Writing and verifying interoperability requirements: Application to collaborative processes

Nicolas Daclin, Sihem Mallek Daclin, Vincent Chapurlat, Bruno Vallespir

▶ **To cite this version:**

Nicolas Daclin, Sihem Mallek Daclin, Vincent Chapurlat, Bruno Vallespir. Writing and verifying interoperability requirements: Application to collaborative processes. Computers in Industry, 2016, Vol. 82, pp.1-18. 10.1016/j.compind.2016.04.001 . hal-01930301

HAL Id: hal-01930301
https://hal.science/hal-01930301

Submitted on 25 May 2021

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.

Writing and verifying interoperability requirements: Application to collaborative processes

N. Daclin[a,*], S. Mallek Daclin[a], V. Chapurlat[a], B. Vallespir[b,c]

[a] LGI2P—Laboratoire de Génie Informatique et d'Ingénierie de Production Site de l'Ecole des Mines d'Alès, Parc Scientifique G. Besse, 30035 Nîmes cedex 1, France
[b] Univ. Bordeaux, IMS, UMR 5218, F-33400 Talence, France
[c] CNRS, IMS, UMR 5218, F-33400 Talence, France

ABSTRACT

Interoperability analysis is highly correlated with interoperability requirements, the ability to grasp, structure, author and verify such requirements has become fundamental to the analytical process. To this end, requirements must be: (1) properly submitted in a suitable and usable repository; (2) written correctly by stakeholders with relevance to the studied domain; and (3) as easily verifiable as possible on various models of the system for which interoperability capabilities are being requested. The purpose of this article is to present both a structured repository for interoperability requirements and a Domain Specific Language to write and verify interoperability requirements – within a collaborative process model – using formal verification techniques. The interoperability requirements repository, which serves to structure interoperability requirements and make them available, is itself structured through abstraction levels, views and interoperability life cycle dimensions. Additional parameters detailing the requested information and the known impacts of requirements on behavior of the studied system have also been included. The Domain Specific Language provides the means for writing interoperability requirements. Afterwards, these requirements – more specifically the temporal requirements – are re-written into properties by transforming the temporal logic TCTL to allow for their effective verification by using the model checker UPPAAL. The overall approach is illustrated in a case study based on a collaborative drug circulation process. The article also draws conclusions and offers an outlook for future research and application efforts

Keywords:
Interoperability requirements
Repository for interoperability requirements
Requirements verification
Domain Specific Language

1. Introduction

In the field of Systems Engineering (SE) [1], like in any specialized engineering field (mechanical, Information Systems, mechatronics, etc.), requirements engineering is a critical activity dedicated to ensuring that a given system meets all expressed requirements (i.e. original requirements corresponding to stakeholder[1] expectations and prescriptions, as well as requirements induced by technical choices and decisions throughout the system design phase).

For one thing, a requirement assigns, without ambiguity and in a coherent manner, designers' tasks and constraints when devising a solution. A requirement can be described using standards [2], reference models and vocabularies [3]. However, the existing vocabulary and requirement checklists commonly adopted in the SE field tend to be understandable, yet perfectible and abstract when taking a particular category of requirements into account (e.g. non-functional requirements such as interoperability) or a particular system (e.g. enterprises involved in collaborative processes). For another thing, the interoperability concept [4] remains a key factor of success for enterprises that share and exchange processes, service data and resources in a collaborative context; moreover, a number of existing works have sought to characterize [5], implement [6] and assess this very concept [7–9]. Nevertheless, compliance with interoperability requirements within a partnership is neglected and constitutes a challenge that can provide: (1) structure to interoperability requirements, (2) a

* Corresponding author.
E-mail addresses: surname.name@mines-ales.fr (N. Daclin), bruno.vallespir@ims-bordeaux.fr (B. Vallespir).

[1] We have adopted the definition in [3] that defines a stakeholder as a "*party having a right, share or claim in a system or in its possession of characteristics that meets said party's needs and expectations*".

means for expressing requirements, and (3) the tools needed to detect possible interoperability flaws.

The paper's focus is threefold: grasping and structuring interoperability requirements, identifying the means of writing such requirements, and performing verifications with formal techniques [10]. The definition and verification of requirements involves a collaborative process model for the purpose of highlighting interoperability issues that may lead to dysfunctions and interfacing problems from technical, organizational (including human) and conceptual points of view. Following this brief introduction, the problem statement and expected outcomes will be presented. The relevant research work will be provided and discussed in Section 3. Section 4 will then lay out the proposed repository for interoperability requirements, featuring its dimensions, their relationships and the steps allowing for its utilization. Section 5 will offer a case study to illustrate the value of such an approach; lastly, Section 6 will assess the approach and its possible enhancements.

2. Interoperability requirements engineering

2.1. Problem statement and expected outcomes

Nowadays, a technical problem involving interoperability in the fields of computer science and Information and Communication Technologies [11] basically includes organizational aspects (i.e. are the organization and its personnel able to collaborate efficiently?) and conceptual aspects (do the data being exchanged share a common semantics?) [12–14]. While initially focused exclusively on data exchange and sharing, topics such as process interoperability [15] are also receiving consideration at present. Moreover, interoperability encompasses other aspects, like for instance the interfacing issue, which extends to the autonomy and reversibility of partners involved in the collaborative venture [16]. Interoperability therefore is an important and mandatory capability to ensure effectiveness in terms of: exchanging and sharing information, products and resources; aligning and orchestrating collaborative processes; and establishing decisions or policies. Among the numerous relationships concepts (collaboration, cooperation, coalition . . .) and their associated time scale, interoperability is for instance necessary in organizations such as Collaborative Networked Organizations [61] which rely on collaborative business processes (which can be coordinated, orchestrated or else, synchronized) which themselves rely on interoperable activities, resources, applications (through collaboration points [62]) which themselves, for instance, exchange data. However, the better the understanding and definition of interoperability, the more complex its implementation, monitoring, control and improvement. This statement actually leads to considering interoperability requirements and their verification from a more suitable perspective [17].

Requirements engineering practices must consider two major aspects [18], namely the requirements management (access, versioning, change, traceability, etc.) and the actual engineering steps (elicitation, writing, refinement,). Requirements must be checked, throughout a system's life cycle from design phase to execution phase via the corresponding components and subsystems development phase [2], in order to prove expectations have been satisfied and avoid problems (e.g. drift from expected objectives, cancellation in worst cases). Similarly, some requirements must be verified during the actual operations phase and until the system is decommissioned (or at least partially redesigned). Requirements engineering therefore plays a major role in the success or failure of a project [19,58], yet it is often neglected by actors [20,59,60].

The requirements engineering process continues to be considered as time- and resource-consuming and without clear added value. Stakeholders however should always keep in mind that as more errors or omissions are carried to the upstream engineering phases, the remedial costs in downstream phases will increase (modification to the existing system) [21] (Fig. 1). On this figure, an important aspect is the “cost to extract defects” in relation to the different steps of development which shows that the more a defect is identified belatedly, the more the correction cost is important. The requirements engineering belongs to the field of the definition of the problem so, it is beneficial to spend time defining clearly what is expected in order to avoid (as much as possible) problems in the later phases of development. Interoperability is a non-functional requirement (NFR) to be incorporated throughout the system life cycle [22] that affects both the functioning and quality of system service yet that has remained neglected [23]. Hence, interoperability requirements engineering is a key to handling, improving and ensuring that interoperability capabilities are being properly controlled.

First of all, a simple requirements baseline is unsuitable. Requirements need to be combined into a structured reference, i.e. a repository. The overall objective is to structure requirements for them to be easily: (1) traceable throughout the system life cycle (defined, verified, allocated, satisfied), (2) modifiable/removable/addable, (3) usable for determining relevant solutions, and (4)

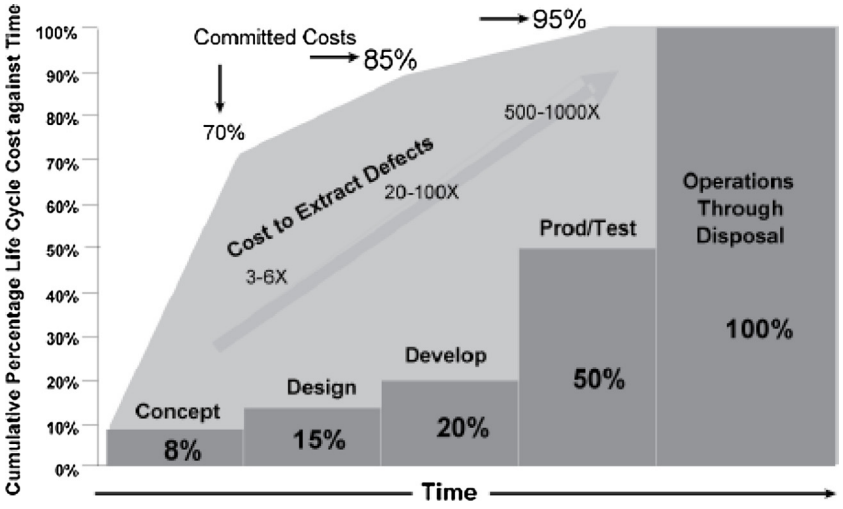

Fig. 1. Commited Life-cycle cost against time (extracted from [21] from Defense Acquisition University 1993).

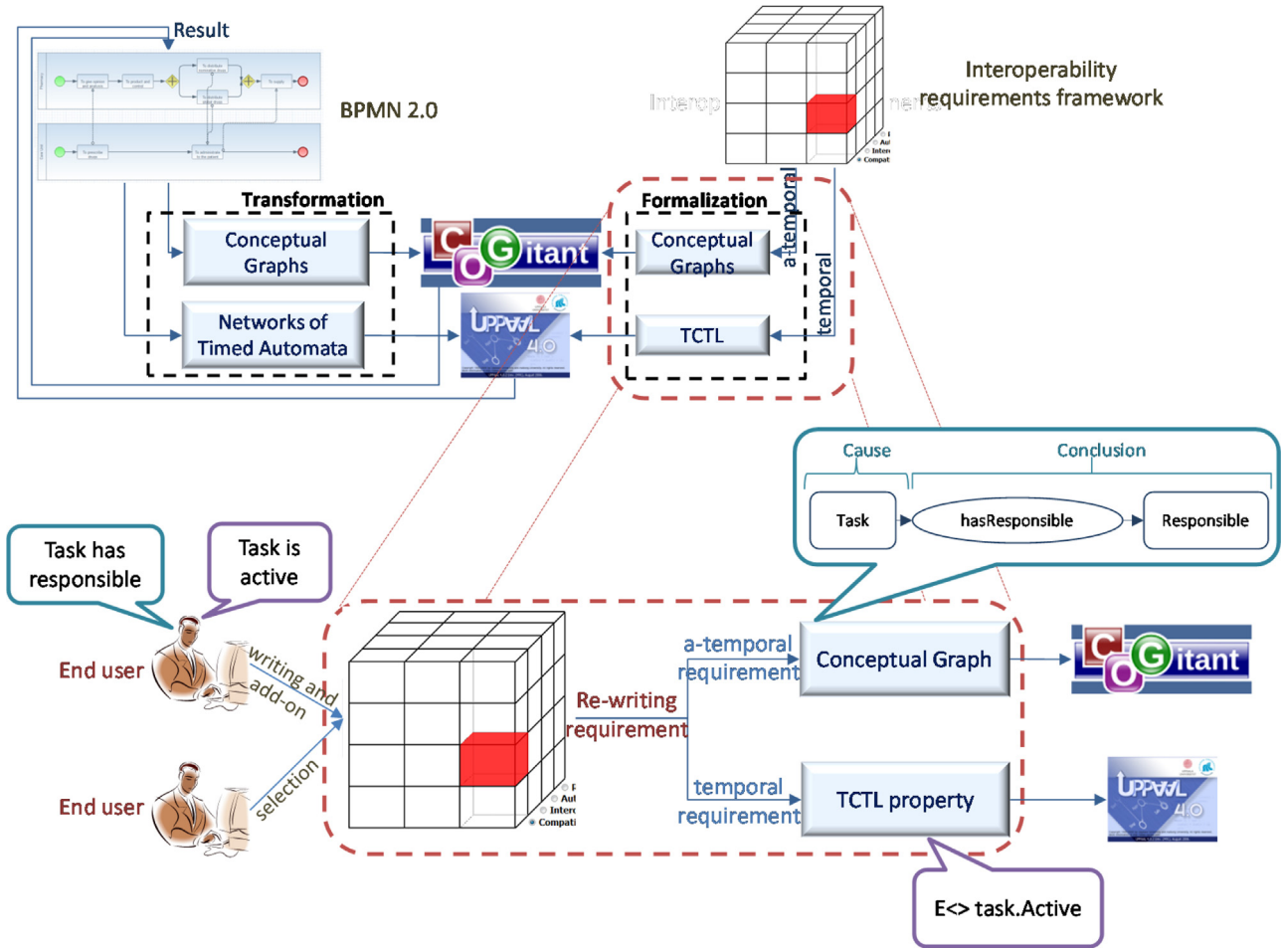

Fig. 2. Overall process of managing and expressing interoperability requirements.

identifiable. Moreover, a repository is capable of supporting the requirements specification from a well-defined basis, rather than with a jumbled list, and is bound to comply with quality criteria from a set of requirements, as defined in [24].

The assumption here is that the qualities inherent to a set of requirements are more easily respected with a repository that structures the interoperability requirements. As a first and obvious point, the repository must be in agreement with the interoperability concept, specifically in corporate sectors concerned with issues of interoperability, interoperability problems and life cycle (when interoperability is anticipated). Second, a repository can lead to considering interoperability requirements that might be generic enough for application to any collaborative processes and in different fields. This kind of requirement would then be identified and permanently positioned in the repository to be used at stakeholders' subsequent request. Third, an overview like a repository can simplify conflict identification between requirements. Such is the case in collaborative processes involving various activities and resources from various processes belonging to different organizations with the potential for conflicting expectations. Lastly, a repository in agreement with existing interoperability frameworks and including requirements that satisfy interoperability needs could ultimately yield interoperability solutions that are fully adapted to the initial expectations.

Stakeholders must subsequently express their requirements. To this end, a number of practical rules for adequately expressing, reviewing and sharing requirements are available in the literature. Let's cite the well-known acronyms M.U.S.T,[2] S.M.A.R.T[3] and ISO recommendations [25]. The spectrum of languages to express requirements extends from natural languages to formal languages. Natural languages offer readability (with requirements being understandable), flexibility (possibility of writing the same requirement using different terms), extensibility (definition of new concepts, terms, attributes), though their principal drawbacks are low levels of consistency and precision plus a risk of ambiguity. Furthermore, the verification of these requirements relies, for instance, on human expertise, thus implying deficiencies (misinterpretation of requirements, an untruthful result, time allocation to verify requirements). Formal languages offer precision (mathematical basis), consistency and are often supported by computer tools. Conversely, these languages are barely readable and prevent stakeholders from using their own business vocabulary and culture. These languages and tools require a solid knowledge, and only a few stakeholders are accustomed to these techniques in favoring natural languages.

[2] M.U.S.T. stands for: **M**easurable, **U**seful, **S**imple and **T**raceable.

[3] S.M.A.R.T. stands for: **S**pecific, **M**easurable, **A**chievable, **R**elevant and **T**raceable (or **T**ime-bound).

[4] For instance, the modalities in temporal logic, where E stands for "it exits", A for "for all", [] for "always" and <> for "sometimes".

Regardless of the language, a dedicated language – as formal as possible – can assist actors in specifying interoperability requirements. The proposed language must be concise, accurate and simple enough to create requirements that are understandable, distributable, unambiguous and entirely readable for humans (as a formal language[4]). The language must then allow writing interoperability requirements within a collaborative process, i.e. by including limited concepts related to: attributes for checking interoperability (data, responsibilities, duration); an expression of requirements (modalities, state, cause, conclusion); the language used to model the collaboration. Moreover, the language needs to guide the writing, by making available both proposals and information regarding possible error. The language must therefore comply with a strictly defined grammar.

Consequently, the main goal here is to support the process that entails managing, writing and verifying interoperability requirements (see Fig. 2). The verification step is performed on a collaborative process model – compliant with BPMN 2.0 [26] – transformed into an equivalent model for applying the verification routine. Two formal verification techniques have been used: conceptual graphs [27] using the COGITANT tool [28] to verify non-temporal requirements and model checking [29] using UPPAAL [30] as well as temporal requirements. At this stage, the collaborative process model is transformed into the model used by verification tools [31]. Moreover, requirements are written directly in the formal language, i.e. conceptual graphs and TCTL (temporal logic—Timed Computation Tree Logic). The purpose then is to enable: (1) managing requirements in a structured form, whereby a set of requirements can be selected/added/removed/modified; and (2) writing requirements using a human readable language and addable into a dedicated repository. As a final condition, requirements must be transformed into the targeted formal language for verification.

This stage will solely consider the writing of temporal requirements. Non-temporal requirements have already been written and positioned within the repository for selection and verification. Hence, the goals are to: (1) render the writing of temporal requirements feasible with a dedicated stakeholder-oriented language; (2) rewrite this requirement into the language used in formal verification techniques; and (3) perform the verification step.

2.2. Managing and writing interoperability requirements

Interoperability is applicable in various fields (e.g. medicine [32,33], armed services [34], computer science [35], crisis management). As regards corporate interoperability, research has sought to: (1) define interoperability, (2) develop methods and tools for interoperability evaluation, and (3) propose methodologies for its implementation. Some work can serve as a basis to manage interoperability requirements and elicit requirements.

Interoperability frameworks structure interoperability according to its characteristics (problems, approaches). Most frameworks [55] consider interoperability [4,36,37] as a conceptual, organizational and technological issue. Some however include additional dimensions in order to develop interoperability solutions [4]; they mainly consider the "how" (solution) rather than the "what" (what needs to be done?), yet they do allow structuring interoperability, and this step needs to be considered in grasping (a solution is a response to a requirement(s)), organizing and handling interoperability requirements.

Interoperability evaluation also involves interoperability requirements. As regards maturity evaluations [38–41], maturity models describe expectations to be met regarding a given interoperations capability that may evolve throughout maturity levels. The notion of requirement is implicit and not formalized (i.e. not structured or expressed as a requirement), yet these models remain a key input from which interoperability requirements can be extracted. Operational measurements (i.e. effective exchange and sharing phase) [8,42] define and provide metrics for an interoperability evaluation. These results are compared with respect to expected results describing requirements in terms of operational interoperability. Further details (maturity/operational evaluation) are available in [43].

Regarding research on interoperability requirements, [44] defines interoperability as a fundamental "ility" that enhances Systems of System capabilities. In a similar manner, the latest version of TOGAF [45] highlights the need to define interoperability requirements and provides guidelines in support of their definition. Moreover, [46] proposed a set of interoperability requirements (related to gas and electricity Smart Metering Systems), according to a defined and common format. With respect to the structuring of interoperability requirements, [31] proposed a set of requirements and a 3-dimensional model to provide a means for verifying interoperability requirements. This work was mainly directed at the effective verification of requirements supported by formal techniques. Let's note that [4], with its interoperability engineering phase dimension, considers defining requirements, yet this has not been developed any further for relevant application and moreover may be extended to consider interoperability requirements. Finally, other works focusing on the notion of interoperability and collaboration between organizations can be also considered. Along these lines, let's mention the service level agreements (SLA) that contractually define the expected quality of services between parties.

To address these issues surrounding requirements expression, various vocabularies and their semantics can be employed in order to abandon the natural language and its weaknesses. These vocabularies must support designers not only in expressing the requirements, but also in decomposing or refining them from a more relevant and formal presentation into a set of sub-requirements. Various approaches are available to handle these vocabularies; however, it remains a difficult task to choose and develop a dedicated language that is concise and capable of: properly writing requirements, specifying interoperability, performing verifications, and being simple enough for widespread comprehension. From this perspective, mental maps or guided interviews may be used to specify requirements among the less formal techniques. More formally speaking, the KAOS method [47], boilerplate approaches [48], Use Case Map notation [49], standardized requirement checklists [3] or the REGAL approach [50] have all been recognized as suitable methodologies. These approaches however offer methodologies to write requirements using natural language. In this sense, the use of standards offering a vocabulary based on natural language, in addition to limiting and structuring the writing of requirements, seems to be more appropriate. Let's also mention URN [51], GRL [52] or SBVR [53]. For instance, Semantic Business Vocabulary Rules (SBVR) propose a defined vocabulary to write business rules that represent requirements. 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*" [53] and moreover is based on a natural language that simplifies writing requirements in comparison with a formal language [54]. Thus, SBVR offers a clear representation of requirements without necessitating specific skills since it is based on natural language with a limited vocabulary. These semi-formal notations, either graphical or textual, feature a well-defined syntax and are easily understood and handled by stakeholders. Conversely, the majority of these languages are only minimally automated and still present a risk of

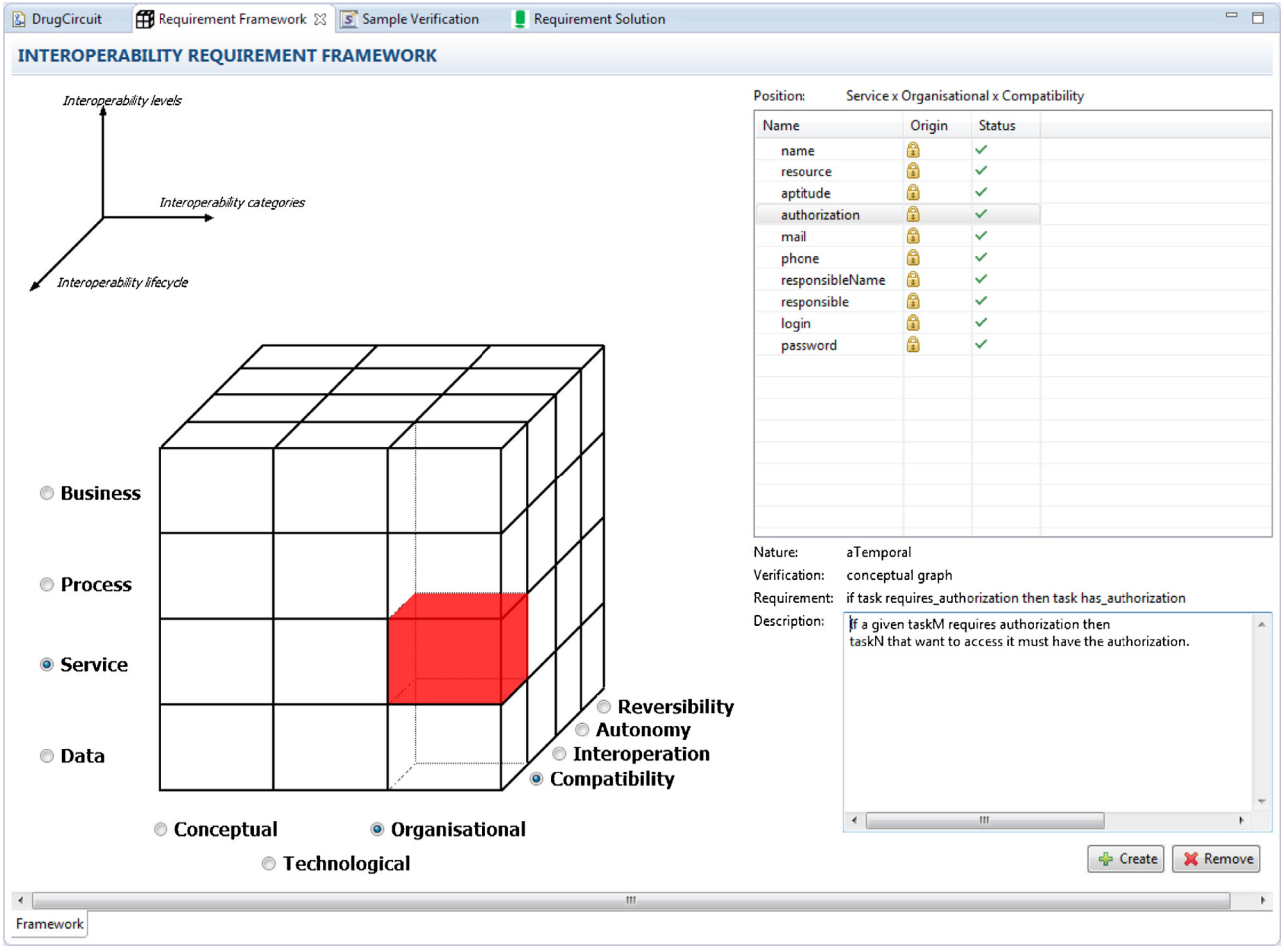

Fig. 3. The Interoperability Requirements repository (requirements shown with a lock are predefined and unmodifiable).

ambiguity. Formal languages (CTL, LTL, TCTL, etc.) also make it possible to write requirements with great precision by offering a defined syntax and semantics with a high level of automation. In contrast, these languages are difficult to learn, read and understand without extensive expertise. Lastly, let's mention the Formal Description Techniques (FDTs) which offer languages to describe a system, especially for computer system development. FDTs ensure to get a set of specifications which respect characteristics such as clearness, concision, completeness, consistency, tractability, and conformance. FDTs are addressed to developers, implementers, testers and end users and let's quote ESTELLE, ALBERT II, LOTOS... [65]. These FDTs remain close to the programmatic languages structure based on mathematical symbols that can hardly usable but they allow to reach an effective and reliable system in an efficient and rigorous manner. Thus, it could be interesting to extend these kinds of approaches to allow – beyond the developers and implementers – the business users in charge of the management of collaborative processes and, here, the management of interoperability aspects, to benefit from formal techniques.

From these considerations, it is necessary at first to clearly structure identified (or written) interoperability requirements according to: (1) interoperability problems, and (2) the interoperability definition. This structure must ultimately provide feasible interoperability solutions that satisfy stakeholder expectations. Second of all, it is necessary to write requirements in adhering to strict guidelines and rules applicable to stakeholders in their specifications task and the resulting requirements must be verified with formal techniques. This has to lead to the development of a language sufficiently "non formal" to be used without strong knowledge and sufficiently formal to be linked with a tool allowing formal verification. Thus, the expecting result is a Controlled Natural Language [63] enables non engineers or experts in requirements engineering to take advantage of reliable verification means based on formal language while keeping, as possible, the comfort and the power of expression of the Natural Language, i.e., "*to bridge the gap between a natural language and a formal language by the use of a controlled natural language (CNL) that can mediate between these languages*" [64].

3. Repository for interoperability requirements

3.1. Fundamental dimensions

A repository is "*a place where things are stored and can be found*[5] ". Regarding interoperability requirements engineering, the repository provides a means for structuring, organizing and managing requirements according to interoperability features, in continually verifying their compliance throughout a partnership and guiding the selection of a best interoperability solution [9]. In other words, it offers a set of interoperability requirements that respect the quality criteria of a set of pre-established requirements, thus making it possible to:

[5] http://dictionary.cambridge.org/.

- grasp and structure interoperability requirements—stakeholders may set up their own requirements in accordance with dimensions properly related to interoperability;
- specify the requirement author, modify and remove interoperability requirements all in an efficient manner;
- include a basic set of interoperability requirements—existing requirements from the literature applicable to any collaborative process (i.e. independent of the context) are already positioned and proposed;
- satisfy all stakeholder expectations, avoid redundancy and conflict between requirements, ensure that a set of (selected) requirements can be satisfied (otherwise, provide for early detection of infeasibility), and ensure that requirements are easily accessible and available for consultation (structuring).
- generate a set of interoperability solutions when a requirement is not satisfied – the targeted repository must be consistent with existing interoperability frameworks yielding interoperability solutions.

Consequently, the proposed repository is in agreement with the framework in [4] and [36]. This one can act as a base to collect interoperability requirements to get a relevant repository. Moreover the original version of the framework makes available a set of interoperability solutions – according to its dimensions – that can be related with interoperability requirements. Thus, the repository considers three basic dimensions: abstraction, views, and interoperability life cycle (Fig. 3), in agreement with [4]. *Abstraction levels* are included within the interoperability problems. *Interoperability views* are the companies' fields affected by interoperability. Lastly, a third dimension is added and defined as *interoperability life cycle* which is the decomposition of interoperability into agreements during the partnership life cycle. For instance, requirements at the beginning of a partnership concern interfacing while requirements at the end pertain to unplugging.

The dimension of *abstraction levels* considers interoperability problems in the three realms of Conceptual (meaning of information, capability of expressing and communicating information), Organizational (responsibilities, authority decision-making processes, policies, organizational processes and regulations) and Technological (physical interface compatibility issues).

The dimension of *interoperability views* identifies the main fields needed to develop in-company interoperability [36]: Business, Process, Service and Data. Initially limited in its original version to the data exchange, sharing and the exploitation of exchanged and shared data exclusively, this last level of the repository has currently been extended to consider other flows, so as to enable specifying interoperability requirements. For our purposes, this level includes:

- **Resources**. The interoperability of resources relates to the specific resources (human, software, hardware) that belong to the various partners and that can be shared among them. This aspect mainly pertains to the synchronization of resources deployed, their aptitudes and capacities;
- **Material**. Material interoperability refers to exchanges of the materials to be processed by the various partners. This aspect mainly pertains to the volume, length, span, etc. of the material being exchanged.

The dimension of *interoperability life cycle phases* relates to the occurrence of interoperability during the partnership. Interoperability requirements may indeed evolve, depending on the phase of the partnership, i.e. the beginning (partner connections), the operational phase (interoperation exchange, sharing, performance) or the end (dismantling). Each phase therefore is consistently correlated with interoperability. As such, this dimension is characterized by: compatibility (interfacing issues), interoperations (running of the partnership), autonomy, and reversibility [31].

As shown in the previous figure, a set of interoperability requirements is already positioned within the repository. The stakeholder must obtain information on the requirements and is unconcerned with any writing or verification process. For instance, the selected requirement is positioned at the service/organizational/compatibility level; moreover, it is non-temporal and verified with conceptual graphs (pre-loaded). Moreover, its goal consists of ensuring that a person associated with a task of a given process holds the possible required authorization to access another process (information, tool, database). The stakeholder then merely has to instantiate the requirement in specifying the considered tasks (shown in the description). Lastly, the following table presents some typical interoperability requirements positioned into the repository. This table presents the two dimensions interoperability levels and interoperability categories, the interoperability lifecycle phase is positioned beside the requirement (C-compatibility, I-interoperation, A-autonomy and R-reversibility) (Table 1).

Table 1
Examples of interoperability requirements (available at: https://tel.archives-ouvertes.fr/tel-00666099/document).

	Conceptual	Organizational	Technological
Business	–	• The decisional structures are defined (C) • A common strategy is defined (I) • A strategy of a partner is not altered by the common strategy (A)	–
Process	• The internal processes are modeled (C)	• The collaborative process is modeled (I)	–
Service	• The services are described (C)	• The responsible are identified (C) • The receiver acknowledges reception (I) • The internal functions are performed after the partnership (R) • A service can stand in for the service involved into the partnership (A)	• A communication protocol is defined (C) • The duration to exchange is less than a defined value (I) • The duration to use exchanged data is less than a defined value (I) • The duration to execute a task post-partnership is equal to the duration ante-partnership (R)
Data Resource Material	• Data are unambiguous (C) • Received data are conform to required data (I) • Data are not degraded (R)	• Internal data are secured (C) • Data to exchange are available (I)	• A data mapping language is defined (C) • The quantity of required data is equal to the quantity of received data (I)

All these requirements are either (1) expressed and verified with conceptual graphs due to their a-temporal nature, (2) expressed and verified with the model checking technique due to their temporal nature or (3) kept like this whatever their nature and verified by expertise since it is impossible to express them with a more formal language.

Lastly, let's mention that the following point of view is adopted to consider a requirement as an interoperability requirement: *when heterogeneous organizations work together and require to share, exchange, use, modify, have permission . . . functionalities, resources, data . . . from each other, and when objects and attributes from different organizations occur in a requirement that can impact the relation, the requirement falls into the category of interoperability requirements and are positioned into the repository.*

3.2. Additional parameters

Other parameters must also be considered by the repositrory in order to: (1) ascertain with precision the impact of interoperability requirements on collaborating partners and/or on the partnership itself; (2) provide information about verification techniques and the means for writing requirements; and (3) ensure availability of adapted solutions with respect to the verified requirements. These parameters are known as "*Granularity*", "*Analysis*", "*Solution proposal*" and "*Means of verification*" [17].

The *Granularity* parameter represents the level of detail on the object affected by interoperability. The selection and implementation of an efficient interoperability solution depends on how accurate the particular object has been identified. Interoperability solution can indeed influence partnership operations or efficiency (mission, objectives, etc.) or a partner (mission, objectives, components, resources). As an example, "*partners hold the necessary authorization to access shared data*" (CxOxD[6]) does affect partnership activities (risk of lost time while obtaining the information required by an activity). The requirement "*function f, performed by resource r involved in partnership, is still performed*" (AxOxS) exerts an impact on a partner (execution of a function). Partners can thus select/adapt/build interoperability solutions or, potentially, relax requirements.

Analysis levels constitute the main characteristics affected by the implementation of interoperability. The satisfaction of a requirement can affect criteria (e.g. performance affecting the partnership or a partner). Requirements must therefore be defined according to these criteria in order to highlight interoperability expectations and their impacts on partner and partnership. These criteria are defined as: performance, stability, and integrity. *Performance* refers to the ability of a system to achieve its objectives. Requirements positioned on this characteristic concern the expected interoperability performance and the impact of interoperability on partner/partnership. For instance, the requirement "*the duration to connect application is less than x times units*" impacts performance (CxTxS). More precisely, it affects partnership performance by extending for example partners' interface duration and the time required to deliver the expected service/product. *Stability* refers to a system's ability to maintain its viability and adapt to its environment (external change). For purposes of illustration, the requirement "*a resource r is used to change the partnership mission*" (IxTxS) indicates that a modification in the mission (e.g. new service expected by the customer) must be performed by the initially allocated resource. In this case, the stability of the partnership is affected since the resource must be capable of participating in the new mission, under satisfactory conditions and with the expected performance level (e.g. agility, flexibility, reactivity). *Integrity* refers to a system's ability to remain consistent and carry out its functions in case of modifications (e.g. loss of resources – internal change). For instance, the previous requirement "*function f, performed by resource r involved in partnership, is still performed*" alters the integrity of a partner since its own resource is being allocated to the partnership and an internal function also needs to be performed. In this case, the partner must take action to sustain its own integrity (e.g. overload of another internal resource, subcontracting, overtime hours).

Solution proposal parameters are intended to make available a set of solutions (this set is fully related to [4]) in accordance with the considered requirement. This parameter also includes: (1) information on the potential problem(s) resulting from requirement non-verification; and (2) the potential impact on the verification of other requirements once a solution is implemented. For instance, non-verification of the requirement "*if task requires aptitude, then resource has aptitude*" (CxOxS) can lead to partial task achievement or, in the worst case, to its non-achievement. The solution might then consist of either replacing the allocated resource by one with the requested aptitude or training the current resource. The choice of solution (resource change) can however produce an impact on the verification of other requirements, namely those related to the achievement of other tasks and the availability of shared resources (e.g. "*It is possible that task is starting and resource is available*").

Verification means stakeholder guidance during selection of a predefined requirement or in writing a requirement. It therefore provides information about the technique used to verify requirements (expertise, conceptual graphs, model checking), depending on both the nature of the requirement (temporal or non-temporal) and its ability to be verified automatically (conceptual graphs and model checking) or not (human expertise). It also provides the dedicated domain specific language, according to a selected means of verification, for writing a requirement. For instance, the previous requirement "*a resource r gets used to carry out a change in the partnership mission*" is verified by an expert because no verification can be conducted with any formal technique. Consequently, stakeholders must have the possibility to write requirements depending on the chosen means of verification. The next section will present a language to write interoperability requirements for verification using model checking techniques.

4. Process of writing interoperability requirements

4.1. Principles of DSL[7] applied to writing interoperability requirements

First, the domain specific language must be easily manipulated and accessible without any sophisticated knowledge (in comparison with temporal logic, for instance) and allow stakeholders to properly create and verify requirements. Hence, the proposed language must be limited yet sufficient to write requirements, i.e. it must be developed in accordance with the field under study. In this research, DSL includes limited concepts, such as:

[6] We have adopted the following convention regarding the position of a requirement:

- The first letter denotes the interoperability life cycle (e.g. "C" stands for compatibility).
- The second letter denotes the abstraction level (e.g. "O" stands for organizational).
- The third letter denotes the interoperability level (e.g. "D" stands for data).

For instance, CxOxD means the requirement concerns the interfacing aspect (*Compatibility*) at the *Organizational* level and is required for the exchange/sharing of *Data*.

[7] DSL stands for Domain Specific Language.

Table 2
DSL grammar for temporal requirement writing.

```
Requirement: (mod=Modality p+=Proposition+) | (p+=Proposition+ lead=LeadTo q+=Proposition+);
Proposition: obp+=OpenBracket* (iter=Iter? | neg='not'?) fact=Fact cbp+=CloseBracket* bool=Bool?;
Fact: sT=StartFact (cbf+=CloseBracket* | eF+=EndFact*);
StartFact: (stateTerm=StateTerm (verb=Verb | timeTerm=TimeTerm)) | tab=Tab | val=Value | c=Clock;
EndFact: (comp=Comparator | op=Operator) ((stateTerm=StateTerm timeTerm=TimeTerm) | val=Value |
tab=Tab);
StateTerm: st=("task" | "resource");
Clock: c="clock";
Value: val="value";
TimeTerm: tt=("timeMax" | "timeMin");
Verb: v=('is_waiting' | 'is_working' | 'is_starting' | 'is_finished' | 'is_stopped' | 'is_available' | 'is_active');
Comparator: comp=('is_less_than' | 'is_greater_than' | 'is_equal_to' | 'is_not_equal_to');
Operator: op=('plus' | 'minus');
Bool: b=('and' | 'or');
Iter: it=('for all' | 'it exists') itType=IterType;
IterType: id_it=('tasks,' | 'sequence_flows,' | 'message_flows,');
Tab: tab=('task_start [index]' | 'task_end [index]' | 'emission_message_start [index]' |
'emission_message_end [index]' | 'reception_message_start [index]' | 'reception_message_end [index]' |
'reception_sequence_start [index]' | 'emission_sequence_end [index]');
Modality: reachable="It is possible that" | invariantly="Invariantly," | inevitable="It is inevitable that" |
potentially="There is potentially always ";
LeadTo: lt='lead to';
CloseBracket: cb=')';
OpenBracket: ob='(';
```

1. The language used to model the collaborative process (BPMN concepts, like task, resource, event) [26];
2. The verification technique (UPPAAL) to model process behavior and temporal requirements [30]. All automaton and states corresponding to the BPMN object behavior (e.g. task in a "Working" state) are taken into consideration, as is the specification language (reachability "E<>").
3. Interoperability concepts, i.e. all concepts absent from the previous point but still relevant to writing interoperability requirements [16] (aptitude, *is_less_than*, authorization, responsibilities, etc.).

The proposed DSL is constrained to write interoperability requirements and formalized in the following formula:

$$InteroperabilityRequirement\ \ DSL = \{BPMNconcepts, UPPAALconcepts, Interoperabilityconcepts\} \quad (1)$$

Second, the DSL must provide: (1) proposals throughout the writing process, (2) information about possible mistakes, and (3) a human readable language rather than a formal language. On the basis of this DSL therefore, stakeholders must be able to write a temporal interoperability requirement for verification on a collaborative process model. For example, the interoperability requirement '*it is possible that a task is working and a resource is active*' is built based on BPMNconcept (task, resource) and UPPAALconcept (working, active). Furthermore, instead of using the formal quantifier "E <>", this one is expressed by '*it is possible that*', which is a more easily understood construction.

4.2. The syntax of interoperability requirement DSL

The proposed syntax allows stakeholders to express requirements from an intelligible language while respecting the fundamental characteristics of a requirement. It makes fundamental objects available, such as the modalities that express the type of proposition to verify (e.g. "it is inevitable that"), the elements involved in the collaborative process (resources), and the state of a given element in the collaborative process (resource availability), operators (Boolean as well as comparative). In turn, all these elements serve to build and create a set of interoperability requirements. From a formal perspective, an interoperability requirement can be simply written in order to respect the following syntax:

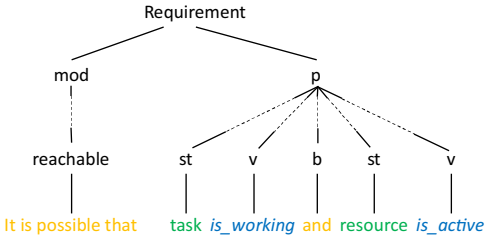

Fig. 4. Example of a syntax tree for a requirement conforming to DSL grammar.

$$interoperabilityRequirement: = modality\ p \mid p\ leadTo\ q \quad (2)$$

where:

$$modality: = \{reachable, invariantly, inevitable, potentially\} \quad (3)$$

$$lead\ To: = \{\text{lead } to\} \quad (4)$$

and:

$$p,q: = proposition \quad (5)$$

A proposition can also include various operators (e.g. Boolean, iterator) to build a more complex interoperability requirement. For this purpose, the proposition integrates *BPMNconcepts*, *UPPAALconcepts* and *Interoperabilityconcepts*. Moreover, let's note that the proposed syntax can evolve in accordance with the specific needs of stakeholders in terms of requirement writing. The resulting implementation of the DSL grammar is given in Table 2 below.

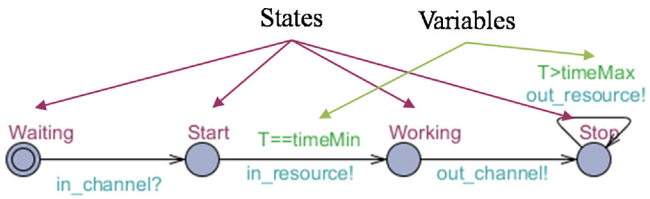

Fig. 5. Task automaton in UPPAAL.

The following example shows that a requirement conforms to the grammar defined above. The syntax is implemented within the xText[8] framework, which allows defining an entire specific language that includes aspects leading to completion and mistake avoidance during requirement writing (Fig. 4).

4.3. TCTL syntax

The UPPAAL tool is used to process a behavioral model defined as a set of templates that communicate by synchronization using channels and syntax such as sent/receive. Templates are given locations and transitions [30]. The principle of a model checker is to exhaustively verify requirements with temporized and possibly constrained automata that describe the behavior of a system. Furthermore, temporal requirements must be formalized into the TCTL (Timed Computation Tree Logic) properties used to consider several possible futures based on the state of a system. The UPPAAL model checker features four TCTL quantifiers (A: for all paths, E: a path exists, []: all states in a path, <>: some states in a path) introduced to write a property. Formally, a TCTL property (a query in UPPAAL) can be simply written in respecting the following syntax.

$$Query: = quantifier\ p|\ p\ leadTo\ q \tag{6}$$

where:

$$Quantifier: = (pathQuantifier,\ temporalQuantifier) \tag{7}$$

with:

$$pathQuantifier = \{[], <>\} \tag{8}$$

$$temporalOperator = \{E,\ A\} \tag{9}$$

$$leadTo = \{\rightarrow\} \tag{10}$$

and:

$$p,\ q = expression \tag{11}$$

An expression (p) is written in accordance with existing automata (states and variables), as presented in Fig. 5 for the automaton of a task described with four states (Waiting, Start, Working and Stop) and two variables (timeMin and timeMax) used in a clock T.

Furthermore, an expression may also include various operators (e.g. Boolean, iterator) to build a more complex property to verify.

The resulting implementation of this TCTL grammar is shown in Table 3 (adapted from [30]).

As per the previously defined TCTL query syntax, Fig. 6 describes a given requirement using TCTL logic.

4.4. Rewriting of interoperability requirements

Once a requirement has been written using DSL, it then needs to be rewritten into the language used to apply the verification technique (TCTL). Both syntaxes are based on the use of modalities (interoperability requirement DSL)/quantifier (TCTL) followed by a proposition.

The first mapping involves the link between the modalities of the interoperability requirement DSL and the quantifier used in TCTL. As such, four modalities are defined in keeping the meaning of the quantifier. These modalities are human readable and express path quantification (E, A) and temporal operators ([], <>) (Table 4).

The second mapping pertains to the link between a proposition expressed using DSL and a proposition using TCTL. For instance, a *stateTerm* (e.g. "task") will be mapped with an *iValue*. In the same manner, a *verb* (e.g. "is_waiting") will also be mapped with an *iValue*. Beyond this simple mapping however, both the structure of the written requirement and the targeted language must be considered as well. For instance, the *stateTerm* "task" in a requirement means that the mapping must be established with the automaton Task. In addition to this initial mapping, a subsequent mapping of the *Verb* "is_waiting" must be performed with the corresponding state of the automaton Task ("Waiting"). This second mapping must still respect the structure of the writing using TCTL, which means that the dot symbol (".") must be inserted between the name of the automaton and the considered state. Consequently, this second mapping also considers the aspect of query building correctness by incorporating the "*pID*" rule found in the grammar.

The mapping therefore depends not only on a simple relationship between the elements of a requirement and a query (one-to-one mapping) but also on how they are arranged in the requirement and the correspondence of this arrangement within a query. A partial representation of the proposed solution is given in the following table (Table 5).

Based on this mapping, all elements of an interoperability requirement (DSL grammar) need to be transformed into a query (TCTL grammar). To this end, rules have been developed to automatically rewrite a requirement. For instance, a mapping can be derived to rewrite the interoperability requirement "*It is possible that task is_working and resource is_active and clock is_less_than timeMax*" into TCTL, as: "$E <>$ *task.Working and resource.Active and* $T < timeMax$". In considering and respecting the proposed mapping, the modality can indeed be rewritten into a quantifier and the DSL proposition rewritten into a TCTL proposition, as shown in Fig. 7.

Lastly, to prepare a requirement for verification entails instantiating it with existing elements in the studied collaborative process model. This implies that before the rewriting step, elements within the process must be proposed from the elements that compose the requirement to be verified. Thus, for a given requirement involving both tasks and resources for example, then each task and resource belonging to the collaborative process model must be proposed to instantiate the requirement. In pursuing the current example "*It is possible that task is_working and resource is_active*", all names of tasks and all names of resources involved in the process need to be proposed in order to instantiate the requirement.

[8] Available online at: http://www.eclipse.org/Xtext/.

Table 3
Implementation of the TCTL query grammar (adapted from UPPAAL).

```
Query: ((quantifier = QuantifierPath) p = AbstractExpression) |
        (p = AbstractExpression lead = QuantifierLead q = AbstractExpression);
AbstractExpression: atom = Atom (binary = Binary | atom1 += Atom | point += Point)*;
Point: pID = '.' ident = Ident;
Word: w = wValue;
enum wValue: deadlock ='deadlock' | false = 'false' | true = 'true';
Atom: (unary = Unary? | iter = Iteration) (ident = Ident | word = Word);
Unary: unaryValue = uValue;
enum uValue: plus = '+' | minus = '-' | no='!' | not = 'not';
Binary: binaryValue = bValue;
enum bValue: inf = '<' | infE = '<=' | strictE = '==' | dif = '!=' | supE = '>=' | sup='>' | plus = '+' | minus = '-' |
mult = '*' | div = '/' | percent = '%' | sAnd = '&' | sOr = '|' | exp = '^' | hInf = '<<' | hSup = '>>' | and = '&&' | or
= '||' | infD = '<?' | supD = '>?' | orL = 'or' | andL = 'and' | imp = 'imply';
Iteration: iterOp = ('forall' | 'exists') '(' iterBody1 = Ident ':' iterBody2 = Type ')';
Ident: iValue = ID;
Type: prefix=Prefix typeId = TypeId;
TypeId: identType = Ident | typeWord = TypeWord | scalar = Scalar | int = Inte | struct = Struct;
Struct: s = 'struct' '{' field1 = FieldDecl field2 += (FieldDecl) '}';
FieldDecl: Type ID array1 += ArrayDecl (',' ID array2 += ArrayDecl)* ';';
ArrayDecl: '[' a = Atom ']' | '[' t = Type ']';
Inte: i = 'int' '[' exprei = Atom ',' exprei1 = Atom ']';
Scalar: scal = 'scalar' '[' expres = Atom ',' expres1 = Atom ']';
TypeWord: typeWordValue = twValue;
enum twValue: int = 'int' | clock = 'clock' | chan = 'chan' | bool = 'bool';
Prefix: prefVaLue = pValue;
enum pValue: urgent = 'urgent' | broadcast = 'broadcast' | meta = 'meta' | const = 'const';
QuantifierLead: qL = '-->';
QuantifierPath: quant = QuantValue;
enum QuantValue: invariantly = "A[]" | inevitable = "A<>" | potentially = "E[]" | reachable = "E<>";
```

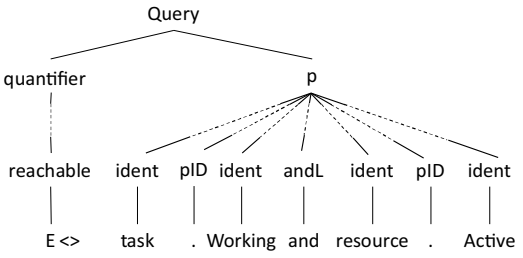

Fig. 6. Example of the syntax tree for a query conforming to TCTL grammar.

5. Application case study: the drug circulation process[9]

Drug circulation is a critical process in hospitals since it is mandatory to provide the right drug to the right patient in time and in the right dose. While apparently simple, proper execution of this circulation primarily depends on good interactions between participants and precise interactions between all resources involved. This process therefore must closely involve stakeholders in order to improve pharmacy practices and strengthen the role of the Medicine Committee (care unit). A drug circuit is generally composed of three main steps: prescription, delivery, and administration performed by both the care unit and the pharmacy. Furthermore, three resources are involved: a nurse (Care Unit), a medical practitioner (Care Unit), and a pharmacist (Pharmacy). All data relative to the process, e.g. resource declaration, resource allocation, attribute declaration, are provided by the stakeholders in charge of modeling. Tasks and interactions are modeled with a BPMN 2.0 modeler (Fig. 8). The following diagram (known as a collaboration diagram in BPMN [26]) sets up the model for various processes from various entities (Pharmacy and Care Unit) and their interactions (message flows between processes), with the entire set-up constituting the drug circulation process (upper part of Fig. 8). Various resources are involved and ascribed to a single entity. These resources are declared (a point to extend the BPMN language has been developed, lower part of Fig. 8) and allocated to the various process tasks (a resource ascribed to one entity can in fact be allocated to a task of another entity process).

It is now proposed to show how interoperability requirements are managed through the designated repository. This discussion will concern the selection and use of a requirement already existing in the repository. Afterwards, the case study will demonstrate how stakeholders can write, select and verify their own requirements. Two interoperability requirements expressed in natural language will be considered:

1. *When a given task requires a specific aptitude to be deployed, the allocated resource must have the appropriate aptitude.*
2. *A resource is available to perform its allocated task when required.*

Interoperability is often simplified to the exchange and sharing of information and technical compatibilities. In a collaborative process context, beyond the exchange of information, the exchange and sharing of resources (hardware, software or human) may be required as well. Such is the case in this process, for which the "medical practitioner" resource is declared as belonging to the

[9] A demonstration of the developed tool is available at: goo.gl/chclk7, in offering details of the application case study.

Table 4
Mapping between DSL modalities and the TCTL quantifier.

Interoperability requirement modalities	TCTL quantifier
It is possible that (p is satisfied in some possible worlds)	E <> (p is true in one reachable state)
Invariantly (p is satisfied in all worlds)	A[] (p is true in all reachable states)
It is inevitable that (p will inevitably become satisfied)	A <> (p is true in some states of all paths)
There is potentially always (p is potentially always satisfied)	E[] (exists a path in which p is true in all states)

Table 5
Mapping between DSL proposition and TCTL proposition (partial representation).

Interoperability requirement DSL proposition	TCTL proposition
StateTerm (st)	Ident (iValue)
Verb (v)	Ident (iValue)
StateTerm Verb (st v)	Point (pId)
Operator (op)	Binary (binaryValue)
Bool (b)	Binary (binaryValue)
Iter (it)	Iteration (iterop)

"Care Unit" participant. Beyond his/her involvement in the activities of the "Care Unit", the "medical practitioner" is indeed also involved in the activities of the participant "Pharmacist", so as "to provide opinion and analysis". The first requirement therefore entails ensuring that the resource is in effect available (in terms of time) to perform the activity. The second requirement concerns ensuring that the allocated resource is capable (in terms of aptitude, as the medical sector requires numerous qualifications) of performing the activity.

The first requirement is predefined and non-temporal with the possibility of being selected in the repository. It is positioned on the service view, at the conceptual and compatibility level. Moreover, it is expressed such that "*if task requires aptitude then resource has aptitude*". As a predefined requirement, it can be directly selected by stakeholders before its instantiation and verification.

The conceptual graph corresponding to this requirement also exists and can be applied in its current form (Fig. 9).

Hence, the stakeholders are not concerned by the writing of this kind of requirement. They must merely ensure, via the information provided by the requirement repository, that the requirement matches its expectations in terms of collaborative process model verification and has been selected. The previous requirement is already positioned in the repository and cannot be modified by users (locked). The nature of the requirement, the means used for its verification, and its expression and information are all used to guide the selection (Fig. 10).

The second requirement is temporal and can be written or chosen should it exist in the repository. If it is written, then stakeholders must position it in the repository according to the relevant view, abstraction level and life cycle level. The following figure shows the graphical user interface that implements the DSL to allow writing a requirement. The writing process is guided through a set of proposals that are consistent with the defined grammar. As regards the requirement presented in Fig. 11, once the stakeholders have selected the nature of this requirement (temporal vs. non-temporal) as well as its means of verification (expertise vs. model checking vs. conceptual graphs), the corresponding defined grammar is proposed (Fig. 11a). As for writing this requirement and in agreement with the syntax (see Section 4.2), the *modalities*, *iterations* and *negation* are proposed first. Since the stakeholders have chosen a modality ("*It is possible that*"), "*iterations*", "*negation*" and "*stateTerm*" are all proposed. Let's note that additional constraints have been included in the syntax in order to detect the previous terms and propose only the verb related to this term (e.g. the verbs "*is_waiting*", "*is_working*", "*is_starting*" and "*is_stopped*" only relate to the state term "*task*"). Moreover, the syntax tree is built according to the same principle and until the stakeholders complete their writing process. In this example, the last proposals are the verbs related to the state term "*resource*", namely "*is_active*" and "*is_available*". Lastly, let's note that the stakeholders are constrained when they build a requirement and cannot write nonsense; otherwise, the requirement could not be validated (errors are highlighted, see Fig. 11b) or verified.

This requirement, expressed previously in natural language, is now being expressed with the dedicated language; it uses the modality "*it is possible that*", which indicates that proposition must be satisfied along one path of the collaborative process (stemming from the originating requirement "*when it is required*"). It is also composed of the proposition "*task is_starting and resource is_available*", which means that when a task begins to fit into the working state, the allocated resource must be positioned in the state available at the same time. The set, modalities and proposition taken together make up the interoperability requirement to verify. Moreover, the expressed requirement can be saved and added to the repository for further verification (Fig. 12).

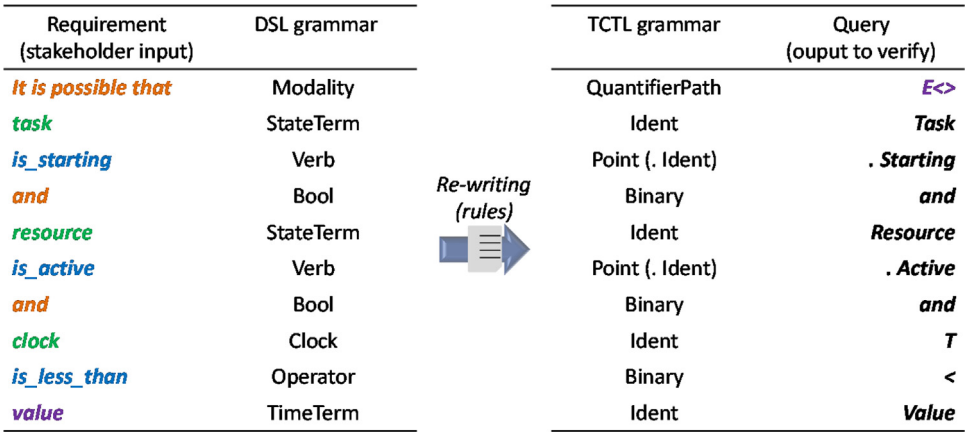

Fig. 7. Principle of mapping between an interoperability requirement and a TCTL query.

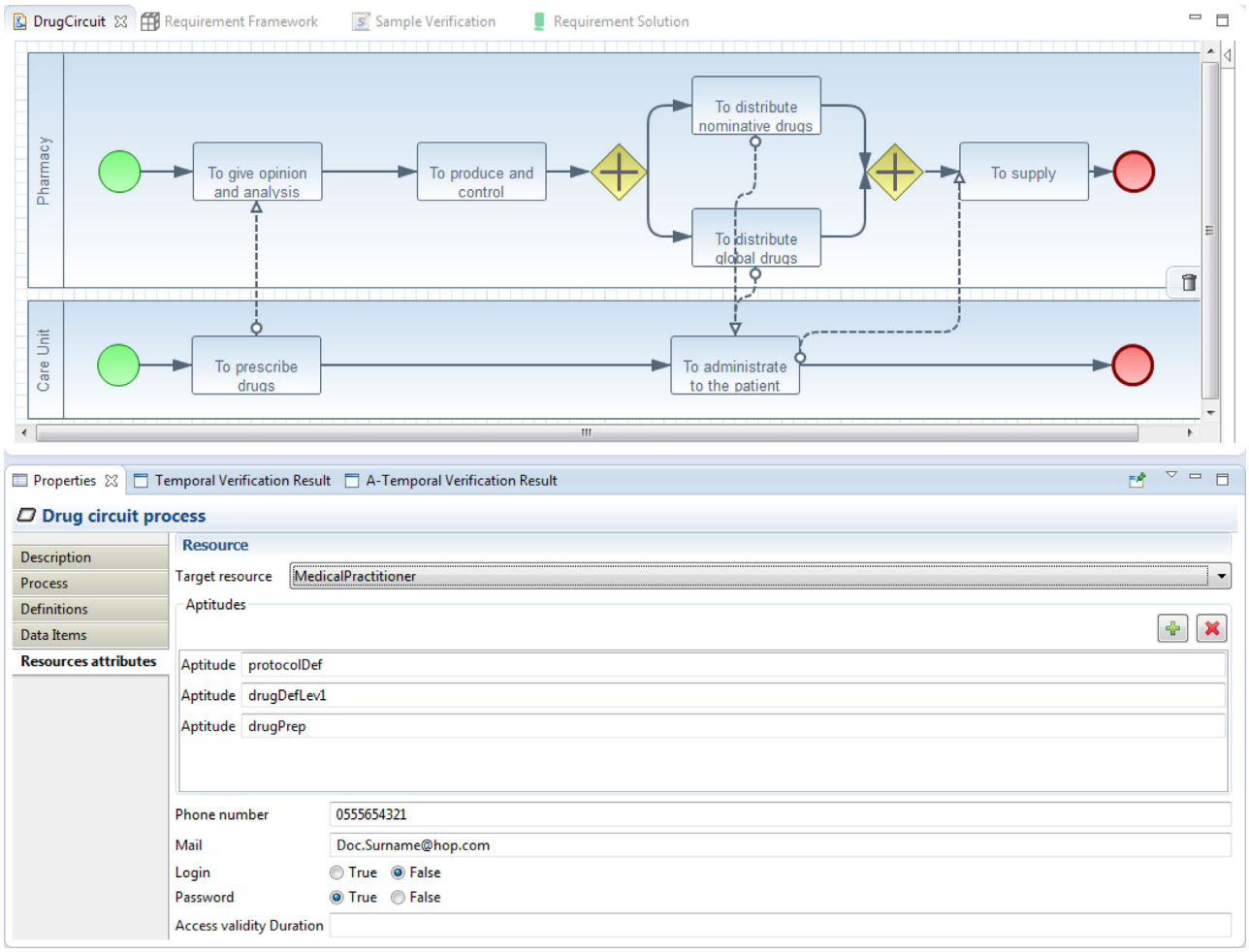

Fig. 8. The drug circulation process modeled with BPMN 2.0, including the declaration of a human resource.

Requirement expression	Conceptual graph representation
If task *requires* aptitude	Task → *requires* → Aptitude (cause)
then resource *has* aptitude	*uses* → Resource → *has* (conclusion)

Fig. 9. Expression of a non-temporal requirement and mapping with the conceptual graph.

It is to note that some requirements are predefined and directly usable. This is typically the case for the requirements which are domain-independent and which can be applied on any collaborative processes. Other requirements are defined directly by the stakeholders with the help of the proposed grammar to write temporal requirement and verified by model checking. For instance, the selected requirement (receipt) is a predefined and unmodifiable (lock) requirement. It is expressed by a conceptual graph and allows verifying that an acknowledgement of a receipt has to be performed when a task receives a message flow from another entity: "*if task receive a message flow then a mechanism of confirmation exists*" (Fig. 13, using the language used by COGITANT on the left and the graphical representation on the right). Conversely, the requirement previously written (ResourceAvailabilityError) is intentionally built with an error. In that case, the requirement is added into the repository, can be modified (without lock) but cannot be used for verification.

To extend the case study from the originating requirement to the writing of a requirement and finally its verification, let's consider the stakeholder requirement:

"*The duration of interactions between participants must not exceed x time units and the duration of interaction between the activities within an internal process must not exceed y time units*".

In this format, the stakeholders have two ways to verify this need. First, they can use expertise and perform a complete review of the model and its inputs related to the times and durations of the activities and interactions to be sure that they meet the expectation. This is relevant but it can require time and several resources with the possibility to get an error. Second, they can take time to build a requirement based on a defined grammar that guides the writing. They have to write the originating requirement according to the elements in the grammar that allows considering and expressing the duration between participants and within a participant. Thus, the proposed language is enough constrained to

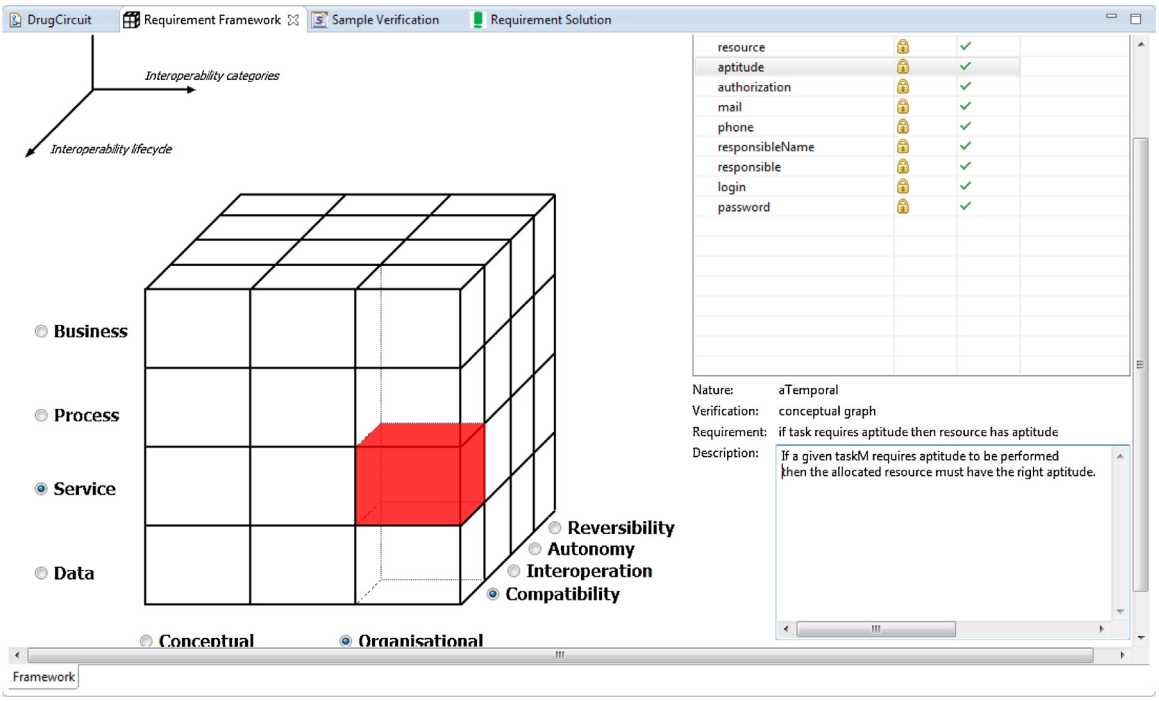

Fig. 10. Positioned requirements in the repository and the set of information provided.gr10

express a requirement accurately and to be mapped with a formal language to be verified. Obviously, the use of such a language (as for any languages) requires a minimum of knowledge and time to be mastered but as mentioned in introduction, the time spent in the expression of the problem allow to avoid mistakes in the later phases. Hence, although the writing step is guided, it is required to understand the semantic and the syntax of the grammar to be sure that the written requirement is the right expression of the originating requirement. As a consequence, the requirement

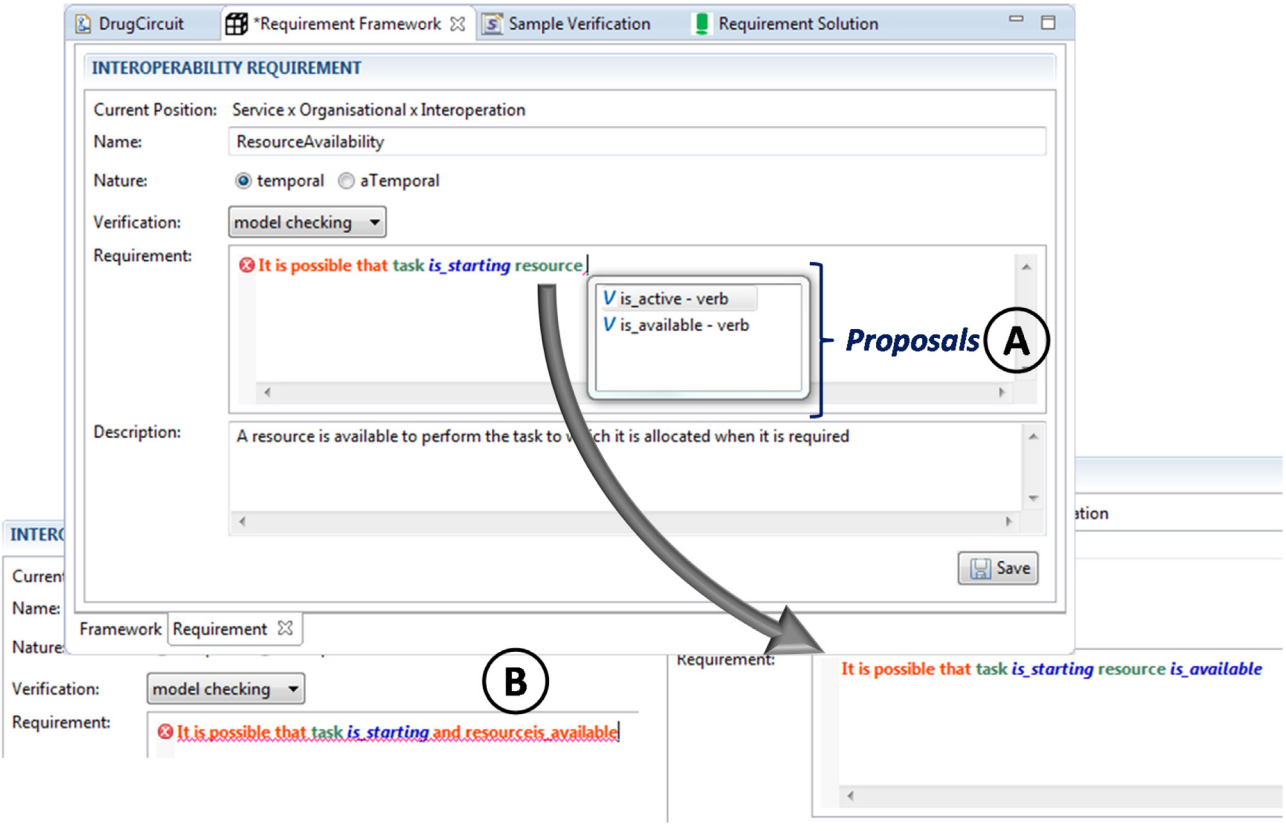

Fig. 11. Writing of a temporal interoperability requirement.

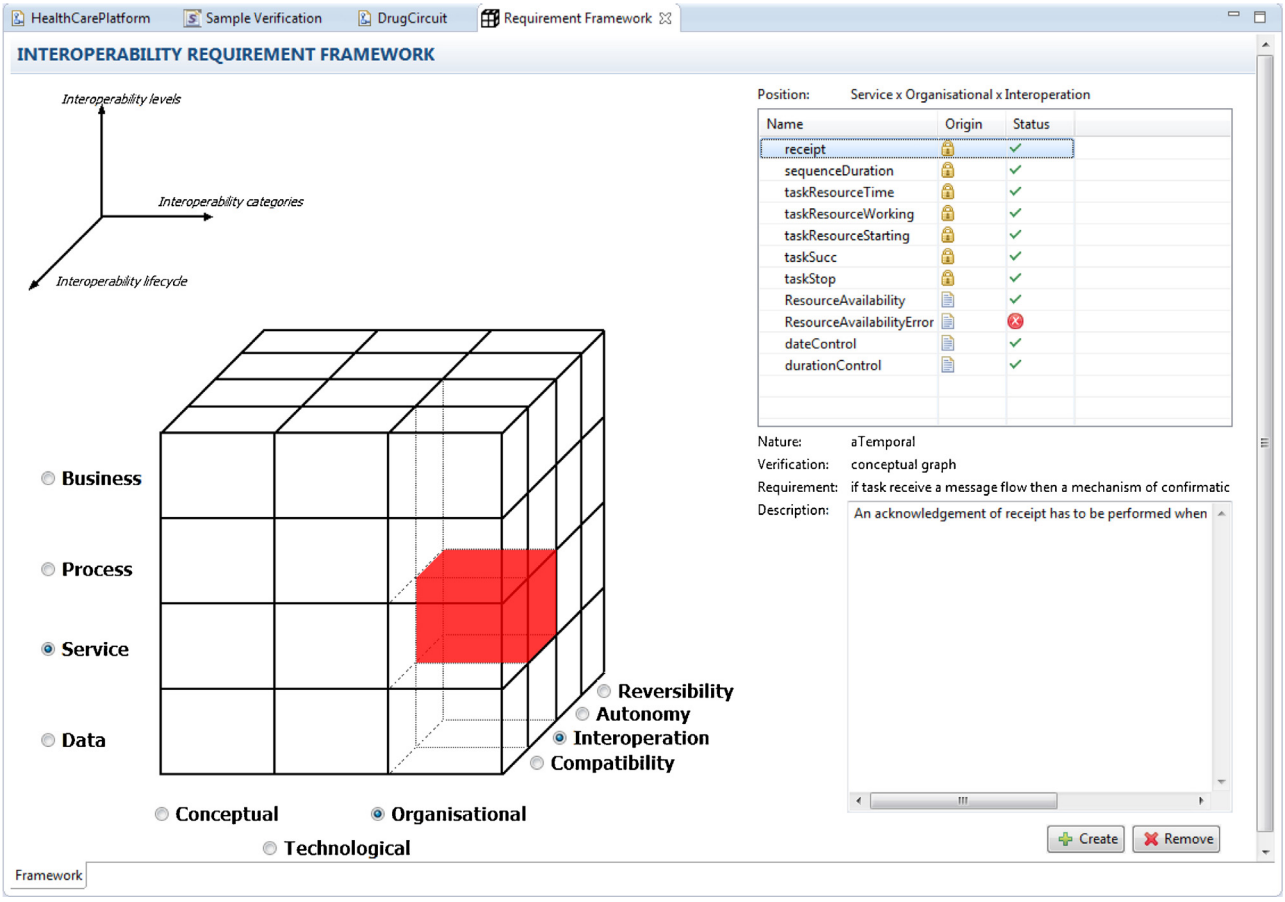

Fig. 12. Requirements added to the repository (without locks, one correct and one with an error).

shown in natural language can be write with the proposed controlled language as:

"*It is possible that for all message_flows, emission_message_end [index] minus reception_message_start [index] is_less_than value and for all sequence_flows, reception_sequence_start [index] minus emission_sequence_end [index] is_less_than value*"

In this requirement, "message_flows" represents the interactions between the participants, "sequence_flows", the interactions

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cogxml>
  <graph id="receipt" nature="PositiveConstraint">
    <concept id="_C1" labelType="Task" color="0"/>
    <concept id="_C2" labelType="MessageFlow" color="0"/>
    <concept id="_C3" labelType="ExtensionAttributeValue" color="1"/>
    <concept id="_C4" labelType="value" color="1"/>
    <concept id="_C5" labelType="Receipt" color="1"/>
    <concept id="_C6" labelType="receiptMechanism" color="1"/>
    <concept id="_C7" labelType="Value" labelMarker="true" color="1"/>
    <relation id="_R1" labelType="MessageFlow#targetRef#InteractionNode" color="0 "/>
    <relation id="_R2" labelType="BaseElement#extensionValues#ExtensionAttributeValue" color="1"/>
    <relation id="_R3" labelType="hasAttribute" color="1"/>
    <relation id="_R4" labelType="hasValue" color="1"/>
    <relation id="_R5" labelType="hasAttribute" color="1"/>
    <relation id="_R6" labelType="hasValue" color="1"/>
    <edge label="2" cid="_C1" rid="_R1"/>
    <edge label="2" cid="_C3" rid="_R2"/>
    <edge label="2" cid="_C4" rid="_R3"/>
    <edge label="2" cid="_C5" rid="_R4"/>
    <edge label="2" cid="_C6" rid="_R5"/>
    <edge label="2" cid="_C7" rid="_R6"/>
    <edge label="1" cid="_C2" rid="_R1"/>
    <edge label="1" cid="_C1" rid="_R2"/>
    <edge label="1" cid="_C3" rid="_R3"/>
    <edge label="1" cid="_C4" rid="_R4"/>
    <edge label="1" cid="_C5" rid="_R5"/>
    <edge label="1" cid="_C6" rid="_R6"/>
  </graph>
</cogxml>
```

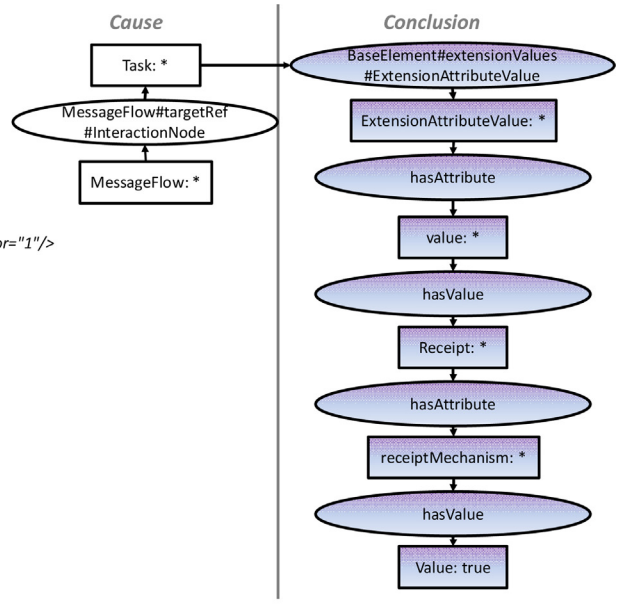

Fig. 13. Predefined requirement using the conceptual graph technique.

between activities inside a process, “reception_message_start” (“emission_sequence_end” . . .) represents the dates of emission/reception in the collaborative process, “[index]” the considered message flow(s) and sequence flow(s) and, “value”, the time to not exceed.

Then, the written requirement must be instantiated and mapped (automatically) into the targeted formal language to be verified. Regarding to the example the following property is getting:

E<> forall(i:NbMessageFlow)emission_message_end [i]-reception_message_start [i]<5 and forall(i:NbSequenceFlow)reception_sequence_start [i]-emission_sequence_end [i]<10

In this formal requirement, the stakeholders are interested in all interactions (index “i”) and choose different value to not exceed (5 time units between participants and 10 time units inside a process). This process – from an originating requirement expressed in the Natural Language to a formal requirement expressed in a Formal Language – shows that a controlled natural language to express a requirement can act as pivot between the Natural Language with its imperfection and its lack of reliable verification means and a Formal Language hard to use and understand but having a reliable support tool to perform the verification.

To return to the initial case study, requirements must be instantiated (Fig. 14) according to their constituent concepts (e.g. task, resources, clock, time) and elements existing in the collaborative process (e.g. “to provide opinion and analysis”, “to supply”, Nurse, Pharmacist). For this purpose, a set of instantiation proposals are directly submitted, in accordance with the selected requirement (coming from its position in the repository), leaving stakeholders to choose the relevant proposals with respect to their verification goal. Hence, the first requirement *“if task requires aptitude then resource has aptitude”* must be instantiated with the name of the set of tasks and resources defined on the collaborative process model (instance proposals). The resulting instantiation leads to the requirement being defined as *“if To give opinion and analysis requires aptitude then MedicalPractitioner has aptitude”* (instantiation result). The instantiation of the second requirement, i.e. *“it is possible that task is_starting and resource is_available”* (interoperability requirement) follows the same principle and must be instantiated with the tasks and resources (instance proposals). The resulting requirement is automatically generated and, in this example, becomes *“It is possible that To give opinion and analysis is_starting and MedicalPractitioner is_available”* (instantiation result).

Lastly, temporal and non-temporal requirements can be verified, which can then be provided to the stakeholders as results. Let's note that the mapping and verification steps are not visible to stakeholders, only the verification result is given.

As shown in Fig. 15, the temporal requirement (upper frame) is satisfied, which implies that the resource (medical practitioner) is effectively available when requested by the activity (“to give opinion and analysis”). More precisely, the model checker explores all process paths (transformed into a network of timed automata [54]) and ensures that a path exists whereby the activity is in the “start” state and the resource in the “available” state. In contrast, the non-temporal requirement *“aptitude”* (lower frame) is not verified, which indicates that the resource cannot appropriately perform its assigned activity due to an aptitude defect. More specifically, the projection mechanisms offered by conceptual graphs confirm that all aptitudes requested by the activity to be performed are effectively held by the allocated resource. According to the verification result, stakeholders must decide, define and apply corrective actions to ensure that the requirement is satisfied. Note that once satisfied, the non-temporal requirements called “responsible” (not presented herein) were intended to ensure that the Medical Practitioner would interact with the right person at the right time and right place in the pharmacy.

6. Discussion

The proposed approach aims to facilitate and guide the use of interoperability requirements within collaborative processes by means of comparison with hazardous approaches. Its support of this aspect involves an attempt at reducing the cognitive load

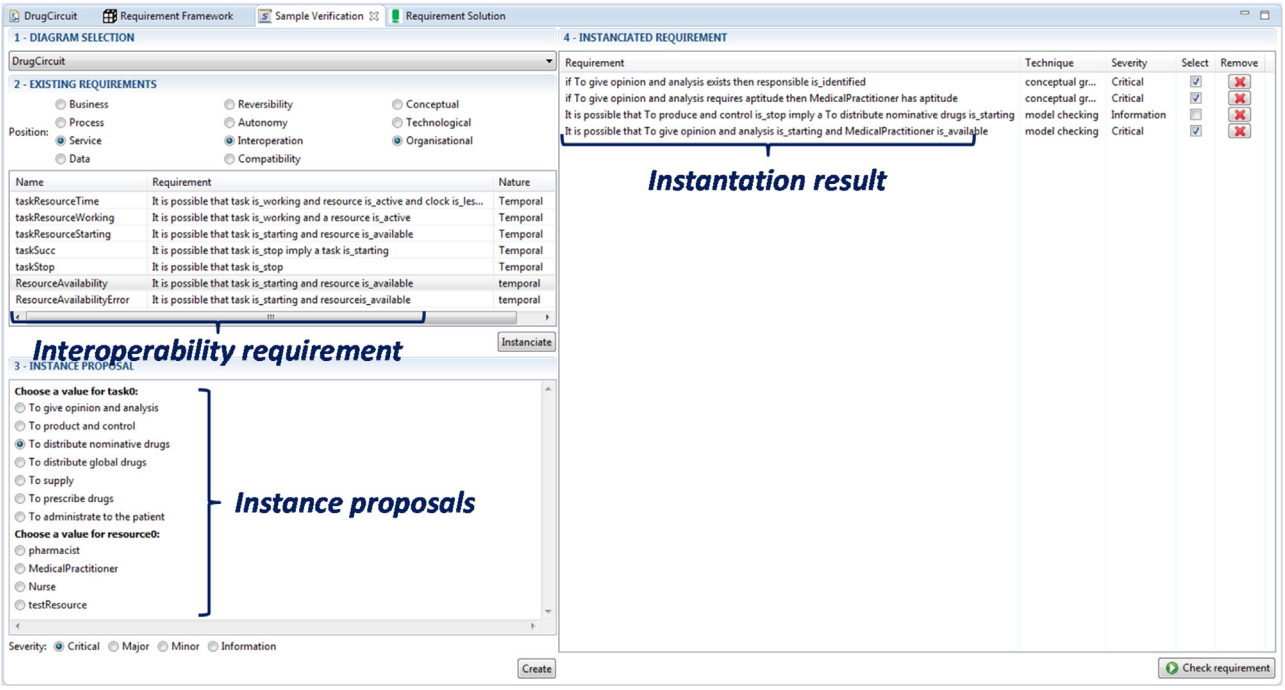

Fig. 14. Interoperability requirement instantiation.

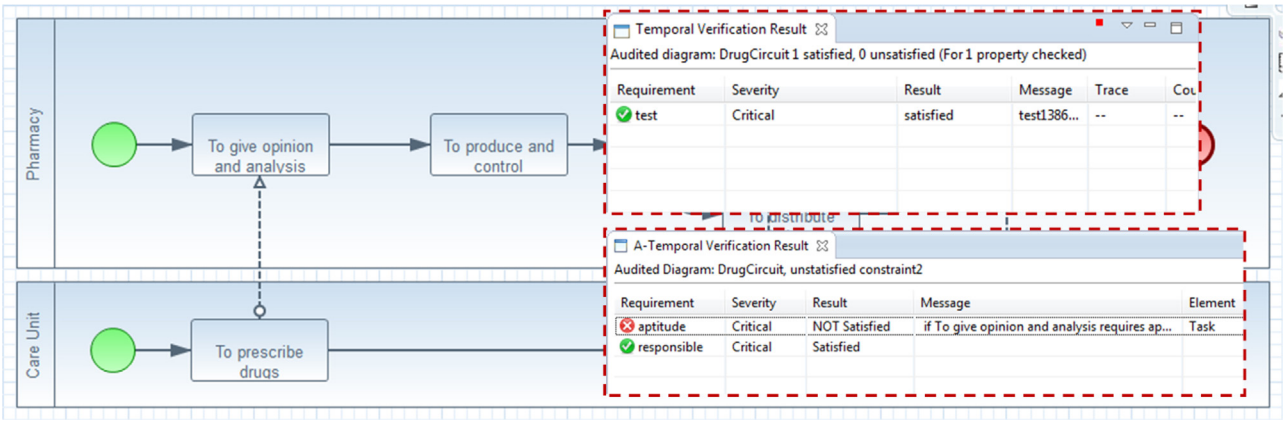

Fig. 15. Results of interoperability requirement verification.

required to manage-write-verify requirements, in order to add value to the process. The repository introduced allows properly structuring the requirements in agreement with the studied field and encompassing key requirements in the collaborative process so as to save (or at least not lose) time for their verification rather than deal with an endless inventory. Writing/instantiation serve to express requirements without strong knowledge using formal technique and mechanisms that avoid mistakes, either with rules during the writing process or with proposed objects belonging only to the studied collaborative process model during instantiation. Moreover, the verification process including rewriting and model transformation is not visible for the end user and merely outputs the final result. Obviously, all requirements cannot be verified with formal techniques, but those that can are no longer being assigned to stakeholders.

Although this approach has laid the initial concepts and tools, numerous points can still be improved in order to further support stakeholders' efforts to avoid bad practices. Despite the fact that conflict identification between requirements stems from different stakeholders and that this kind of structure supports redundancy, the repository remains dependent upon human expertise and its possible absence, especially when the set of requirements becomes substantial. Automated means can be implemented to address this issue; to this end, the work performed on ontologies or semantic similarity measurements may prove useful and substantiate this aspect of requirement engineering. The solution issue however is still inadequately developed; hence, a solution can only be linked to a requirement. As mentioned previously, interoperability is part of a broad set of requirements to be satisfied, and any given solution can affect the other requirements (functional as well as non-functional) and further the collaboration of a given partner. Hence, the link between requirements in terms of dependence and the possible nature of this link in terms of impact/influence must be established. The work presented in [57] has been performed along these lines and reveals the possible dependencies among "-ilities", including interoperability. The final goal consists of making available the most appropriate solution by taking into account the environment of interoperability requirements and stakeholder priorities (e.g. investment/gain tradeoff). Another aspect to be considered is the existing approaches [56] to develop interoperability that may also be useful in choosing a relevant solution according to the temporal aspect of a partnership (long vs. short term). The choice of a solution is therefore not solely related to the non-verification of an interoperability requirement but also to numerous parameters of varying importance and for various partners. The link between interoperability requirements and interoperability solutions needs to be refined, and the choice of solution is to be guided with as much finesse as possible.

Regarding the proposed DSL, it is highly correlated with the process modeling language and extensions performed on it. As such, this initial version must consider, if possible, other objects belonging to BPMN as well as potential extensions to be added (e.g. equipment and their attributes). The DSL thus needs to be refined to take these objects into consideration within a requirement. The mapping rules between DSL and the language used by the formal verification technique (model checking plus the conceptual graph) must consequently be refined. One expected characteristic of the proposed DSL is to be usable by anyone with a minimal learning process. For instance, the modalities currently expressed are understandable, but other objects remain difficult to use as is (e.g. *Invariantly, for all tasks, task_end[index] minus task_start[index] is_less_than value*) and only a few have been developed or proposed. Furthermore, the TCTL grammar offers other advanced objects (e.g. urgent), requiring knowledge of their relevance and utility, which in turn will require developing their expression close to the natural language and implementing them in the DSL without any loss of the original meaning.

Finally, the use of such an approach and, more widely, the concepts stemming from Systems Engineering and Requirements Engineering falls under the responsibility of actors involved in the collaboration. This set-up fits into a change management process that extends from experiential approaches to more pragmatic approaches. Stakeholders however must keep in mind that the main point is to strike the right balance between the weaknesses (e.g. loss of a requirement) along with the broad freedoms (e.g. possibility to express anything) offered on the one hand and the difficulties (e.g. constrained grammar) despite reliable results and supporting tools (e.g. formal verification) offered by the other.

7. Conclusion and outlook

Requirements engineering is crucial to designing a system that achieves its own missions and reaches its performance goals. Prior to producing any system design, it is necessary to establish requirements that are: (1) well written, (2) appropriate and relevant to the studied field, (3) verifiable, and (4) accessible and understandable. The lack of a clear repository for managing requirements and dedicated languages for writing requirements leads to taking the risk of failing to meet certain expectations (non-related requirement, conflict, omission, lack of traceability, etc.). In the work presented herein, these characteristics have been applied to the field of collaborative process analysis in order to detect interoperability problems before implementing the process or while running the process so as to make the proper adjustments.

This article has presented a repository for structuring interoperability requirements that: (1) constitute a set of predefined and

available requirements (some 30 in all, temporal as well as non-temporal, present in the literature or derived from stakeholders); and 2) may be used as a guide to elicit other interoperability requirements. Another challenge is to allow stakeholders, either with or without a weak knowledge of formal verification techniques, to accurately write interoperability requirements. To this end, a specific DSL has been proposed to assist and guide stakeholders in writing their requirements and overcoming the problems of ambiguity, redundancy and inconsistency. Finally, the proposed approach also allows instantiating requirements in accordance with the studied collaborative process model and rewrite them (using mapping rules) into properties, in the case of temporal requirements, so as to enable their verification using UPPAAL.

Although the repository for interoperability requirements makes it possible to select and verify non-temporal requirements (based on the conceptual graphs), no dedicated language is available for writing non-temporal requirements. As such, it would be necessary to develop this DSL, in establishing proper mapping and mechanisms for its instantiation, in order to enable verification with COGITANT. The challenge here is to tie a human readable DSL with a graphical language.

References[10]

[1] Guide to the Systems Engineering Body of Knowledge (SEBoK), version 1.0, available at: http://www.sebokwiki.org/wiki/Main_Page.
[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] ISO, Advanced Automation Technologies and Their Applications—Part 1: Framework for Enterprise Interoperability, International Organization for Standardization, 2011 (ISO 11354, ISO/TC 184/SC 5).
[5] D. Chen, G. Doumeingts, F. Vernadat, Architectures for enterprise integration and interoperability: past, present and future, Comput. Ind. 59 (7) (2008) 647–659.
[6] R. Chalmeta, V. Pazos, A step-by-step methodology for enterprise interoperability projects, Enterp. Inf. Syst. 9 (4) (2015) 436–464.
[7] C4ISR Architecture Working Group (AWG), Levels of Information Systems Interoperability (LISI), USA Department of Defense, 1998.
[8] C.T. Ford, Interoperability Measurement, Ph.D. Thesis, Department of the Air Force Air University, Air Force Institute of Technology, 2008.
[9] N. Daclin, S. Mallek, Capturing and structuring interoperability requirements: a framework for interoperability requirements, Enterprise Interoperability VI Proceedings of the I-ESA Conferences Volume 7 (2014) 239–249.
[10] P. Dasgupta, A Roadmap for Formal Property Verification, Springer, 2010 (ISBN: 978-90-481-7185-9).
[11] IEEE, IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries, Institute of Electrical and Electronics Engineers, New York, 1990.
[12] X. Wang, X.W. Xu, DIMP: an interoperable solution for software integration and product data exchange, Enterp. Inf. Syst. 6 (3) (2012) 291–314.
[13] R. Jardim-Goncalves, A. Grilo, C. Agostinho, F. Lampathaki, Y. Charalabidis, Systematisation of interoperability body of knowledge: the foundation for enterprise interoperability as a science, Enterp. Inf. Syst. 7 (1) (2013) 7–32.
[14] H. Panetto, J. Cecil, Information systems for enterprise integration, interoperability and networking: theory and applications, Enterp. Inf. Syst. 7 (1) (2013) 1–6.
[15] B. Vallespir, D. Chen, Y. Ducq, Enterprise modelling for interoperability, 16th IFAC World Congress, Prague, Czech Republic, July, 2005.
[16] D. Chen, M. Dassisti, B. Elveaeter, Enterprise Interoperability Framework and Knowledge Corpus—Final Report, Interop Deliverable Di.3, May, 2007.
[17] N. Daclin, S. Mallek, Towards a sustainable implementation of interoperability solutions: bridging the gap between interoperability requirements and solutions, 6th International IFIP Working Conference on Enterprise Interoperability (IWEI 2015), Nîmes, France, 28–29 may, 2015.
[18] J. Favaro, H.S. De Koning, R. Schreiner, X. Olive, Next generation requirements engineering, 22th Annual INCOSE International Symposium, Rome, Italy, 2012.
[19] The Standish Group, The Chaos Report, The Standish Group International, Inc, 1995, 2016.
[20] G.V. Bochmann, Writing Better Requirements, University of Ottawa, 2010 (slide only).
[21] INCOSE. *Systems Engineering Handbook—A guide for System life cycle processes and activities*, INCOSE Systems Engineering Handbook v. 3.2.2INCOSE TP 2003 002 03.2.2, October 2011.
[22] A.M. Ross Defining and using the new ilities, Systems Engineering Advancement Research Initiative (SEARi) working paper series, 2008.
[23] J. Willis, Systems Engineering and the forgotten '-lllities, 14th Annual Systems Engineering Conference, San Diego, CA, the United States of America, 2011.
[24] INCOSE. Guide for writing requirements, v1.0, Requirements WG-INCOSE, INCOSE-TP-2010-006-01, 2012.
[25] ISO/IEC/IEEE. Systems and software engineering – Life cycle processes – Requirements engineering, ISO/IEC/IEEE 29148:2011, 2011.
[26] Open Management Group (OMG): Business Process Model and Notation (BPMN) – version 2.0, available at: http://www.omg.org/spec/BPMN/2.0/, January 3rd (2011).
[27] J.F. Sowa, Conceptual graphs, IBM J. Res. Dev. 20 (4) (1976) 336–357.
[28] Cogitant: CoGITaNT Version 5.2.0, Reference Manual (http://cogitant.sourceforge.net) (2009).
[29] Edmund M. Clarke Jr., O. Grumbereg, A.P. Doron, Model Checking, The MIT Press, 1999.
[30] G. Behrmann, A. David, K.G. Larsen, A Tutorial on Uppaal, Department of Computer Science, Aalborg University, Denmark, 2004.
[31] S. Mallek, N. Daclin, V. Chapurlat, The application of interoperability requirement specification and verification to collaborative processes in industry, Comput. Ind. 63 (7) (2012) 643–658.
[32] NEHTA. 2007. Towards a Health Interop Framework. Version 1.0. www.nehta.gov.au.
[33] A.A. Sinaci, G.B. Laleci Erturkmen, A federated semantic metadata registry framework for enabling interoperability across clinical research and care domains, J. Biomed. Inf. 46 (5) (2013) 784–794.
[34] Department of Defense, Department of Defense Dictionary of Military and Associated Terms, US Department of Defense, 2001.
[35] ISO. 2002. ISO 16100, 1 – Industrial Automation Systems and Integration – Manufacturing Software Capability Profiling for Interoperability—Part 1: Framework. Geneva : International Organization for Standardization. ISO/TC 184/SC 5/WG 4, 2002.
[36] ATHENA Integrated Project. Requirement for interoperability framework, product-based and process-based interoperability infrastructures, interoperability life-cycle services, ATHENA deliverable A4.1, 2005.
[37] EIF, European Interoperrability Framework for pan-European eGovernment services, European Commission, version 2.0, 2008.
[38] C4ISR, Levels of Information Systems Interoperability (LISI), USA Department of Defense, Architecture Working Group (AWG), 1998.
[39] A. Tolk, S.Y. Diallo, C.D. Turnitsa, Applying the levels of conceptual interoperability model in support of integratability, interoperability, and composability for system-of-systemsengineering, Syst. Cybern. Inf. 5 (5) (2007) 65–74.
[40] S. Fewell, T. Clark, Organisational interoperability: evaluation and further development of the OIM model, 8th International Command and Control Research and Technology Symposium, Washington DC, 2003.
[41] IEC, Common Automation Device—Profile guideline, IEC 32390, TC 65: Industrial Process Measurement and Control, 2005.
[42] M. Kasunic, W. Anderson Measuring systems interoperability: challenges and opportunities, Software engineering measurement and analysis initiative, Technical note CMU/SEI–2004–TN–003, 2004.
[43] R. Rezaei, T.K. Thiam Kian Chiew, S.P. Sai Peck Lee, Z.S. Zeinab Shams Aliee, Interoperability evaluation models: a systematic review, Comput. Ind. 65 (1) (2014) 1–23.
[44] Kang Shian Chin, Pee Eng Yau, Sim Kok Wah, Pang Chung Khiang, Framework for managing System-of-systems ilities, DSTA Horizons, 2013/14.
[45] The Open Group Architecture Framework. TOGAF version 9, chapter 29 Interoperability requirements, 2011.
[46] S. Harrison, J. Brogden Interoperability requirements, Supplier Requirement for SMART Metering (SRSM) project report, 2007.
[47] A. Van lamsweerde, A. Dardenne, B. Delcourt, F. Dubisy, The KAOS Project, Knowledge Acquisition in Automated Specification of Software, AAAI Spring Symposium Series, American Association for Artificial Intelligence, 1991.
[48] G. Fanmuy, J. Llorens, A. Fraga, Requirements verification in the industry, CSDM (2016) 2011.
[49] Use Case Map (UCM) notation, available at: http://jucmnav.softwareengineering.ca/ucm/bin/view/UCM/AboutUseCaseMaps.
[50] INCOSE, REGAL. Requirements Engineering Guide for All, but applicable to Systems Engineering, 2008.
[51] 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).
[52] University of Toronto. Goal-oriented Requirement Language, available at: http://www.cs.toronto.edu/km/GRL/.
[53] 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 2008.
[54] S. Mallek, N. Daclin, V. Chapurlat, B. Vallespir, Enabling model checking for collaborative process analysis: from BPMN to Network of Timed Automata, Enterp. Inf. Syst. (2015), doi:http://dx.doi.org/10.1080/17517575.2013.879211.

[10] All links in the references are accessible as of September 2015.

[55] ISO (2002). *Enterprise Integration—Framework for Enterprise Modelling*, International Organization for Standardization, ISO/CEN 19439CEN TC310/WG1 and ISO TC184 SC5/WG1.
[56] ISO. ISO 14258—Industrial Atomation Systems—Concepts and Rules for Enterprise Models, ISO TC184/SC5/WG1, 1999.
[57] A.M. Ross, D.H. Rohdes, Towards a prescriptive semantic basis for change-type ilities, Procedia Comput. Sci. 44 (2015) 443–453.
[58] D. Norfolk *Reducing the risk of development failure . . . with cost-effective capture and management of requirements*, a white paper by Bloor Research, available online at http://www.ibm.com/, July 2013.
[59] Dassault Systèmes, *ENOVIA Requirements Central Essentials*, student guide, 2008–2011.
[60] INCOSE, A world in motion—Systems Engineering Vision 2025, 2014.
[61] L. Camarinha-Matos, H. Afsarmanesh, Collaborative networks: a new scientific discipline, J. Intell. Manuf. 16 (2005) 139–452.
[62] Q. Li, Z. Wang, W. Li, J. Li, C. Wang, R. Du, Application integration in a hybrid cloud computing environment: modelling and platform, Enterp. Inf. Syst. 7 (3) (2016) 237–271, doi:http://dx.doi.org/10.1080/17517575.2012.677479,2013.
[63] A.Wyner,K.Angelov,G.Barzdins,D.Damljanovic,B.Davis,N.Fuchs,S.Hoefler, K. Jones, K. Kaljurand, T. Kuhn, M. Luts, J. Pool, M. Rosner, R. Schwitter, J. Sowa, On Controlled Natural languages: Properties and Prospects, Workshop on Controlled Natural Language, CNL. 2009, Marettimo Island, Italy, June 8–10, 2009.
[64] R. Schwitter, Controlled natural languages for knowledge representation, Proceeding COLING '10 Proceedings of the 23rd International Conference on Computational Linguistics: Posters, Beijing, China, 23–27 August, 2010.
[65] K.J. Turner, Using Formal Description Techniques—An introduction to ESTELLE, LOTOS and SDL, John Wiley and Sons Ltd., 1993.