HAL open science

Requirements formalization for systems engineering: an approach for interoperability analysis in collaborative process model

Sihem Mallek, Nicolas Daclin, Vincent Chapurlat, Bruno Vallespir

To cite this version:

Sihem Mallek, Nicolas Daclin, Vincent Chapurlat, Bruno Vallespir. Requirements formalization for systems engineering: an approach for interoperability analysis in collaborative process model. Workshop IWEI International IFIP Working Conference on Enterprise Interoperability 2013, Mar 2013, Enschede, Netherlands. pp.243-257. hal-00804283

HAL Id: hal-00804283

https://hal.science/hal-00804283v1

Submitted on 20 Jun 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Requirements formalization for systems engineering: an approach for interoperability analysis in collaborative process model

Sihem Mallek[1], Nicolas Daclin[1], Vincent Chapurlat[1], Bruno Vallespir[2],

[1] LGI2P - Laboratoire de Génie Informatique et d'Ingénierie de Production
site de l'Ecole des Mines d'Alès, Parc Scientifique Georges Besse,
F30035 Nîmes Cedex 5, France
{Sihem.Mallek, Nicolas.Daclin, Vincent.Chapurlat}@mines-ales.fr

[2] Université Bordeaux, IMS, UMR 5218, F – 33400 Talence, France
Bruno.Vallespir@ims-bordeaux.fr

Abstract. In the field of requirement's engineering, writing requirements is a fundamental stage. Indeed, requirements must be (1) written correctly by the user, (2) relevant to the studied domain and (3), verifiable. In this research work, the studied domain is related to the verification of interoperability requirements on a collaborative process model by formal verification techniques. In this case, it is necessary to offer to a user the mean to write its own interoperability requirements easily, correctly and human readable in order to be re-writing into properties and to allow their verification. Precisely, this communication presents the mapping between interoperability requirements expressed in SBVR into properties expressed in TCTL.

Keywords: interoperability requirements, formalization, rules mapping, SBVR, UPPAAL.

1 Introduction

In Systems Engineering domain (SE) [1] as in any more specific engineering domain (mechanical, information systems, mechatronic…), requirements description, analysis and verification are crucial activities.

On the one hand, a requirement results from stakeholders' expectations and prescriptions *i.e.* from stakeholders' needs analysis. It can be also induced by technical, technological or organizational choices made by designers. So, a requirement fixes without ambiguities, in a coherent way, and even constraints, what designers have to respect when designing a solution. First, in order to describe a requirement, these designers can be helped by standards *e.g.* [2] and [3] that propose reference models and reference vocabularies. In the same way, they promote splitting up requirements into various categories. There are based classically on the distinction between functional requirements (what a system S has to do?) and non-functional requirements (how this must be done?). However, the resulting vocabularies and

requirements check lists commonly adopted in the SE domain are more or less perfectible when taking into account interoperability problematic.

On the other hand, a requirement can evolve, being refined or decomposed all along the System Engineering project. These refinements or decompositions are requested, and then done by designers from various domains having different points of view about the system under design. So, they require various domains vocabularies having to be coherent for describing a requirement.

Last, design activities are considered generally as model based or model guided activities [4]. Indeed, designers' work is oriented on modeling and analysis activities of the obtained models. However, verification, and even partial validation, activities of these models are required for two reasons. First, it is necessary to assume the quality of any model of a system S under design before performing any analysis. For instance, analysis can consist to simulate the behavior of S in order to evaluate the performance and the relevance of a given communication protocol between S and another system. Second, when verified, a model has to be used in order to check if a given part of the requirements is really respected [5]. This is classically the aim of verification and validation activities. The goal is to prove and to justify that the system meets these requirements based only on models of this system we have. There is obviously no magic bullet to ensure completeness of modeling and then of the requested proofs. However, it should help designers to ensure that at least some of these requirements are met.

The research work herein presented focuses on writing, verifying, partially validating and justifying requirements in a model based design environment by using various formal techniques *e.g.* those promoted by [6]. This communication applies and illustrates a research work in order to help designers for the writing and proving interoperability requirements on a sociotechnical system. This one is a collaborative process aiming to involve various partners from various business domains, to share activities, data and time. By assumption, in this case, interoperability requirements allow us to focus precisely on issues dreaded but often implied a non-interoperable partner involved in a collaborative process will sooner or later lead to malfunctions and interfacing problems both from technical, organizational or also human points of view.

As a consequence, this paper is organized as follow. Section 2 presents the problematic of writing interoperability requirements and related state of the art. Section 3 presents our proposed approach to write correctly and easily interoperability requirements and its re-writing to make their verification possible. Section 4 gives a case study in order to illustrate the re-writing of interoperability requirements.

2 State of the art and problematic

From [7], four categories of interoperability requirements have to be considered (compatibility, interoperation, autonomy and reversibility). In this study, an interoperability requirement can be qualified as a-temporal *i.e.* it is independent of time and has to be verified in all steps of system behavior. Conversely, it can be qualified as temporal *i.e.* it is dependent of temporal hypotheses and has to be verified

only at some stages of the system life cycle. The problem addressed in this article is triple that means it is necessary to ensure that:

1. **Interoperability requirements are well written *i.e.* written correctly.** By hypotheses, interoperability requirements can be form heterogeneous nature (functional as non-functional), from various origins (for whom or what purpose has this requirement is written?) and can appear, sometimes change or even become obsolete. For these reasons, natural language is often preferred and classically used for writing them rather than using formal language such as logic which can be difficult to understand by all end users. This induces, however, classical rewording problems and ambiguities. At the opposite, various business vocabularies and their respective semantics can be employed to abandon natural language. These vocabularies help designers not only to write the requirement itself, but also to decompose or refine it from a more relevant and formal manner into a set of sub-requirements. There exist various approaches allowing to handle these vocabularies. We can site mental maps and guided interviews among the more non formal ones. KAOS method [8], Boiler plates approaches [9], Use Case Map notation [10], standardized requirements check lists [3] or the REGAL approach [11] are now recognized as good methodologies. Last, it seems more adequate here to use standards and implanted tools which remain conform to these standards. We propose then to use Semantic Business Vocabulary Rules (SBVR) [12] but let's mention also URN [13] or GRL [14] as potential candidates.
2. **These requirements are good and relevant** when regarding the system to be designed. In this way, requirements have to be expressed according to all concepts used to describe the system to analyze. Thus, it is necessary to ensure that the used language to write requirements is sufficient, complete and correctly built to guide the user in the writing process of its requirements.
3. **The system model can allow to check all or part of these requirements** taking into account their nature (functional or non-functional) and other characteristics. It is proposed here to make a mapping between the used language to describe requirements and the used verification technique to analyze them.

This approach is applied, in order to guide designer in the definition and the expression of interoperability requirements. Precisely, the final goal is to allow verification of interoperability requirements in a collaborative process model using formal verification techniques. In this case, the collaborative process is modeled using the BPMN 2.0 language [15] and has to be translated - using model transformation rules -into an equivalent model upon which the formal verification techniques can be applied as presented in [7]. In this way, one of the used formal verification techniques is based on model checking [16] using the model checker UPPAAL [17] to verify temporal interoperability requirements. As a consequence, it is necessary to write these interoperability requirements into properties with TCTL (temporal logic used by UPPAAL). However, it is difficult to ensure compliance and quality of expression of a property. In addition, the user must have the mastery explicit representation languages properties. As a consequence, it is proposed in this paper (1) to allow the user to express correctly its requirements in a readable language, (2) to establish mapping rules to re-write correctly these interoperability requirements into TCTL properties and facilitate the use of our tool for the end user as presented in the following figure.

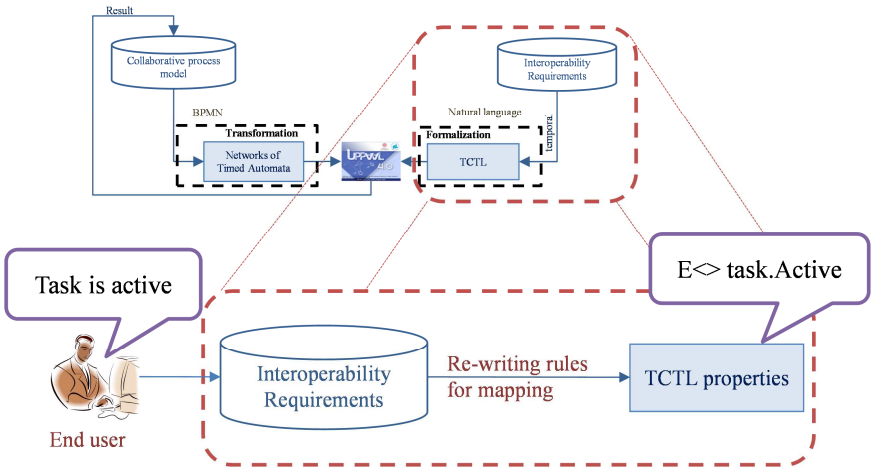

Fig. 1. From the expression of an interoperability requirement to the corresponding TCTL property to verify.

Based on these hypotheses, next sections present the proposed approach of re-writing interoperability requirements expressed in structural language into TCTL based on the use of mapping rules.

3 Interoperability requirements writing

As mentioned previously, it is necessary to help stakeholders to write their interoperability requirements with a simplified language such as natural language. So, it is important to offer a language readily understandable and easily accessible for each stakeholder. This research work is based on the use of SBVR (Semantic of Business Vocabulary and Rule) [12]. In fact, SBVR "*defines the vocabulary and rules for documenting the semantics vocabularies, business facts and business rules for the interchange of business vocabularies and business rules among organizations and between software tools*". SBVR is based on natural language allowing to write requirements easier rather than with a more formal language – such as temporal logic as presented in [18].

The SBVR language allows to define a limited but sufficient vocabulary to write rules and to ensure, further, their verification supported by a verification technique. Thus, this vocabulary has to be made in accordance with the studied domain. In our case the vocabulary is based on:

1. The modeling language used to model the collaborative process, *i.e* BPMN [15]. Thus, the proposed vocabulary has to allow considering all concepts included in BPMN (e.g. task, resource, event…). For instance, each BPMN

object and its attributes are well considered in the *BPMNVocabulary* as "*terms*".

2. The verification technique *i.e.* UPPAAL language [17] used to model the process behavior as well as properties. Thus, the vocabulary has to allow to consider all the automaton and their states that corresponds to the behavior of each BPMN object (*e.g.* task is in state "Working", resource is in state "Active"…). As an example, states of an automaton are described as in *UPPAALVocabulary* as "*verbs*".
3. Interoperability concepts that represent all concepts that are not proposed in the previous defined vocabularies but that allow to write an interoperability requirement [19] (aptitude, is_less_than, authorization…). Depending of the nature of the interoperability concepts, they can be either a "*term*" or a "*verb*". For instance, the interoperability concept "aptitude" is an attribute added to enrich the BPMN language, thus, it is *de facto* a "*term*". Furthermore, the interoperability concept "*is_less_than*" represents an operator and so a "*verb*" in *InteroperabilityVocabulary*.

As a consequence the proposed vocabulary can be formalized as shown in the following formula:

$$InteroperabilityRequirementVocabulary = \{BPMNVocabulary, UPPAALVocabulary, InteroperabiltyVocabulary\} \quad (\mathbf{1})$$

Based on this vocabulary, a user (*e.g.* stakeholder) can write an interoperability requirement which can be verified on a collaborative process model. For instance, the simplified interoperability requirement "*it is possible that a task is working and a resource is active*" is built based on *BPMNVocabulary* (task, resource) and *UPPAALVocabulary* (working, active).

Once the vocabulary is defined, and the SBVR rule is written, it is necessary to re-write the rule into a property expressed in the target property verification language (TCTL in our case). In order to perform this step, it is mandatory to dispose of the SBVR syntax to write an SBVR rule, to dispose of the TCTL syntax to write TCTL property, and finally to establish mapping rule between SBVR rule and TCTL property.

In this way, the following sections describes (1) the syntax of SBVR rules, (2) the syntax of TCTL property and (3) the mapping rules of re-writing at the current stage of our research work.

3.1 SBVR rules syntax

SBVR defines an SBVR rule such as "*a rule always tends to remove some degree of freedom*" [12]. The advantage to write a rule with SBVR is that it allows to write a requirement that follow good practices such as SMART [20]. Therefore, SBVR rules are based on "*facts*", and "*facts*" are based on "*terms*". The syntax of an SBVR rule is described in this section, without going into details (for more details, reader may wish to refer to [12]). Formally, an SBVR rule can be written such as:

$$Rule ::= modality?\ p \quad (\mathbf{2})$$

Where:

$$Modality = \{necessity, possibility, contingency, obligation, permission, optionally\} \quad (3)$$

and:

$$p=\{fact_i, quantifiers_j, logicalOperation_k, keyword_l\}, i \in [1, n], j, k, l \in [0, m], n, m \in \mathbb{N}^+ \quad (4)$$

with:

- *fact* ∈ *Fact*, *Fact* ⊂ *InteroperabilityRequirementVocabulary*
- *quantifier* ∈ *Quantifier*, *Quantifier* = *{each, some, at least, at most, exactly, at least n and at most n, more than one}*
- *logicalOperation* ∈ *LogicaOperation*, *LogicalOperation* = *{it is not the case that, and, or, but not both, if p then q, if and only if, neither, nor, whether or not}*
- *keyword* ∈ *Keywords*, *Keywords= {the, a, an, another, a given, that, who, of, what}*

Following the previous defined SBVR rule syntax, the Fig. 2 describes a given rule that is conform to the SBVR syntax.

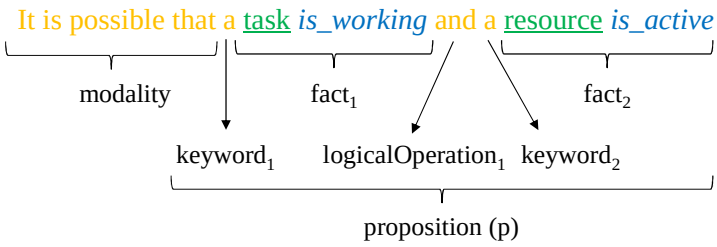

Fig. 2. Syntax of SBVR rule.

3.2 TCTL property syntax

A property in UPPAAL is written using a fragment of the TCTL logic [17]. Therefore, the TCTL syntax is presented in this section without going into details (for more details, reader may wish to refer to [17]). Formally, a TCTL property can be written respecting the following syntax.

$$Property ::= quantifier\ p \mid p \rightarrow q \quad (5)$$

Where:

$$Quantifier = (pathQuantifier, temporalOperator) \quad (6)$$

With:

- *pathQuantifier ={[], <>}*

- *temporalOperator* = {∃, ∀}
- *leadTo* = {→}

and

$$p, q = expression \tag{7}$$

An expression (p) is written according to existing automaton and their states and variables such as presented in the following figure with the automaton of a task which has 4 states (Waiting, Start, Working and Stop) and two variables (timeMin and timeMax) used in a clock T.

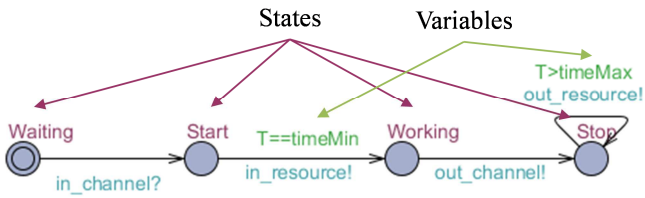

Fig. 3. Task automaton in UPPAAL.

Following the previous defined TCTL property syntax, the Fig. 4 describes a given property using TCTL logic.

$$\underbrace{\text{E<>}}_{\text{quantifier}}\ \underbrace{\text{task.Working and resource.Active}}_{\text{expression (p)}}$$

Fig. 4. Syntax of TCTL property in UPPAAL.

3.3 Mapping rules from SBVR rule to TCTL property

According to the syntax previously defined (SBVR rule syntax and TCTL property syntax), it is possible to observe that an SBVR rule and a TCTL property have both a proposition/expression which can be precede by a modality in the case of an SBVR rule and by a quantifier in the case of TCTL property. In this way, it is proposed to develop two mappings to re-write SBVR rules into TCTL property. Thus, the first mapping allows to re-write modality into quantifier if this one exists or if it is possible. Then, the second mapping allows to re-write SBVR rule proposition into TCTL property expression.

For the first mapping, it is possible to highlight that the modality and the quantifier are issued from the modal logic [21]. From this consideration and the definitions of modalities from SBVR rules and quantifiers from TCTL properties, it is proposed to develop the following mapping as presented in the Table 1.

Table 1. Mapping between SBVR modality and TCTL quantifier.

SBVR modality	Definition	TCTL quantifier	Definition
It is necessary that	Necessity. The meaning of its embedded logical formulation is true in all possible worlds	A<>	Inevitable. The proposition will inevitably become true.
It is obligatory that	Obligation. The meaning of its embedded logical formulation is true in all acceptable worlds	A[]	Invariantly. The proposition is true in all reachable states.
It is permitted that	Permission. The meaning of its embedded logical formulation is true in some acceptable worlds.	E[]	Potentially. The proposition is potentially always true.
It is possible that	Possibility. The meaning of its embedded logical formulation is true in some possible worlds.	E<>	Reachability. It is possible to reach a state in which the proposition is satisfied

Furthermore, an SBVR proposition can be written using *keywords*, *logicalOperations* and *facts* that are based on *terms* and *verbs*. As presented previously, *interoperabilityRequirementVocabulary* was defined according to (1) BPMN language, (2) the used verification technique (UPPAAL) and (3), the interoperability concepts. In this way, in order to perform the second mapping, a given *term* corresponds to an automaton or variable. Furthermore, a *verb* corresponds to a state of an automaton or an operator. The reason of this mapping is related to the transformation from BPMN to a Network of Timed Automata where all concepts from BPMN related to the behavior of the process are transformed into automaton such as (resource, task…) with several states representing the evolution of the process. Finally, we define that a *logicalOperator* corresponds to a logical operator into TCTL as presented Table 2. It is to note that *keywords* are not considered in the following mapping.

Table 2. Mapping between SBVR proposition and TCTL proposition.

SBVR proposition	TCTL proposition
term	automaton or variable
verb	state or operator
logicalOperator	logical operator

For instance, according to the presented syntaxes of SBVR rule and TCTL property, a mapping can be done to re-write the following interoperability requirements "*It is possible that a task is_working and a resource is_active and a clock is_less_than timeMax and clock is_greater_than timeMin*". This requirement is expressed as an SBVR rule based on a defined vocabulary and can be re-written into TCTL such as "*E<> task.Working ans resource.Active and T<timeMax and T>timeMin* ". Indeed, considering and respecting the proposed mapping the modality is re-written into a quantifier and the SBVR rule proposition is re-written into a TCTL property expression as presented in the following figure.

	SBVR rule decomposition (source)	TCTL property decomposition (target)	
modality	It is possible that	E<>	quantifier
proposition (p)	task (term)	task. (automata)	expression (q)
	is_working (verb)	Working (state)	
	and	and	
	resource (term)	resource. (automata)	
	is_active (verb)	Active (state)	
	clock (term)	T (variable)	
	is_less_than (verb)	< (operator)	
	time_max (term)	timeMax (variable)	
	is_greater_than (verb)	> (operator)	
	time_min (term)	timeMin (variable)	

Fig. 5. Mapping between an SBVR rule and a TCTL property.

The proposed mapping, in its current stage, is limited to re-write an SBVR rule into a TCTL property. In the case of another expression of an SBVR rule, this one can be verified with another verification tool such as, for example, COGITANT [22] based on Conceptual Graphs [23] in the case of a-temporal requirements or with the expertise for requirements that cannot be verified using formal verification techniques. In fact, the objective of this research work, in a first time, is to ensure the verification of 15-20% of interoperability requirements thanks to formal verification techniques.

To illustrate the proposed approach, an application case is given in next section to re-write interoperability requirements expressed in SBVR rules into TCTL to make their verification possible using model checker UPPAAL.

4 Application case: drug circuit process

The drug circuit is a critical process (*e.g.* it is mandatory to provide the right drug to the right patient in time and in right measure) inside a hospital. Although this process seems simple, its good execution depends primarily on good interactions among its participants and precisely interactions between resources used by the process. Thus, this process has to closely involve stakeholders in enhancing pharmacy practices and strengthening the role of the Medicine Committee (care unit). A drug circuit is typically composed of three main steps such as prescription, dispensation, and administration performed by both the care unit and the pharmacy following several tasks. These tasks and their interactions on the drug circuit are modeled thanks to a BPMN 2.0 modeler, as represented in the following figure.

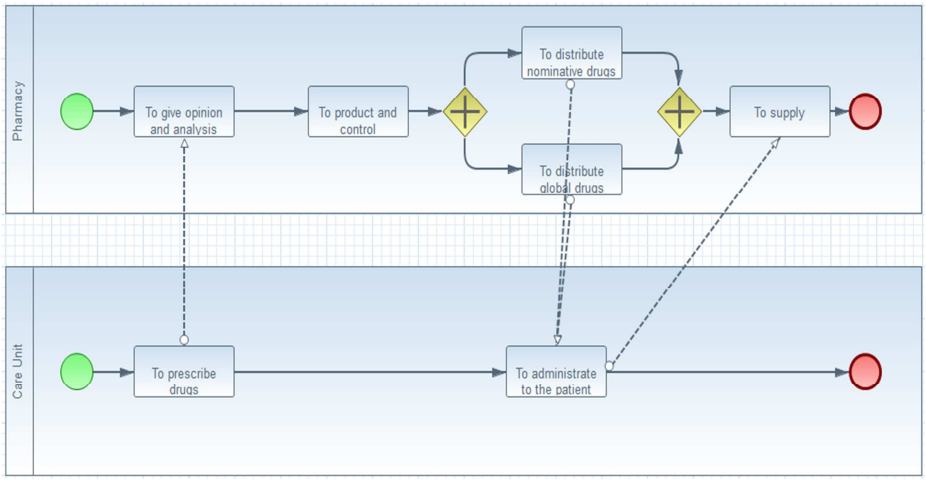

Fig. 6. Drug circuit process modeled with BPMN 2.0.

To demonstrate the usability of this approach, it is proposed here to verify the following temporal interoperability requirements expressed by the user into natural language.

1. *"Medical practitioner is available to write a medical prescription when it is required."*
2. *"Nurses confirm to the pharmacy the administration of drugs to the patient."*

To make verification of these requirements possible, it is proposed, (1) to write them into SBVR rules in order to (2) re-write these ones into TCTL properties respecting the presented mapping rules.

Let us consider the first interoperability requirement. This requirement considers the task "To prescribe drugs" and the resource "Medical practitioner". As a consequence, the end user can easily write or choose (into an interoperability requirement repository) an SBVR rule which corresponds to its initial requirement. Thus, the corresponding SBVR rule is expressed such as: "It is possible that a task *is_starting* and a resource *is_available*". Furthermore, the proposed approach offers the possibility to instantiate a rule with the consideration of elements that are present in the process. This one is then instantiated with the considered task and resource. For instance, it can be instantiated such as "*It is possible that a* ***To prescribe drugs*** *is_starting and a* ***MedicalPractitioner*** *is_available*".

In the same way, the second requirement is expressed with the following SBVR rule: "It is possible that if a task *is_stop* then a task *is_starting*". Then, it is instantiated with the corresponding tasks: "To administrate to the patient" and "To supply".

After the instantiation, the mapping can be done to get TCTL properties that can be verified using UPPAAL on the equivalent behavior model as a Network of Timed Automata. For information, the previous SBVR rules are written into TCTL properties such as:

1. "E<> Topresicibedrugs_.Start and Medicalpractitioner.Available"

2. “E<> Toadministratetothepatient_.Stop imply Tosupply_.Start”

The different steps of the approach supported by an application tool are presented in the following figure.

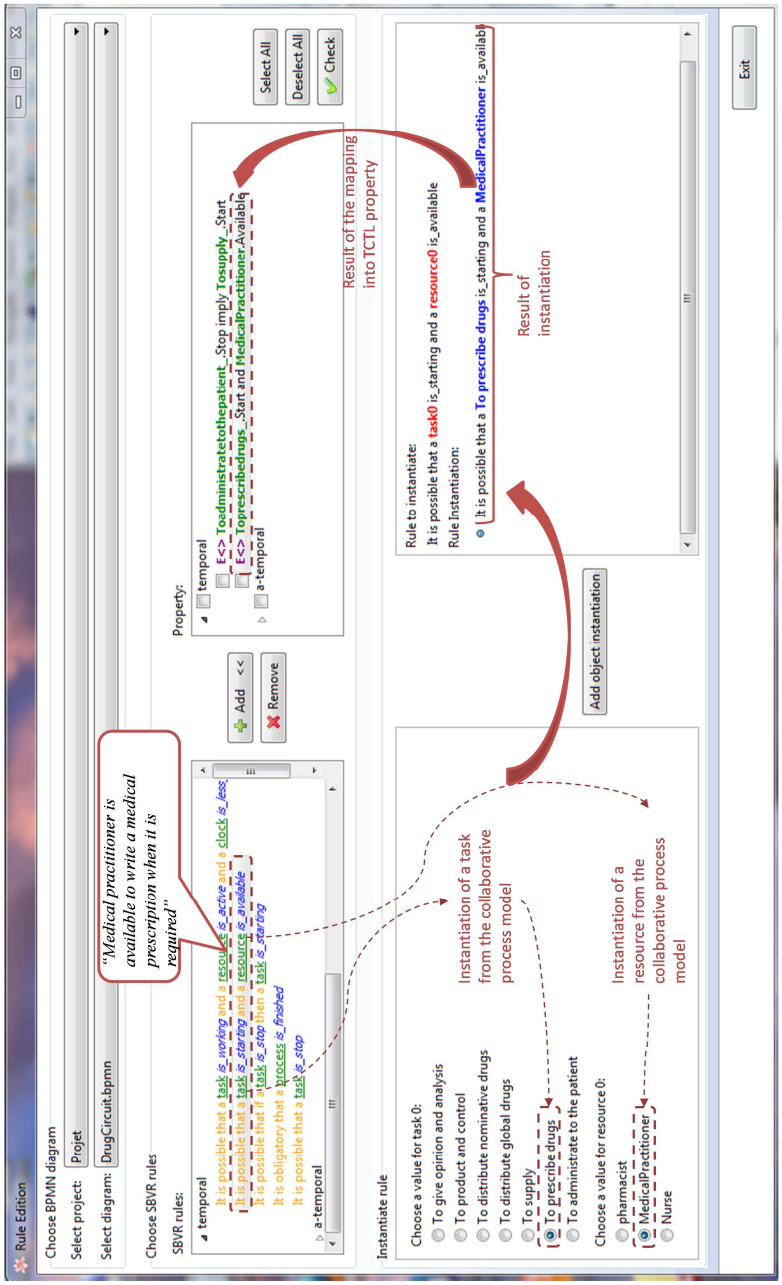

Fig. 7. From the instantiation of SBVR rules to the equivalent TCTL properties to verify.

Finally, the verification (checking task) of both requirements can be done and gives the result that the two properties are satisfied by the collaborative process model as presented on the following figure.

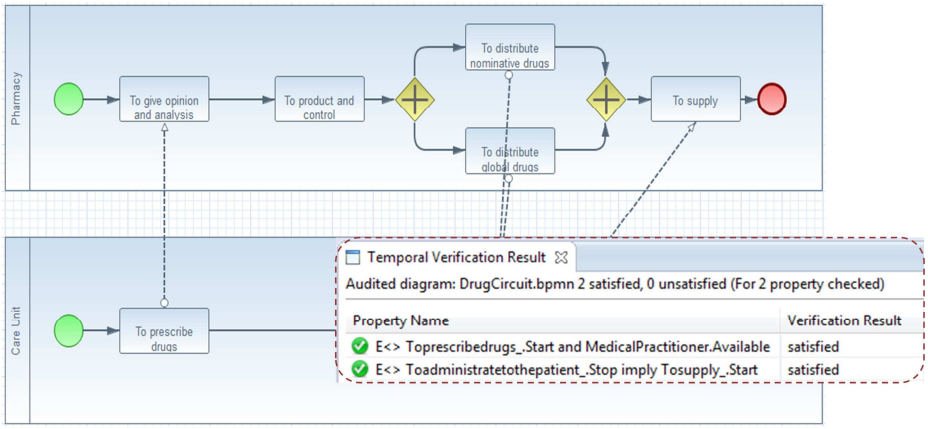

Fig. 8. Results of TCTL properties verification

Finally, it is to note that the mapping and the verification steps are not visible to the end user.

5 Conclusion

In Systems Engineering domain, requirements engineering is one of the most important step and precisely in writing requirements. Indeed, before any conception of a system, it is necessary to dispose of requirements that are (1) well written, (2) good and relevant to the studied domain and (3) verifiable. In our research work, these characteristics are applied in the field of collaborative process analysis to verify interoperability requirements.

One of the most challenges is to allow the users - who do not necessarily have knowledge about the language used by the formal verification techniques - to write interoperability requirements correctly. The use of a language such as SBVR should help and guide these users in writing their requirements and overcome the problems of ambiguity, redundancy and inconsistency ... The presented approach, in its present development, provides a set of interoperability rules that are consistent with a vocabulary which currently includes approximately one hundred "*terms*" and "*verbs*". This approach also allows (1) to instantiate the interoperability rules according to the studied collaborative process model and (2) to re-write them - using mapping rules established and demonstrated in this communication - into properties in order to make their verification possible using UPPAAL. Currently the audit coverage of verification using formal verification techniques is 20% of all identified interoperability requirements.

Although the proposition of a set of interoperability rules (with SBVR), that the user can choose, facilitates the expression of a requirement and its verification, in the future, the goal is to enable the user to write directly its own interoperability rules with SBVR using the defined vocabulary. Furthermore, future works are related to the proposition of mapping rules to re-write interoperability requirements expressed with SBVR into Conceptual Graphs to make the verification of a-temporal requirements possible using the tool COGITANT.

References

1. Guide to the Systems Engineering Body of Knowledge (SEBoK), version 1.0, see http://www.sebokwiki.org/index.php/Main_Page (last visited 16/11/2012)
2. ANS/EIA 632 Standard: Processes for Engineering a System (1998)
3. ISO/IEC 15288:2008(E) / IEEE Standards 15288.2008 – Systems engineering – System life cycle processes (2nd edition), February (2008)
4. Estefan, J. A.: Survey of model-based systems engineering (MBSE) methodologies. Incose MBSE Focus Group, 25, 1-70 (2007)
5. Grady, J.O.: System Verification: proving the design solutions satisfies the requirements, Academic Press, Elsevier editor, ISBN: 978-0-12-374014-4 (2007)
6. Dasgupta, P.: A roadmap for formal property verification, Springer, ISBN: 978-90-481-7185-9 (2010)
7. Mallek, S., Daclin, N., Chapurlat, V.: The application of interoperability requirement specification and verification to collaborative processes in industry. International Journal Computers in Industry, COMIND, May (2012)
8. Van lamsweerde, A., Dardenne, A., Delcourt, B., Dubisy, F.: The KAOS Project : Knowledge Acquisition in Automated Specification of Software, AAAI Spring Symposium Series, American Association for Artificial Intelligence, (1991)
9. Fanmuy, G., Llorens, J., Fraga, A.: Requirements verification in the industry, CSDM 2011
10. Use Case Map (UCM) notation, available at : http://jucmnav.softwareengineering.ca/ucm/bin/view/UCM/AboutUseCaseMaps (last visited 16/11/2012)
11. REGAL: Requirements Engineering Guide for All, but applicable to Systems Engineering, 2008 (access granted to any INCOSE member at http://www.incose.org/REGAL/)
12. Open Management Group (OMG): Semantics of Business Vocabulary and Business Rules (SBVR) – version 1.0, available online at http://www.omg.org/spec/SBVR/1.0/PDF (last visited, November 12 2012), January (2008)
13. Itu-T Z.151 - Telecommunication Standardization Sector Of Itu: Series Z: Languages And General Software Aspects For Telecommunication Systems - Formal description techniques (FDT) – User Requirements Notation (URN), Language, Definition Recommendation (2008)
14. Goal-oriented Requirement Language, available at : http://www.cs.toronto.edu/km/GRL/ (last visited 16/11/2012)
15. Open Management Group (OMG): Business Process Model and Notation (BPMN) - version 2.0, available online at: http://www.omg.org/spec/BPMN/2.0/ (last visited 16/11/2012), January 3rd (2011)
16. Edmund, M., Clarke, Jr., Grumbereg, O., Doron A.P.: Model checking. The MIT Press (1999)
17. Behrmann, G., David, A., Larsen, K. G.: A tutorial on Uppaal. Department of Computer Science, Aalborg University, Denmark (2004)

18. Mallek, S., Daclin, N., Chapurlat., V.: Formalisation and Verification of Interoperation Requirements on Collaborative Processes. *18th IFAC World Congress (IFAC'11)*, Milano, Italy, (2011)
19. Chen, D., Dassisti, M., Elveaeter, B.: Enterprise interoperability framework and knowledge corpus – final report. Interop deliverable DI.3, may (2007)
20. Mannion, M., Keepence, B.: SMART Requirements. Software Engineering Notes Vol 20 N2. April (1995)
21. Zalta, E., N.: Basic Concepts in Modal Logic, Center for the Study of Language and Information, Stanford University (1995)
22. Cogitant: CoGITaNT Version 5.2.0, Reference Manual (http://cogitant.sourceforge.net) (2009)
23. Sowa, J.F.: Conceptual Graphs. IBM Journal of Research and Development, (1976)