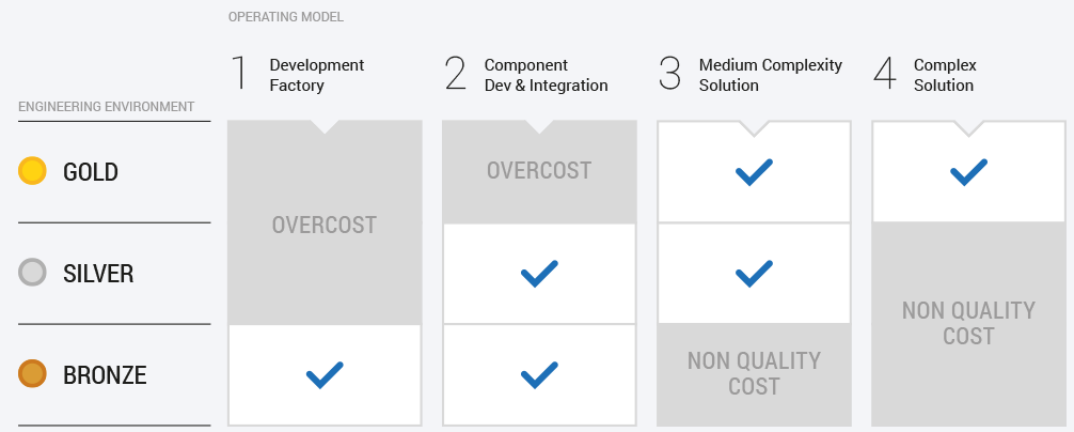

Figure 5: Orchestra segments according to complexity

Formose aims to design a Requirement Engineering method for complex systems. As a consequence, we will present in Section 6.1, some of the tools associated with the gold segment of the Orchestra workbench. We will focus on the tools linked to requirement engineering and will not present the whole toolkit (50 tools). In Section 6.2, we will present some of Thales' definitions and practices regarding requirement engineering with respect to what has been defined in Section 6.1.

6.1 Tools

When using the gold segment of the Orchestra workbench, two tools can be directly linked to requirement engineering : DOORS and Reqtify. In the following, we will briefly present those tools and some of the adaptation that were made to better suit Thales needs.

6.1.1 DOORS T-REK

DOORS is a tool developed and distributed by IBM. Thales uses an add-on to Doors called DOORS Telelogic/Thales Requirements Engineering Kit (T-REK). DOORS T-REK provides:

– Tools to structure the information in a Data Model,

– Specific module types in order to be compliant with Thales development processes,

– Modularity in order to be able to support different levels of project complexity.

DOORS object contains the atomic information through a template of attributes : an object heading containing a title, an object text containing a description, and a paragraph style describing the style information.

T-REK objects contains all the classical DOORS attributes plus a set of attributes that answers to specific needs of the system engineering. Figure 6 presents an example of a template for a User Requirement.

Typical attributes for a requirement are for example:

– An attribute (a short, unique and persistent tag)

– Name (unique and specific)

– Description (short content)

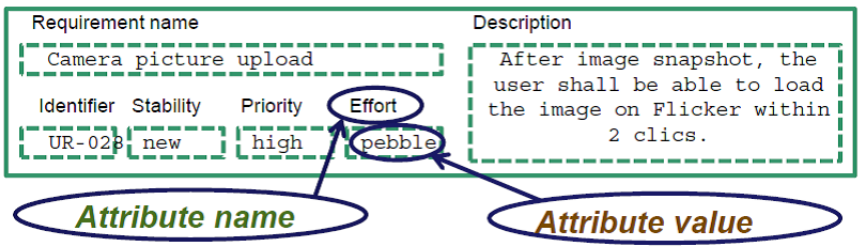

Figure 6: Example of a T-REK template for a User Requirement

- Author
- Source
- Stability (describing expected change)
- Criticity (estimated damage if not satisfied)
- Priority (requirement selection criteria)
- Contractual scope (must, should, may)
- Status, version, allocation…

DOORS T-REK allows to trace different objects in your database with different type of links. A link is a always directed from a source object to a target object. There exists 5 different types of links:

- « satisfies » that links a requirement from a lower level requirement to a upper requirement. It is used during requirement analysis. Its utilization is illustrated in Figure 7.
- « is justified by » that links a requirement to a decision in the decision and/or justification. It is used during requirement analysis and design phases. Its utilization is illustrated in Figure 8.
- « refers to » that links a critical point or a conception choice to a requirement. It is used during requirement analysis and conception phase. Its utilization is illustrated in Figure 8.
- « verifies » that links a verification activity to a requirement. It is used during IVVQ phase. Its utilization is illustrated in Figure 7.
- « is allocated by » that links a component to a requirement. It is used during the allocation of requirements to components. Its utilization is illustrated in Figure 7.

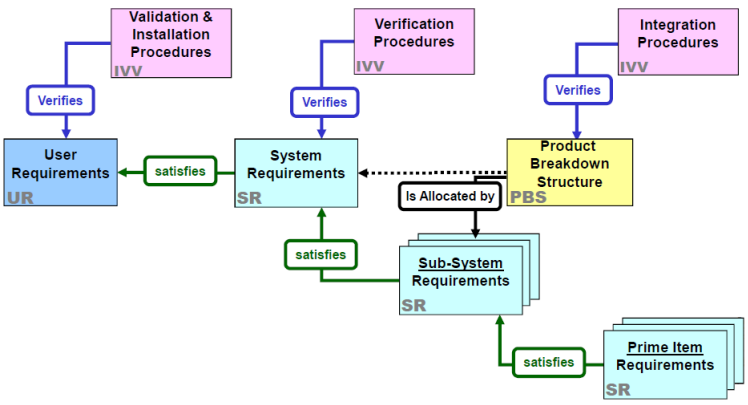

Figure 7: Illustration of the links between Doors objects (1)

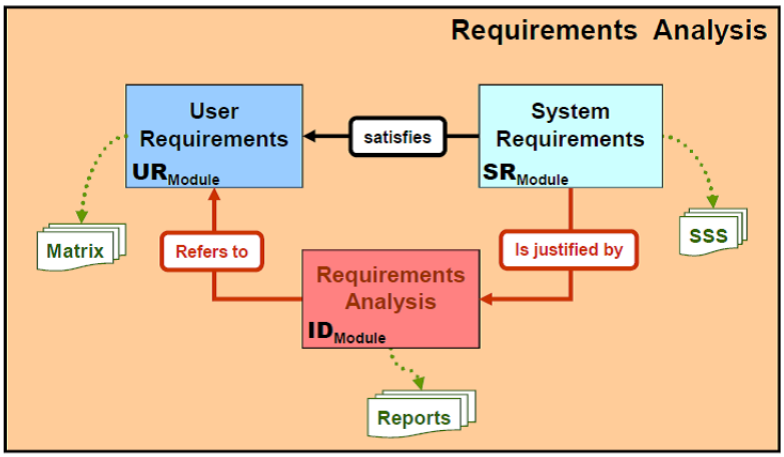

Figure 8: Illustration of the links between Doors objects (2)

6.1.2 Reqtify

Reqtify is a tool developed and distributed by Dassault Systèmes.

A specific version of Reqtify (called 'Reqtify in Orchestra') has been developed to provide all the features that allow it to be well integrated into the workbench Orchestra. The main features of Reqtify regarding requirement management are coverage analysis and impact analysis of requirement changes.

6.2 Requirement Engineering Activities

Using the tools presented in Section 6.1, Thales has defined several process and «Tooled-Up Practices » for defining, designing, producing, and testing a system.

6.2.1 Requirements Elicitation

In Thales processes, requirement elicitation is an iterative activity. Requirements are collected through various means (workshops, interviews, technical document reviews, market analysis, protypes…) and therefore are of different nature. They can be expressed in a natural language, through a semi-formal or formal model, or thanks to a combination of different means (textual and model requirements).

6.2.2 Requirement Analysis

First of all, requirements at Thales should verify a number of properties defined (quality criteria) defined with the provider of the requirements. Typically, these properties include:

- Clearly and properly stated
- Uniquely identified
- Appropriate to implement
- Verifiable (through test, demonstration…)
- Traceable to an upper level requirement

Moreover, Thales processes requires some additional work to be done when receiving and analysing requirements:

- Prioritization

  Projects are done with limited ressources. Therefore, it is important to have priorities on the requirements to allocate ressources to the high-priority requirements in case of conflict. It is also used to solve inconsistencies, plan releases…

- Separation between key and critical requirements

  At a system level, key and critical requirements are identified. Key requirements are those that impact the greater part of the costs. Critical requirements are those that may affect the development success.

- Requirement categorization

  Requirements are segregated between:

  - Functional requirements

    Functional requirements define what are the system's behaviors, the functions or tasks to be performed as well as the data exchanged with external systems.

  - Non-functional requirements

    Non-functional requirements define the qualities of the system. They can cover performance, ergonomics, reliability, maintainability, availability, security and privacy, safety, integrity…

  - Constraints

    Constraints define the limitation of the solution space for the system design. Constraints can cover : physical limitations (dimensions, weight…), interfaces with external devices, environment (mechnanicals, climatic…), domain-related standards, logistics, development processes and strandard, legal obligations, cost and lead time…

6.2.3 Requirement Specification

Thales requirement structuring is done through DOORS T-REK as described in Sect. 6.1.1. It is suggested that requirements follow a specific template such as the one in Figure 9.

[In this context] the system shall [do something][delivering this performance]

Figure 9: Illustration of a requirement template

Moreover, the structural architecture of all the different layers of requirements (from system requirement to lower levels) is inspired by the MIL-STD-498 standard.

6.2.4 Requirement Validation

Validation of a set of requirement covers several activities in Thales processes. Each activity must ensure one of the following property:

- Completeness : Defined as reached when the requirement set no longer needs amplification according to all stakeholders
- Consistency : Defined as reached when no individual requirements are contradictory, requirements are not duplicated, and when the terms used throughout the document are well defined and non redundant
- Affordable : Defined as reached when the complete set of requirements can be fulfilled by a solution that meets the life cycle constraints.
- Bounded :Defined as reached when the set of requirements does not go beyond what is needed to satisfy the stakeholders needs.

6.2.5 Requirement Management

Thales uses a broader definition of Requirement Management than the one defined in Section 6.1. Activities in this category include:

- Organization of the requirements by using DOORS T-REK as seen in Section 6.1.1

– Managing requirements changes. Using DOORS T-REK, when new requirements are received, the requirement database is updated with the new requirements or the new version if modified. DOORS is also used to maintain the requirement change history with a rationale for each change. Using Reqtify, impact analysis on the whole set of document and activities is also identified.

– Maintaining a bi-directional traceability of the requirements. This is done by using DOORS T-REK and the « is justified by » links between set of requirements.

7 Conclusion

This document was an introduction to requirements engineering and its application with formal methods and in industry. Our aim is to define a requirements modeling language which takes into account the specific characteristics of critical complex systems: cross-references between functional and non-functional requirements, traceability, modularity and formal verification.

Based on this state of the art and on the first case studies, we have chosen to start from the SysML system modeling language and the goal-oriented KAOS requirements modeling language. The first meetings with the industrial partners lead us to add new views like ontology, context and domain models, in order to determine the vocabulary and the concepts used in the system. The definition of the new requirements engineering language will be described in Deliverable D2.1.b.