An Approach for Interoperability Requirements Specification and Verification

Sihem Mallek, Nicolas Daclin, and Vincent Chapurlat

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

Abstract. Enterprises are today involved in collaborative processes with other partners sharing common economical interests in confidence. This allows these enterprises to focus on their core business, to optimize, and to be effective to respond to customers' needs. Implicitly, a partner that wishes to become involved in a partnership must demonstrate numerous qualities and enable to gain the confidence of other partners. Among other ones, demonstrate its ability to be interoperable is a major issue. This research work aims to define, to formalize and to analyze a set of interoperability requirements that each partner of a collaborative process have to satisfy prior to any collaboration. This paper focuses and illustrates how interoperability requirements related to the static and dynamic aspects of the collaboration may be formalized and verified by the use of a formal verification technique.

Keywords: interoperability, interoperability requirements, compatibility, interoperation, verification, model checker, conceptual graphs, collaborative process.

1 Introduction

A collaborative process can be defined as "*a process whose activities belong to different organizations*" [1]. It is a way allowing to formalize how partners (enterprises for inter organizational collaborative processes or team for intra-organizational processes) may work together regarding a common objective that is usually defined to provide - faster and efficiently - products and services (to design, to produce, to deliver…) to their stakeholders. However, before being involved with confidence in a collaborative structure, each partner may have to assume and, if needed, to demonstrate that it possesses relevant qualities and it respects needs regarding the type, the requested role and the nature of the collaboration. One of them is related to its ability to interoperate harmoniously and efficiently with other partners, in other words to be interoperable as defined in [2] as the "*ability of enterprises and entities within those enterprises to communicate and interact effectively*". Therefore, to help partners involved in a collaborative process to find their interoperability problems, this research work focuses on the detection from an anticipative manner – *i.e.* before the implementation of the collaborative process - of interoperability problems that can be induced by characteristics or behaviors of partners. In this perspective, the anticipation of a problem requires to perform analysis on a model of the collaborative process. Then interoperability problems are extracted and characterized from interoperability needs of partners. Finally, to demonstrate that a need is satisfied or covered, several verification techniques can be implemented.

From these considerations, this research work aims, first, to define, to structure and to formalize interoperability needs that have to be satisfied by the partners. Second, it aims to promote and implement a set of formal verification techniques that can be used prior to any concretization of the collaborative process.

This paper focuses on the formalization and verification of interoperability requirements to be verified including static and dynamic aspects of the collaboration. It is structured as follows. Section 2 reminds the principles and classification of interoperability requirements. Section 3 introduces the proposed mechanisms used to analyze interoperability requirements. Section 4 presents the verification of static requirements using conceptual graphs. The verification of dynamic requirements using model checker is given section 5. To illustrate the verification of these kinds of requirements, an application case is given in section 6.

2 Interoperability Requirements Definition

A requirement is defined as "*a statement that specifies a function, ability or a characteristic that a product or a system must satisfy in a given context*" [3]. In other words, a requirement translates from an unambiguous manner any need. With regards to (1) the interoperability barriers and interoperability concerns proposed in the interoperability framework [4], (2) the maturity models [5] [6] [7] and several projects such as ATHENA [8] and, (3) an investigation made from enterprises to collect their interoperability needs, three classes of interoperability requirements have been defined such as:

- **Compatibility requirements:** A compatibility requirement is defined as "*a statement that specifies a function, ability or a characteristic, independent of time and related to interoperability barriers (conceptual, organizational and technological) for each interoperability concerns (data, services, processes and business), that enterprise must satisfy before collaboration effectiveness*". Compatibility means to harmonize partners (method, organization, tool...) in order to be ready to collaborate. For instance, a compatibility requirement can be given as: "*A right access to shared data is allowed to external partners*". However, compatibility focuses on a static point of view of the collaboration and remains insufficient to determine if enterprises are interoperable during the execution of the collaborative process. It is necessary to consider the evolution of the context and of the situation of each partner.
- **Interoperation requirements:** An interoperation requirement is defined as "a statement that specifies a function, ability or a characteristic, dependent of time and related to the performance of the interaction, that enterprise must satisfy during the collaboration". These requirements focus on the ability of the enterprise to be able to adapt its organization, its functioning modes and its behavior when it interacts. For example, an interoperation requirement can be described as: "For each data received, a receipt must be returned".

– **Reversibility requirements:** A reversibility requirement is defined as "a statement that specify functions, abilities or characteristics related to the capacity of enterprise to retrieve its autonomy and to back to its original state (in terms of its own performance) after collaboration, that enterprise must satisfy". Reversibility means that an enterprise may maintain or retrieve easily its autonomy and performance (including positive and/or negative variations that are accepted) at the end of any collaboration. For instance, a reversibility requirement for cost criterion at the level of service is described by: "the cost of a given service after the collaboration corresponds to the cost before collaboration including variations (e.g. admissible increase of cost)".

An interoperability requirement can be qualified as a static or not temporal requirement, *i.e.* independent of time, and has to be verified all along the process evolution. Conversely, it can be qualified as a dynamic or temporal requirement, *i.e.* dependent of temporal hypotheses and time evolution, and has to be verified only at some stages of the collaboration. Thus, compatibility requirements are static, interoperation requirements are dynamic and reversibility requirements can have both aspects. The description, formalization and understanding of a requirement can be difficult for many reasons: complexity, comprehensiveness, quantity of requirements, etc. To tackle this first obstacle, a requirement reference repository is proposed and is described as a causal tree, as illustrated in Fig. 1 (for more details, we refer the reader to [9]).

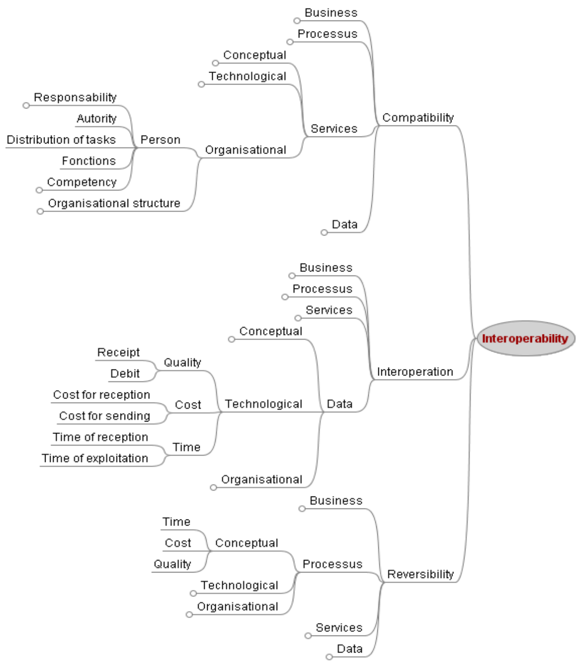

Fig. 1. Reference repository of interoperability requirements (partial view) [9]

A set of requirements represented as nodes in the causal tree and refinement relations from an abstract requirement (node) to a set of more precise requirements (subnodes) allows to obtain an oriented causal tree. The causal tree allows by successive refinement to reduce the ambiguity of requirements that may exist for each level.

Thereafter, to prove that each requirement is satisfied by the collaborative process model and, by the process itself in a formal manner, this research work proposes to apply verification activity. The objective is to ensure "*the confirmation by examination and proof that specified requirements have been satisfied*"[10]. Several verification techniques are presented in the next section.

3 Interoperability Requirements Verification

The objective of the verification is to demonstrate that a set of selected interoperability requirements is satisfied. Indeed the reference repository presented in [9] allows users to select relevant requirements to be checked. In order to be able to perform this verification before the runtime of the collaborative process, this one is done on a model of the collaborative process. Several verification techniques exist in the literature such as simulation, tests, or formal verification techniques [11] [12] [13].

The simulation is done on a theoretical model whose behavior is considered as similar to the behavior of the pointed out system. It is done before implementation of the system. However, simulation is unable to assume all behavioral scenarios of the system. It requires human expertise to analyze results and formulate the demonstration. Nevertheless, simulation is now a well known technique more and more developed and used in enterprise. A test is directly done on an existing system. It allows to check, for example, capacity and relevance to detect errors before system implementation.

On the other side, formal verification techniques allow to explore exhaustively a formal model *i.e.* a model obtained with a modeling language using a formal semantic. In this case, it is possible to provide a formal proof of the respect (or not) of a requirement independently from any human interpretation. In this way, it is proposed to use in a complementary way two formal verification techniques and to associate also a technical expertise as summarized in Fig. 2.

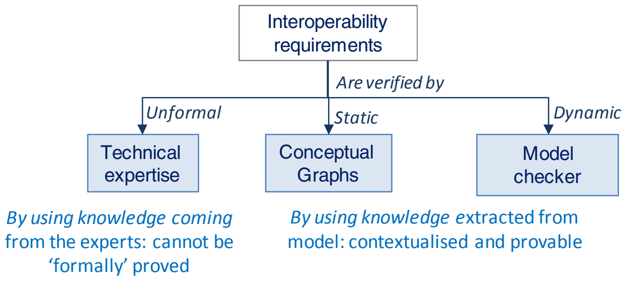
Interoperability requirements
Are verified by
Unformal
Static
Dynamic
Technical expertise
Conceptual Graphs
Model checker
By using knowledge coming from the experts: cannot be 'formally' proved
By using knowledge extracted from model: contextualised and provable

Fig. 2. Proposed verification techniques

The first verification technique is based on Conceptual Graphs [14] to verify static requirements. The advantage to use Conceptual Graphs is (1) to describe the collaborative process and interoperability requirements on the same formalism, (2) to dispose of a convenient graphical form to handle and, (3) to dispose of a mathematical foundation and mechanisms (projection, principles of rules and constraints principles) which are used to check static requirements.

The second one is based on model checking [15] for the dynamic requirements. The advantage to use model checker is (1) to include temporal aspect of the collaboration, (2) to consider all states of the collaborative process all along the collaboration and (3) to verify dynamic requirements exhaustively.

Applying these techniques requires to assume that the modeling language used to build the process model allows the description of interoperability requirements. The chosen modeling language is BPMN (Business Process Modeling Notation) [16]. It provides a standardized notation that is readily understandable by all actors involved in the design, development and monitoring of a collaborative process. However, it is necessary to enrich this modeling language to embed the interoperability requirements model. The proposed enrichments detailed in [17] include interoperability concepts such as the nature of the exchanged flow (information, energy, material and person), the availability of resources and their aptitudes.

Furthermore, the use of these verification techniques requires to translate the collaborative process model in enriched BPMN - thanks to model transformation rules - into an equivalent model upon which the formal verification techniques can be applied as shown Fig. 3. Indeed, the proposed enriched version of BPMN suffers yet from a lack of formalization and verification techniques cannot be applied directly, regarding to interoperability. The first equivalent model is obtained using a formal knowledge representation of Conceptual Graphs for static requirements proof as presented in [17] and [18]. In this case verification is performed with COGITANT tool [19]. The second equivalent model is obtained using a behavioral modeling language named Networks of Timed Automata for dynamic requirements proof. In this case, the model checker UPPAAL is used for various reasons: richness of TCTL temporal logic, open source, user friendly, and stand alone tool [15]. In both cases of target models, the required rules for model transformation are developed with ATL (Atlas Transformation Language) [20] in order to re-write the collaborative process model into Conceptual Graphs and Networks of Timed Automata. In the case of Conceptual Graphs, the transformation from enriched BPMN is of course not semantically preserved due to the rewriting hypothesis adopted. The objective is therefore to assume the coherence of the process model that is to say to prove that each BMPN modeling entity used and then instantiated in the process model is well and completely defined. In the case of Networks of Timed Automata, the transformation rules have been established respecting an equivalence between BPMN entities behavior and a standardized state model behavioral semantic. Therefore, under these equivalence hypothesis, these rules preserve the behavioral semantic of the enriched version of BPMN. Interoperability requirements are formalized to make their verification possible. Thus, static requirements are formalized with Conceptual Graphs and dynamic requirements are formalized with TCTL.

In other cases, if interoperability requirements highlight particular points of view of the process and cannot be described due to a limitation imposed by the modeling language, the technical expertise of the model is required. This aspect of checking is not considered in this work.

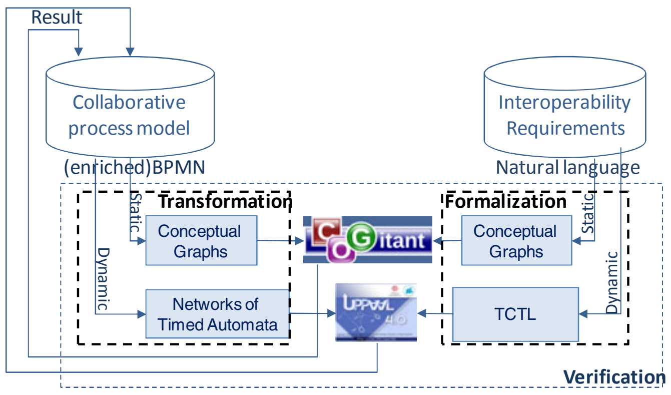

Fig. 3. Verification process for interoperability requirements

4 Verification Process for Static Requirements

A Conceptual Graph is defined as a graph with two kinds of nodes: concepts and oriented relations as shown in Fig. 4 with a conceptual graph that can be read as: *"Any activities* (concept) *begin* (relation) *at a beginning date* (concept)*"*. Concepts and relations are described in hierarchical structures called concepts and relations lattices. Individual markers are added to obtain the model ("*" means generic concept on the following figure).

[Activity: *] → (Begin) → [Date: BeginningDate]

Fig. 4. Example of a conceptual graph

The tool COGITANT allows to handle Conceptual Graphs and to make formal graph transformations upon which the verification process of static requirements is based. The principle is to use a graph operation named projection in order to check if a constraint graph (*i.e.* a requirement) is really projected in the conceptual graph that describe the static model of the system. If projection operation fails the requirement is not verified and the causes can be highlighted by analyzing the resulting conceptual graph. To make verification with COGITANT, three types of files are necessary and known as: support, fact and constraint graph.

The "*Support*" represents all the concepts and relations from enriched BPMN metamodel and markers representing all the instances of these concepts and relations defined in the process model. The *"Fact"* contains the equivalent conceptual graph of the model obtained by applying ATL transformation rules and respecting the support. Finally, the *"requirements"* to verify are modeled in other conceptual graphs called "*constraints*". The verification is performed using the projection of a positive or a negative constraint on the conceptual graph that represents the model of studied process. A positive constraint is described with a cause and a conclusion and its projection is performed according to the following interpretation: "*If the cause is true, then the conclusion must be true as well*". A negative constraint is a single conceptual graph and its projection is interpreted as: "*If a negative constraint is not projected on the fact model, it is verified*".

As mentioned the verification of a positive and negative constraint is made using the mechanism of projection. This mechanism involves to project a given property translated in conceptual graph on the obtained conceptual graph in the fact file that represents the translation of the model. If the projection fails, then the modeled constraint (*i.e.* requirement) is not verified and the causes are highlighted.

As a consequence, the transformation from process model to COGITANT requires to perform three ATL transformations. Hereafter, Fig. 5 represents the principle of the first transformation to get the support model (the two others transformations are based on the same principles and are not detailed here).

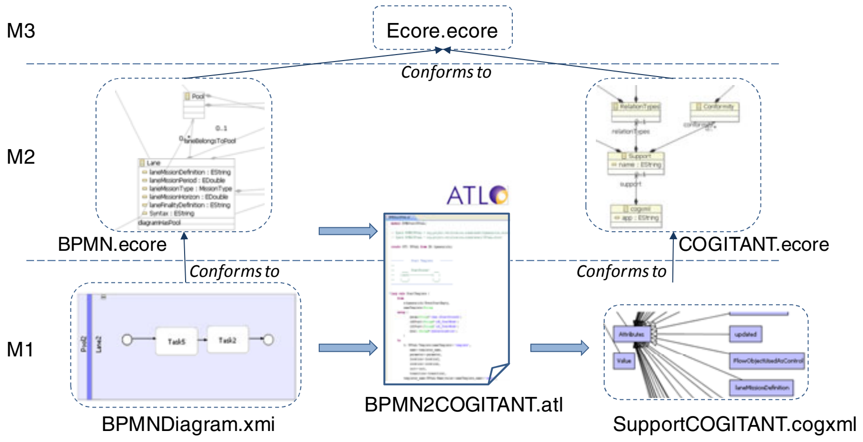

Fig. 5. Transformation from enriched version of BPMN to Support in COGITANT

The first transformation procedure to obtain the support file (level M1) starts with the consideration of the meta models (level M2) of the enriched BPMN language and COGITANT which are conform, as well, to the ecore model (level M3). Thus, each class (including its attributes) is translated into concept and each relation of the meta model is translated into relation in the support file. This transformation is made in order to provide all the needed concept and relations used and deployed, further, in the fact file.

The second transformation allows to obtain a representation into Conceptual Graph of the considered model (fact). Finally, the last transformation is performed to obtain constraints (representing the requirements) that have to be projected onto the equivalent graph model. In this case, the requirement is translated in a positive or negative conceptual graph constraint depending of the user intention.

For example, the compatibility requirement described as: "*Any task uses resources*" can be formalized into a positive constraint as shown Fig. 6. The verification of this constraint using the projection is performed with the projection of the cause (uncolored concept on left side) on the fact model. If the cause is projected on the fact model, the conclusion (colored concept and relation on right side) must be projected too in order to respect the requirement.

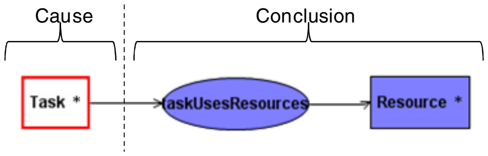

Fig. 6. Positive constraint representing a compatibility requirement

5 Verification Process for Dynamic Requirements

The principle of a model checker is to verify properties exhaustively with temporized and eventually constrained automata that describe the behavior of a system. Obviously, the system is here the collaborative process model.

Verification with model checkers requires two phases. The first phase consists to define a set of equivalent behavioral models of the collaborative process model and to define the collaborative process model transformation rules to be applied. The second phase consists to reformulate the dynamic requirements under the form of properties respecting the formal language adopted by the chosen model checker (in this study, a temporal logic) [21].

The chosen tool, UPPAAL, allows to handle a behavioral model defined as a set of templates, which communicates with synchronization (either on the form *Expression*! for sending or *Expression*? for receiving synchronization), using channels and syntax like sent/receive. Each template has locations and transitions to link a location source to a target source [15].

The enriched BPMN model must be transformed into Networks of Timed Automata to perform verification of dynamic requirements. In Fig. 7 the transformation procedure of models (level M1) starts with the consideration of the meta models (level M2) of the enriched BPMN language and UPPAAL which are conform, as well, to the ecore model (level M3). This transformation is made in order to provide all the needed concepts used and deployed in the Networks of Timed Automata. In this way, it is mandatory to consider all the modeling entities which will be used in the checking task. Thus, each class (including its attributes) of the meta model is translated into templates. Respecting this consideration, each BPMN element can be extracted from the collaborative process model in order to produce the corresponding template representing Networks of Timed Automata. Thus, these templates gather all the knowledge described in the model and represents the behavioral model of the collaborative process.

The proposed process model transformation is based on [22] which proposes the transformation of the few BPMN elements: start and end event, gateway (AND and XOR) and the Task. For instance, the task is transformed using four locations and two synchronizations as presented in Fig. 8 (a). In this figure, (b) and (c) consider the message flow between two tasks thus the single transformation proposed in the literature is extended. In the same manner, other own transformations such as resource, multi start/end event and so on, are fully developed in the frame of this research.

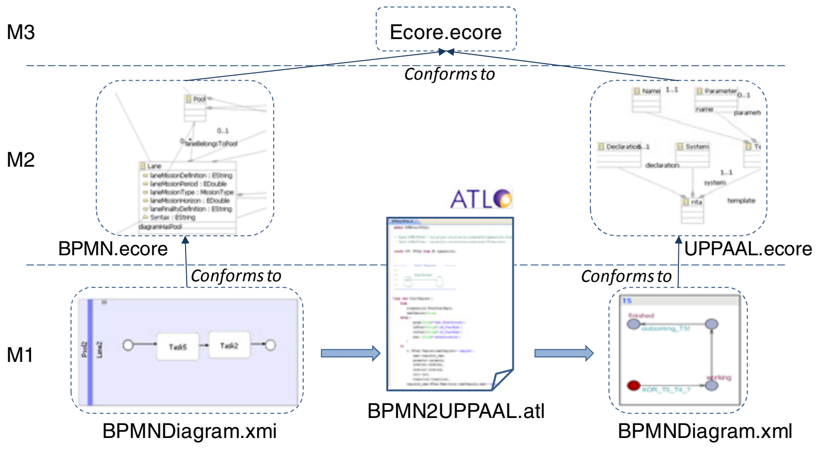

Fig. 7. Transformation from enriched version of BPMN to Networks of Timed Automata

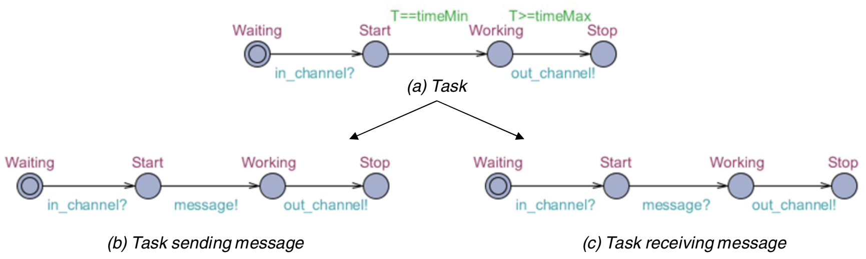

Fig. 8. Single Task model template (a) extended to consider the BPMN message flow (b and c)

To enable the implementation of formal verification techniques, the dynamic requirements are formalized into TCTL properties (Timed Computation Tree Logic *i.e.* the UPPAAL property specification language) [23]. TCTL is an extension of CTL (Computational Tree Logic) which allows considering several possible futures from a state of a system. The model checker UPPAAL has four TCTL quantifiers (A: for all paths, E: it exists a path, []: all states in a path, <>: some states in a path) allowing to write a property p:

– E<> p: reachability *i.e.* it is possible to reach a state in which p is satisfied.
– A[] p: invariantly p *i.e.* p is true in all reachable states.
– A<> p: inevitable p *i.e.* p will inevitable become true.
– E[] p: potentially Always p *i.e.* p is potentially always true.
– P $\rightarrow$ q: p leads to q *i.e.* if p becomes true, q will inevitably become true.

According to the templates defined above, the dynamic requirements written in natural language are manually re-written into properties using TCTL. Then the model checker UPPAAL verifies exhaustively properties in TCTL through all execution paths of the behavioral models that are reachable.

For instance, a requirement described as "*a task is working between T=5 time units and 10 time units*" can be formalized into a property using TCTL as:

"E<> Task.Working and T>5 and T<10"

This property indicates that a path can exists where a task is in the state Working between 5<T<10. This property can be verified on the template representing a task shown Fig. 8 (a).

To illustrate the proposed approach, an application case is given in next section to formally verify several static and dynamic interoperability requirements.

6 Application Case

To illustrate the proposed approach, an example of a collaborative process representing an European project called PABADIS'PROMISE is proposed [24]. This project extends the idea of distributed control to an innovative architecture which incorporates both resources and products. Furthermore, this project combines International forces to provide this architecture.

The project is composed of 8 work packages. A work package consists of several independent tasks which all together have to be performed in order to achieve the work package's goal. Each work package is under the authority of the Work Package Leader who will lead the work package team throughout the period of activity of all tasks involved in the work package. Each work package executes on its own control over internal tasks within the allocated resources. A task consists of a subset of activities within one work package.

Let us consider, a partner 1 that has to assume the work package 3 with three other partners (this application focuses only on the work package 3). This work package covers the development and implementation of the PABADIS'PROMISE manufacturing ontology, the manufacturing process and product description language based on it. To perform this work package, four tasks led by the four partners are highlighted. Each partner can be involved in all tasks. The task 1 titled "*development of manufacturing ontology*" is led by the partner 1. The task 2 "*Specification of product*", led by the partner 2, has to specify the PABADISE'PROMISE product and Production Process Description Language enabling the detailed description of products. The task 3 "*Implementation of product*", led by the partner 3, aims to implement the product and the process description and comparison systems. The last task "*Data protection and security aspect*", led by the partner 4, has to cover privacy, data protection, security and trust aspects related to technologies and organizational structures used on the PABADIS'PROMISE architecture. All interactions between tasks in this work package are presented Fig. 9 where the second and the last tasks work in parallel. The first task is triggered after receiving a message from a task of the work package 1. At the end of the work package 3, the work package 8 can start. The collaborative process shown Fig. 9 is a part of the full collaborative process between all work packages.

To illustrate the verification of the interoperability requirements using formal verification techniques previously presented, compatibility and interoperation requirements are effectively verified on this collaborative process.

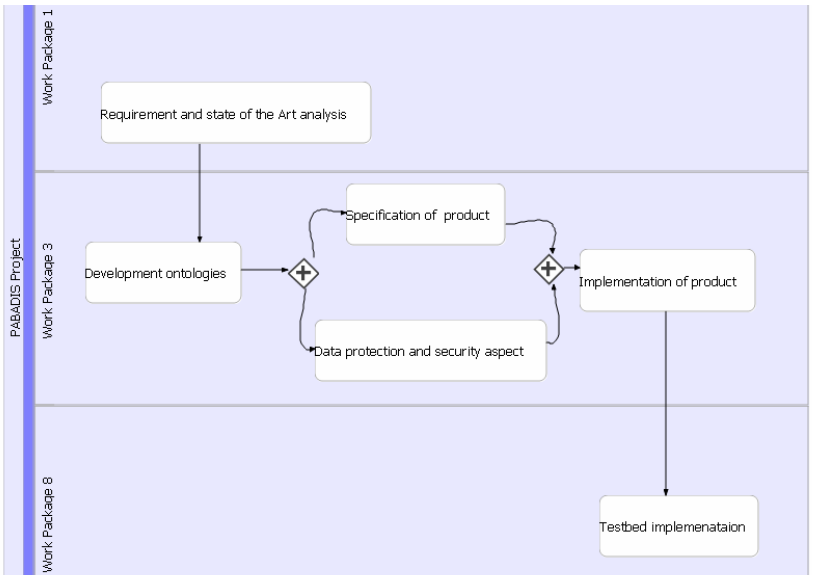

Fig. 9. Collaborative process for the work package 3 in PABADIS'PROMISE project

Before any collaboration between partners, each responsible or leader of each task must be clearly identified which can be typically a problem of interoperability. In fact, this identification is necessary to avoid loss of time to identify and to find the right responsible, that a non responsible person has access to confidential information... As a consequence, a compatibility requirement defined as: "*all tasks have an identified responsible"* must be verified using conceptual graphs thanks to COGITANT tool. This requirement is formalized into the positive constraint (a) as shown Fig. 10.

In PABADIS'PROMISE project, each leader has authority on the other partners. Therefore, another compatibility requirement defined as: "*each responsible of task has the authority on the other partners*" can be formalized into the positive constraint (b) shown Fig. 10 and verified on the fact model.

The verification of these compatibility requirements is performed using the projection of the cause (not colored concepts and relations) and the conclusion (colored concepts and relations) on the fact model. If causes are projected on the fact model and the conclusions are not projected, constraints are not verified which means that compatibility requirements are not satisfied. If these requirements are not satisfied on the fact model, it may dread mistakes during the transmission of orders that could lead to a deterioration of the collaboration performances (loss of time to convey the right order to the right person ...).

During the collaboration, partner 2 and partner 4 are involved in the two parallel tasks ("*specification of product*" (task 2) and "*Data production and security aspect*" (task 4)). The partner 2 is required on these two tasks simultaneously. As a consequence, it is necessary to verify if these two tasks do not use the partner 2 at the same time (*i.e.* to verify that a resource conflict does not exist on these tasks). The interoperation requirement described as: "*task 2 and task 4 uses the human resources of partner 2*" can be verified. If this requirement is verified as a static requirement using

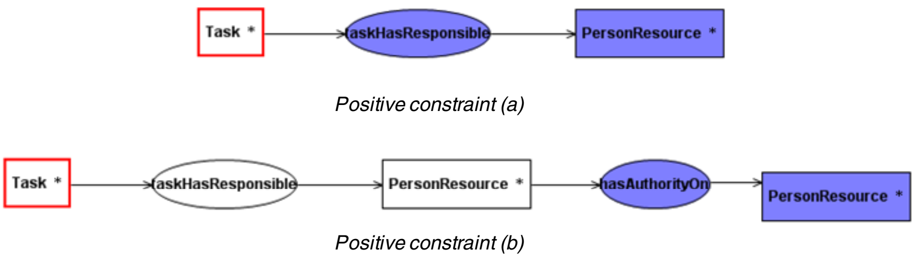

Fig. 10. Compatibility requirements formalized into properties as positive constraints

conceptual graph presented previously in section 4, it will be satisfied because conceptual graphs does not take into account the dynamic aspect of the collaboration. But if this requirement is verified using model checker, it will be not satisfied as demonstrated hereafter.

This requirement is verified on the dynamic model of the collaborative process presented by the task template and the resource template shown Fig. 11.

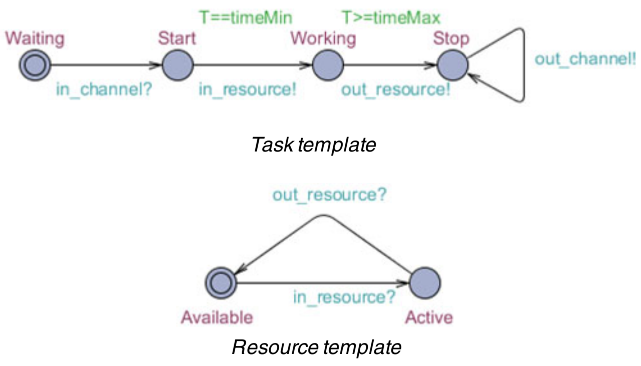

Fig. 11. Task template and resource template

To verify this requirement on UPPAAL, it must be formalized into property using TCTL as:

E<> ResourcePartner2.Active and Task2.Working and Task4.Working

This property indicates that it exists a path where a resource is active when the two tasks are working. The verification of the property will go through all possible paths and answering true or false. In this case, the response of the model checker is false, because the resource cannot be used by the two tasks on the same time. Furthermore, if a time condition is added on the property, it is possible to use the same resource at different time. For instance if the partner 2 is involved in task 2 between 2 and 5 time units and after 6 time units on task 4, the requirement can be satisfied with the verification of two properties. Then, the properties to verify on the dynamic model will be given by:

E<> Resource Partner2.Active and T>2 and T<5 and Task2.Working

for the task 2 and by:

E<> Resource Partner2.Active and T>6 and Task4.Working

for the task 4 where T represents a clock. In this case, the properties are satisfied and the interoperation requirement is satisfied.

As a consequence, it is to note that the consideration of the temporal aspect of collaboration is a primordial aspect since it can changes the result of the verification.

7 Conclusion

In a collaborative context, interoperability takes a preponderant part. During the life cycle of any collaboration between partners, these partners aim to detect and to solve quickly interoperability problems. The proposed approach aims to verify static and dynamic interoperability requirements using different verification tools. In this way, formalization, and verification of interoperability requirements to help enterprises to find their interoperability problems can be a solution to improve collaboration.

This verification is performed using formal verification techniques. This paper focuses on the verification of static and dynamic interoperability requirements. Static interoperability requirements are verified using Conceptual Graphs. To make the verification of dynamic requirements, the verification technique used is model checking. In summary, two formal verification techniques are used. The usefulness of these verification techniques required to make transformation of models and to formalize interoperability requirements into properties using a formal language.

Future works are related first, to the verification of reversibility requirements using formal verification techniques. Second, it wills intent to define the link with a complementary simulation approach based on distributed multi agents systems [25] to improve interoperability problems detection.

References

1. Aubert, B., Dussart, A.: Système d'Information Inter-Organisationnel. Rapport Bourgogne, Groupe CIRANO (March 2002) (in French)
2. ISO/DIS 11345-1: Advanced automation technologies and their applications. Part 1: Framework for enterprise interoperability (2009)
3. Scucanec, S. J., Van Gaasbeek, J. R.: A day in the life of a verification requirement. U.S Air Force T&E Days, Los Angeles, California (February 2008)
4. INTEROP: Enterprise Interoperability-Framework and knowledge corpus - Final report. INTEROP NoE, FP6 – Contract n° 508011, Deliverable DI.3 (May 21, 2007)
5. Tolk, A., Muguira, J.A.: The Levels of Conceptual Interoperability Model. In: Proceedings of Fall Simulation Interoperability Workshop (SIW), Orlando, USA (2003)
6. C4ISR Architecture Working Group: Levels of Information Systems Interoperability (LISI). United States of America Department of Defense, Washington DC, USA (March 30, 1998)
7. Clark, T., Jones, R.: Organisational Interoperability Maturity Model for C2. In: Proc. of Command and Control Research & Techn. Symposium, Newport, USA (1999)

8. ATHENA Integrated Project : Requirement for interoperability framework, product-based and process-based interoperability infrastructures, interoperability life-cycle services, ATHENA deliverable A4.1 (2004)
9. Mallek, S., Daclin, N., Chapurlat, V.: Toward a conceptualisation of interoperability requirements. In: IESA 2010: Interoperability for Enterprise Software & Applications, April 14-15 (2010)
10. ISO 8402: Quality management and quality assurance. Vocabulary, Second edition 1994-04-01, International Standard Organization (1994)
11. Balci, O., Ornwsby, W.: Expanding our horizons in verification, validation and accreditation research and practice. In: Yücesan, E., Chen, C.-H., Snowdon, J.L., Charnes, J.M. (eds.) 2002 Winter Simulation Conference (2002)
12. Edmund, M., Clarke Jr., Grumbereg, O., Doron, A.P.: Model checking. The MIT Press, Cambridge (1999)
13. Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, P., McKenzie, P.: Systems and Software verification: model checking techniques and tools. Springer, Heidelberg (2001)
14. Sowa, J.F.: Conceptual Graphs. IBM Journal of Research and Development (1976)
15. Behrmann, G., David, A., Larsen, K. G.: A tutorial on Uppaal. Department of Computer Science, Aalborg University, Denmark (2004)
16. BPMN: Business Process Modeling Notation, V1.2 (2009), `http://www.bpmn.org/`
17. Roque, M., Chapurlat, V.: Interoperability in collaborative processes: Requirements characterisation and proof approach. In: Camarinha-Matos, L.M., Paraskakis, I., Afsarmanesh, H. (eds.) PRO-VE 2009. IFIP Advances in Information and Communication Technology, vol. 307, pp. 555–562. Springer, Heidelberg (2009)
18. Chein, M., Mugnier, M.-L.: Conceptual graphs: fundamental notions. Revue d'intelligence artificielle 6(4), 365–406 (1992)
19. Cogitant: CoGITaNT Version 5.2.0, Reference Manual (2009), `http://cogitant.sourceforge.net`
20. ATLAS Groupe INA & INRIA Nantes: ATL Atlas Transformation Language. Specification of the ATL Virtual Machine. Version 0.1 (2005)
21. Schnoebelen, P.: The Complexity of Temporal Logic Model Checking. In: Advances in Modal Logic, vol. 4, pp. 1–44 (2002)
22. Gruhn, V., Laue, R.: Using Timed Model Checking for Verifying Workflows. In: Computer Supported Activity Coordination 2005, pp. 75–88 (2005)
23. Alur, R., Courcoubetis, C., Dill, D.: Model-Checking in Dense Real-Time. Information and Computation 104(1), 2–34 (1993)
24. PABADIS'PROMISE STREP FP6: Plant Automation Based on DIStributed Systems (2008), `http://www.pabadis.org/`
25. Rebai, A.S., Chapurlat, V.: System interoperability analysis by mixing system modelling and MAS: an approach. Agent-based, Technologies and applications for enterprise interoperability (ATOP). In: Eighth International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS 2009), Budapest, Hungary (May 2009)