Formal Requirements Engineering for Smart Industries Toward a Model-Based Graphical Language

Conference Paper · July 2016

DOI: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0160

4 authors, including:

Alexandre Le Borgne
Ecole des Mines d'Alès
1 PUBLICATION **1** CITATION
SEE PROFILE

Nicolas Belloir
IRISA - Institut de Recherche en Informatique…
35 PUBLICATIONS **136** CITATIONS
SEE PROFILE

Jean-Michel Bruel
Institut de Recherche en Informatique de To…
122 PUBLICATIONS **1,005** CITATIONS
SEE PROFILE

Formal Requirements Engineering for Smart Industries

Toward a Model-Based Graphical Language

Alexandre Le Borgne, Nicolas Belloir
LIUPPA / University of Pau
BP 1155
64013, Pau, France

Jean-Michel Bruel
IRIT/University of Toulouse
118, route de Narbonne
Toulouse, France

Thuy Nguyen
EDF R&D
Chatou, France

***Abstract*— Requirements engineering is a most critical activity in the engineering process of complex cyber-physical systems. To avoid the ambiguity of natural or semi-formal languages and to provide tool support for engineering activities all along a system lifecycle (from scoping studies to system operation and maintenance), EDF has developed FORM-L, a formal constraints-based language for the modelling of assumptions, requirements and preliminary designs. This language can be interfaced with detailed design languages such as Modelica [7]. We present in this paper an ongoing work that aims at developing a graphical representation of FORM-L. The main objective is to facilitate the understanding of models, in particular by persons not familiar with FORM-L.**

Keywords— Requirements engineering, formal languages, graphical languages, early validation

I. Introduction

Development of smart and sustainable cities is among the main challenges for the next years [11], mainly due to the necessity to fully integrate networks, sensors, applications and humans into a sustainable cooperating connected environment [6]. Smart industries are following the same way, trying to better take human aspects and external environment (e.g., global warming) into account. In our domain, we used "Smart" as defined in [12]: "systems are able to cope with a changing and unknown environment, assist human operators, or self-organize to provide unanticipated products & services. Systems engineers must integrate social, functional and physical demands to create valuable system solutions that are resilient in their operational environment". In this context, engineering approaches must be able to deal, especially at the requirements level, with contextual and human elements in addition to the traditional technical specifications.

Combining those complex systems with contextual and human elements tends to add very complex non-functional parameters into the system, increasing the likelihood of undesired emergent properties. To deal with this problem, it is necessary to make the best specification of the system as early as possible. Combining formal requirement engineering within simulation is one of the best ways to explore different specifications by simulation, allowing validating the more convincing one.

In this context, the EDF Company has defined a formal language called FORM-L [1] to formally express requirements. Those requirements can be simulated using tools like StimuLus [13] in order to validate it earlier as possible. Nevertheless, specifying requirements with FORM-L is not an easy task, especially for domain-specific engineers not familiar with the language. To solve this problem, we propose to define a graphical language, called FORM-GL, formally based on FORM-L, helping them to specify and / or understand FORM-L models and validate them using tools such as StimuLus.

On the other hand, using models to support the different activities of the engineering lifecycle is essential when dealing with large, complex systems. This approach is called Model-Driven Engineering (MDE). One of the proposed roadmaps for both system and system-of-systems engineering is to follow this way (using specific models and transformation between those models) [12]. Hence, we propose to use MDE to translate requirements expressed using FORML-GL into FORM-L. The main idea is to provide an interface allowing engineers to express requirements using both their own natural languages and FORM-L.

This paper presents our early steps toward the definition of FORM-GL. This graphical language will provide a graphical syntax to express requirements for smart industries incorporating external elements such as environmental context and/or human interactions. The paper is organized as follow. Section II presents the industrial context of this work. Section III provides a short state-of-the-art section. Section IV describes our approach and our first works. We conclude in section V on the first feedbacks and define the remaining tasks toward our goal.

II. Industrial context

Nowadays urban areas more and more attract people. This attraction is leading to such an urban population growth that today more than the half of the people in the world is living in urban and metropolitan areas [5]. This situation is obviously becoming a new challenge for cities development, and implies a new kind of development based on the idea of an interactive and connected world where mobility comes to be an important part of everybody's lives. It is at the same time a great opportunity to see how we can adapt our technologies such that we can "upgrade" cities as smart cities, making possible to these smart urban areas to improve the quality of life while minimizing the impact on the environment and costs [6].

Nevertheless, developing smart cities implies a lot of new constraints. Among those constraints we can highlight the wide interconnection of objects and numerous interactions with humans and environment [6]. Also, it seems quite obvious that to be smart a city needs smart industries that can make possible for smart cities to provide a better life area that impact as little as possible the environment.

To be able to handle seamlessly the full system lifecycle, EDF needs to take into account a huge amount of different factors that can be either internal or external to the system. Indeed, when a nuclear power plant is built, we need to understand that it is in fact a 60-years lifecycle (at least) that will lead at the end to the dismantling of the plant. As a consequence, the plant operator faces numerous and wide uncertainties. It is essential that these uncertainties and their envelopes are explicitly stated and validated. To provide tool support (e.g., massive simulation to explore the huge number of possible situations), EDF intends to model them as formal assumptions using FORM-L. The simulation may include physical aspects, controls, failures, costs, but also exceptional events such as earthquakes, flooding, malicious attacks, etc. For instance EDF needs to ensure the stability of the French electric power grid despite the ever increasing share of renewable but intermittent production means (wind and photovoltaics).

With this in mind, EDF has defined a new requirement modeling language, called FORM-L [1] that can be simulated. However, to build smart industries, we need to be able to get easily understandable requirements. This is where our work starts with FORM-GL, giving a new graphical domain specific dimension to FORM-L.

III. State-of-the-art

There are numerous ways of expressing requirements specification. We present in this section only some of them. We first analyze SysML, which is becoming a reference in Model-Based System Engineering (MBSE). Next, we present Kaos, a popular goal-oriented notation, used to elicit requirements as goal in software engineering. Last, we introduce FORM-L which is the language developed by EDF.

A. SysML/UML

UML is *de facto* the actual modeling language standard used in software engineering to model software. However, it is not well adapted to specify requirements. Indeed, the Use Case diagram which is the dedicated and more used diagram to specify requirements is better adapted to illustrate a requirement than to specify it. Use cases are mainly applied to functional requirements and are not useful to design non-functional ones [14].

SysML [2] is an extension of the UML language and is designed to model complex systems that are not necessarily software systems but also large systems involving human interactions, hardware, software and even other systems. This language is meant to be very generalist, allowing engineers to model a wide range of systems. In SysML, the notion of requirement (which does not exist in UML) has been introduced. A SysML requirement is modeled with the class stereotype `<<requirement>>` that is actually a UML class that contains an identifier that indexes the requirement and a text field that is used to define the requirement. A requirement can also be derived with the stereotyped relation `<<deriveReqt>>`. One of the notion SysML has also introduced is the notion of *traceability* that is useful to specify which model elements satisfy (relation stereotype: `<<satisfy>>`) or verify (relation stereotype: `<<verify>>`) a given requirement. However, even if SysML seems to be complete, it does not allow performing simulation over its requirements since they are only defined as text fields where no strict formalism exists and so does not allow direct exploitation by simulation dedicated tools. Moreover, one of the big limitations of SysML is that its notion of requirement is intrinsic to the language itself, which does not allow reusing requirement models that have been written in other requirement specification languages.

B. KAOS

KAOS (Knowledge Acquisition in Automated Specification) as well as I* are goal oriented requirement specification methods. A KAOS goal may be defined as the goal that the system and its environment must realize thanks to the cooperation of agents (hardware, software, human) [3]. KAOS allows requirement refining and traceability all along the model. But as SysML, its requirements are textual requirements that do not allow performing simulation on them.

C. FORM-L

FORM-L (Formal Requirements Modelling Language) has been specified in the framework of the ITEA2 MODRIO project. This language aims to fill some gaps that exist in systems engineering and especially those which are related to simulation of requirements and assumption of a system. FORM-L provides a formal way for modeling properties (requirements and assumptions) of a given system, that means that this language provides formal syntax and semantics that can be used as input to a dedicated simulation tool [4]. This language is also intended to support the coordination of the many teams and disciplines involved in the design and operation of a large and complex system, as shown in Fig. 1.

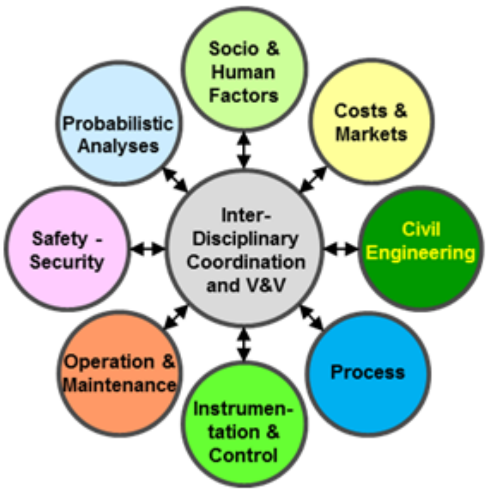


Fig. 1. A neutral integration ground for the teams & disciplines working on the system [4]

IV. Approach

A. Goal

First of all, it is important to keep in mind that the main goal of our work is not only to propose a graphical representation for all requirements (and assumptions) that can be expressed in FORM-L: this representation should also be understandable by engineers with limited training in FORM-L. Indeed, UML-like models are too difficult to understand for untrained people. So the first objective of the FORM-GL notation is to be adaptable to the culture and capabilities of future users, with the use of graphic and text boilerplates. Thus, for example, the FORM-L keywords can be replaced by the user's own native language keywords. As FORM-GL needs to be easily understandable, it should rely only on a limited number of graphical conventions. For instance, we will try to not have a lot of arrow styles that could be very disturbing for an untrained user or any user who would not know UML. We also want to keep at the lowest the number of different boxes that represent FORM-GL concepts even if we obviously need to differentiate these concepts to make sure that modeling will be simple for users. Of course as the aim of FORM-L is to do simulation all over a project lifecycle from prospective studies to the maintenance and even the death of this project (nuclear center dismantling), our FORM-GL tool thereby has to generate directly from a model the FORM-L code that will be given to simulation tools that can interpret FORM-L.

B. Tools

As mentioned earlier, we aim at benefiting from the MDE techniques and tools. Here are the main ones we are using on our work:

EMF (Ecore Modeling Framework) [8] is the reference eclipse tool for designing metamodels for domain specific modeling languages and provides different views of a model as a tree view and a graphical UML-like view which is useful to represent an understandable metamodel of FORM-L.

Sirius [9] is a tool created and maintained by Obeo for the eclipse platform which aims to provide capabilities for creating modeling tools after an EMF metamodel. This tool seemed to be the most appropriate for this project since it includes, thanks to an extension, Xtext support and allow users of a graphical modeling tool to directly generate code relying on the Xtext grammar that is link to the EMF metamodel.

Xtext [10] is also an eclipse based tool that is designed to create grammars and provide an eclipse based editor that follow this syntax. We can notice that this tool is capable to generate an EMF model after the syntax that is defined and is also capable to go through the opposite, taking an EMF model and transform it to an Xtext grammar.

C. Ongoing approach

We could have used EMF and Sirius to directly model the FORM-L language in order to transform it into a metamodel that would have allowed us to develop the graphical language. But a complete metamodel of FORM-L is not the best way to develop a user-friendly graphical language. Instead we want to consider FORM-GL as a graphical DSL.

Our ongoing effort is then to define a formal grammar of FORM-L. Using Xtext in this purpose (around 1K LoC) have allowed us to automatically derive the FORM-L ecore metamodel. In addition it allows us to define at the same time a FORM-L editor. In this grammar, we defined the main concepts of the FORM-L language. FORM-L is composed of different kind of `models`:

- The *contract model*: allows defining a contract between at least two models by defining engagements and expectations of each of these models.
- The *binding model*: meant to be a model that will link black box systems to other models by only knowing or translating their inputs and outputs.
- The *behavioral model*: allows FORM-L including other non-FORM-L models (e.g., economic models)
- The *property model*: a set of declarations and definitions that represent the system and its requirements.
- The *configuration model*: an architecture model that binds all the previous models.

Fig. 2 represents a small part of the Ecore metamodel (that contains right now 195 metaclasses) we obtained. Nevertheless, this subpart of the metamodel represents some of the key concepts of the language. First of all, the `StatementDef` metaclass represents statements, which consists in defining the properties (assumptions, requirements, guards) of a `model`. The `expression4` metaclass represents all kind of expressions that may be written in FORM-L (integer expressions, boolean expressions, string expressions…). Finally, the metaclass named as `TimeLocator` is used to allow definition of time (which may be discrete, continuous or event defined as a sliding window) concepts of FORM-L. Fig. 3 is an example of FORM-L code that illustrates the definition of some of FORM-L properties (assumptions and requirements). In this part of FORM-L code, assumptions `a1` and `a2` assume that the temperature will be comprised between `0°C` and `tCWMax` which is an external variable. In requirements `r1` and `r2` we check that a property will become true just after an event occurs and within a given duration: for instance `r1` checks that the system will leave its `startUpCold` state within `15` minutes after it entered into this state. The following requirements `r3` and `r4` verify that the system stays into a domain of temperature during all the time it is in its `normalOp` state whereas `r6` checks a boolean constraint for all the periods the system is not in the `stopped` state. `r7` and `r8` requirements (that are actually expressing the same system requirement in two different ways) are the same kind of requirements that check that a boolean condition must be true at the end of a period of time and for a set of `pumps`. The requirement `r5` checks a condition that must always be true, if this condition becomes false even once then the requirement `r5` will not be satisfied.

All the previous requirements are based on continuous period of physical time, but in FORM-L we can also describe discrete time (e.g., state becomes `startUpCold` is a discrete time locator).

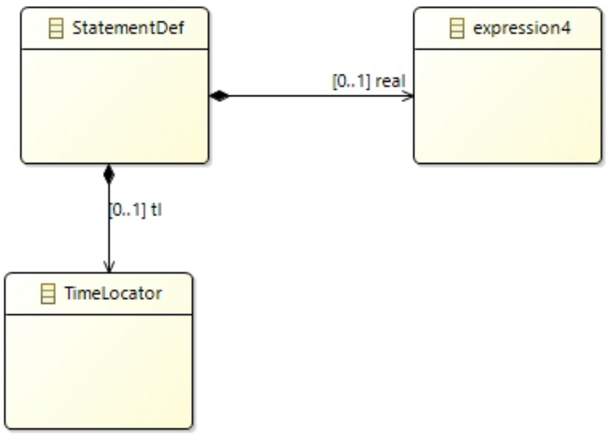
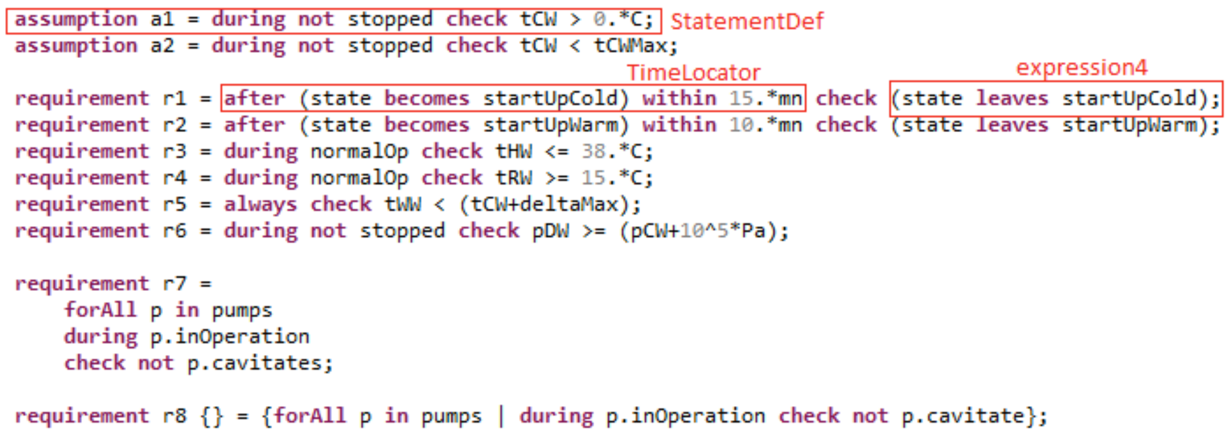


Fig. 2. Part of the FORM-L metamodel



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

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

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

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

Fig. 3. Example of FORM-L requirements, using our generated Xtext editor

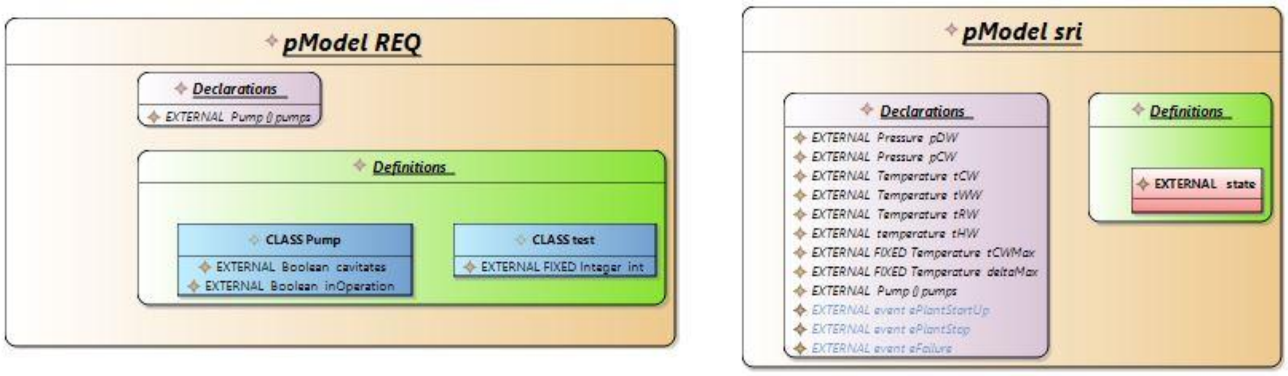


Fig. 4. Figure 4: Illustration of FORM-GL property models in Sirius

While formally defining the FORM-L grammar, we have experienced that making a graphical language directly based on how the language is written taking into account the understandability we want to reach is probably not realistic because in a previous attempts, we could not even represent an integer expression such as `1+1+2*3` in a simple way that would have implied immediate comprehension of any user. A conclusion we have reached is that we now need to define a new FORM-GL metamodel as an extension of the FORM-L metamodel. Indeed, by doing this, we can reach a satisfying user-friendliness level. So the FORM-GL metamodel will have to be defined on the concepts.

The main constraint for FORM-GL is that it must be easily understandable to allow anyone to develop or read a FORM-GL model. Hence we have chosen that the graphical frame of FORM-GL must be seen as an empty phrase where we only choose the keywords (e.g., some `TimeLocator`) and where the user only has to complete those graphical phrases with specific expressions and values (e.g., `tRW >= 15.*C`). Moreover, as we said before, this graphical language must not have too many arbitrary kinds of formalisms (we want to avoid too many different kinds of diagrams or even arrows) so a FORM-GL model will be meant to be arrow free as much as possible. That choice implies that models and their components will be deployed as adaptable boxes as illustrated in Fig. 4.

V. Conclusion

We have presented in this paper our ongoing work aiming at defining FORM-GL, a new formal graphical requirements language dedicated to express requirements for smart industries. We are working in the context of the French EDF Company that need such languages for specifying smart systems of systems. This language is designed to allow end-user to express requirements in a graphical manner but using both their own notation and graphical elements of the existing formal FORM-L language.

Our first step has been to formally define the FORM-L grammar. One of the benefits of working at the metamodel level is that we can more easily express constraints on the model elements. One of our future works will then be to illustrate this benefit by adding some OCL constraints to the FORM-L metamodel. Another goal is to refactor the metamodel (e.g., defining packages, using inheritance, …) without modifying the existing syntax.

Currently, we are focusing on the graphical language FORM-GL. Once available, we plan to evaluate it on an in-house EDF industrial case study that would help assessing how our approach helps end-users to better express requirements. Some of our long term goals include: (i) to benefit from MDE context to automatically generate state automata; (ii) to integrate this front-end graphical language with powerful simulation tools currently used at EDF and (iii) to use FORM-L as a formal semantics for Requirements in other modeling notations such as SysML.

Acknowledgment

This work is supported by a neOCampus M.Sc. grant (https://www.irit.fr/neocampus/) and a EDF collaboration.

References

[1] T. Nguyen, "FORM-L: A Modelica Extension for Properties Modelling Illustrated on a Practical Example," Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden.

[2] Object Management Group: *SysML V1.4* ; Disponible ici : http://www.omg.org/spec/SysML/, 2015.

[3] A. van Lamsweerde, L. Willemet. Inferring Declarative Requirements Specifications from Operational Scenarios. IEEE Transactions on Software Engineering, 24(12), December 1998.

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

[5] United Nations, D.o.E.a.S.A., Population Division, World Urbanization Prospects: The 2014 Revision, Highlights (ST/ESA/SER.A/352). 2014.

[6] A. Ghannem, M. S. Hamdi, W. Abdelmoez and H. H. Ammar, "A context model development process for smart city operations," Service Operations And Logistics, And Informatics (SOLI), 2015 IEEE International Conference on, Hammamet, 2015, pp. 122-127. doi:10.1109/SOLI.2015.7367605.

[7] Modelica, https://www.Modelica.org/ .

[8] Eclipse Modeling Framework (EMF), https://eclipse.org/modeling/emf/ .

[9] SIRIUS, https://eclipse.org/sirius/ .

[10] XText, https://eclipse.org/Xtext/ .

[11] Manville et al., "Mapping smart cities in the UE", UE technical Report, http://www.europarl.europa.eu/RegData/etudes/etudes/join/2014/507480/IPOL-ITRE_ET%282014%29507480_EN.pdf , 2014.

[12] INCOSE "System Engineering Vision 2025", http://www.incose.org/AboutSE/sevision, 2014.

[13] B. Jeannet, F. Gaucher. Debugging Embedded Systems Requirements with STIMULUS: an Automotive Case-Study. https://hal.archives-ouvertes.fr/hal-01292286

[14] M.S. Soares and J. Vrancken, "Requirements specification and modeling throught SysML". In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, pp 1735-1740, 2007.