Checking models based on an iterative co-specification process of a critical system

Fabien Bouffaron, Pascale Marangé, Gérard Morel

▶ To cite this version:

Fabien Bouffaron, Pascale Marangé, Gérard Morel. Checking models based on an iterative co-specification process of a critical system. International Conference on Industrial Informatics, INDIN 2014, Jul 2014, Porto Alegre, Brazil. pp.1. hal-00702417v2

HAL Id: hal-00702417

https://hal.science/hal-00702417v2

Submitted on 18 Jul 2012 (v2), last revised 12 Sep 2014 (v3)

Revisiting the interoperation relationships between Systems Engineering collaborative processes

Fabien BOUFFARON, David GOUYON, Dragoş DOBRE, Gérard MOREL

Nancy Research Centre for Automatic Control (CRAN), Lorraine-University, CNRS UMR 7039
Campus Science, BP 70239, 54506 Vandœuvre-lès-Nancy Cedex, France
{fabien.bouffaron; david.gouyon; dragos.dobre; gerard.morel}@univ-lorraine.fr

Abstract:
Systems Engineering (SE) best practices are currently guided by standardized processes which must be adapted by skill rules in order to specialize domain-dependent SE workflows as well as domain-independent standardized languages. This paper aims to revisit first the interdisciplinary relationships within a SE process as specification relationships between Problem Space (PS) and Solution Space (SS) across collaborative domains. This SE rationale is then applied on a Requirement Specification (RS) workflow formalized with high-level Petri nets and verified on a human-robot protection case-study.

Keywords: Systems Engineering, Specification, Domain, Requirement.

1 INTRODUCTION

According to the SEBoK (Pyster et al., 2011), Systems Engineering (SE) is defined as “an interdisciplinary approach and means to enable the realization of successful systems. It focuses on holistically and concurrently understanding stakeholder needs; exploring opportunities; documenting requirements; and synthesizing, verifying, validating, and evolving solutions while considering the complete problem, from system concept exploration through system disposal”. These SE activities are recursively organised according to five views and iteratively within each view related to “Business”, “Requirement”, “Architecture”, “Integration Verification Validation Qualification” and “Configuration”. These SE activities are guided by SE standardized processes as those of (ISO/IEC 15288, 2008). Overall, SE processes and standardized languages as SysML defined in OMG (2010), are designed as general purpose processes and languages. However according to Arthurs (2008), “practioners want to solve specific problems, so the challenge becomes determining what modelling artefacts to use and how/when to use them efficiently to solve problems”.

Among many mandatory prerequisites such as “system thinking” to understand and to apply SE basics, one of the main is the *problem-solution spaces* partition which must be formally defined to become a SE basic construct. This partition means that any confusion between the roles assigned to any responsible specifying problems and any responsible specifying solutions would be to the detriment of meeting target system objectives (AFIS, 2009).

Section 2 states on the iterative and recursive nature of this *problem-solution spaces* relationship defined as a specification key-artifact for SE right practices. This rationale is applied in section 3 to propose a generic workflow for the RS process. This workflow is tested in section 4 on a case-study related to the protection of human using robot. Section 5, draws a first assessment of this preliminary approach towards the transition to the industrial scale.

2 INTEROPERATION RELATIONSHIP WITHIN A SYSTEMS ENGINEERING PROCESS

This *problem-solution spaces* basic construct is studied in software development and automation domains before being generalized to the SE domain.

2.1 Software development issues

The *problem-solution spaces* interoperation has been studied by Berg et al. (2005) for traceability issues across domains whereas Czarnecki (1998) states that any domain can be divided into a PS and SS. According to Bjørner (2009), “by a domain we shall here understand a universe of discourse, an area of nature subject to laws of physics and study by physicists, or an area of human activity subject to its interfaces with other domains and to nature”.

This interoperation relationship has been the subject of many researches in the field of computer science. Hall et al. (2002) extend the problem frames approach (Jackson, 2001) towards the Twin Peaks model (Fig.1) in order to define the iterative nature of the software development process as an iterative specification between problem and solution structures.

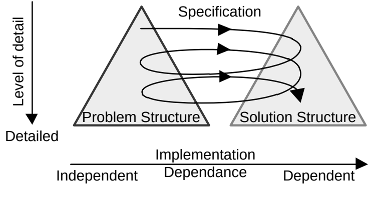

Fig.1. Variation of the Twin Peaks model by Hall et al. (2002)

Another works within the problem frames approach defined precisely the nature of a specification (Gunter et al., 2000). The proposed reference model (Fig.2) gives five artifacts distributed among the environment domain and the system domain: the *world knowledge W* as a description of the relevant environment, the statement of *requirements R*, the *specification S* that mediates between the environment and the machine, the description of the *machine M* and the *program P* which executed on the *machine M* implements the *specification S*. In a more general way, the *machine M* with the *program P* represents the system to be constructed. Jackson (1997) distinguishes two types of description of the environment: *Description optative mood* "expresses a condition over the phenomena of the environment that we wish to make true by installing the machine" and may be associated with *R*; *Description indicative mood* may be associated with *W* and "expresses a condition over the phenomena of the environment that we know to be true irrespective of the properties and behaviour of the machine". The separation between environment and system, according to Gunter et al. (2000) allows the separation of phenomena (states, events, individuals), owned and controlled by the environment e={e_v, e_h} or by the system s={s_v, s_h} (Fig.2).

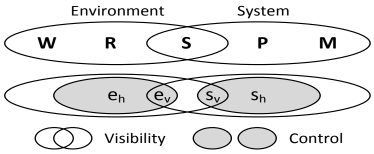

Fig.2. A reference model for requirements and specifications (Gunter et al., 2000)

On the one hand, there are phenomena e_h, e_v and s_v visible to the environment and used in W and R. On the other hand, there are phenomena s_h, s_v and e_v visible to the system and used in P and M. Therefore the specification S is expressed in the common phenomena s_v and e_v, and defined by (1):

$$\forall\ e, s.\ W \wedge S \Rightarrow R \qquad (1)$$

Regarding this with a domain engineering point of view, Bjørner (2010), expressed that " Before Software can be designed we must understand the Requirements, and before Requirements can be expressed we must understand the Domain " and provides a formalization of the specification *S* in Domain *D* (2), where $S \vDash R$ means that S is a model of R:

$$D, S \vDash R \qquad (2)$$

2.2 Automation issues

The above-mentioned interoperation relationship has been also studied in the field of automation in order to revisit the Fusaoka's automatic control synthesis condition (3) (Fusaoka, 1983) arguing that the design of any automation systems consists in prescribing the (unknown) control rules of the (known) dynamics of a physical system from the behavioural (known) goals to be met:

$$\mathit{Dynamics} \wedge \mathit{Unknown\ Control\ Rules} \supset \mathit{Goal} \qquad (3)$$

This weak prescription $\supset$ is more broadly interpreted by Lamboley (2001) as a predicate implemented with a B-method based process:

$$\mathit{Control\ Specification} \wedge \mathit{Pr\ ocess\ Specification} \Rightarrow \mathit{System\ Specification} \qquad (4)$$

To ensure the *a priori* correctness of each term of this automation predicate and their coherence as a whole, Pétin, et al. (2006) propose a model-driven specification approach ensuring the predicate:

$$P_C \wedge P_O \Rightarrow S \qquad (5)$$

where the architecture $P_C \wedge P_O$ of automation P_C of a process P_O must satisfy the specification *S*.

2.3 Systems Engineering issues

Dobre (2010) combines recently these above mentioned works focusing mainly on specialist engineerings in order to take into account the interdisciplinary process required to engineer a system as a whole.

The predicate (5) must be re-formulated according to problem frames approach as:

$$W \wedge P_C \wedge P_O \Rightarrow R \qquad (6)$$

where *W* represents the context of the system-of-interest within the existing *SS* of the domain-of-interest. This interpretation considers that the domain-of-interest (*i*) is partitioned in two *spaces* of *problem* (Ps_i) and *solution* (Ss_i) and generalizes this partition to any domain involved in a collaborative SE process. Each interoperation relationship between a problem space Ps_a and a solution space Ss_b is considered as a contractual *descriptive / prescriptive specification* S_{ab}. The resulting generalized SE specification process applies this rationale iteratively between Ps_a and Ss_b, and recursively (Herzog, 2004) during the whole SE process (Fig.3).

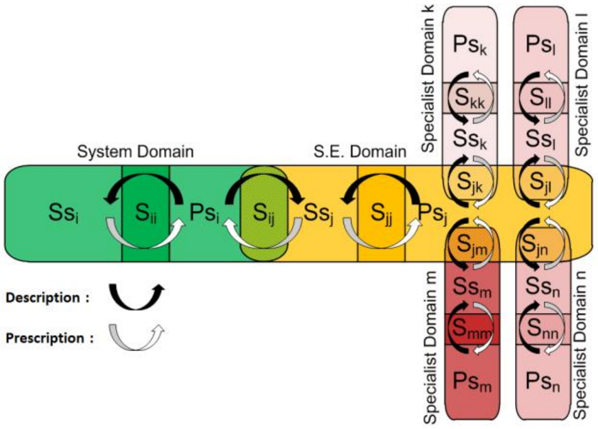

Fig.3. Iterative and recursive specification process

3 ITERATIVE INTEROPERATION RELATIONSHIP FOR REQUIREMENT SPECIFICATION PROCESS

SE process manipulates objects which are structured in metamodels. Among these metamodels, the one proposed by Holt and Perry (2008) places requirements as key objects for SE. Considering requirements, this section focuses on the relationship between system domain (seen as *a PS*) and SE domain (seen as *a SS*). It proposes a workflow, based on the evolution of requirement types and states that we see as an artifact of SE process, to aid the process of RS. According to Caron (2005), a requirement type is defined by a "set of rules characterizing the relationship that the requirement of this type must or may have with other engineering data". Requirement state is defined by the fact that "some of these links are instantiated or not".

As the workflow is built upon various objects (presented in section 3.1) and various structured activities (presented in section 3.2), the language chosen to model it has to be able to express such specificities. Among candidates, high level Petri nets have been chosen because they enable to represent the variety of objects manipulated using colours and indexes, and to model structured activities using hierarchy in places. This workflow has then been formalized with CPN Tools (Jensen et al., 2007).

3.1 Objects used by the workflow

This section defines the various objects used during RS:

- *Stakeholder requirement:* expresses the expected system interactions with its environment;
- *Skill:* is seen as the indicative mood within the meaning of Jackson (1997). It represents the known properties of system environment validated by an expert. Skills are required at each stage of the specification process, and can be for example characteristics of good requirements including characteristics about syntax or semantics (INCOSE, 2010), domain skills for transformation...
- *Optative requirement*: expresses a condition that is to be on the phenomena of the environment domain (Jackson, 1997). An initial optative requirement corresponds to the description of a stakeholder requirement. An optative requirement can be *verified* or *not verified*, according to a verification activity which is detailed in the next section;
- *Transformation*: corresponds to the requirement modification based on SS skills;
- *System requirement*: a requirement which has to be satisfied by the future system, prescribed to PS. A system requirement can have several states: *{Not Prescribed, Not Validated}*, *{Prescribed, Not Validated}* and *{Not Prescribed, Validated}*. These states are defined by prescription and validation activities;
- *Specification*: the set of all system requirements.

A static view of these objects and their relations, due to definition, description, transformation, prescription and validation activities, is formalized as a meta-model presented in (Fig.4). Packages of meta-model represent different repositories to ensure traceability throughout the process of requirement specification.

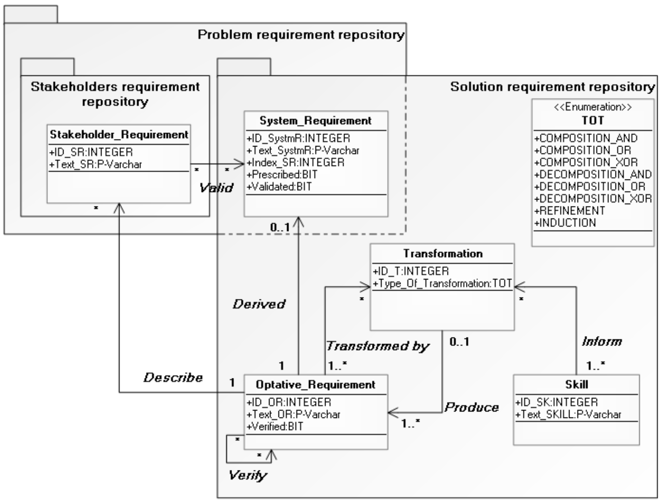

Fig.4. Meta-model used by the workflow

3.2 Iterative workflow between problem and solution spaces

The activities of RS, which compose the proposed workflow and are described below, are modelled by hierarchical transitions in a high level Petri net, while sets of requirements, including repositories, are modelled by places (Fig.5).

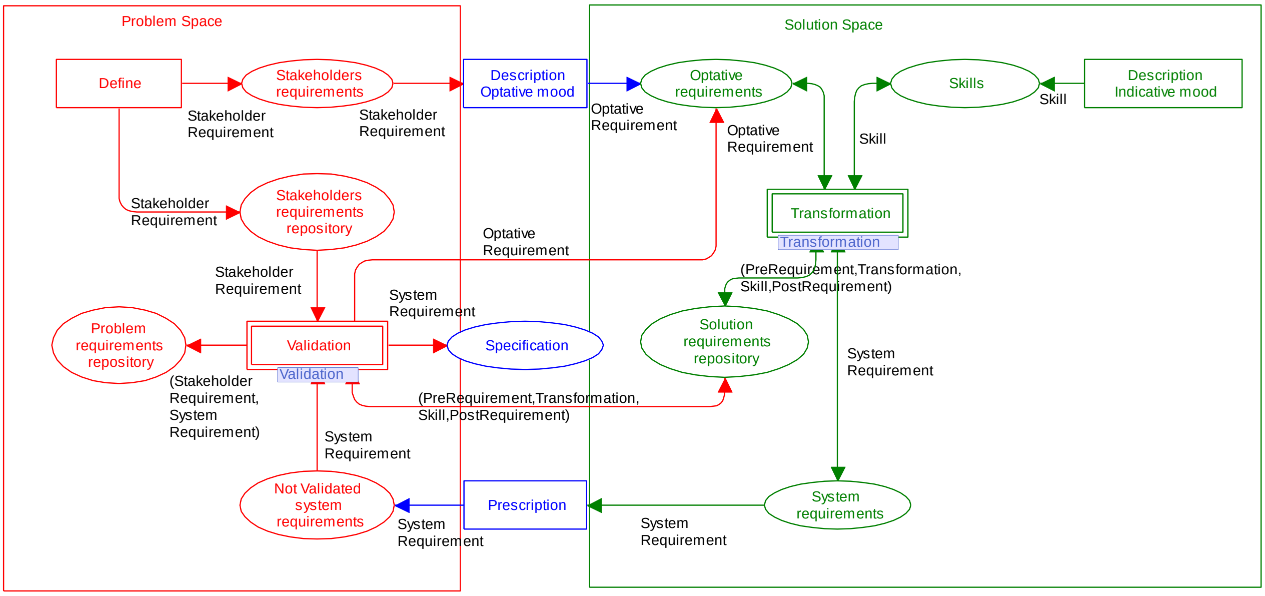

Fig.5. Main structure of the iterative workflow proposed for RS

Requirements and skills are represented using tokens, which colour depends of their type (stakeholder, optative, system …). Indexes are used to uniquely identify them. In this paper only places and transitions related to the workflow are shown. For readability purposes, places allowing the execution of the model have been hidden.

We propose in this section to define the different activities performed during the requirement process specification:
- *Stakeholder requirement definition*: during this activity, PS defines its need in the form of several stakeholder requirements. The definition of a new requirement produces a new couple token (*Stakeholder_requirement(i), ID_SR(i))* in place *Stakeholders requirements*. At this stage, *ID_SR(i)* is redundant information because it represents the identifier of the stakeholder requirement, but it ensures traceability between the stakeholder requirement and the different requirements resulting from transformations of it. Moreover a couple token (*Stakeholder_requirement(i),ID_SR(i))* is stored in place *Stakeholders requirements repository*, which represents a requirement repository ensuring the traceability of stakeholders requirements;
- *Description Optative mood*: description transforms a stakeholder requirement (from PS) into an optative requirement (into SS), but its content is not altered. This change of requirement type expresses that the requirement is addressed in another space. During description, a token (*Stakeholder_requirement(i),ID_SR(i))* becomes a token *(Optative_requirement(i),ID_SR(i))*. Traceability between stakeholder and optative requirements is ensured because they have the same identifier;
- *Transformation*: we have identified four transformation mechanisms: refinement, induction, decomposition and composition. For the last two, it is possible to add a suffix (AND, OR, XOR) to precise the relation between the requirements which are produced or used. For this reason, the transition "Transformation" of (Fig.5) can be decomposed as presented in (Fig.6) and an enumeration of types of transformations is given (Fig.4).

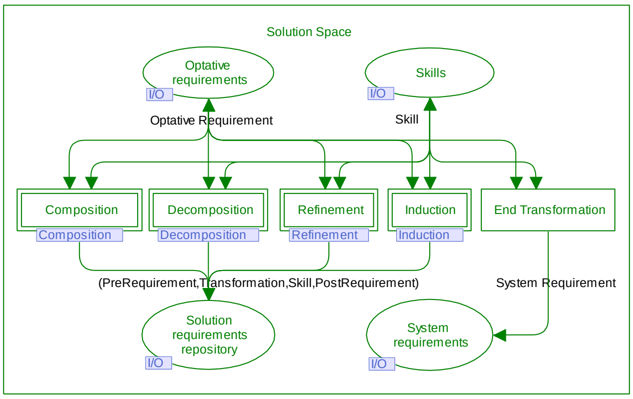

Fig.6. Types of requirement transformations

All these transformations follow a same pattern, using (Pre)requirements and skills to generate (Post)requirements which have to be verified (Fig.7).

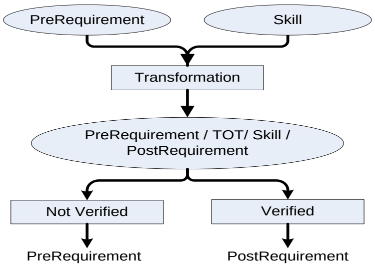

Fig.7. Transformation pattern

Let *R*, *PreR*, *PostR* and *SK* the sets of elements manipulated during transformations:

- $R = \{preR_1, \dots, preR_n, postR_1, \dots, postR_m\}$, $(n, m) \in \mathbb{N}$, the set of optative requirements
- $PreR = \{preR_1, \dots, preR_n\}$, $n \in \mathbb{N}$, the set of optative requirements to transform, $PreR \supset R$;
- $PostR = \{postR_1, \dots, postR_m\}$, $m \in \mathbb{N}$, the set of optative requirements transformed, $PostR \supset R$;
- $SK = \{sk_0, sk_1, \dots, sk_i\}$, $i \in \mathbb{N}$, the set of domain skills. For each transformation skill $sk_i \in SK$ allows to justify the transformation performed.

Considering these sets, the transformations are defined as:
- *Refinement:* the requirement « preR$_n$ » refined with the skill « sk$_i$ » produces the refined requirement « postR$_m$ ». The skill « sk$_i$ » bridges the gap between « preR$_n$ » and « postR$_m$ ».
- *Induction:* the requirement « preR$_n$ » with the skill « sk$_i$ » induce a new requirement « postR$_m$ » while retaining the requirement « preR$_n$ ». « k$_i$ » is a skill from which is induced « postR$_m$ »;
- *Decomposition*: (Decomposition_AND, Decomposition_OR and Decomposition_XOR) the requirement « preR$_n$ » can be decomposed into a set of requirements {postR$_m$,postR$_{m+1}$,…, postR$_{m+n}$} where n+1 represents the number of decomposition, using the skill « sk$_i$ » to justify the decomposition;
- *Composition*: (Composition_AND, Composition_OR and Composition_XOR) the set of requirements {preR$_n$, preR$_o$,..., preR$_x$}» can be composed into a requirement « postR$_m$ », using the skill « sk$_i$ » to justify the composition.

For each transformation, a quadruplet token (PreRequirement, TOT, Skill, PostRequirement) is created. PreRequirement is a *List[Optative_requirement(i), ID_SR(i)]*. TOT is the Type Of Transformation completed, Skill is the *Skill(i)* that enabled the transformation. PostRequirement is a *List[Optative_requirement(j), ID_SR(j)]*.
- *Verification:* after each transformation, the compliance of the set *PostRequirement* with the set *PreRequirement* is verified. If it is unverified, the set *PreRequirement* is returned to the place *Optative requirements*. Otherwise if it is verified the set *PostRequirement* is returned, and a quadruplet token (PreRequirement, TOT, Skill, PostRequirement) is stored in the repository *Solution requirements repository* thereby making the traceability between the requirements of different levels;
- *Transformation End:* after several transformations, the set of optative requirements obtained is, according to the skill (*Skill(i)*), at a system level. Accordingly a token (*Optative_requirement(i),ID_SR(i))* becomes a token *(Sys-*

tem_requirement(i),ID_SR(i)). Traceability between optative and system requirements is ensured because they have same identifiers;

- *Prescription:* during this activity, system requirements issued from a same stakeholder requirement are simultaneously prescribed to *PS* for validation. The statement of requirements remains the same but the requirements change from state *Not Prescribed* to *Prescribed*;
- *Validation:* during validation, the client validates all systems requirements issued from a same stakeholder requirement, considering the links between them which are stored in the *Solution requirements repository* (Fig.8). This is to ensure that the *SS* clearly understands and expresses the stakeholder requirements, and that the compromises made are acceptable. If it is not validated, a token (*Optative_requirement(i), ID_SR(i))* is returned in place *Optative requirement.* This optative requirement corresponds to the stakeholder -requirement described (*Stakeholder_requirement(i),ID_SR(i))* from which the validation was performed. If it is validated, the set of system requirements *(System_requirement(i), ID_SR(i))* is sent to the place *Specification*, and changes from *Not Validated* to *Validated.* Moreover, a couple token (List[Stakeholder_requirement(i)], List[System_requirement(i)]) is stored in *Problem requirement repository*. This token links stakeholder requirements described and system requirements prescribed as answers.

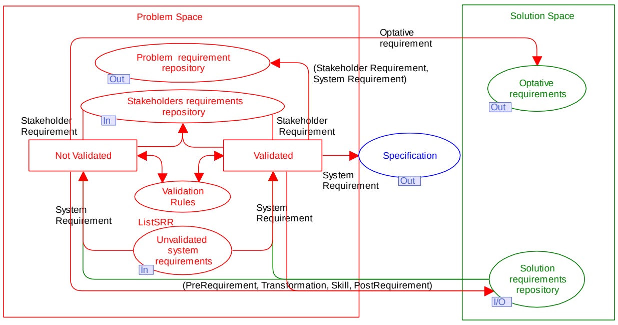

Fig.8. System requirements validation

4 WORKFLOW APPLICATION ON A CASE STUDY

The proposed workflow is illustrated on a case study from the training center AIP-Primeca Lorraine (http://www.aip-primeca.net). To be compliant with the French labour code, the AIPL develops and implements security systems to prevent risky situations. For example, the power supply of the 6-axis articulated robot should be cut if a door of its protective enclosure is open.

In the *PS*, this need can be defined by a stakeholder requirement: *Stakeholder_requirement(1):* "Robot power supply must be cut if a door of the protective enclosure is open". A token *(Stakeholder_requirement(1),[ID_SR(1)])* is put in place *Stakeholders requirements* and is also stored in the repository *Stakeholders requirements repository.*

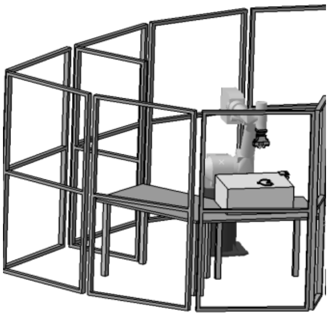

Fig.9. 6-axis articulated robot with a protective enclosure

Firstly, the PS describes to the SS the couple *(Stakeholder_requirement(1),[ID_SR(1)])* that becomes the couple *(Optative_requirement(1),[ID_SR(1)]).* Secondly, a *Skill(1):* "The robot is equipped with a power input board with two inputs FN1 and FN2 on which is connected a switch installed on the door. They enable to cut the power supply of the robot if the switch is open" informs that *Optative_requirement(1)* can be refined into a new requirement *Optative_requirement(2)* "Security access door enslavement must be connected with inputs FN1 and FN2 of the power input board". Considering the pattern of transformation previously defined, we have {*PreRequirement = [(Optative_requirement(1), [ID_SR(1)])], TOT = Refinement, skill = Skill(1), PostRequirement = [(Optative_requirement(2), [ID_SR(1)])]*}. This is presented by place *"3-PreRequirement/TOT/Skill/PostRequirement"* of (Fig.10), in which the contained token means that the *Optative_requirement(1),* in connection with the stakeholder requirement of index *ID_SR(1),* can be refined thanks to *Skill(1)* to produce a new requirement *Optative _requirement(2)* in connection with the stakeholder requirement of index *ID_SR(1)*. Once completed the *Refinement*, the correctness of the transformation is checked using verification rules. Considering that the transformation is correct, a new *Optative_requirement(2)* is obtained. Moreover a new token *([(Optative_requirement(1), [ID_SR(1)])], Refinement,Skill(1),[(Optative_requirement(2), [ID_SR(1)])], [ID_SR(1)])* is put in place "*Solution Requirement Repository"*, to ensure traceability between *Optative_requirement(1)* and *Optative_requirement(2)* according to the *Skill(1).*

After transformation, a token (*Optative_requirement(2), ID_SR(1))* is in place *Optative requirements*. A skill, *Skill(2):* "Sensors are in the interface between the system to design and the environment represented by the robot" defines that this requirement is at system-level, accordingly transformations can be stopped, and this optative requirement becomes a system requirement (*System_requirement(2), ID_SR(1)).* Once the *SS* has defined system requirements to answer all initial optative requirements, they are prescribed to the PS for validation. Thus, *(System_requirement(2), ID_SR(1))* is prescribed to *PS*, and is finally validated with respect to the stakeholder requirement *(Stakeholder_requirement(1),ID_SR(1))* from which it comes. Then, *(System_requirement(2), ID_SR(1))* is placed in place "*Specification*". Finally, a couple token *([Stakeholder requirement(1)], [(System_requirement(2), ID_SR(1))])* is stored in place *"Problem Requirement Repository"* ensuring traceability between stakeholder requirement and system requirement.

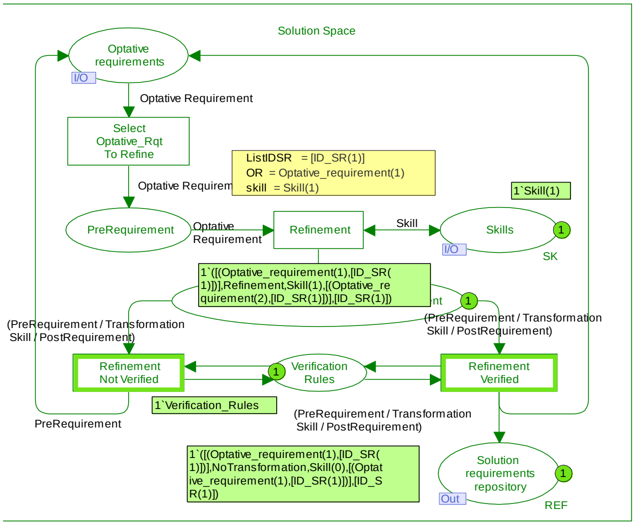
Solution Space
Optative requirements
I/O
Optative Requirement
Select Optative_Rqt To Refine
ListIDSR = [ID_SR(1)]
OR = Optative_requirement(1)
skill = Skill(1)
Optative Requirem
1`Skill(1)
PreRequirement
Optative Requirement
Refinement
Skill
Skills
1
I/O
SK
1`([(Optative_requirement(1),[ID_SR(1)])],Refinement,Skill(1),[(Optative_requirement(2),[ID_SR(1)])],[ID_SR(1)])
nt
1
(PreRequirement / Transformation Skill / PostRequirement)
(PreRequirement / Transformation Skill / PostRequirement)
Refinement Not Verified
1
Verification Rules
Refinement Verified
1`Verification_Rules
PreRequirement
(PreRequirement / Transformation Skill / PostRequirement)
1`([(Optative_requirement(1),[ID_SR(1)])],NoTransformation,Skill(0),[(Optative_requirement(1),[ID_SR(1)])],[ID_SR(1)])
Solution requirements repository
1
Out
REF

Fig.10. Refinement of requirement Optative_requirement(1) from Skill(1)

5 CONCLUSIONS AND PERSPECTIVES

Although these works are preliminary, well formulating basic SE constructs is of importance for engineering use as well as for training purposes. As example, SysML, the de-facto domain-independent system modelling language defined in OMG (2010), remains controversial for Model Based Systems Engineering domain dependent because of its too general semantics. Applying our approach leads to improve this semantics for specification issues by manipulating modelling objects (dependencies, boundary-box,...) based on defined constructs as optative-indicative moods, problem-solution spaces, World/Domain which turn out to be already efficient in practice within the requirement definition process. Current work aims to enrich SysML metamodel by stereotyping the proposed constructs and by developing new system modelling objects.

Transition to more operational context is on-going in order to prove the relevance of the proposed workflow at the industrial scale. First application is related to railway embedded control system, in order to formally control passenger access through train door. A second application is related to the formal specification of a computer-aided device to improve human-based plant operation (Dobre, 2010).

6 REFERENCES

AFIS (2009), Discover and understand systems engineering (v3), Association Française d'Ingénierie Systèm (in french).

Arthurs, G (2008), Model-Based System Engineering, Elements for deploying an Efficient Development Environment, Telelogic White paper, IBM Company.

Berg, K., Bishop, J. (2005). Tracing Software Product Line Variability – From Problem to Solution Space. 2005 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries, White River, South Africa.

Bjørner, D. (2009). *From Domains to Requirements. On a Triptych of Software Development*. www.complang.tuwien.ac.at/bjorner/book.pdf

Bjørner, D. (2010). Domain engineering. In *Formal Methods;State of the Art and New Directions*, P. Boca, J. P. Bowen, and J. I. Siddiqi, Eds. Springer-Verlag, London, pp. 1-41, ISBN 978-1848827356.

Caron, F. (2005), Collaborative management of engineering data system, *Génie logiciel*, 75, pp. 2–6, (in french).

Czarnecki, K. (1998), Generative programming. Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models, PhD thesis, Technical University of Ilmenau.

Dobre, D. (2010), Contribution to the modelling of an interactive system driving assistance of an industrial process, PhD thesis (in french), Nancy University.

Fusaoka, A., Saki H., Takahashi, A. (1983). Description and reasoning of plant controllers in temporal logic. International Joint Conference on Artificial Intelligence. Karlsruhe 8-12/09, pp. 405-408

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

Hall, J.G., Jackson, M.A., Laney, R.C, Nusbeibeh, B., Rapanotti, L. (2002). Relating Software Requirements and Architectures using Problem Frames, IEEE RE 2002.

Herzog, E. (2004), An approach to systems engineering tool data representation and exchange, PhD thesis, Linköping University.

Holt, J., Perry, S. (2008). *SysML for Systems Engineering Using a Model-Driven Development Approach*. White Paper, I_Logix, Andover, MA.

INCOSE (2010), *Systems Engineering Handbook : a guide for system life cycle processes and activities (v 3.2.1)*, International Council on Systems Engineering.

ISO/IEC 15288 (2008). *Systems and software engineering – System life cycle processes*. International Organisation for Standardization.

Jackson, M. (1997), The meaning of requirements, *Annals of Software Engineering*, 3 (1), pp. 5–21.

Jackson, M. (2001), *Problem Frames: Analysing & Structuring Software Development Problems*, ISBN 020159627X

Jensen, K., Kristensen, L.M., Wells, L. (2007), Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems, *International Journal on Software Tools for Technology Transfer*, 9(3), pp. 213–254.

Lamboley, P. (2001), Production systems automation formal method proposal, PhD thesis (in french), Nancy University.

OMG, (2010), OMG Systems Modeling Language (OMG SysML) (v1.2).

Pétin, J.-F., Morel, G., Panetto, H. (2006), Formal specification method for production systems automation, *European Journal of Control* 12 (2), pp. 115-130.

Pyster, A., Olwell, D., Squires, A., Hutchison, N., Enck, S., Eds. (2011) *A Guide to the Systems Engineering Body of Knowledge (SEBoK). Version 0.5*. Stevens Institute of Technology, Hoboken, NJ, USA.