A Formal Framework for Process Interoperability in Dynamic Collaboration Environments

Malik Khalfallah

Numéro d'ordre 272-2014 Année 2014

Université Claude Bernard Lyon 1
Laboratoire d'InfoRmatique en Image et Systèmes d'information
École Doctorale Informatique et Mathématiques de Lyon

Thèse de Doctorat de l'Université de Lyon

Présentée en vue d'obtenir le grade de Docteur,
spécialité Informatique

par

KHALFALLAH Malik

A Formal Framework for Process Interoperability in Dynamic Collaboration Environments

Thèse soutenue publiquement le 03 décembre 2014 devant le jury composé de :

Mme.	Corine Cauvet	Présidente
M.	Khalil Drira	Rapporteur
M.	Mourad Chabane Oussalah	Rapporteur
M.	Ricardo Jardim-Goncalves	Examinateur
Mme.	Genoveva Vargas-Solar	Examinatrice
M.	Mahmoud Barhamgi	Co-encadrant
M.	Nicolas Figay	Co-encadrant
Mme.	Parisa Ghodous	Directrice de Thèse

Laboratoire d'InfoRmatique en Image et Systémes d'information
UMR 5205 CNRS - Lyon 1 - Bât. Nautibus
69622 Villeurbanne cedex - France

Abstract

Designing complex products such as aircrafts, helicopters and launchers must rely on well-founded and standardized processes. These processes should be executed in the context of dynamic cross-organizational collaboration environments.

In this dissertation, we present a formal framework that ensures sustainable interoperability for cross-organizational processes in dynamic environments. We propose a declarative modeling language to define contracts that capture the objectives of each partner in the collaboration. Contract models built using this language under-specify the objectives of the collaboration by limiting the details captured at design-time. This under-specification decreases the coupling between partners in the collaboration. Nevertheless, less coupling leads to the creation of mismatches when partners' processes will exchange messages at run-time. Accordingly, we develop an automatic mediation algorithm that is well adapted for dynamic environments. We conduct a thorough evaluation of this algorithm in the context of dynamic environments and compare it with existing mediation approaches which will prove its efficiency. We then extend our framework with a set of management operations that help realize the modifications on the collaboration environment at run-time. We develop an algorithm that assesses the impact of modifications on the partners in the collaboration environment. Then, this algorithm decides if the modification can be realized or should be postponed to wait for appropriate conditions. In order to figure out how to reach these appropriate conditions, we use the planning graph algorithm. This algorithm determines the raw set of management operations that should be executed in order to realize these conditions. A raw set of management operations cannot be executed by an engine unless its operations are encapsulated in the right workflow patterns. Accordingly, we extend this planning algorithm in order to generate an executable workflow from the raw set of operations. We evaluate our extension against existing approaches regarding the number and the nature of workflow patterns considered when generating the executable workflow. Finally, we believe that monitoring contributes in decreasing the coupling between partners in a collaboration environment. Accordingly, we extend our framework to support monitoring queries. We define a new execution plan for these queries that is more efficient. We conduct a thorough evaluation for this plan and compare it with existing industrial implementation. We implement a prototype for our framework based on standardized software components and illustrate it through a real world use case from the European project IMAGINE[1].

Keywords: Collaboration, Process Interoperability, Mediation, Dynamic Manufacturing Networks, Complex Event Processing.

[1] http://www.imagine-futurefactory.eu

Résumé

Concevoir les produits complexes tels que les avions, les hélicoptères, et les lanceurs requière l'utilisation de processus standardisés ayant des fondements robustes. Ces processus doivent être exécutés dans le contexte d'environnements collaboratifs inter-organisationnels souvent dynamiques.

Dans ce manuscrit, nous présentons un cadre formel qui assure une interopérabilité continue dans le temps pour les processus inter-organisationnels dans les environnements dynamiques. Nous proposons un langage de modélisation déclaratif pour définir des contrats qui capturent les objectifs de chaque partenaire intervenant dans la collaboration. Les modèles de contrats construits avec ce langage sous-spécifient les objectifs de la collaboration en limitant les détails capturés durant la phase de construction du contrat. Cette sous-spécification réduit le couplage entre les partenaires de la collaboration. Néanmoins, moins de couplage implique l'apparition de certaines inadéquations quand les processus des partenaires vont s'échanger des messages lors de la phase d'exécution. Par conséquent, nous développons un algorithme de médiation automatique qui est bien adapté pour les environnements dynamiques. Nous conduisons des évaluations de performance sur cet algorithme qui vont démontrer son efficience par rapports aux approches de médiation existantes. Ensuite, nous étendons notre cadre avec un ensemble d'opérations d'administration qui permettent la réalisation de modifications sur l'environnement collaboratif. Nous développons un algorithme qui évalue l'impact des modifications sur les partenaires. Cet algorithme va ensuite décider si la modification doit être réalisé à l'instant ou bien retardé en attendant que des conditions appropriées sur la configuration de l'environnement dynamique soient satisfaites. Pour savoir comment atteindre ces conditions, nous utilisons l'algorithme de planning à base de graphe. Cet algorithme détermine l'ensemble des opérations qui doivent être exécutées pour atteindre ces conditions. Un ensemble d'opérations ne peut être exécuté par un moteur d'exécution à moins que ces opérations soient encapsulées dans les bons workflow patterns. Par conséquent, nous étendons l'algorithme de planning pour générer un workflow exécutable. Nous évaluons notre extension vis à vis des approches existantes concernant le nombre et la nature des workflow patterns considérés quand nous générons le workflow exécutable. Au final, nous considérons que la technique de suivi de la collaboration peut être un moyen efficace pour réduire encore plus le couplage entre les partenaires. Par conséquent, nous étendons notre cadre pour supporter les requêtes de suivi de collaboration. Nous définissons un nouveau plan d'exécution pour ces requêtes qui est plus efficient. Nous évaluons ce plan et nous le comparons avec les solutions industrielles existantes. Nous réalisons une implémentation d'un prototype pour ce cadre en se basant sur des composant logiciels standardisés et illustrons son utilisation via une étude de cas réelle tirée du projet Européen IMAGINE.

Mots Clés: Collaboration, Interopérabilité des Processus, Médiation, Réseaux de Manufacturing Dynamiques, Gestion des évènements complexes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1.1 Context and Motivation

Today considerable effort is expended in the aerospace development value chain to reduce throughput times in the development processes and to enhance the quality of digital products. Indeed, during the design phase of products many disciplines and partners could be involved. One of the key challenges to be surmounted is to successfully implement concurrent changes which could be requested during the design phase. More specifically, every change of the product properties should be validated by all involved partners working in different disciplines in order to allow the requested change to take place. To address this key challenge, a standardized process was developed that ensures that a change request has followed the whole validation process prior to its implementation. This standardized process is the Engineering Change Management (ECM) [OMG11].

For a product that is designed within a single enterprise, implementing the ECM process within this enterprise is straightforward. The reason is that the person who requests a change and the person who evaluates the impact of this change are supposed to use the same software applications as well as the same work methodologies. Nevertheless, we have observed a shift from mass production systems where a single enterprise performs all design activities to a more open model. This model is the extended enterprise where the Original Equipment Manufacturers (OEMs) keep the core design activities locally while outsourcing the remaining activities to external partners [EME00]. In this case, implementing the ECM process becomes more challenging. For example suppose that for an aircraft design scenario, three engineers belonging to the OEM (e.g. Airbus) simultaneously ask the subcontractor's engineer that is in charge of the design of the *Fuselage* of the aircraft to change three properties of this *Fuselage*. Concretely, these engineers will simultaneously send three messages with the values of the properties that they want to modify as it is prescribed by their internal process. Suppose also that the subcontractor engineer should follow the subcontractor existing process. This process handles all change requests on the *Fuselage* properties in sequence. In this case the processes of both partners do not match and thus they have an interoperability issue as depicted in Figure 1.1[1].

1.1.1 Business Process Modeling

The current practice is to define (from scratch) a cross-organizational process that ensures that the way change requests' messages are sent corresponds to the way that these

[1] *NpaxFront*: Number of pax in the front. *Nailes*: Number of ailes within the fuselage

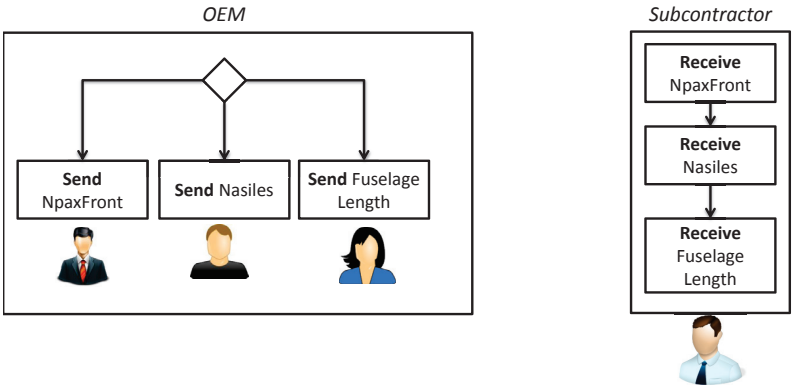


Figure 1.1: Difference between the OEM process and the Subcontractor process

messages are expected. When relying on the existing standards to define such a cross-organizational process, engineers who define change requests on the *Fuselage* would use languages such as BPMN[2] , WS-CDL[3]. Nevertheless, these languages use many low-level concepts (activity, message, choice split etc.) and thus they increase the mental effort of engineers when defining the process [TS09]. Worse, when defining the cross-organizational process using these languages, engineers completely shift their focus. Hence, instead of focusing on how to reach the desired configuration of the *Fuselage*, they will focus on how to exchange messages correctly to achieve that desired configuration of the *Fuselage*.

1.1.2 Mediators

Another possible approach consists in connecting existing partners' processes. In this case, engineers will be asked to define mediators [Wie92] that ensure the reordering and transformation of exchanged messages while allowing each partner to keep his own process unchanged. Such an approach is more practical because it is no longer acceptable that an organization imposes to another organization to modify its internal processes to join the collaboration [VO11]. Nevertheless, engineers will still be asked to acquire new skills in order to implement this approach. These skills include analyzing existing processes and developing mediators.

Although the mediation approach seems to be acceptable to resolve the *interoperability* issue between partners' processes, the proposed approaches in the literature are still too rigid when considering the dynamicity of collaboration environments. Indeed, we have seen recently the emergence of Dynamic Manufacturing Networks (DMNs) [IMA11]. In these environments, unexpected events could occur during the run-time

[2]www.bpmn.org
[3]www.w3.org/TR/ws-cdl-10/

phase of the collaboration. These unexpected events could lead to the replacement of partners or the modification of the messages being exchanged between partners. Accordingly, in DMNs defining mediators will become a repetitive task. Indeed, a mediator developed at instant t will no longer be effective at instant $t + \epsilon$ if the subcontractor has been replaced and the new subcontractor has a different configuration of his process. In this case engineers should develop a new mediator and repeat this task many times during the collaboration life cycle.

Furthermore the decoupling between the partners' processes that mediators are supposed to accomplish remains unsatisfactory especially in DMNs. Indeed, extending our previous example by including the design of the *Nacelle*. Although, change requests on the *Fuselage* properties are independent from change requests on the *Nacelle* properties, the model of the cross-organizational process supporting the execution of these change requests is common to both components as depicted in Figure 1.2. This cross-organizational process model interconnects (i) the model of the sub-process supporting change requests on the *Fuselage* and (ii) the model of the sub-process supporting change requests on the *Nacelle* while allowing a parallel execution of both sub-processes since they are independent. If the sub-contractor responsible of handling change requests on the *Fuselage* has to be replaced, then several actions have to be performed. Among these actions there is the removal of this sub-contractor from the cross-organizational process model and its replacement by a new one. During the time interval between the removal of the sub-contractor from the process model and its replacement, the partners working on the *Nacelle* will no longer be able to collaborate because the whole cross-organizational process model is impacted.

The removal of the sub-contractor from the cross-organizational process introduces syntactic errors in the model of this cross-organizational process and thus this model can no longer be executable by an engine until the new sub-contractor is added.

From this example we can observe that even though handling change requests for the *Fuselage* is (from an engineering point of view) independent from handling change requests on the *Nacelle*, a coupling still exists between their sub-processes. Accordingly, mediators as developed currently could not achieve a satisfactory decoupling level.

1.1.3 Monitoring

Another important aspect of the ECM process in a cross-organizational environment is monitoring. Indeed, when conducting parallel change requests as presented in the previous example, a chief engineer belonging to the OEM might need to monitor the

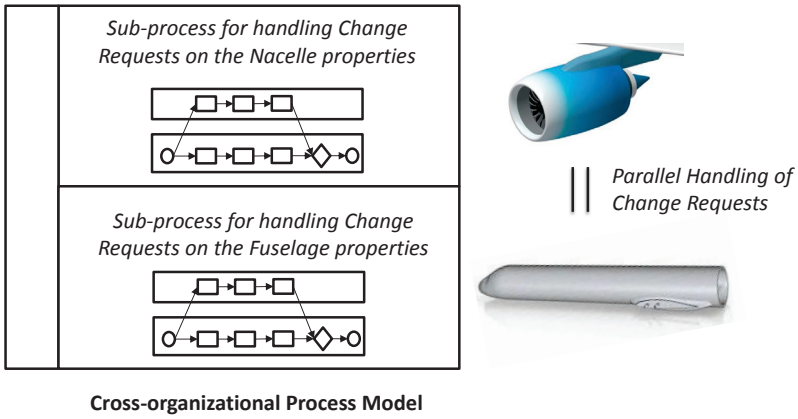


Figure 1.2: Cross-organizational process to handle change requests

unfolding of parallel changes. His aim could be to ensure the coherency of the properties of the final product that is the aircraft. To realize this monitoring, the chief engineer needs to obtain copies of the exchanged messages in the cross-organizational process and then to perform further processing on these messages. To achieve this processing he may call a service or a composition of services in order to obtain higher level information that he can use to take decisions.

1.1.4 Airbus Group Innovations Strategy

To address the mentioned issues, Airbus Group Innovations has been involved in several European projects where the aim was to develop a collaborative platform that supports the execution of the cross-organizational process of the Engineering Change Management. An industrial implementation of such a platform has already been realized but the capabilities of this platform are being enhanced continuously. However, enhancing such a platform is particularly difficult due to the following reasons:

First, existing cross-organizational process modeling standards do not provide the right constructs at the right level of abstraction in order to facilitate the work of business actors who need to build the process model. Thus the difficulty consists in providing these actors with a language that has two properties: (i) it has the right constructs at the right level of abstraction so that business actors do not need to acquire new skills to use it. (ii) The modeling language should allow the business actors to focus on specifying the objectives of their collaboration (i.e. to change the properties of product component) and not on specifying how to achieve these objectives.

Second, business actors should not be considered as the platform administrators. Indeed, using a mediator deployed within the platform is necessary to ensure interoperability [BCG+05]. Nevertheless, business actors should be shielded from defining and

managing mediators in the platform.

Third, in the context of DMNs, changing the configuration of the collaboration environment should be allowed and facilitated. However, due to the coupling that could exist between partners' processes, implementing this capability is especially difficult. The reason of this difficulty is how to make these changes transparent to other partners.

Fourth, there is a plenty of software applications that can be used to provide the chief engineer with a monitoring capability so that he can watch the unfolding of the collaboration between partners. However these software applications and their underlying monitoring mechanism are not always efficient and in some cases they fail to provide the right information at the right time. Thus, there is a need for a monitoring approach that efficiently informs the chief engineer.

In the next section, we detail these challenges and then we survey the thesis contributions that address them.

1.2 Scientific Problems and Contributions

1.2.1 Challenges

The issues encountered by engineers involved in the ECM process as presented in the previous section are instances of more general problems in the domains of Business Process Management (BPM) and Service Oriented Computing (SOC).

Modeling Language : providing a modeling language that uses high-level concepts understandable by business actors is a problem that has been addressed by several works. Indeed, numerous frameworks have been developed to allow business actors to focus on the business value of a process rather than on the steps and activities of that process. The Commitment Framework [TS09] and the language Let's Dance [ZBDtH06] are examples of such frameworks.

Specifying the objective of the collaboration rather than how this collaboration has to be conducted is a typical problem of using declarative languages rather than imperative languages when modeling the collaboration. On one hand, imperative languages specify the steps to be followed in order to reach an objective but they make abstraction on the specification of that objective. On the other hand declarative languages specify the objective to be reached and make abstraction on the steps to reach this objective. Each paradigm has its advantages and drawbacks depending on the addressed problem. Declarative languages are more appropriate when more flexibility in reaching the objective is required. This flexibility facilitates the modifications on the process to reach

that objective. Imperative languages are more rigid, inappropriate when changes are frequent but are easier to automate.

Mediation : developing mediators to ensure interoperability between two communicating processes is a problem addressed by several works since decades [Wie92, YS97]. Nevertheless, assuming that the involved processes can evolve at run-time which is the case in DMNs, implies that mediators should evolve at run-time as well. Thus, a potential solution to mediators' evolution in DMNs would be to create a unique mediation approach that could be used for the different cases of mismatches. In this case, the problem is how to develop the algorithm of this approach that runs efficiently?

Management System : managing the modifications that can occur on a process model at run-time is a problem of adaptability of business processes. Many standardized process modeling languages are of imperative nature and thus they are still considered to be rigid [vdAP06]. However, efforts have been conducted in order to incorporate flexibility in processes modeled using these languages for example using the change patterns [WRR13]. Although, existing works address the adaptability problem for a process involving one partner, adaptability of cross-organizational processes involving several partners has received less attention. Indeed, modifying a cross-organizational process generates new issues that have not been considered for a single partner process. One of these issues is: when a modification concerns a single partner process, how can this modification be realized without impacting other partners not concerned by it but participate to the same cross-organizational process?

Changing the configuration of a collaboration environment by replacing partners in the cross-organizational process requires removing these partners and then adding the new ones. When many removals are performed it would be better to generate a set of actions that will semi-automatically guide the process designer in adding the missing partners. To achieve this, planning algorithms can be used. Indeed, a planning algorithm will analyze the current state of the process model and finds out that there are partners who are missing. Then it generates a solution plan that defines the actions to be executed in order to add the missing partners. Nevertheless, existing planning algorithms generate a raw set of actions. If one wants to semi-automate the execution of this set of actions, it will be challenging because the sequencing of these actions should be determined before an engine can execute them. Accordingly a problem that is still pending is how to generate an executable workflow of actions from a solution plan?

Monitoring System : monitoring the exchanged messages in a cross-organizational process provides valuable information for the monitor. To realize this monitoring, it is possible to use a Complex Event Processing approach. Indeed, the exchanged messages

can be seen as events and a CEP software application can be used to capture these events, process them and provide the final result to the monitor. The processing of the received events involves that the CEP invokes a composition of services that will generate the required information. Existing CEP frameworks provide the possibility to express processing on the received events but the problem is whether the required execution time and space for this processing is optimal or not. If not how they can be optimized?

Implementation : realizing a software platform that answers the mentioned problems should use standardized implementation of the underlying software modules as much as possible. This makes our platform vendor-independent. Nevertheless, ensuring that a software module is compliant with a standard specification is currently done either manually or by relying on the software documentation. These verification approaches are not well-founded since the assessment results are only qualitative and not quantitative. Accordingly, a problem that needs to be addressed in order to use only standardized software modules is how to compute quantitative evaluation criteria and report them so that the most conformant software module to a standard can be selected?

1.2.2 Contributions

To address the identified scientific challenges we have made the following key contributions:

High-level Modeling and Automatic Mediation : we define a formal model of the framework to specify the objective of the collaboration between two partners regarding the design of a product component. This specification allows involved partners to focus exclusively on the objective of their collaboration without specifying how it will be achieved. Of course this under-specification will leave room for heterogeneous interpretations on how to realize the objective. To overcome this heterogeneity we propose an efficient mediation solution that uses a single mediation algorithm applicable for the possible situations. Thus it prevents partners from replacing it when changes occur on the configuration of the collaboration [KFBG13b, KFB$^+$13, KBFG14].

Example 1. *Following the example of Figure 1.2, the proposed modeling approach will allow the engineers to specify that the collaboration will be on the Fuselage involving the properties :* $\{Npaxfront, Naisles, Fuselage\_Length\}$ *without requiring additional details. The cross-organizational process supporting the delivery of the exchanged messages will be generated automatically while upholding each partner internal rules in organizing message exchange activities. In addition, the mediation on the fly will efficiently deliver the right message at the right moment*

without asking the involved engineers to intervene.

Management System : we enrich our framework with a set of management operations that help partners realize modifications on their interactions at run-time. In order to shield partners that are not concerned by the modifications, we developed an algorithm that checks whether a modification will not impact other partners. If this is the case it realizes the modification otherwise it postpones it until other modifications are requested. Hopefully all these modifications assembled together can be realized and allow the carrying of the collaboration between the involved partners [KFBG13a].

Example 2. *In the context of dynamic collaborative environment that we will detail in the next section, when the OEM engineer in Figure 1.2 wants to replace the subcontractor designing the Fuselage, our management system will ensure that other parallel design processes belonging to the same cross-organizational process (for example the one related to the design of the Nacelle) will not be impacted since the design of the Nacelle is independent from the design of the Fuselage.*

In order to fasten the resuming of the collaboration between partners impacted by multiple change operations, we use AI planning to generate a plan (a succession of sets of actions) that should be executed in order to resume the collaboration. We extend the planning-graph algorithm [NGT04] in order to semi-automate the execution of the generated plan. This extension generates an executable workflow that guides partners through the actions that should be executed in order to realize the recovery and resume the collaboration.

Example 3. *Following the scenario of replacing the subcontractor in Figure 1.2. If we assume that the OEM engineer decided to replace the subcontractors involved in the design of many other aircraft components (Engine, Nacelle, etc.). When trying to replace all these partners, it is better to guide him by using a semi-automated workflow. As we will see, the set of actions of this workflow can be generated by the planning-graph algorithm, but this* ***unordered*** *set of actions cannot be executed by an engine. Our extension of the planning-graph algorithm will add enough details to this* ***unordered*** *set so that it can be executed by an engine.*

Efficient Monitoring : as monitoring can be realized by using monitoring queries in CEP software applications, we define a new execution plan for monitoring queries. This execution plan is more efficient in comparison to an existing industrial implementation since it consumes less time and less memory [KFBG14b].

Example 4. *If the OEM engineer wants to study the interaction between the Engine and the Nacelle during their design, he asks to obtain copies of the exchanged messages in their corresponding design processes. He may also want to perform some computations on the received*

copies to generate the desired information. Our plan, that runs this computation, will reduce the time and the memory space required to deliver the desired information in comparison to existing CEP software.

Conformance to Standards Implementation : to implement a software platform that provides the desired features and that has software modules compliant with their respective standards, we developed a formal approach. This approach assesses the conformance of a software module with a standard specification. It uses vector calculus to generate quantitative measures that help us decide what module to choose among several ones claiming to be standard compliant [KFBG14a].

Example 5. *In our prototype, we will use a workflow engine that will execute the cross-organizational process. This engine cannot be selected randomly. We need to check its conformance to the workflow community standard. Our approach provides us quantitative metrics that allow us to choose the most conformant software to that standard.*

1.3 Dissertation Organization

The reminder of this dissertation is organized as follows:

In **Chapter** 2 we detail the engineering design process for products and show how this process is implemented in the industry through the ECM process. We give a real application case of the ECM process and then we derive the required features of the platform that supports the execution of this process in cross-organizational context. In addition, we review the closest works inside the Airbus Group that cover the identified features and show their limits. From these limits we review the challenges and the key contributions of the thesis.

In **Chapter** 3 we review the related work regarding high-level frameworks to model cross-organizational processes and identify their limits, we also review change management frameworks for business processes and identify their limits as well. We explore the work on complex event processing and how existing frameworks execute the processing of monitoring queries. This chapter also introduces the main concepts of the theoretical frameworks used in subsequent chapters.

In **Chapter** 4 we propose a conceptual framework to model the objectives of the collaboration between two partners; we extend this framework to model the objectives of the collaboration in the whole collaborative environment and their temporal relationships as well. From this specification, we elaborate on the approach to generate the cross-organizational process. Key inputs for this generation are the business rules

of each partner that define how the partner process activities are organized. Then we elaborate on the mediation solution and show how it resolves heterogeneity between processes in DMNs.

In **Chapter** 5 we extend our conceptual framework with a set of management operations that can be used to change partners, objectives, and the relationships between objectives. We define the precondition and post-condition of these operations and also the underlying mechanism to support the concurrent call of a management operation. After that, we show how these operations can be used by a planning algorithm to generate a solution plan that aims at recovering several removals and then we extend this solution plan to make it executable by an engine.

In **Chapter** 6 we extend the conceptual framework so that it supports defining monitoring queries on the cross-organizational process. We identify a set of monitoring patterns and show how they can be used to generate the query execution plan that performs better in comparison to those of existing works.

In **Chapter** 7 we present the implementation of our collaboration platform. We present the implementation techniques that we have used to realize the different functionalities. In addition we elaborate on the methodology used to test the conformance of software modules with standards specifications and show how it helped us select the most compliant workflow engine software.

In **Chapter** 8 we summarize our contributions and discuss future directions of our research.

Chapter 2

Thesis Context and Problematic

Contents

2.1 Introduction

This thesis takes place in an industrial context at the AIRBUS Group Innovations (AGI) and more specifically in the context of the European project IMAGINE[1]. AIRBUS Group is a leading aircraft designer and manufacturer with products ranging from airplanes and helicopters to space launchers and satellite systems. To design and manufacture these products, several frameworks were developed to support the products' development. In this chapter we present the engineering design process in a collaboration environment and its realization through the ECM process. Then we derive a set of challenges that are still not addressed neither by frameworks available inside AGI nor by frameworks from the research community. First we start by presenting the engineering design process from a conceptual point of view. We detail the main activities of this process in order to achieve a desired design of the final product. Second we elaborate on the industrial and standardized implementation of the design process in a collaboration environment and we analyze the main activities of this standardized process. We present a practical scenario that illustrates this process and from that scenario we identify the need for a collaboration platform that supports the different parties in achieving the design. We sketch the features that this platform should provide and then we assess the available frameworks inside AGI regarding the coverage of these features. From this assessment, we derive a set of scientific challenges that still need to be addressed and we conclude with the thesis contributions that address the identified challenges.

2.2 Engineering Design

An engineering project often starts by capturing the customer requirements. From these requirements, the project planning takes place and its outcome consists in two structures: the system breakdown structure and the work breakdown structure. Using these two structures, the design process is performed in order to design a system that responds to the customer requirements.

2.2.1 Breakdown structures

During the design process, different groups of engineers work on different systems. Thus, in order to allocate the appropriate team to design the appropriate subsystem, the system breakdown structure as well as the work breakdown structure needs to be defined at the early phases of the project. Figure 2.1 gives a generic example of a system

[1] http://www.imagine-futurefactory.eu/

breakdown structure and Figure 2.2 gives an example of a work breakdown structure for an aircraft.

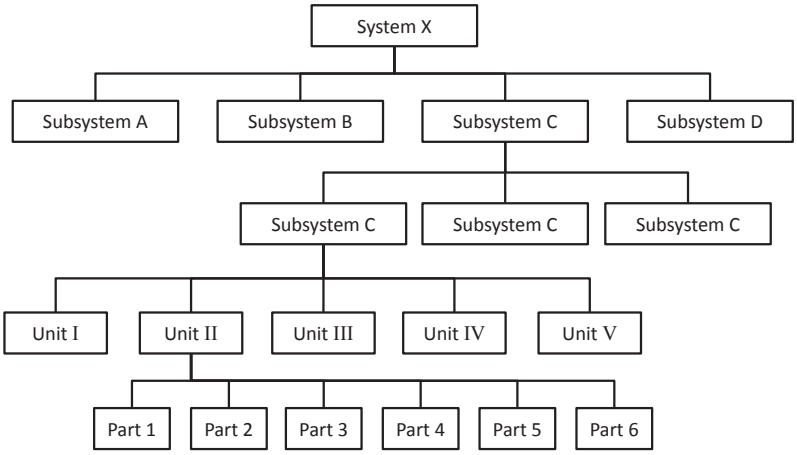


Figure 2.1: System breakdown structure [Sad12]

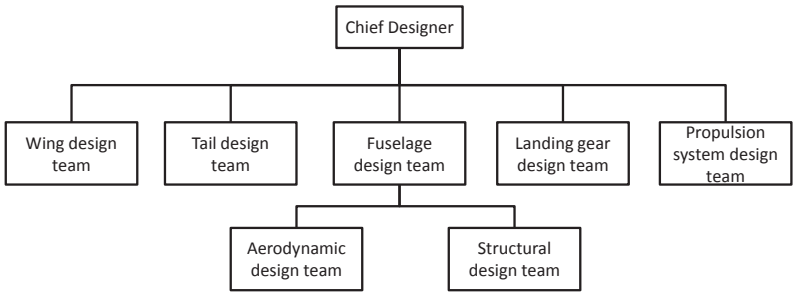


Figure 2.2: Work breakdown structure

When both breakdown structures are ready, three major activities are executed in order to design the system that meets customer requirements [Ray06] as depicted in Figure 2.3.

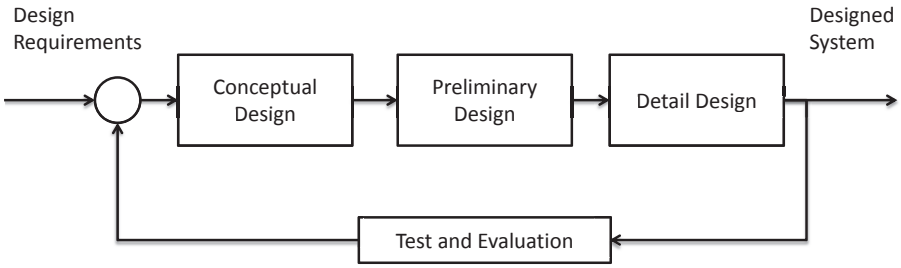


Figure 2.3: Relationship among four major design activities [Sad12]

2.2.2 Conceptual System Design

Conceptual Design usually begins with either a specific set of design requirements established by the prospective customer or a company-generated guess as to what future customers may need. The aim of this activity is to define the optimum configuration of the whole system without going into detail on its subsystems. Thus, in this activity, the main question to be answered is "Can an affordable system be built to meet the customer requirements?" [Ray06].

2.2.3 Preliminary System Design

In the preliminary design, the aim is to determine features of the basic components. More specifically, it consists in preparing the design requirements for the detail design at subsystems level as depicted in Figure 2.1. This phase also aims to prepare the different software packages, design tools, and technologies that will be used during the detail design of subsystems [Sad12]. The ultimate objective of the preliminary design is to pave the road for the detail design phase.

These two first phases are characterized by the involvement of a few highly qualified individuals with broad technical knowledge. As noticed by [Sad12] for aircraft design, most of engineers who go to work for an aerospace company will work in detail design. Thus as the design progresses, the number of engineers involved increases and the need for an organized collaboration is more required.

2.2.4 Detail System Design

The detail design is the most extensive phase in the whole design process [Sad12]. Its purpose is to produce the data necessary for the manufacture of hardware. In the case of a sophisticated system (e.g. an aircraft) many tens of thousands of Computer Aided Design (CAD) files are needed to define the aircraft adequately [Ray06].

Prior to starting the detail design process, chief designers (depicted in Figure 2) need to plan the goals of the detail design from the design requirements. Then at run-time, they need to manage the design process by introducing some changes. Finally they need to monitor the detail design process [Sad12].

2.2.4.1 Converting design requirements into problem statements

In order to fulfill the design requirements determined in earlier phases for a particular component, the chief designer needs to translate the design requirements for that

component into problem statements and then submit these problem statements to the component design team. Problem statements tell the design team what constraints they should take into account while designing the component and what results are expected by the chief engineer to evaluate their design.

[Sad12] formulated a problem statement as a triple: $\langle Goal, Objectives, Constraints \rangle$.

- **Goal** is a brief, general and ideal response to the need statement. The need describes the current, unsatisfactory situation, while the goal describes the ideal future condition to which the chief engineer aspires in order to improve on the situation described by the need.

- **Objectives** are quantifiable expectations of performance which identify those performance characteristics of a design that are of most interest to the chief engineer of the subsystem. In addition, the objectives must include a description of conditions under which a design must perform. In other terms objectives aim to specify the *whats* and not the *hows*.

- **Constraints** are restrictions on performance measures. They limit the freedom of design. Constraints define the permissible conditions of design features and the permissible range of the design and performance parameters.

Problem statements support the chief engineer in specifying what needs to be achieved by collaborative design with the subsystem design team. It lets the chief engineer focus on the final result of the design by specifying the *what* while making abstraction on *how* the design process will be conducted.

2.2.4.2 Temporal constraints between subsystems designs

Specifying problem statements for subsystems design is necessary but not sufficient. The chief engineer should uphold a certain order when defining problem statements for sub-systems because temporal design constraints could exist between the different subsystems. Indeed, for a wide range of systems the associated subsystems and components can be classified into two groups: (i) primary or dominant subsystems, and (ii) secondary or servant subsystems.

Example 6. *In an aircraft, the wing, fuselage, tail and engine are assumed to be dominant components, but the electric system, air conditioning system, cabin, cockpit and landing gear are categorized as servant components.*

This categorization of subsystems has a direct impact on the design process. Indeed, the dominant systems need to be designed first and their characteristics need to be determined, then the design or the choice of the servant systems can be validated.

Accordingly, besides specifying problem statements for the different subsystems, the chief engineer needs to be able to specify temporal constraints between problem statements of subsystems in order to uphold the categorization that separates between dominant and servant subsystems.

Example 7. *In an aircraft design project, the aircraft aerodynamic design leads the aircraft structural design, since the structure is a servant subsystem.*

2.2.4.3 Iterative detail design

Once the detail design phase is achieved for a particular component, an additional step is required that is the test and evaluation step as depicted in Figure 2.3. The outcome of this step is an answer to the question whether the design is satisfactory for the chief designer and an agreement has been reached with the team responsible for this component. In this case the process can carry on for the manufacturing or for the detail design of the servant components of this particular component. When an agreement has not been reached, a new iteration of the detail design process for this component is required.

2.2.5 Changing the Specification of Problem Statements

During the detail design of a component, modifications can be initiated on the problem statement specification of this particular component for example by adding new objectives/constraints or removing some objectives/constraints for different reasons, e.g., to correct a design deficiency, improve the component, incorporate a new technology, respond to a change in operational requirements, compensate for an obsolete section etc. Modifications may be initiated from within the design process, or as a result of some new externally imposed requirements [Sad12].

2.2.6 Monitoring the Unfolding of other Problem Statements

An important task of the chief designer is to monitor the unfolding of the design of the components. This monitoring could be performed on each component separately, but it can also be performed collectively by monitoring the unfolding of the design of a subset of interacting component together.

2.2.7 Summary

In this section, we gave a high-level and conceptual view on how the design process is conducted. We saw that a chief designer submits problem statements specifications to design teams and expects to receive characteristics of the components in order to analyze whether it is possible to carry on to the manufacturing phase or to redo another iteration of the design. Nevertheless in an industrial project it is recommended to rely on a standardized process when designing a system and its components. This process is the *Engineering Change Management*.

2.3 Engineering Change Management

In the previous section we described the conceptual framework of engineering design based on sharing problem statements between the chief designer and the teams responsible for designing the components. Nevertheless, with the extensive use of software applications during the engineering design, Product Lifecycle Management (PLM) has emerged as a strategic business approach. This approach consistently manages all lifecycle stages of a product including the design stage [TLM06]. Thus, the chief designer and the different teams should follow the PLM strategy when carrying the design process.

When sharing problem statements, the chief designer and the design teams should use their Product Data Management Systems (PDM) that implement the PLM strategy[MR09]. More specifically, PDMs implement a standardized process to support the engineering design of a product that is the Engineering Change Management process (ECM).

The ECM is executed when a partner (e.g. chief designer) identifies the potential need for an engineering change on the configuration of a product component [OMG11]. Hence requesting an engineering change is equivalent to submitting a problem statement. The ECM standardized process can be internal when the component to be modified is designed internally within the enterprise or external when the change concerns a component designed by an external organization that is known as a Configuration Item (CI).

There are several reasons that can lead to changing a product and its components properties. For example a change can be required due to a possible optimization detected during the lifecycle of the product. When realizing the change following the ECM process, this change will be evaluated, implemented and released in the detail design process. It is also possible that the idea of a change will be rejected because it

appears unpromising [SAS10].

Figure 2.4 depicts a phase diagram of the ECM process. It starts by the need to *perform change on a certain component*.

The second phase *Development of Alternative Solutions* consists in determining the technical possibilities that can meet the change need. For example, if a change request is on the engine of an aircraft, there could be three different alternative solutions, an engine from *Snecma*, an engine from *Rolls Roys*, or an engine from *General Electric*, since these are the main engine providers for *Airbus*.

Once the potential technical solution is identified, it will be analyzed with respect to technical considerations in the phase *Specification and Decision on Change*. This phase is the core of the ECM process. During this phase, the chief designer exchanges messages that correspond to the problem statement specification with the component design team until reaching an agreement on the change to implement on the product component. The Engineering Change Request (ECR) object is generated at the end of this phase.

If the ECR is approved, the change is implemented in the fourth phase *Engineering Implementation of Change*, for example by making changes to relevant documents such as CAD models, drawings, Bill of Materials or product structures. The ECR object forms the basis of these activities and provides the necessary authorization.

Finally, when a change is implemented by engineering and released, the phase *Manufacturing Implementation of Change* begins. It consists in producing the changed component with the new configuration.

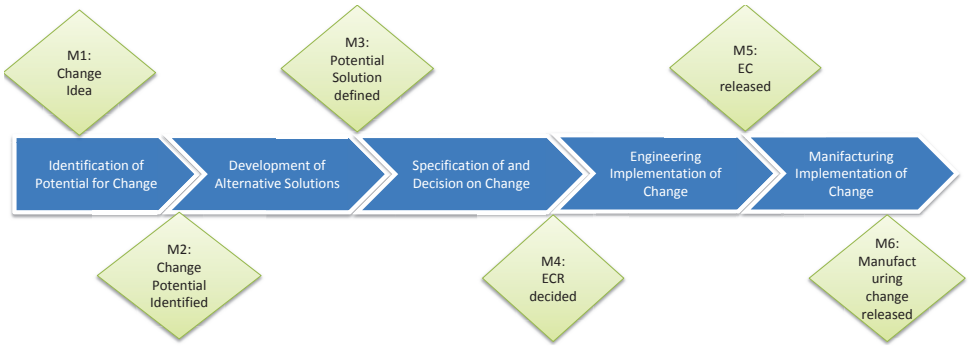


Figure 2.4: ECM Process overview [SAS10]

In the subsequent sections, we detail the second phase that is the *Specification and Decision on Change* since it is during this phase where interesting collaboration between partners takes place.

2.4 Specification and Decision on Change

2.4.1 Basic Concepts

2.4.1.1 Engineering Change Request (ECR)

Initiating an ECR consists in requesting an evaluation on a change on a particular component of the product [SAS10]. It includes changing the component dimensions or other properties [OMG11].

Elaborating the ECR involves two standardized types of partners: *coordinators* and *participants*. *Coordinators* are those that issue the ECRs, they generally represent the chief designer and *participants* are those that process the ECRs and they represent the teams responsible for different components [SAS10]. *Partners* refer to both, *coordinators* and *participants*.

2.4.1.2 Interaction

To realize an ECR there is a need of message exchange between involved partners. An interaction is generally bidirectional where a message of type request and a message of type response are exchanged. A unidirectional interaction comprises a send action from the sender and a receive action from the recipient. A bidirectional interaction is comprised of send and receive actions with sender and receiver swapped [SAS10].

2.4.1.3 Interaction Scenario

It is a description of the sequences used and the conditions under which data is exchanged between partners to address an ECR [SAS10]. Each partner in an interaction scenario has a process that supports him in processing the ECR. This process is decomposed into [SAS10]:

- **private process** that describes the partner-specific process used internally
- **public process (protocol)** that defines the permitted message sequence per interaction scenario from a *coordinator*'s and from a *participant*'s point of view.

2.4.2 Engineering Change Request Generation

When a change has been identified, the *coordinator* creates an ECR on the associated component (ECR created in Figure 2.5). Then the *coordinator* in association with the *participant* define the messages to be exchanged and their sequence in order to detail

(iteratively) the ECR created. Since *coordinators* and *participants* could already have existing processes to address ECRs on specific components, they need to map their existing processes with the messages that have been defined in order to create the interaction scenario associated to this particular ECR as depicted in Figure 2.5.

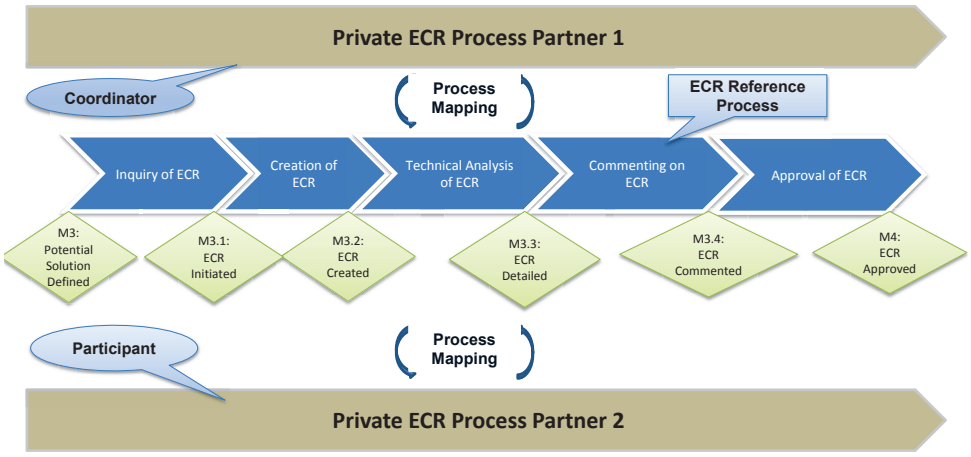


Figure 2.5: Mapping between the ECR process and private processes [SAS10]

Once the *interaction scenario* is created, the collaboration between the ECR *coordinator* and the ECR *participant* begins. In this case, the ECR *coordinator* instantiate a problem statement and submit it to the ECR participant following the *interaction scenario* specification. The ECR *participant* executes his private process in order to analyze the problem statement and submits the response to the *coordinator*. This process iterates until the *coordinator* takes a decision and notifies the *participant* to stop, as depicted in Figure 2.6.

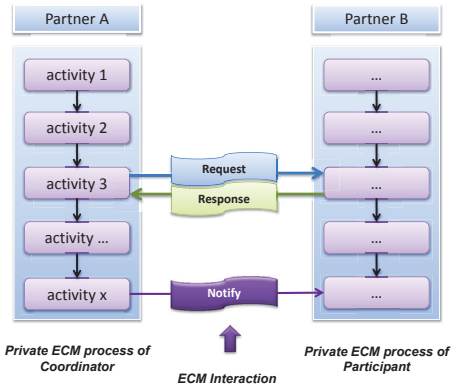


Figure 2.6: Integration between the processes of two partners [SAS10]

2.4.3 Cascading Processing of Change Requests

The collaborative environment is organized into layers. Partners located in layer 1 are those that handle the change requests expressed by the OEM on a particular compo-

nent. Partners located at layer 2 are those that handle the change requests on the sub-components of the previous component and so on. Figure 2.7 depicts the layered structure of the collaborative environment.

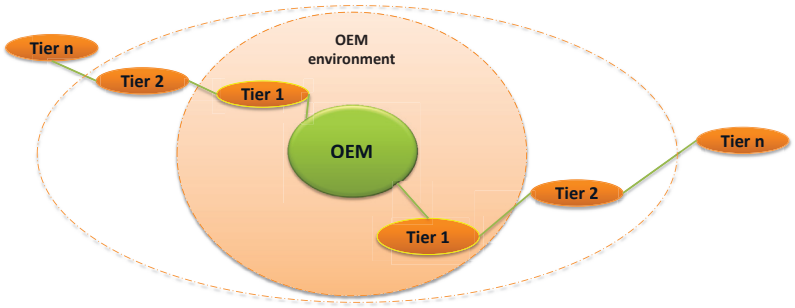


Figure 2.7: Layered structure of the collaborative environment

Each partner within a collaborative environment may have the role of *coordinator* or *participant* depending on the specific cooperation [SAS10]. Indeed, during the processing of an ECR on a particular component A requested by a *coordinator* α, the partner β playing the role of *participant* may also need to create ECRs on sub-components of A. These ECRs aim at analyzing the impact of the change desired by α on the sub-components of A. In this case, the partner β will play the role of *coordinator* in the new interaction scenario and the same standardized ECM process will be followed.

Nevertheless, there could be some synchronization constraints between the different ECM processes related to the component and its sub-components. For example a **lead to** constraint that indicates that a change on a component cannot be validated until the change on its sub-component is validated. Hence, it is necessary to be able to specify this kind of constraints between the change requests on components and their sub-components in order to maintain the constraints between changes explicit. Figure 2.8 shows an example of a cascading processing of an ECR.

2.4.4 Summary

In this section we detailed the specification of an ECR by focusing on interaction scenarios and how they are mapped to private processes of partners. We also discussed the cascading nature of the processing of ECRs and that an explicit temporal link between ECRs should be expressed.

In the next section we provide an industrial scenario that shows how the ECR process is used and then we motivate the need for a PLM Hub to enhance the collaboration.

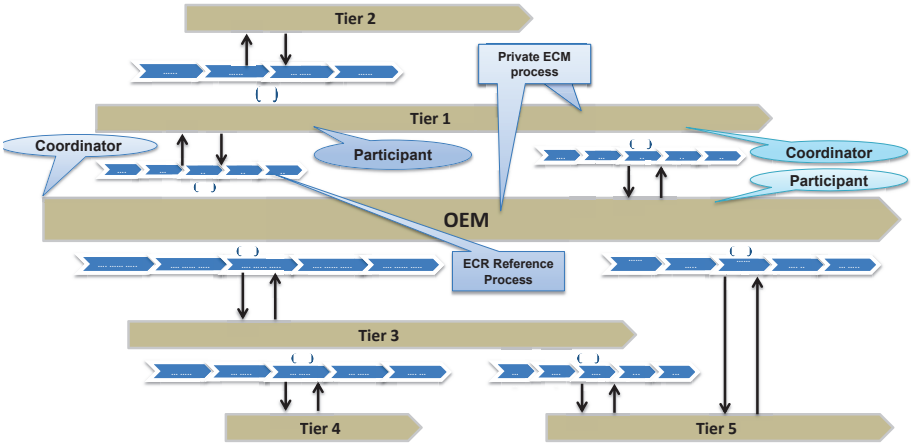


Figure 2.8: Cascading ECR processing [SAS10]

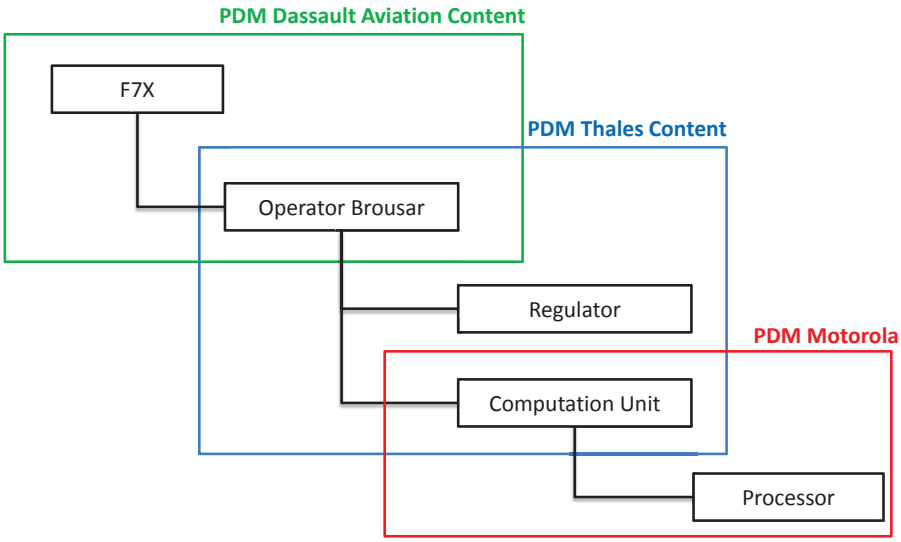


Figure 2.9: F7X breakdown excerpt

2.5 Practical Scenario: Operator Broussard Design

The *Broussard operator* is a telecommunication system designed by Thales (TH) and used by Dassault Aviation (DA) for the *F7X* whose breakdown is depicted in Figure 2.9. This equipment constitutes a component in the whole breakdown of the *F7X* aircraft. In order to achieve a desired configuration of this component so that it becomes well adapted for the *F7X*, the *coordinator* (pertaining to DA) creates an ECR on the *Broussard Operator* and then in collaboration with the *participant* (pertaining to TH) iteratively detail this ECR until both partners reach an agreement. Figure 2.10. depicts how currently the collaboration between these partners is performed.

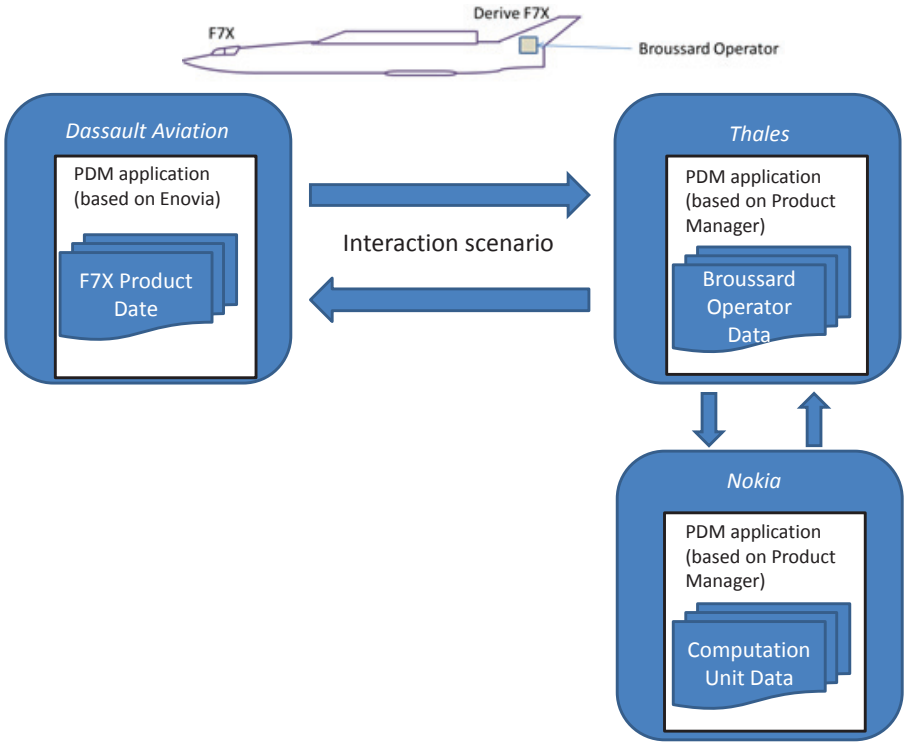


Figure 2.10: Current industrial collaboration

2.5.1 The Storyboard of the Collaboration between the Involved Actors

- The *coordinator* in (DA) creates an ECR on the *Broussard operator* using his internal PDM system *Enovia*[2].

- The *coordinator* defines an interaction scenario (messages and their sequence) with the *participant* from TH in order to prepare the ECR. In this case, the *coordinator* would oblige the *participant* to adapt his existing process that handles ECRs on the *Broussard Operator* in order to iteratively detail the ECR. This is necessary because the *coordinator* and the *participant* processes should be fully compatible to detail the ECR.

- The *participant* in TH receives a problem statement and notices that if this change is implemented, it will have an impact on the configuration on the computation unit of the *Broussard Operator*

- The *participant* in TH creates a new ECR on the computation unit for which he is the *coordinator* and starts collaboration with the *participant* from Motorola. Nevertheless, he has no formal way to specify that the ECR on the *Broussard Operator*

[2]http://www.3ds.com/products-services/enovia/

cannot be validated until the ECR on the computation unit is validated. He is the one who has this information and it remains implicit

- Once the *participant* in Motorola has validated the ECR coming from TH, the *participant* in TH can validate the ECR of the *coordinator* in DA.

- Finally, both ECRs are released to be implemented during the manufacturing phase

2.5.2 Conclusion: The Need for a PLM Hub

After this analysis of the industrial implementation of the engineering design process through the standardized ECM process, we conclude that there is a need for a collaborative PLM platform (PLM Hub) shared among all partners located at all layers in the collaborative environment. The purpose of a PLM Hub is to support the execution of ECM processes involving multiple couples of partners as depicted in Figure 2.11. The following points summarize the reasons of behind this need:

- **Specification of interaction scenarios:** we have seen that the *coordinator* needs to collaboratively define the messages that should be exchanged with the *participant* [SAS10] in order to detail an ECR. Thus they need a shared software environment that allows them to specify the interaction scenario.

- **Interoperability of processes:** we have seen that the *coordinator* as well as the *participant* will map their existing private processes with the specification of the interaction scenario (Figure 2.5). Nevertheless, it is no longer acceptable that a partner imposes to another partner the way he should define and run his private process [MR09]. Accordingly, the interaction scenario specification should be abstract enough so that each partner keeps his private process unchanged and this possibly creates mismatches between the *coordinator* process and the *participant* process that must be resolved so that the collaboration could take place. A PLM Hub will help in resolving these mismatches

- **Cascading ECM:** we have seen that some constraints could exist between an ECM process on a component and ECM processes on its sub-components or other components as well. Thus, there is a need to maintain an explicit relationship between these ECMs in order to ensure that no ECR will be validated without the realization of all its preceding ECRs. A PLM Hub will help in defining these constraints and watching that they are upheld.

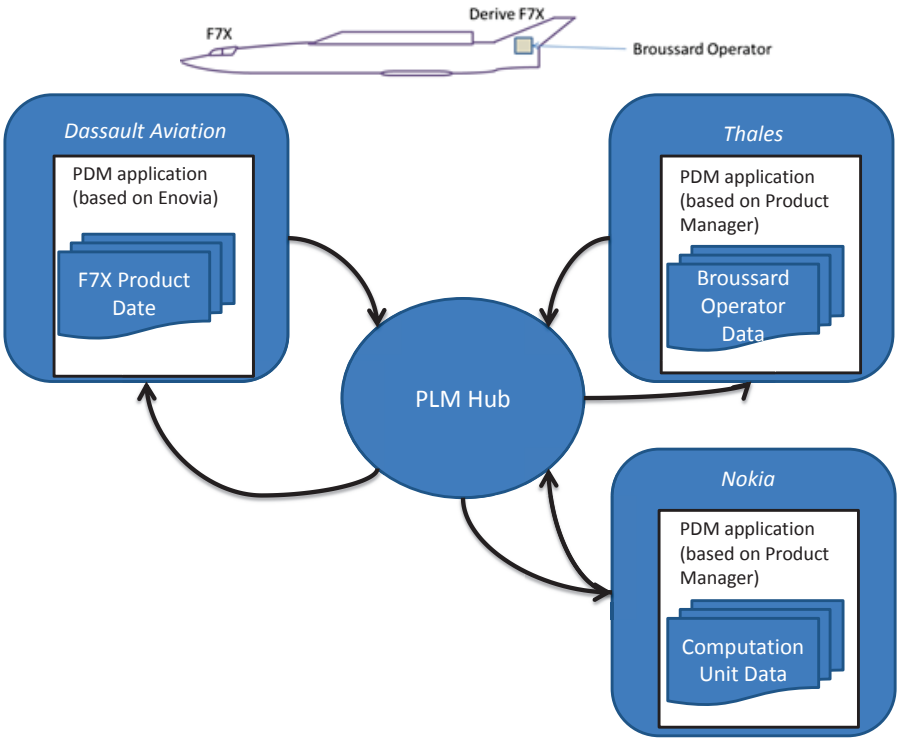


Figure 2.11: PLM Hub

2.6 Background for a PLM Hub

In this section, we define the main technical concepts that will be used in this thesis.

2.6.1 Business Processes and Workflows

A *business process* consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal. Each business process is enacted by a single organization, but it may interact with business processes performed by other organizations [Wes10].

A *cross-organizational process* is a business process involving more than one partner. A cross-organizational process can be divided into two categories of processes. The public processes that contain only the data send/receive activities and that allow the communication between the participants. The private processes that cover the activities that will achieve the goal and each private process is specific to one partner.

A *workflow* is the automation of a cross-organizational process, in whole or part, during which documents, information or tasks are passed from one participant to another for action action, according to a set of procedural rules [DvdAtH05].

2.6.2 Interoperability and Interoperability Levels

Interoperability is the ability of a collection of communicating entities to (a) share specified information and (b) operate on that information according to an agreed operational semantics [LMSW08].

Sustainable interoperability focuses on novel strategies, methods and tools to maintain and sustain the interoperability of enterprise systems in networked environments as they inevitably evolve with their environments [JGPG12].

Interoperability can be classified into four levels [CD10]:

1. *Data interoperability* refers to operate together different data models.

2. *Service interoperability* is concerned with identifying, composing and operating together independent software.

3. *Process Interoperability* aims to make various processes work together in a cross-organizational process.

4. *Business Interoperability* refers to work in a harmonized way at the levels of organization despite the difference of decision-making methods, methods of work etc.

In this thesis the focus will be on Process Interoperability.

2.6.3 Mediation and Mediation Levels

Fensel et al. [FB02] described mediation as a process for settling a dispute between two parties where a third one is employed whose task is to find common ground that will resolve inconsistencies between their respective conceptualizations of a given domain.

There are three levels of mediation that have emerged, namely:

1. *Data mediation* [NVSM07, SCMF06] that addresses signature of messages exchanged in a cross-organizational process.

2. *Process mediation* [KMSF09, SCMF06] that addresses protocol level mismatches.

3. *Functional mediation* [KMSF09, CMS06] which refers to concepts level mismatches. To resolve the interoperability issues of each level, transformation functions should be defined and activities executing these transformation functions should be implemented. This ensemble defines a software component called the mediator.

2.6.4 Dynamic Manufacturing Network

A Dynamic Manufacturing Network (DMN) is a coalition, either permanent or temporal, comprising production systems of geographically dispersed small and medium enterprises. These partners collaborate in a shared value-chain to conduct joint manufacturing [IMA11]. A particular characteristic of a DMN is the possibility to conduct changes on the DMN configuration at run-time for example by modifying the problem statements, or by replacing the partners.

2.7 Features of a PLM Hub

This thesis aims at providing the *coordinator* of an ECM process with a software environment to support him in planning, managing and monitoring the collaboration between different teams (*participants*). In their work, *coordinators* use the product breakdown structure and the work breakdown structure in order to assign the appropriate *participant* to the appropriate component prior to starting the ECM process. Ideally, a cross-organizational process will support the data exchanges of the interaction scenario involving *coordinators* and *participants*. In this section, we present the main features that a software environment should provide so that the *coordinator* could specify problem statements and enact the collaboration with *participants*.

2.7.1 Feature 1: Access to the Product Structure Breakdown

During detail design, the product structure is a working tool for thousands of engineers [GD07]. *Coordinators* need to access the product breakdown structure as depicted in Figure 2.1 in order to assign the right *participant* to a component and then create an ECR on this component. The source from which this product breakdown structure is obtained is the PDM System of the *coordinator*.

In the context of a cascading ECM, the partner that is a *participant* for an ECR while being a *coordinator* for another ECR should be able to define the constraint between the two ECRs interaction scenarios.

2.7.2 Feature 2: Defining the Cross-organizational process

The *coordinator* needs to define the cross-organizational process that will support the exchange of messages with the *participant* and map the activities of this process with his private process and the private process of the *participant*. Nevertheless, business

processes cannot be defined or changed without considering their compliance with imposed business rules [KRM+12]. Accordingly, the PLM Hub should assist the *coordinator* in defining this cross-organizational process and perform an automatic mapping with the private processes using the business rules of each partner.

2.7.3 Feature 3: Defining the Mediator to Run the Cross-Organizational Process

Mediators are required even though the collaboration is based on standards [BCG+05]. The reason is that it is no longer practical to consider that the *coordinator* will impose a cross-organizational process model to all *participants* [VO11]. Thus, the *coordinator* should expect heterogeneity when chaining the *participants* existing processes to support the data exchanges. Accordingly, the *coordinator* needs to resolve this heterogeneity prior to starting the collaboration. The *coordinator* will define a mediator that will intercept the exchanged messages and eventually perform transformation operation on them in order to deliver the exact expected data to the *participant*. Consequently, a fundamental element of the collaboration is the presence of mediators within the PLM Hub to resolve process interoperability issues.

2.7.4 Feature 4: Managing the changes in the context of a DMN

The *coordinator* and the *participant* carry out their tasks in a DMN. Thus, at run-time, the *coordinator* should be able to perform modifications on the current and future configurations of the collaboration environment. These modifications must be applied in a controlled fashion. The aim of this feature is to minimize inconsistencies and disruptions by guaranteeing seamless interoperation of the cross-organizational business process [ABP09a]. The PLM Hub should provide capabilities to partners to modify the configuration of the collaboration environment. In addition it should minimize the impacts of these modifications on the whole collaborative environment.

2.7.5 Feature 5: Monitoring the Collaboration

During the collaboration, a *coordinator* might need to monitor parallel design activities of components that are performed by *participants*. This is an important feature for *coordinators* since it helps them detect possible dysfunctions much earlier. *Coordinators* should be able to specify what they want to monitor and also, they should be able to perform processing on the received messages in order to obtain high-level information transparently.

Business Requirements for a PLM Hub and Features	**Planning** the processing of an ECR	**Running** the interaction scenario of an ECR	**Managing** the processing of an ECR	**Monitoring** the processing of parallel ECRs
F1: Access to product features	Yes		Yes	
F2: Definition of the cross-organizational process	Yes	Yes	Yes	
F3: Mediator	Depends on the mediation strategy	Yes	Depends on the mediation strategy	
F4: Management of changes			Yes	
F5: Monitoring				Yes

Table 2.1: Summary of business needs coverage by software features

2.7.6 Summary

We presented the main features that a PLM Hub should implement in order to respond to *coordinators* needs. Table 2.1 summarizes what business needs each feature will cover. In the next section, we analyze how much functionality the current available frameworks at Airbus Group Innovations are providing to respond to the identified features.

2.8 Existing Frameworks to Ensure PLM Hub Features

Extensive work is being carried out inside the Airbus Group Innovations to address collaboration issues at detail design phase and to support the collaborative execution of the ECM process. Indeed, Airbus Group Innovations has participated to multiple European projects that address interoperability issues at detail design phase (Seine project[3], Crescendo project[4], Athena project[5]). The main results of these projects are: (i) a framework for interoperability developed in [Fig09]. This framework has partially led to the development of an operational collaborative platform of industrial level called PHUSION. In this section we review the AGI frameworks.

[3]http://www.eads-iw.net/web/plminterop/demonstrators
[4]www.crescendo-fp7.eu/
[5]athena.modelbased.net

2.8.1 The Aeronautic Interoperability Framework

The main contribution of this framework consists in defining an innovative concept to ensure data interoperability of the messages exchanged during an interaction scenario of an ECM process. Indeed, [Fig09] defined the *extended hyper-model* that ensures the preservation of the semantic of messages when they are exchanged between different partners. It formalizes the syntactic operations that should be performed on the exchanged messages so that the receiving partners could operate on the received message using their own representation.

2.8.1.1 Building Blocks of the Extended Hyper-model

The extended hyper-model is defined as a tuple $\langle datamodel, ground, landscape, hyper-graph \rangle$ where:

- The **data model** serializes the information being exchanged between two partners in the collaboration environment.
- The **ground** is the software environment that has produced the data model. This data model is assumed to be compliant with a versioned standard.
- The **landscape** of a standard is the set of available grounds that serialize their data models using this specific standard.
- The **hyper-graph** formalizes the multi-representation of a data model using the accepted standards in its community. The vertices of this hyper-graph capture the landscapes and the edges capture the operations that should be performed to travel from one vertex to another.

Figure 2.12 depicts the basic concepts of the extended hyper-model and Figure 2.13 depicts an operation on the data model.

2.8.1.2 Advantages of the Extended Hyper-model

The noticeable advantage of the extended hyper-model is that it ensures data interoperability between the partners' processes. Thus, it helps decrease the coupling between these partners. Indeed, this framework allows partners to keep their own representation of data models. Then, during the collaboration it is up to the extended hyper-model to perform the required transformations in order to deliver the data model in the right standard and in the right format.

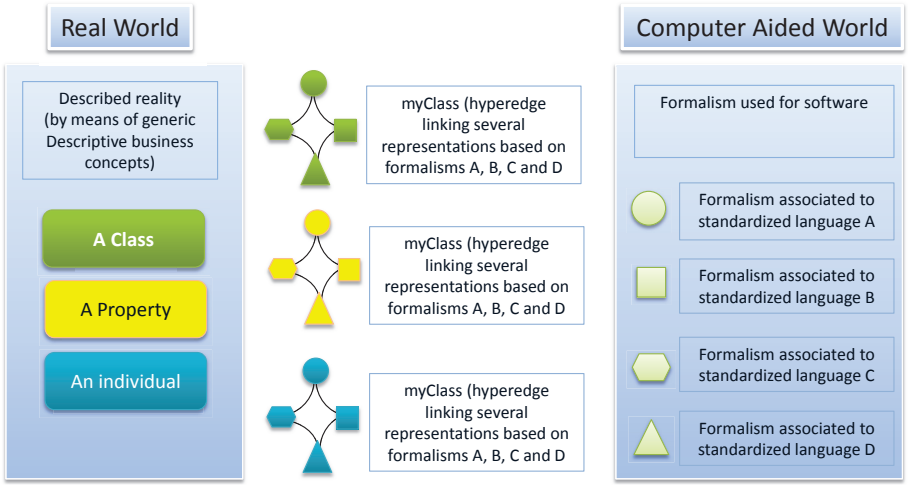


Figure 2.12: Hyper-model concepts

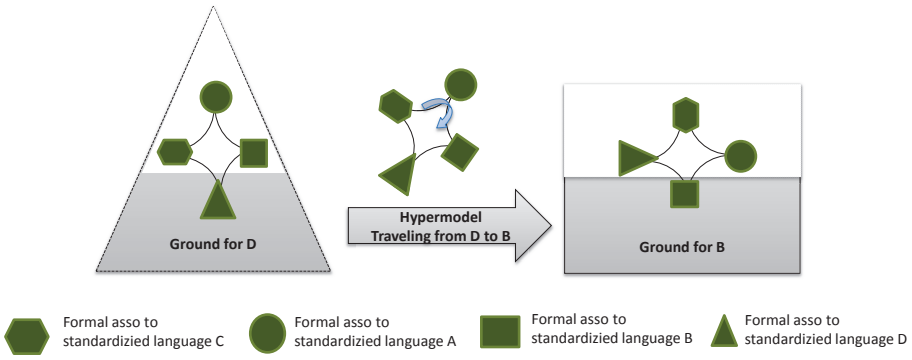


Figure 2.13: Hyper-model operation

2.8.1.3 Limitations of the Extended Hyper-model

Although the extended hyper-models plays a major role in ensuring data interoperability during the collaboration, it lacks capabilities in ensuring processes interoperability. Indeed, whether using the extended-hyper model or using a totally conformant standardized data models, process interoperability issues still persist [BCG+05]. The reason is that on one hand, the data model standards that will be used within a collaborative environment can be forecasted in a certain extension prior to the run-time phase. However if a non-expected standard is introduced at run-time, the extended hyper-model needs to be modified extensively. On the other hand the sequence of activities in the cross-organizational process supporting the collaboration is not always predictable. For example a *participant* joining the collaboration has an already implemented process different from the implementation expected by the *coordinator*. Thus, attaching this *participant* process to the existing cross-organizational process would create mismatches and the extended hyper model does not address this type of mismatches.

2.8.2 IMAGINE Platform

In the European project IMAGINE (2011-2014) a major concept has been developed that is the Dynamic Manufacturing Network (DMN)[6]. The platform developed in this project supports the dynamicity of the collaboration environment during the manufacturing phase of the product. Our contribution is to address interoperability issues at the design phase of the product. Indeed, the dynamic nature of DMNs introduces new challenges regarding the problem of interoperability especially at the process level.

2.8.3 Business Process Management Platforms

Business process management (BPM) is a software environment that supports the design, administration (management), configuration, enactment of business process [Wes10]. BPMs provide the necessary features to *coordinators* to plan, manage and monitor the collaboration. However, they have major lacks that we can summarize in the following points:

- Although the concepts used to define the cross-organizational process are standardized[7], it is known that they are low-level concepts [TS09]. Thus, they are not well-adapted with *coordinators/participants*' way of thinking and the way they want to specify their goals.
- When tailoring the cross-organizational process definition at run-time, this modification will not only impact the partners associated to this change (e.g. when redefining the sequence of his activities). Instead, all partners of the collaboration environment will be impacted since there is a single running workflow that should be stopped and restarted to realize the modifications.
- To allow a *coordinator* to monitor the messages exchanged between *participants*, new activities need to be added to the workflow. This will impact other *participants* in the same cross-organizational process and will create a strong coupling between the *participants*.

2.8.4 PHUSION Global Collaboration Project[FTG+14]

The last motivation that led the development of the framework presented in this thesis is the shift from OEM centric model where the OEM imposes proprietary processes, methods and tools to a network based model where all participants are supported by a

[6]http://www.imagine-futurefactory.eu/uploads/banner/imagine_animation_3_v6-presentation.swf
[7]WfMC Glossary: http://www.aiai.ed.ac.uk/project/wfmc/ARCHIVE/DOCS/glossary/glossary.html

hub in their collaboration as depicted in Figure 2.14. In this environment, heterogeneity between participants processes should be tolerated but should be resolved with the least cost possible.

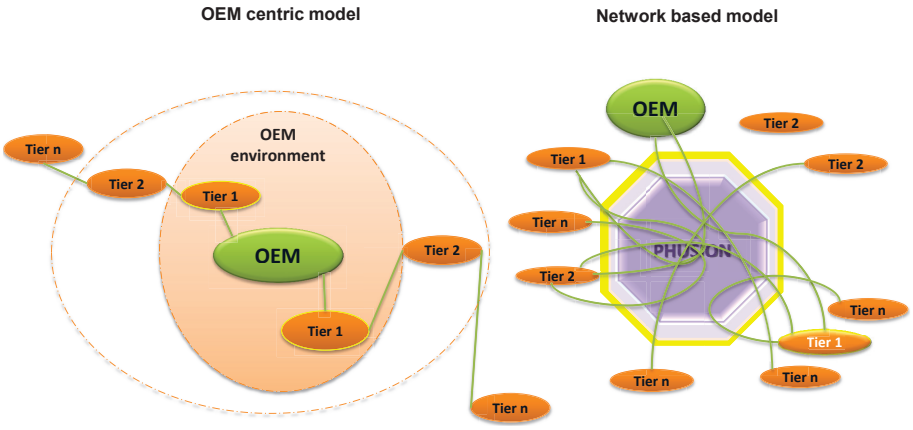


Figure 2.14: PHUSION Platform Objective

This shift is demonstrated by the development of the operational PHUSION platform. Airbus Group has launched the project PHUSION in order to develop a collaborative platform that supports engineers in exchanging data models during the detail design phase and it is currently being used for operational projects.

2.8.5 Summary

Table 2.2 summarizes to what extent the previously developed frameworks implement the features expected from a PLM Hub that supports partners during their collaboration.

This section presented the main industrial frameworks that had impacted the proposals in this thesis. Nevertheless, rigorous and formal models had also impacted the choices made in this thesis in order to develop verifiable and valid solutions. In addition using formal models has facilitated the implementation of the developed algorithms.

2.9 Challenges for a Collaborative Software Environment

In the previous sections, we have analyzed the needs of *coordinators* when defining ECRs in a collaboration environment. Furthermore, we have identified the main features of a PLM hub that a *coordinator* would like to have access to in order to perform his work with *participants*.

In this section, we summarize the main scientific challenges that should be addressed by a PLM Hub framework.

Features and Industrial Framework	Aeronautic Interoperability Framework	IMAGINE Platform	BPM Platforms	PHUSION
F1: Product Breakdown	✖	✔	✖	✔
F2: Cross-organizational Process	✖	✔ (predefined process)	✔ (low level concepts)	✖
F3: Mediator	✔(only for data)	✖	✖ (compatible processes)	✖
F4: Management	✖	✔(for manufacturing only)	✖ (impact all participants)	✖
F5: Monitoring	✖	✔(for manufacturing only)	✔ (for the unfolding of the process only)	✖

Table 2.2: Summary of PLM Hub features coverage by current frameworks in Airbus Group Innovations

2.9.1 Challenge 1: High-level Modeling Language

When creating a new ECR, the *coordinators* should access their PDM Systems and select the component on which they want to perform a modification. Then they define the interaction scenarios with the *participant* in terms of problem statements. The definition of the interaction scenarios could be achieved by Business Process Modeling (BPM) languages by modeling a cross-organizational process. However, BPM languages are known to use low level concepts and in general do not have the adequate business concepts [TS09, vdAP06]. Moreover, these languages are of imperative nature and thus *participants* could be obliged to tailor their private business processes in order to be able to exchange messages with the *coordinator*. Nevertheless, modeling interaction scenarios using these languages provide well-founded models that ensure a correct interoperation from design-time. Since *coordinators* and *participants* are bound through a contractual link, the first challenge consists in developing a contract modeling language that balances between the following two conflicting requirements. On the one hand, we need a modeling language that should be close to the *coordinator* vocabulary and can capture problem statements related to interaction scenarios. On the other hand, this language should consider that *participants* are autonomous and have the freedom to define internal business rules of their business processes to which contracts with *coordinators* will be mapped as depicted in Figure 2.5.

2.9.2 Challenge 2: Mediation on-the-fly

Considering the hypothesis that *coordinators* can no longer impose a cross-organizational process to *participants*, in this case *coordinators* when mapping contracts' terms to private processes should define *mediators* by themselves to resolve the interoperability issues that could occur. Since this work could be tedious for *coordinators* and out of their expertise, the second challenge consists in discharging *coordinators* of this work by automating the mediation mechanism inside the PLM Hub.

2.9.3 Challenge 3: Managing the changes

In the context of a DMN, *coordinators* may need to perform modifications on an ECM process for example by replacing the *participant* responsible of a particular component. Since all ECM processes are interconnected to constitute a unique cross-organizational process, the previous replacement of a *participant* could have a global impact. Indeed, other *participants* collaborating with the *coordinator* could be impacted even though they are not concerned by the replacement. Thus, the third challenge consists in developing capabilities for the PLM Hub so that it minimizes or avoids that a modification in the DMN impacts other *participants*.

2.9.4 Challenge 4: Advanced Monitoring Capability

When multiple Engineering Change Requests are triggered in parallel on different components, a particular *coordinator* may need to monitor the unfolding of the processing of ECRs on the different components. This feature can be fulfilled using existing monitoring tools (e.g. Esper[8]). Nevertheless, the time that the monitoring tool generates the expected information by the *coordinator* can be significant. Thus the fourth challenge consists in developing an efficient monitoring mechanism that delivers the right information on the unfolding of different ECRs to the *coordinator* as fast as possible.

2.10 Thesis Contributions

In order to address the previously identified challenges, we develop a formal framework that supports *coordinators* in specifying ECR interaction scenarios with *participants*. This framework allows *coordinators* to focus on what they want to achieve through the ECR collaboration. It gives them the right constructs to define their goals while discharg-

[8] esper.codehaus.org/

ing them from tedious tasks related to the collaborative platform management. More specifically, the framework developed in this thesis discharges *coordinators* from:

1. Modeling the cross-organizational process corresponding to the interaction scenario by defining the right sequence of message exchanges. Instead, *coordinators* will only specify the obligations that *participants* should uphold when processing the ECRs on components and what characteristics the *participants* will give back to *coordinators* once a processing attempt has been terminated.

2. Looking after ensuring sustainable interoperability in a continuously evolving collaboration environment (DMN). Instead, the framework proposes an on-the-fly mediation approach that is independent from the communicating processes. Indeed, it gives *coordinators* the freedom to make changes on the configuration of the collaboration (e.g. replacing a *participant*) without caring about how their processes and the *participant*'s process will interoperate. All the adaptations will be performed automatically.

3. Caring about the impact of changes on other *participants*' processes. Instead, *coordinators* are allowed to perform modifications that they deem useful for the product design. All the impacts on the cross-organizational process in terms of interoperability or executability are managed by our framework.

4. Struggling to watch the design evolution of other components that they may be interested in. Instead the framework gives *coordinators* the capability to specify declaratively what exchanges they want to monitor and what processing they want to perform on these data. Then, the framework executes optimally the monitoring query.

The first contribution of this thesis aims at combining declarative specifications with on-the-fly mediation [KFBG13b, KFB+13, KBFG14]. Indeed, declarative specifications give *coordinators* the necessary expressiveness to specify the goal that they want to achieve through the collaboration with *participants*. However, it does not give them the expressiveness to specify how the activities that lead to this goal will be organized in the cross-organizational process. Here the on-the-fly mediation enters into action. Indeed, since the organization of activities is not known, there is a high probability that a process interoperability issue will occur between *coordinators'* processes and *participants'* processes. The on-the-fly mediation has the advantage to ensure interoperability between the possible configurations of processes derived from the goal specified by the *coordinator*.

The second contribution consists in proposing a mechanism that maintains the collaboration running for a maximum number of couples of *coordinator/participant* when a modification occurs in the collaboration environment [KFBG13a]. We developed a verification algorithm that aims at deferring the impact of modifications on the cross-organizational process until the algorithm ensures the soundness of this cross-organizational process after the realization of the modification.

Furthermore, a *coordinator* could perform many modifications for example by removing many *participants* in the perspective of replacing them by other participants having equivalent capabilities (i.e. making a recovery). Accordingly, we developed a mechanism that analyzes the state of the configuration of the collaboration environment. Then, it proposes a semi-automated solution to the *coordinator* so that he can recover the modifications he performed as fast as possible. The objective is to resume the collaboration with the new *participants*. This mechanism extends the planning-graph algorithm to generate an executable solution plan.

The third contribution of this thesis consists in providing *coordinators* with an efficient monitoring mechanism to observe what is happening during the design of a particular set of components while performing processing on the received data. We found that current (industrial) tools that could be used to implement the monitoring mechanism do not execute the processing on the continuously incoming data in an optimal fashion. Thus, we proposed an algorithm that optimizes the processing time of the incoming data [KFBG14b].

In order to prove the feasibility of our theoretical contributions, we developed a comprehensive prototype of a collaborative platform that implements the contributions. We used a use case to evaluate the performance of our algorithms. Nevertheless, prior to realizing our platform prototype, the industrial context of the thesis constrained us to select exclusively the software components that uphold the community standards (e.g. the workflow engine that will run the cross-organizational process should be compliant with the WfMC standard). Accordingly, in order not to make our choices arbitrary or just by counting on the software provider 'word', we developed a formal framework that allowed us to assess how much a software implementation is compliant with the standard specification using quantitative metrics [KFBG14a]. Thus at the end we could say that we made our best to develop standards-based collaborative platform.

2.11 Conclusion

Developing PLM Hubs to support the processing of multiple Engineering Change Requests (ECRs) on product components involving multiples *participants* in the collaboration environment is challenging. Until the present time, there is a lack of formal frameworks that address issues related to process interoperability while considering the network dynamicity. In this chapter we have presented the context of this thesis. We started by presenting the generic engineering design process and how it is implemented in the industry through the Engineering Change Management (ECM) process. We have detailed the Engineering Change Request (ECR) subprocesses and from an industrial use case, we have identified the need for a PLM Hub. We have derived the set of features that a PLM Hub should provide for *coordinators* to support them in performing their work related to ECRs. From this set of features, we have analyzed the most important existing frameworks inside Airbus Group Innovations that could cover them. From this analysis, we have identified the limits of these frameworks, and we have derived the scientific challenges from the non-covered features. Finally we have summarized the contributions of this thesis that aim at addressing the identified scientific challenges.

In the next chapter we will survey the related works in the literature that can address the identified challenges and we will show their shortcomings.

Chapter 3

State of the Art

Contents

3.1 Introduction

In Chapter 2, we presented the industrial context of the thesis. More specifically, we presented the Engineering Change Management Process and how this process supports *coordinators* and *participants* in designing product components in a collaboration environment. From the presentation of this process, we derived a set of challenges that we can summarize in the following points:

1. The need for a high-level modeling language to model obligations of the *contract* that binds partners.
2. The need for an adequate mediation approach that supports modifications that could occur in the DMN.
3. The need for managing modifications at run-time in the DMN.
4. The need for an advanced monitoring capability.

We analyzed how these challenges are covered by the existing frameworks developed inside Airbus Group Innovations. In this chapter we deepen our state of the art analysis by identifying the frameworks that could be used to address the challenges identified previously. We show the limitations of these frameworks and we derive a set of enhancement requirements that will shape our framework.

3.2 Collaboration in Dynamic Environments

Since decades, several works were performed to address challenges that could appear in collaboration environments for product design. Some of these challenges concern the resolution of data heterogeneity during product design [SMG+06, AGS+14]. Other challenges concern the management of conflicting situations during product design [Lim09].

The main part of existing works addresses collaboration problem in the context of static environments. Nevertheless, we have seen recently several works that consider the dynamicity of the collaboration environment (e.g. a specific track was organized in the IEEE WETICE conference since 2012 to address the new challenges of dynamic environments[1]). These works generally aim at increasing the autonomy of the underlying software platforms that support the collaboration [MHD14, DME+13]. While considering the dynamicity aspect, in the subsequent sections we determine the limits of existing frameworks that address issues in collaboration environments for product design.

[1] http://conf.laas.fr/MADYNE/

3.3 Business Process Modeling and Management

3.3.1 High-level Languages for Business Modeling

There are many languages that can be used to model *interaction scenarios* of ECM processes. In this section we make a survey of these languages and we show their limits regarding the challenges expressed previously.

3.3.2 Product-based Process Design

Although the ECM process is the de jure standard for product design, there are two other fields where products and processes are tightly coupled. (i) The Bill of Material (BOM) that is used in the manufacturing process of products. [vdA99] (ii) Workflows to generate administrative products (e.g. credit, savings, mortgages). For this kind of products several works have been conducted to derive workflows based on administrative products specifications [RLvdA03, VRvdA08, CV11, vdARL01].

[RLvdA03, vdARL01] have split workflow design methods into (i) participative design approach that consists in developing the process design within the setting of a workshop where small groups of consultants, managers, specialist work together, and (ii) analytical design approach that consists in using formal theory and techniques to model and derive the process design.

More specifically, [RLvdA03] have performed a total shift from current practices in process design [AG99] by using the analytical design approach exclusively. They developed a framework (Product Based Workflow Design PBWD) to derive an optimal workflow with a minimum number of tasks given a set of criteria.

Enhancement Req. 1 For our framework, we shall combine both design approaches (participative design and analytical design). In the participative design, partners work together to develop a contract specification by relying on concepts close to their mind. In the analytical design, a set of algorithms are used to derive the workflow that will support them in achieving the contract. These algorithms should take into account the existing processes and rules of each partner and avoid the clean sheet approach which is rarely feasible in real life.

[VRvdA08] extend PBWD in order to add support capabilities. They consider that there is no need for a process to guide the design of a product. The proposed approach starts from the product model and based on the data elements readily available for a

specific component on one hand and the selected strategy for product design (least cost, shortest processing time) on the other hand, the approach determines the next step to be executed.

3.3.3 Declarative Modeling

Imperative models such as BPMN, petri nets, UML activity diagrams are only good in describing the operational way to fulfill the constraints, leaving the constraints themselves implicit. Accordingly, alternative approaches studied by different research groups are based on the use of declarative process models [BJ94, DKRR98, SKT02, SMTA95, vdAPS09, PvdA06].

[vdAP06] pointed out the limits of imperative languages (e.g. BPMN, BPEL, WS-CDL) including their lack to support changes. They have proposed a declarative framework (DecSerflow) to model processes. They used the temporal logic patterns defined by Dwyer et al. [DAC98] to define constraints between the execution of activities in a process as well as constraints on activities themselves (e.g. the number of times an activity can be executed). Despite the advantages that they bring with this framework, there are mainly two drawbacks:

- They still use low level concepts (activity, message flow etc.) to model the process. Using this language in our case obliges *coordinators* to use these concepts that are far from their mind.

- They do not treat interactions as first class concepts but rather as extensions to a core language centered around the notion of activities and dependencies between activities.

- Since the process specification will end up being a temporal logic formula, the valuation of this formula into an executable process (BPMN for example) will induce ambiguity. Indeed, when relying on Dwyer et al. [DAC98] patterns, authors specify $LeadsTo(A, B)$ (where A, B are activities) and interpret it as B can start only when A has finished. However, Knolmayer et al. [KEP00] claim that an activity is time consuming. Thus, the interpretation made by the authors is only one among two possible interpretations (we make the analogy with the intended model and the unintended model for propositional formulas [NGT04]). The second possible interpretation is B can start only when A has started. The reason is that all instants that precede the end of the execution of activity A are considered as a future of the

instant when *A* has started. Thus, in all these instants *B* can start while upholding the specification.

Enhancement Req.2 We shall formally define the mapping between the declarative specifications that use temporal logic patterns and the execution language to prevent ambiguity.

[HMS11a, HMS11b] introduce a new approach to model cross-organization processes. They rely on a declarative approach to avoid using imperative languages like BPMN. Basically, they model the cross-organizational process by:

- Defining events
- Defining relationships between these events. The kinds of relationships are: an event cannot occur if another event occurs (exclusion). An event should be preceded by another event etc. They have formalized the Dwyer et al. [DAC98] patterns not in temporal logic but using a special class of graphs that is Dynamic Condition Response Graphs.

In their prototype architecture, they developed a new workflow engine to run their cross-organizational process model. This language suffers from the same drawbacks of *DecSerflow*. Moreover, the engine that executes the workflow models is proprietary and is not standardized.

3.3.4 Commitments Framework

Business protocols can be specified in different ways. Some representations, like proposals based on Petri nets, finite state machines or on Pi calculus have an algorithmic (procedural) nature that is suitable to capture the desired interaction flows. However, this kind of specification is deemed to be too rigid [CS03, Sin00, WLH04, TKS14]. To address this rigidity, Singh et al. created a framework based on *Commitments* [TKS14, TS12a, MTS13]. Commitments aim at putting forth the business relationships between interacting parties instead of the operational details supporting this relationship. Commitments support a set of business scenario patterns that were defined in [TS12b, TS09].

A worth mentioning commitment scenario is the *Commercial Transaction*. The Commercial Transaction expresses a value exchange between two trading partners. However, the value exchange of this pattern does not capture the possible iterative nature of the relationship which significantly limits its usage.

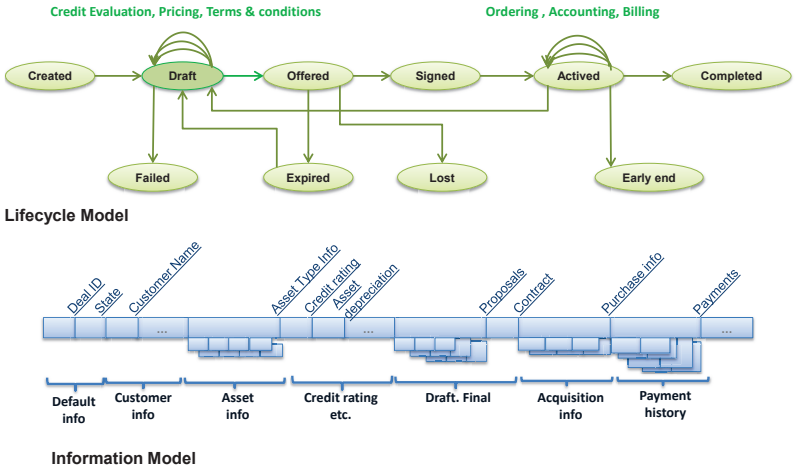


Figure 3.1: Business Entity with Lifecycle

Enhancement Req.3 We shall follow the trend that consists in providing high-level concepts to model interactions between partners. This enhancement can be seen as an additional pattern to the predefined set of patterns established for *Commitments* framework. More specifically, this enhancement shall enrich the *Commercial Transaction* pattern by making it iterative. In addition it shall address interoperability issues that could occur in an efficient manner.

3.3.5 Business Entities with Lifecycle

Nigam et al. [NC03], Bhattacharya et al. [BGH+07] claim that a business process modeling framework should integrate two fundamental aspects *control flow* and *data*. However, from their point of view, typical process modeling approaches emphasis the sequencing of activities, but ignore the informational perspective. To address this shortcoming, they developed the basis of a framework that considers data manipulated by activities (business artifacts) as first class citizens when modeling business processes.

This framework aims at determining all business artifacts that will be manipulated in a business process, additionally; it determines the lifecycle of each artifact by creating an association between the artifact entity and an automaton representing its lifecycles [CH09]. An example of this model is given in Figure 3.1. Subsequent research has been conducted to enhance this framework. Indeed, [HDF+11, VHH+11, HDDM+11, DHV11] found that expressing a life-cycle of an artifact using a finite state machine has limitations. They provided a more elaborated approach to specify the lifecycle of a Business Artifact (BA) that is the Guard-Stage-Milestone (GSM). In the GSM, *stages*

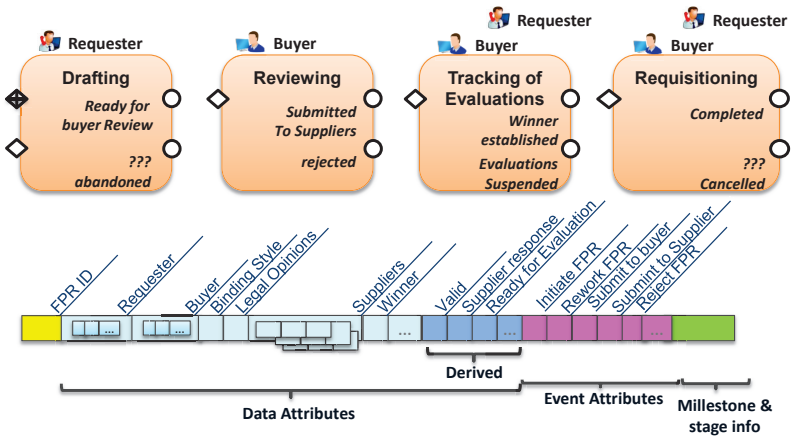


Figure 3.2: GSM specification of an entity lifecycle

provide a mechanism for structuring activities related to a BA instance. Each stage has one or more *Milestones* representing a goal of the stage (true if it has been achieved, false otherwise). *Guards* are rules that control the triggering of the stage. Milestones and guards are represented as rules of the form (**on triggering event if condition**). Figure 3.2 gives an example of a BA lifecycle expressed in terms of GSM.

3.3.6 Business Contracts

Guido et al. [Gov05, GMS06] and Le et al. [LG12] make the link between contracts, business rules and workflows. They developed a language for the specification and implementation of contracts. They defined contracts as mutual agreements between two or more parties engaging in various types of economic exchanges. They also noticed that it is crucial to model contracts in terms of workflows, such that all relevant tasks to achieve contracts can be described as elements of business processes. They defined a logic framework based on Defeasible Logic to specify contracts. The advantage of Defeasible Logic is that it detects any inconsistencies in the contract specification. However, it does not capture the temporal dimension that constrains the contract achievement.

3.3.7 Summary

In this sub-section, we have seen the main frameworks that aim at modeling cross-organizational processes and contracts between partners using high-level specification languages. We have noted that our framework follows the same strategy. We have seen that even though these frameworks provide high-level concepts to model the cross-organizational collaboration, they cannot cover all situations (the case of the commitment framework) and the mapping of the high-level concepts to executable low-level

concepts could be ambiguous.

3.4 Mediators

Mediation (or adaptation) can be derived into two types: interface level mediation that addresses transformation issues related to types of messages [BSBM04]. Such a static compatibility is essential to check, protocol level mediation is more challenging however. Mismatches between protocols [BCG+05, ZGS+13, DSW06] can be classified into the following three categories:

- Attribute granularity difference in message that requires splitting or merging messages to reconcile the mismatch
- Reordering and remembering of messages
- Deadlock

Several works have been carried out in order to resolve the main protocol mismatch patterns that can occur when two processes interact.

Benatallah et al. [BCG+05] developed a set of patterns that capture the possible mismatches that can occur between two communicating business processes.

- Signatures Mismatch Pattern
- Parameter Constraints Pattern
- Ordering Constraint Pattern
- Extra Message Pattern
- Missing Message Pattern
- One to Many message Pattern
- Many to One message Pattern

They assume that analysts will analyze the two processes that will communicate and then they identify the mismatches and the corresponding patterns to resolve them. They also proposed that in the future it will be possible that an automated tool can achieve the identification of what patterns to apply to resolve mismatches. However, as we will see later, the underlying algorithm of such a tool could generate states space explosion especially when it should address the *many-to-one* pattern.

Taher et al. [TABFB09, TPPvdH11] developed a language based on Labeled Transition Systems to formalize the mismatch patterns. The patterns are the same as those that have been defined by Benatallah et al. [BCG+05].

Dumas et al. [DSW06] developed an algebra that defines a set of operators that can be applied on the exchanged messages to reconcile the possible mismatches. Then they defined the visual language associated to these operators that can be used by designers when developing adapters. Authors proposed that in the future they will automate the application of the defined operators. However as we mentioned before, automatically discovering the applicability of the GATHER operator (that corresponds to the many-to-one pattern of [KMNB+09]) is challenging because it could lead to the state space explosion.

Bracciali et al. [BBC05] developed a language to define the possible mismatch patterns between two communicating software components. For a particular couple of software components aiming at establishing a communication, an analyst can use the predefined patterns to specify the adapter behavior for each situation. Later on, an algorithm uses these specifications to derive the adapter. We can notice that the process of specifying the patterns is manual and only the generation of the final adapter is automatic. The mismatch patterns between two software components are classified as follows:

- One-to-one correspondence
- Multiple action correspondence
- Actions without a correspondence
- Non-deterministic action correspondence

Eslamichalandar et al. [EBMN13] identified a set of evolution patterns for two communicating processes. They address the problem of adapters' adaptation when the communicating protocols evolve at run-time. Their approach remains limited because it only covers the case where the sender and receiver are compatible, i.e. they send and expect exactly the same messages but in different orders. The case of the evolution of the many-to-one pattern as defined in [BCG+05] that is more challenging is not addressed.

When two processes communicate, an important mismatch that they might face is the deadlock. Several researches have been conducted to address the problem of deadlock generally by removing the paths in the processes that lead to the deadlock [BBC05]. Motahari Nezhad et al. [MNBM+07] developed an approach that does not remove the

paths that lead to deadlocks which could have an added value for the business but the approach aims at proposing a solution to resolve the deadlock.

Brogi et al. [BP06] aim at building a BPEL process adapter. Given two communicating BPEL processes whose interaction may lock, builds (if possible) a BPEL process that allow the two processes to successfully interoperate. The authors proposed to transform BPEL process models into YAWL workflow. The proposed solution has as inputs two processes C and S whose interaction may lock and it builds (if possible) adapter A, which allows the two processes to successfully interoperate:

- C and S are translated into YAWL
- Build the YAWL workflow of A from the workflows of C and S. It generates the workflow of the adapter that ensures interoperability between C and S.

Even though the proposed solution can resolve processes interoperability issues between several partners (> 2) , it generates for each two partners' processes a mediator, and not a global mediator for all partners. This configuration could lead to maintenance issues since a large number of mediators need to be maintained.

Seguel et al. [SEG09, SEG10] presented a method to construct a minimal adapter for two business protocols containing parallelism and loops. They analyze two BPEL protocols and transform these protocols into trees to automatically identify the minimal set of interactions needing adaptation. Although, the identification method is effective, there is a risk of state space explosion that has not been addressed by the proposed approach when a many-to-one adaptation is required.

Kashlev et al. [KLC13] propose a mediation approach to resolve interoperability issues in scientific workflows. They developed what is called Shims.

The common issue of these approaches is that they focus on defining mediation patterns of the exchanged messages between two processes. However they do not address the problem within a multi-process and dynamic context. In this context, the unique strategy consists in deploying for each couple of processes a mediator. In our case, we follow a different strategy; we deploy a single mediator and create multiple instances of the mediator for each couple. Hence, all mediators will share the same database of mapping functions which fosters reusability.

Table 3.1 concludes this section by summarizing the distinctive features of previous works and comparing them to our work. The main criteria that we relied are (1) whether the framework aims at addressing collaboration issues. (2) whether the framework is flexible enough to support changes at run-time. (3) whether the framework has

	Is the work aimed to address collaboration issues	Flexibility at run-time	Formal Model	Partners Autonomy	Addresses Protocol mismatches	Automatic and efficient mediation	Conformance with standards
Our Work [KFBG13b, KFB+13, KBFG14]	Yes	Yes	Temporal Logic+ Automata	Yes	Yes	Yes	Yes to a certain extent
Kondengha et al. [KNBSP14, KMNB+09]	Yes	No for Many-to-one mapping	Aspect Orientation	No for Many-to-one	Yes	No for Many-to-one	Yes to a certain extent
Let's Dance [ZBDtH06]	Yes	No	Semi-formal	Yes	No	No	No
Van der aaslt et al. [vdALM+10]	Yes	No	Petri nets+ Automata	Yes	No	No	No
Benatallah et al. [BCG+05]	Yes	No	Templates	Yes	Yes	No	No
Business Entities with Lifecycle [HDDM+11]	Yes	Yes but limited	ECA rules	Yes	No	No	No

Table 3.1: The distinctive features of some related approaches

formal basis and (4) maintains the partners' autonomy when realizing the collaboration specification. We also considered (5) whether the framework can resolve protocol level mismatches and possibly (6) using automated mediation. Finally, due to the industrial constraints related to the use of standards, we considered (7) whether the framework implementation is compliant to standards.

3.5 Management of Changes in Business Processes

One of the challenges that we should address during the execution of interaction scenarios of ECM processes are the modifications that could occur at run-time on a cross-organizational process supporting these interaction scenarios. These changes involve

adding/removing activities, lanes in processes etc. To facilitate and rationalize these modifications, multiple works have been conducted that we detail in the next subsections.

3.5.1 Ensuring Processes Flexibility

Process flexibility has been identified as a major challenge in process management research for more than a decade [CCPP98, RRD04a, RW12]. In order to discuss process flexibility, we first introduce some basic terminology.

For each business process to be supported corresponding to *interaction scenarios* a *process type* represented by a *process schema* has to be defined (or in our case generated). For one particular process type several process schemes may exist, representing the different versions and the evolution of this process type over time. A process schema comprises a set of *nodes* representing process *activities* or *control connectors* (e.g. XOR-Split, AND-Join) and a set of control edges (i.e. precedence relations) between them. Activities belong to a lane that represents the actor that will execute these activities. Based on the process schema, at run-time new *process instances* can be created and executed. Completion events related to the activities of a process instance are recorded in a *trace*.

[RRD03, RRKD05, RD98] developed the ADEPT and its evolution ADEPT2 framework that offers a set of advanced features to model workflows and more importantly to realize ad-hoc changes on running workflows (e.g. omit activities, change activity sequences, insert activities). Authors claim that such dynamic changes must not lead to an unstable system behavior. ADEPT ensures the correctness by introducing formal pre- and post-conditions for change operations. In particular, a consistent state must be preserved when a workflow instance is going to be adapted. For example if we consider the ADEPT task insertion operation in a running workflow, in this case ADEPT limits its verification of the workflow correctness to two conditions: (1) Ensuring that the task preceding the task to be inserted has not finished yet and (2) Ensuring that the task following the task to be inserted has not started. When these two conditions are satisfied ADEPT will insert the task as depicted in Figure 3.3 However in the context of a workflow implementing a cross-organizational process, after the insertion of a new task, limiting the verification of the workflow correctness only to the two mentioned conditions is necessary but not sufficient. Indeed, if the inserted task is a send task and if it has no corresponding receive task, then this will create an interoperability issue for the workflow. The latest version of ADEPT does not support this verification.

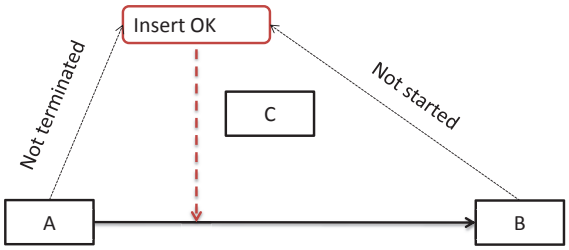


Figure 3.3: Inserting an activity

Enhancement Req.4 We shall perform the verifications related to the cross-organizational nature of the workflow prior to validating the correctness of that workflow after a modification.

Another shortcoming of the adaptation approach proposed by ADEPT is that it verifies the consistency of the workflow after each execution of an adaptation operation. If the operation executed does not generate a correct workflow, it does not accept it (pessimistic approach). Nevertheless, it could be possible that this adaptation operation will be followed by another operation and together, they will lead to a correct workflow.

Enhancement Req5. We shall provide an (optimistic) approach. In other terms, even though an operation leads to an incorrect workflow model we shall keep it and we assume that the designer will run other operations that will make the incorrect workflow correct.

A *choreography* is a global overview on the inter-connection between partners communication imperatives [TBSR06]. [FRMR12] developed an approach to propagate changes on a choreography model involving several partners. It determines all partners affected by a change; it propagates the change to these partners and negotiates with them in order to keep the choreography processes compatible. Authors define the semantics of 4 change operations INSERT, DELETE, REPLACE and UPDATE. Although the authors have detailed the impact of each operation, they have not elaborated on the negotiation process and what would happen if a partner rejects a change proposal. Indeed, if a partner impacted by the change refuses the proposed change, the change requester would no longer be able to collaborate and thus all the choreography is impacted. Moreover, the negotiation could time. During all this period the whole choreography is stopped even though there are partners that should not be concerned by the change. Furthermore, the proposed approach obliges partners to change their processes in order to maintain the

processes of the choreography compatible in case they accept the modification. The approach does not consider mediation solutions and it is not clear how existing mediation solution could be used in this case.

Enhancement Req.6 We shall consider choreography as a contract, as [vdALM+10] did, and we shall provide high-level management operations that keep the changes local and shield all partners not associated to these changes from their effect. Then, if a change is not satisfactory and requires some negotiation, this negotiation remains transparent for all other partners not associated with it.

[WRR13, WPTR13] have worked on the process of process modeling. Following the workflow patterns initiative[2], much of their contributions is on the definition and formalization of *change patterns* on process models. Indeed, instead of considering the process model as a simple graph with primitive change operations (*add/remove node, add/remove edge*) which can be error-prone. They defined a set of 18 high-level change patterns that modify a process model in a controllable way. Figure 3.4 compares between using a change pattern and using change primitives. The original process schema on the left side consists of a single activity A. Assume now that a process change shall be accomplished by inserting activity B in parallel to A. On the one hand this change can be accomplished by using one high-level change operation $parallelInsert(S, B, A)$ which adds activity B parallel to A. To apply this change pattern the user has just to specify a couple of parameters. On the other hand, change primitives can be used to realize the desired adaptation. In the given example, the transformation of the original process schema into the new schema version requires nine change primitives. Using the high-level operation *parallelInsert* instead, from the perspective of the user eight operations can be saved.

[BOAT13a, BOAT13b] developed a set of patterns to manage adaptations for cross-organizational processes. These patterns are similar to those defined by [WRR13]. Nevertheless, they add a set of patterns that consider adaptation of interactions which is not supported by [WRR13]. This research is limited to the identification of the adaptation patterns, it does not detail and does not address the impact of applying these adaptations on the collaboration.

[ARCR14] presented the adaptation in healthcare processes. They identified a specific set of constraints on adaptation in healthcare processes such as legal constraints, environmental constraints and technical constraints. They developed a methodology

[2] http://www.workflowpatterns.com/

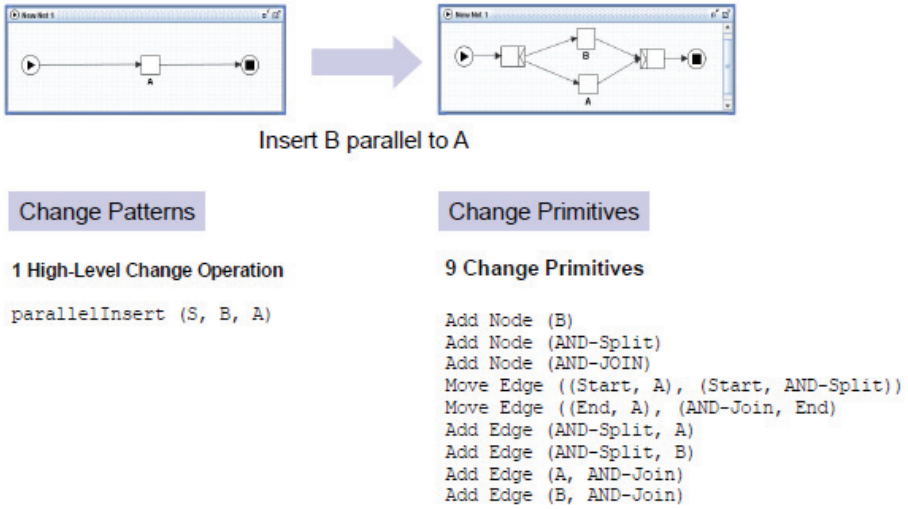


Figure 3.4: Inserting a pattern [WRR13]

to describe flexible processes for healthcare processes called V-BPMI. V-BPMN creates multiples variants of the same process template and at run-time, it instantiates the right variant.

Enhcancement Req.7 We shall go further by providing higher-level management operations to manage contracts between *coordinators* and *participants* in a DMN. In addition, we shall address their impacts on the whole collaboration.

[RWR06] address issues related to the evolution of choreographies. Indeed, if one party changes its process in an uncontrolled manner, inconsistencies on interactions might occur. The approach developed by the authors consists in propagating changes that occur in a process to all related processes of the choreography. Since a process of a partner cannot be modified without his consent, they propose an approach to assist him in driving the modifications that should be performed. Although this approach resolves the inconsistencies that would appear after the changes, it remains functionally limited. The reason is that if the person responsible of the second process refuses to accept the change, then the processes of the choreography will no longer be executable. Using a mediation solution could resolve this problem.

[KR13, KKR12] define the notion of Process View (ProView). They consider that different user groups need customized views on the process models relevant to them and thus there is a need to change process models based on respective views. The aim of their work consists in propagating user changes on a process view to all other views having the same process model as depicted in Figure 3.5.

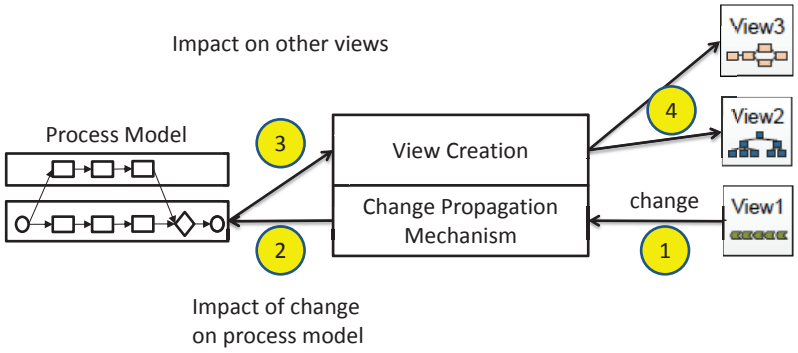


Figure 3.5: Change propagation to views

The propagation mechanism consists in resolving ambiguities that could occur when adding or deleting process blocks. It uses parameterizable view update operations. These parameters allow the resolution of ambiguity when propagating changes.

Table 3.2 compares between our approach that handles the identified enhancement requirements and the existing frameworks.

3.5.2 Generating Recovery Workflows Using AI Planning

Defining the operations of managing cross-organizational processes supporting contracts is necessary but from a functional point of view we can go further. Indeed, if one observes the fourth column of Table 3.2, it shows that existing frameworks just ignore the operation that makes the process syntactically incorrect. They do not consider that in the future other operations might be invoked that will resolve this incorrectness (i.e. they adopt a pessimistic attitude).

We aim at developing an optimistic attitude for our framework. Indeed, it should assume that after the execution of an operation that creates syntactic errors in the process schema, other operations might follow that will resolve these errors. In order to support this optimistic attitude, we should guide users that execute the management operations through a workflow that we call a *recovery workflow*. Nevertheless, the recovery workflow needs to be generated automatically thus we need to identify the mechanism that is able to generate this workflow of management operations.

In the next subsections we give some preliminaries on techniques used to generate the recovery workflow (c.f., AI Planning, Workflow patterns, Abstract State Machines). Then, we will survey the state of the art of the works related to the generation of workflows using the technique identified.

	Considers Syntactic Correctness	**Considers Cross-Org Aspects**	**Ignore a change if it leads to inconsistencies**	**Limit the impact of changes**	**Considers partners' autonomy**	**Considers using a mediation to continue the collaboration**
Our approach	Yes	Yes	No	Yes	Yes	Yes
ADEPT and ADEPT2 [RRKD05]	Yes	No	Yes	Yes	Not for cross-org aspects	No
Fadhila et al.[FRMR12]	Yes	Yes	Yes	No	No	No
Change Patterns	Yes	No	Yes	No	Not for cross-org aspects	Not for cross-org aspects
[RWR06]	Yes	Yes	No	No	No	No

Table 3.2: Comparison of process flexibility frameworks

3.5.2.1 The Planning-graph Algorithm[NGT04]

Planning is the reasoning side of acting. It is an abstract explicit deliberation process that chooses and organizes actions by anticipating their expected outcomes. This deliberation aims at achieving as best as possible some pre-stated objectives.

Given a state transition system Σ, the purpose of planning is to find which actions to apply to which states in order to achieve some objective when starting from some given situation. A plan is a structure that gives the appropriate actions. The objective can be specified as a goal state s_g. The objective is achieved by any sequence of state transition that ends at one of the goal states.

A *planning problem* for a state transition system $\Sigma = (S, A, \gamma)$ where:

- S is a finite set of states
- A is a finite set of actions
- $\gamma : S \times A \rightarrow S$ is a transition function

is defined as a tuple $P = (\Sigma, s_0, g)$, where s_0 is an initial state and g corresponds to a set of goal states. A solution to P is a sequence of actions $(a_1, a_2, ..., a_k)$ corresponding

to a sequence of state transitions $(s_1, s_2, ..., s_k)$ such that $s_1 = \gamma(s_{k-1}, a_k)$, and s_k is a goal state.

The *Planning-graph algorithm* is a technique that can generate a solution plan for a given planning problem. This algorithm as output a sequence of sets of actions e.g. $\langle\{a_1, a_2\}, \{a_3, a_4\}, \{a_5, a_6, a_7\}\rangle$ which represents all sequences starting with a_1 and a_2 in any order, followed by a_3 and a_4 in any order, followed by a_5, a_6, a_7 in any order. Hence, a *solution plan* Π associated to a planning graph is defined as a sequence of sets of actions: $\Pi = \langle\pi_1, \pi_2, ..., \pi_k\rangle$. The set π_1 is a subset of independent actions that can be applied in any order to the initial state s_0 and can lead to a state that is a subset of the first level of the planning-graph. From this state, actions in $\pi_2 \subseteq A$ would proceed and produce a state that is a subset of the second level of the planning-graph and so on until a set π_k whose actions lead to a state meeting the goal g.

3.5.2.2 Workflow Patterns

Workflow patterns is an initiative that aims at providing well founded constructs to realize the assessment of the functionality provided by workflow engines in the market. The development of workflow patterns was accompanied by the development of an extension of Petri nets that is the YAWL language[3]. This language models the workflow patterns and shows how they can be used practically.

Recently, other languages have been used to formalize the workflow patterns depending on the intended usage. There exists formalization of workflow patterns in terms of process algebra, and formalization in terms of abstract state machines exists as well.

Regardless of the language used to formalize the workflow patterns, 43 control workflow patterns have been discovered. They enumerate all possible situations a designer could face when modeling a business process, and also provide a well-founded specification for the development of workflow engines.

These patterns can be classified hierarchically as depicted in Figure 3.6.

3.5.2.3 Abstract State Machines

YAWL was originally used to model the workflow patterns. Other modeling languages have been used to model workflow patterns including Abstract State Machines (ASM) [Bö7]. ASMs are a means of formalizing systems in a state based way [Spioo]. Accordingly workflow patterns can be seen as ASMs that have well-defined states and well-

[3]www.yawlfoundation.org/

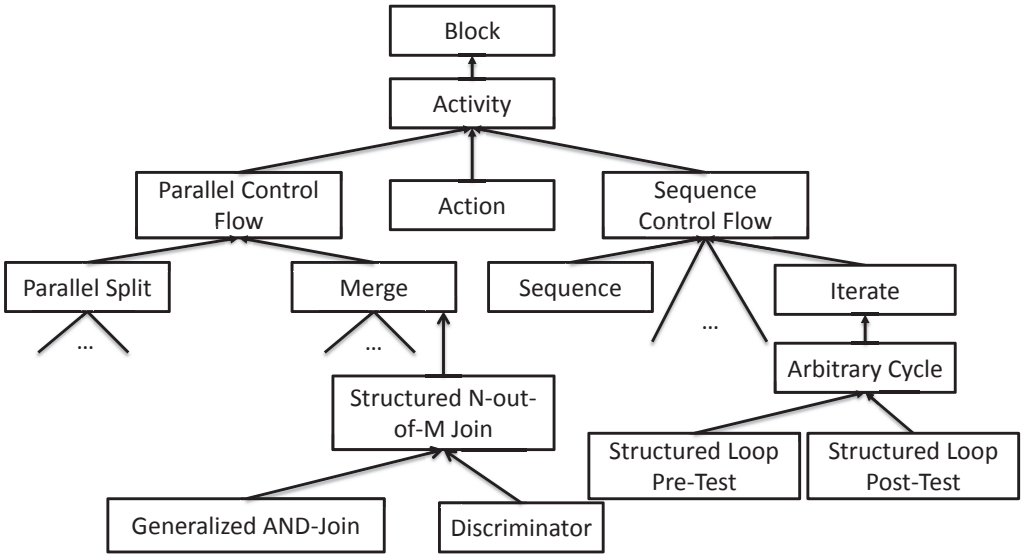


Figure 3.6: The hierarchy of Workflow patterns as established by [Bö7]

defined transitions. For example [Bö7] formalized the **Parallel Split** pattern in terms of ASM as follows:

> **ParallelSplit**(*Activity*, *Thread*, *TriggerExec*) = **forall** $a \in Activity$ **let** $t = new(Thread)$ **in** $TriggerExec(t, a)$

This pattern formalization means that each activity a of the set *Activity* will be executed by a thread (or partner) t in the set *Thread*.

In this sub-section we give the formal definition of an ASM as defined by [Spi00] that we will use in Chapter 5. Further details on ASM can be found in [BS03].

Definition 1. *(Signature)[DvdAtH05] A signature Σ is given as a pair (S, Ω), where $S = \{S_i\}_i$ denotes a set of sorts and Ω is a finite collection of function symbols, each with a fixed arity, i.e.: $f : S_1 \times S_2 \times ... \times S_k \rightarrow S_n$*

Definition 2. *(State)[DvdAtH05] A state in ASMs is a many sorted algebra. For a signature $\Sigma = \langle S, \Omega \rangle$, a many sorted algebra is given by:*

- *(carrier) sets S_i^A for each sort $S_i \in S$*
- *an interpretation for each function symbol $f : S_1 \times S_2, ..., \times S_k \rightarrow S_n$ in Ω is given by a total function: $f^A : S_1^A \times ... \times S_k^A \rightarrow S_n^A$*

Definition 3. *(Expanded State)[DvdAtH05, LMSW08] Given a variable assignment ζ over state A, we denote the pair $B = (A, \zeta)$ as the expanded state. If ζ assigns the values $\overline{a} \in S^A$ to the variables $\overline{v}$ in a state A, we denote the partially expanded state as: $B(\overline{v} \mapsto \overline{a}) = (A, \zeta(\overline{v} \mapsto \overline{a}))$*

Definition 4. *(Fully Expanded State) A fully expanded state is a partially expanded state where every variable is assigned to a value.*

Definition 5. *(State-Term Appropriateness)[DvdAtHo5, LMSWo8] A state A and a term t are* ***appropriate*** *for each other if the signature of A (Σ) contains all function symbols occurring in t and $Var(t)$ is included in the set of variables of A.*

Definition 6. *(Term evaluation) Let $B = (A, \zeta)$ be a fully expanded state and t ranging over terms appropriate for state A. The value $Val_B(t)$ of t at state A is defined as follows:*

- *If the term is a variable $t = v$, then $Val_B(t) = \zeta(v)$*
- *If $t = f(s_1, ..., s_k)$ then its value is given by induction over term structure: $Val_B(t) = f^A(Val_B(s_1), ..., Val_B(s_k))$*
- *If t is a ground term, then $Val_B(t) = Val(t)$*

Definition 7. *(ASM Precondition) Each ASM can be associated to a precondition expressed using the key word* ***where****.*

When generating a workflow (semi-)automatically we need to apply patterns to interconnect the workflow activities. The choice of the pattern to apply when interconnecting two or more activities is important because it captures the behavior of the workflow as desired by the designer. Nevertheless, existing approaches of generating workflow models or service compositions do not consider all patterns depicted in Figure 3.6 but they are generally limited to the patterns at the high-level of the hierarchy. Indeed, a pattern in the hierarchy captures the behavioral of all its children. Thus, when a pattern is appropriate for a certain case, choosing its parent can be risky because the parent will allow certain behaviors not allowed by its child which could lead to inconsistency in the whole workflow. For example, the **Generalized-AND join** pattern can be used to merge multiple flows into one flow. Its specificity is that it waits that *all* input flows arrive to trigger the output flow. If its parent, **Merge**, is used instead, it is generic enough such that it can trigger the merge even though not all input flows have arrived.

Wang et al. [WHR11] proposed extensions of the Partial Order Planning algorithm [NGTo4]. They post-process the output solution plan in order to generate a workflow representation. They identify a subset of workflow patterns from the solution plan to create a workflow diagram. Although they were able to generate the workflow corresponding to a plan, their approach is restricted to use only 7 patterns from 43.

Hatzi et al. [HVN+12] transform a generated plan into an executable composite process specified in OWL-S[4]. Their algorithm uses OWL-S constructs to build composite

[4]http://www.w3.org/Submission/OWL-S/

processes from atomic ones, namely: **Sequence**, **Split** and **Split+Join**. These constructs are too limited in comparison to the number of possible patterns that have been defined in workflow patterns initiative.

Zheng et al. [ZY08] have developed a simplified version of the planning-graph algorithm. They assumed that in the planning graph there is no mutex relationship between actions of the same level of the graph. Nevertheless, redundant services could exist at the same level. Therefore, their effort consists in removing the redundant services in order to produce the service composition that achieves the goal. This composition, however, considers only two patterns among the existing 43 patterns: the **Sequence** and the **Parallel split** patterns.

Marrella et al. [ML13, MMR11] used the partial planning algorithm for automatic adaptivity of workflows. This adaptivity is triggered whenever there is a difference between the expected state and the state reached. The adaptivity consists in generating a workflow (in YAWL) that will reach the expected state from the current incomplete state. Here, the unique pattern in the plan workflow is the **Sequence** pattern.

Moreno et al. [MMK02, RMBCO07] developed a framework to help automate the definition of business processes using AI planning techniques. They defined a set of points of correspondence between plans and workflows meta-models. Nevertheless, these correspondence points remain very limited since they just map plan actions, their preconditions/postconditions to workflow activities preconditions/postconditions respectively. It is straightforward to observe that control flows are not supported.

Beest et al. [VBKB+14] developed a software component called AI Planner that generates a process model from a planning problem. The generated process is a finite partially ordered set of activities. From this definition, it is possible to notice that advanced branching patterns are not considered by the generation algorithm. In fact the AI Planner algorithm considers only two patterns (**Sequence**, **Decision**).

3.5.3 Summary

Table 3.3 concludes this section by summarizing the distinctive features of previous works and comparing them to our work. The main comparison criteria is whether all workflow patterns are taken into account when generating the workflow. The comparison also considers the underlying formal model since we want formal basis for our conceptual framework. Finally, due to industrial constraints, we also consider the compliance to standards for the frameworks' implementation.

	all control flow patterns when generating the process are considered	**Formal model**	**Compliance to standards**
Our approach	Yes	Yes (ASM)	Yes (XPDL)
Wang et al. [WHR11]	No	Yes (Graphs)	No(Directed Acyclic Graphs)
Hatzi et al. [HVN+12]	No	Yes (Logic Formula)	Yes (OWL-S)
Zheng et al. [ZY08]	No	Yes (Graphs)	No (Plain XML)
Marrella et al. [ML13, MMR11]	No	Yes (Petri nets)	No (YAWL)

Table 3.3: Comparison of the workflow patterns coverage by previous work when generating workflows

3.6 Monitoring Cross-Organizational Processes

3.6.1 Preliminaries on Publish/Subscribe Systems

Publish/Subscribe technology encompasses a wide number of solutions that aim at solving a vital problem pertaining to timely information dissemination and event delivery from publishers to subscribers.

Figure 3.7 depicts the architecture of a Pub/Sub system. (i) The Pub/Sub Service stores and manages the subscriptions for one or more publishers. (ii) Publishers push notifications to the Pub/Sub Services. The latter receives these notifications, stores them and then notifies the subscribers that subscribed to this kind of notifications. To notify the right subscribers, it uses a mechanism to match published information and expected information. Basically, there are four approaches to realize the matching between the published information and the expected one namely: content-based, header-based, topic-based and type-based (Figure 3.8):

- Content-based matching compares the whole information in the payload of the published event with the content of the expected information
- Header-based matching compares the header of the event and not its payload with meta-information of the expected information. This matching mechanism is limited in comparison to the content-based matching
- Topic-based matching consists in associating each incoming event to a channel. Then the matching between expected events and the incoming ones relies on the

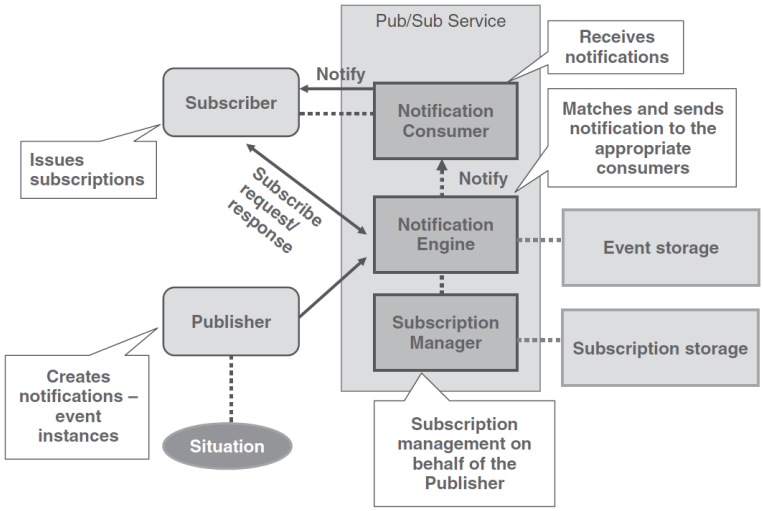


Figure 3.7: The Pub/Sub system (from [Tar12])

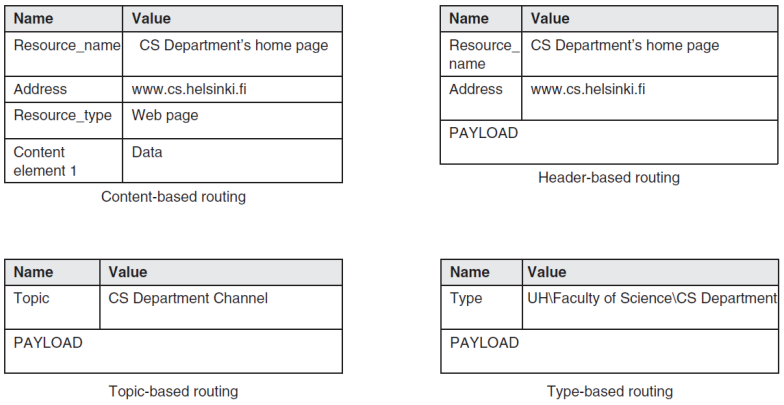


Figure 3.8: Examples of different matching strategies (from [Tar12])

channel identifier

- Type-based matching compares the types of the expected events with the type of the incoming ones. Since types can be organized hierarchically, subscribing to a node in the hierarchy leads to the reception of all events of the type of this node as well as event of type of children nodes in the hierarchy.

3.6.2 Complex Event Processing

Complex Event Processing (CEP) is an approach to implement Pub/sub systems. It refers to any running operations on complex events such as the creation, the reading, the transformation and the abstraction of data carried by these events. In other terms, CEP processes events carrying information that are produced by multiples sources [Bao13].

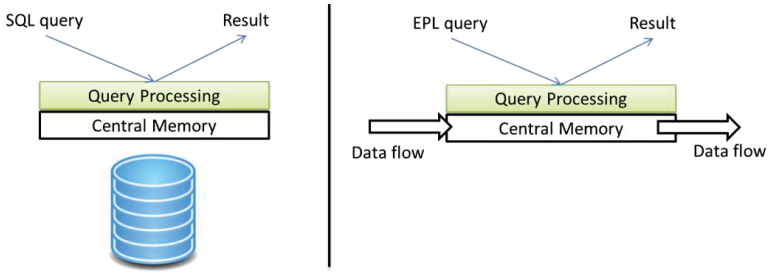


Figure 3.9: Difference between a DBMS and a CEP System (adapted from [Bao13])

The objective of CEP is to satisfy event processing needs in terms of traceability, reactivity, availability and decision making. The complexity of implementing CEP resides in defining efficient algorithms which process and correlate events [Bao13].

The basic terminology used in CEP domain can be summarized in the following concepts:

Event: An event is the recording of a fact. In its simplest form, it indicates the instant and the spot of the fact associated to other data [Bao13]. For example changing the property of a component in the product can be considered as an event that would carry the new value of the property.

Derived Event, or Complex Event: Although simple events are useful by themselves, in many scenarios, some interesting events are inferred only from a combination of simple events occurrence. These are complex events. A complex event is the abstraction of several atomic events. For example the event "Aircraft Structure Airflow Simulation Result" could have as members the events: "Fuselage Airflow Simulation Result", "Wing Airflow Simulation Result", "Empennage Airflow Simulation Result". The relationship between a complex event and its members is called aggregation. The occurrence of a complex event involves the occurrence of all its members. In additions, deriving the complex event from its members could involve performing some processing on its events that could be more or less long.

Continuous Query Processing: A CEP system is able to manage a large number of complex queries. Event processing queries could be written in a SQL-like (e.g. Event Processing Language (EPL)). Using these languages one can define rules for how to process in the inbound streams of events. Processing queries on events in a CEP system follows an inverse logic of processing queries on a database by a Database Management System (DBMS). Figure 3.9 compares between both techniques.

3.6.3 Monitoring Approaches Using CEP

There are many works on CEP frameworks that can be used to monitor cross-organizational processes. [EFGK03, HSS+14] provide a thorough survey on these frameworks. We elaborate on some of them.

Comuzzi et al. [CAV12] developed a monitoring system that allows customers to customize what they want to monitor in a running business process. They identified multiple monitoring dimensions and their associated options. For each dimension, a customer can select the option that fits its requirements and then the monitoring infrastructure will be in charge to deliver the appropriate information. The limit of this framework is that they do not address the issue of data aggregation. They gave an example where a company provides an advertisement process and customers would like to monitor the unfolding of a single piece of information that is the number of clicks on their ad.

Baouab et al. [BFPG12] proposed a decentralized system to monitor the exchanged messages in a process choreography. Their proposal consists in deploying on each partner a software component called the EFM (External Flow Monitor) that manages the incoming messages and generates notifications for other partners. They also proposed a hierarchical organization of partners in order to reduce the flow of notifications and the network overhead that they may generate. Despite that in collaboration environments, partners usually need to filter the notifications they receive [EFGK03], when using this approach, partners cannot select what message instances they want to monitor and thus they will receive all messages. Moreover, if a customer wants to monitor parallel incoming messages and the combine them, this is not specified in the proposed approach. In fact, a customer will receive messages as they are exchanged.

Aurora [ACc+03] is a system for processing data streams. It uses boxes and arrows as constructs to specify the processing to be performed on data flows. Aurora's query algebra (SQuAl) contains built-in support for seven primitive operations for expressing processing requirements. However there is no mention that SQuAl provides the feature to call external services. Thus our conceptual framework can be used to extend the functionality provided by Aurora.

Borealis [AAB+05] is system that succeeds to Aurora. Queries in this system are similar to Aurora queries but they support revision messages. Indeed, in many real-world streams, corrections or updates to previously processed data could be required. The objective of Borealis is to process revisions intelligently, correcting query results that have been emitted in a manner that is consistent with the corrected data. However, since

	Addresses Data Processing	**Maintains Partners Autonomy**
Our Approach [KFBG14b]	Yes	Yes
Comuzzi et al. [CAV12]	No	Yes
Baouab et al. [BFPG12]	No	No
Aurora, Borealis [AAB+05]	Yes but limited to simple operators	Yes
Esper	Yes but inefficient processing	Yes
Composed Service	Yes but inefficient processing	Yes

Table 3.4: Comparison between our approach and the existing ones

Borealis relies on the querying system of Aurora, Borealis system queries are limited to use the Aurora seven primitive operations.

Recently, monitoring has received a particular attention in cloud-based systems. [GKMW11] proposed an approach for monitoring and adapting multi-layered service based systems for cloud platforms. In the monitoring phase, they used EcoWare [BCGG10] that is based on Esper to correlate events. As we will see in Chapter 6, Esper has performance issues when processing incoming events.

3.6.4 Summary

Table 3.4 concludes this section by summarizing the distinctive features of previous works and comparing them to our work.

3.7 Conclusion

In this chapter we reviewed the state of the art on Business Process Modeling, Management and Monitoring. For Business Process Modeling, we argued that the current declarative frameworks have some shortcomings when modeling interactions and we also observed that the available mediation approaches require an important intervention from the side of users. This prevents average users (e.g. engineers) from defining mediators at large. For Business Process Management, we argued that current frameworks address changes for a single actor processes but are limited when it comes to perform changes on cross-organizational processes. We also observed that they generally follow a pessimistic policy regarding changes that impact the correctness of the process schema. Finally, we reviewed some works in the complex event processing area.

We identified a set of enhancement criteria that will guide the design of our framework. This framework aims at meeting these requirements while realizing the functionalities expected for our industrial context.

In the rest of this dissertation we will detail our contributions that address the shortcomings of the related works.

Chapter 4

A Declarative Framework for Product Level Agreements

Contents

4.1 Introduction

Forces of globalization and the ever growing need for differentiation and innovation are moving work from a co-located to a distributed environment. Organizations form collaborations to achieve a goal that none could achieve individually [OBRC10]. This inter-organizational cooperation between enterprises is realized by specifying an abstract description of the overall inter-organizational process that is the choreography. The choreography serves as a common contract between the parties involved in the overall process [vdALM+10]. Examples of choreography languages are WS-CDL[1], Let's dance [ZBDtH06], BPEL4Chor[DKLW07]. Nevertheless, those specific modeling languages do not provide a sufficient level of abstraction to capture the intention behind the interactions [TS09]. Moreover, contracts built with these modeling languages increase the coupling of stakeholders' processes because they capture many details and thus, they over-specify contracts [vdAP06]. For example in a certain choreography the sender of messages has concurrent engineering capabilities and thus can run multiple send activities in parallel. On the other hand, the receiver of these messages does not have concurrent engineering capabilities and thus can run the corresponding receive activities only in sequence. A contract modeled with the mentioned languages constrains the sender to drop his concurrent engineering capabilities and to send messages sequentially. This demonstrates the tight coupling between the sender and the receiver which makes dynamicity in the context of a DMN difficult to achieve.

In this chapter we are interested in developing modeling constructs that facilitate the work of *coordinators* involved in ECM processes. These *coordinators* will define and run contracts supporting Engineering Change Requests with *participants*. The proposed modeling approach under-specifies the collaboration contract by avoiding the inclusion of too much detail. Accordingly, our modeling approach decreases the coupling between partners and facilitates the dynamicity in the context of a DMN. Indeed, this under-specification of the contract leaves the door open for partners to implement internally the processes that support their collaboration in order to achieve the contract terms. This is a positive point, because partners will uphold their internal (business) rules when defining their processes [CM04]. However, mismatches between the communicating processes will definitely appear and thus a mediation solution is mandatory to resolve these mismatches.

Throughout this chapter, we lay down the foundations of a product-based contract modeling approach. We aim at offering *coordinators* of ECM processes constructs at

[1] www.w3.org/TR/ws-cdl-10

the right level of abstraction to model the contract with *participants*. Furthermore, we develop a novel mediation approach that resolves the possible mismatches and that is well adapted to handle the dynamicity of DMNs by limiting the impact of changes on the overall collaboration environment.

4.2 Running Example

Consider that in a collaboration environment for an aircraft design, the *coordinator* wants to issue an Engineering Change Request (ECR) on the Geometry of the Fuselage (*FusGeo*) and the Aerodynamic of the Fuselage (*FusAero*) by changing a set of their attributes as depicted in Figure 4.1. Since the *FusGeo* and the *FusAero* are managed by two different *participants* belonging to an external organization, the *coordinator* needs to create a contract with this organization involving both *participants* to perform the changes on the *Fuselage* (=*FusGeo* + *FusAero*). In this contract the *coordinator* specifies the attributes that he wants to modify and the characteristics of the *FusGeo* and *FusAero* that he wants to obtain after the modifications in order to analyze them.

Tables 4.1, 4.2 summarize the *Fuselage* attributes that the *coordinator* sends to *FusGeo* and *FusAero participants* respectively and the characteristics that he expects from them.

To design the process supporting the ECRs on the *FusGeo* and the *FusAero*, the *coordinator* can use Business Process Management (BPM) languages. He defines activities to send *FusGeo* attributes to be modified to the first *participant* and activities to receive the characteristics of the *FusGeo* when those modifications have taken place.

The *coordinator* defines another independent process model to support the ECR on the *FusAero* as well. Nevertheless, since there is a temporal constraint between changes on the *FusGeo* and changes on the *FusAero*, as we have seen in the previous chapter, the *coordinator* needs to keep this constraint in his mind. In other words, prior to validate the changes made on the *FusAero*, he knows that he should first validate the changes on the *FusGeo*.

In addition, there is a set of existing mapping functions between the Fuselage attributes that are summarized in Table 4.3.

4.3 Challenges

When the *coordinator* defines and manages the previous ECRs, he will face the following key challenges:

Properties of the *FusGeo* to be changed	***FusGeo* Characteristics desired by the *coordinator* after changes**
• Number of paxes in the front ($NPaxFront$) • Fuselage Height ($HFus$) • Number of Aisles ($Naisles$) • Fuselage Length ($LFus$)	• Wetted Area • Fuselage Mass

Table 4.1: Exchanged Fuselage Geo properties between coordinator and participants

Properties of the *FusAero* to be changed	***FusAero* Characteristics desired by the *coordinator* after changes**
• Re • $Aref$ • $LFus$ • Fuselage Width ($WFus$)	• Fuselage Fric Drag

Table 4.2: Exchanged *FuseAero* properties between *coordinator* and *participants*

Mapping Functions
• $Wfus = f(NpaxFront, Naisle)$ • $WettedArea = f(Wfuse, Hfus, Lfus)$

Table 4.3: Some Mapping Functions in the context of Fuselage design

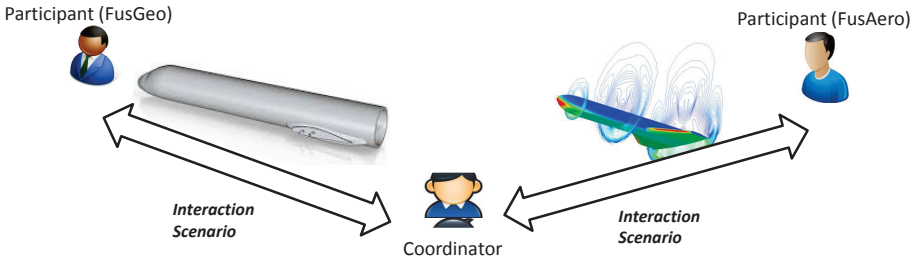


Figure 4.1: ECRs on FusGeo and FusAero

4.3.1 Manual creation of processes

Our example shows that the *coordinator* needs to model the processes that will support the interaction scenarios corresponding to ECRs on *FusGeo* and *FusAero* as depicted in Figure 4.1. This task can be difficult and painstaking in particular for a *coordinator* who typically has no experience with process modeling and the underlying concepts. The *coordinator* needs to learn a new vocabulary and he needs to change his focus from achieving the required configuration of the *FusGeo* and *FusAero* to how to define and deploy the processes that will help him in achieving these objectives. (i.e. when modeling the processes, the *coordinator* changes his focus from what he wants to achieve to how to achieve it).

4.3.2 Finding and applying the right rules

Yellin et al. [YS97] noticed that process activities are organized following a prestablished set of rules. Hence, the second task that the *coordinator* should perform consists in going through all enterprise rules of each *participant* in order to find out the right way in which he should organize the processes' activities of sending and receiving messages. When the *coordinator* finds the right rules, he needs to read and interpret their semantics. Although the (activity ordering) business rules are generally written in an understandable language[KEP00], their large number, however, could make this task error-prone.

4.3.3 Defining mediators

When the public process of each partner is designed by considering exclusively the partner's business rules, in this case the partners' processes could face mismatches when they are interconnected to create the cross-organizational process. For this reason, the *coordinator* needs to develop and deploy mediators that aim to resolve the possible mismatches and that avoid *participants* tailoring their processes. The task of developing mediators can be difficult to achieve since it requires some skills in programming.

4.3.4 Managing the dynamicity in the context of a DMN

All previous challenges that need to be handled by the *coordinator* are repetitive. Indeed, in the context of a DMN the *coordinator* needs to address and readdress them during the whole life-cycle of the collaboration because of the possible modifications that can occur on the collaboration configuration (for example the replacement of partners). More specifically, the *coordinator* needs to maintain processes when *participants* are replaced and redefine mediators when the order of activities is changed.

The aforementioned challenges put in evidence the need for a high-level modeling language to model contracts between *coordinators* and *participants* where the *coordinator* does not need to use low-level concepts to model the interaction scenarios. In addition it shows the need for an automatic mediation mechanism that resolves the possible mismatches that could appear when *coordinators*' and *participants*' processes communicate. Furthermore, it shows the need for a collaborative platform that interconnects all partners' processes so that the *coordinator* can specify temporal constraints between different ECRs. We elaborate on this example when detailing the contributions regarding each challenge.

4.3.5 Overview of the Proposed Approach

We adopt the creation of a high-level modeling framework approach [vdAP06, TKS14, HMS11a]. Specifically, as Figure 4.2 shows, the *coordinator* uses this framework to define a contract with *participants* that will participate in conducting change requests on the selected components of the product. This framework has the advantage to offer the *coordinator* a vocabulary that is close to his mindset. In addition, it has the advantage to be declarative and thus focuses on the objective to be achieved. Once the contract is defined, a procedure that generates a cross-organizational process that will support the partners in performing their collaborative work is executed. This procedure generates for each partner the public process that fits with his business rules stored in business rules repositories. It uses a set of operations (Projection, Reduction) in order to optimize the cross-organizational process generation time.

At run-time, the *mediation on-the-fly* is invoked when mismatches occur between the communicating processes and it aims at resolving these mismatches in an optimal time.

The reminder of this chapter is organized into five sections. Section 4.4 defines the main high-level concepts that are used by *coordinators* and *participants* to build collaboration contracts. Section 4.5 elaborates on the approach to generate the cross-organizational process from a contract specification to support *coordinators* and *partici-*

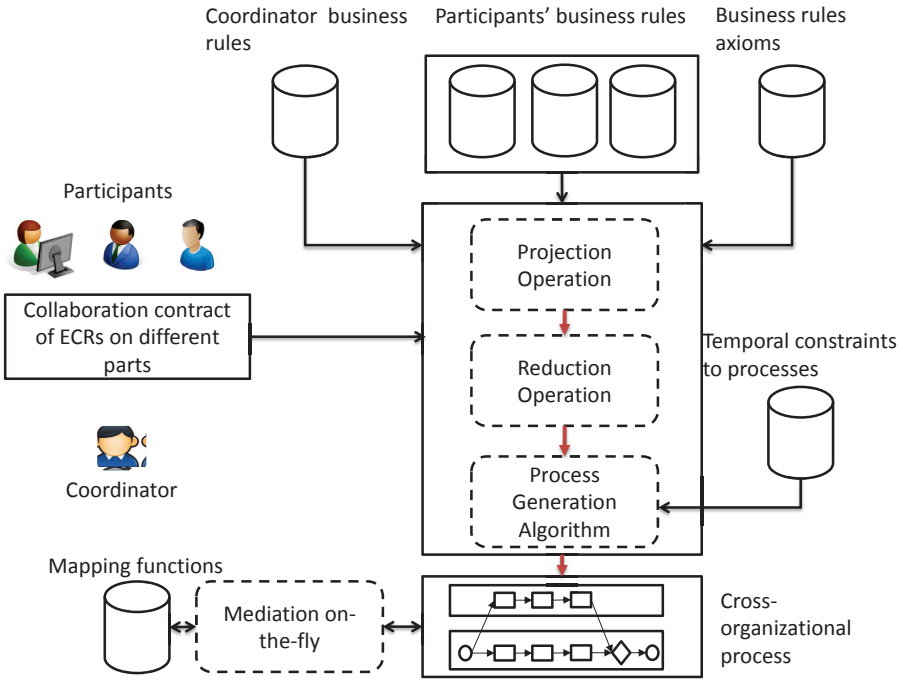


Figure 4.2: An overview of the proposed approach for ECRs in a collaborative environment

pants in fulfilling the contract obligations. Since mismatches could appear in the cross-organizational process, Section 4.6 elaborates on a novel mediation strategy to resolve the mismatches. Section 4.7 evaluates the framework. Finally, Section 4.8 concludes the chapter.

4.4 High-level Concepts for Contract Design

Contracts can be considered as a means to leverage the decoupling between stakeholders in a collaborative environment [ABP09b]. They record the benefits expected by each party from their interaction and the obligations that each party must be prepared to carry out in order to obtain these promised benefits. We define collaboration contracts between partners that formally specify what are the obligations and benefits of each partner. These contracts ensure that all sent data will be received, and dually, all expected data will be sent.

In this section we define the constructs to build contracts. (i) we classify partners depending on the role that they play in the ECM process in the context of a DMN. (ii) we formally define the product model that captures the end product breakdown structure and that is used by all partners in their work. (iii) Partners collaborate in order to reach agreements on the configuration of product components [LFB+12]. Accordingly,

we formally define these configurations and assign them to partners. This assignment determines each partner obligations and benefits regarding the collaboration.

4.4.1 Stakeholders

Stakeholders in a DMN can play two exclusive roles. Namely, *requester* and *supplier*. The requester asks the supplier to design a product component that should fulfill certain conditions. The supplier must consider these conditions when designing the component. Moreover he should provide the requester with a means to evaluate the designed component.

4.4.2 Product Breakdown Model

The product breakdown model is a tree-like structure with the end product as root and product components as nodes [vdARL01]. Standards exist that represent the product breakdown model, for example the product breakdown for support[2]. Nevertheless, in order to keep our approach abstract and independent from any implementation model, we give an abstract definition of the product model adapted from the definition given by [vdA99]. A product breakdown model is defined as a tuple $\langle root, O, N \rangle$ where:

- *root* represents the end product.
- O is a set of objects representing the components of the end product. Each element $o_i \in O \cup \{root\}$ has a set of attributes $\{att_1, att_2, ...att_n\}$.
- $N \subseteq O \times O$ defines the composition relationships between objects.

In the context of DMNs, the product breakdown model has two interesting properties:

- Its decomposition is sustainable. The product's components are determined during early phases of product design. The objective of the collaboration is to find the right properties of each component;
- The vocabulary used to build the product model constitutes a shared ontology upon which all partners agree. Consequently it provides an interesting starting point to define obligations and benefits regarding the collaboration in order to assign them to stakeholders.

[2]http://www.plcs-resources.org/plcs/dexlib/data/dex/

4.4.3 Product Component Configuration

The requester defines a set of constraints on the properties of the end product components. These constraints specify the expected configuration of each component. Given a product component o_i, there are two types of constraints associated to it:

- **Obligations** are parameterized constraints that the component supplier should consider before designing the component.
- **Characteristics** are parameterized constraints that the component requester should consider when evaluating the delivered component model.

Constraints associate variables to components' attributes. During the collaboration, constraints are instantiated by assigning values to their variables. The collaboration aims to determine the appropriate values of these variables. The combination of the supplier obligations and the requester benefits regarding a component defines the *product component configuration*.

Definition 8. *A product component configuration (PCC) of a component o_i is the couple $\langle Obligations, Characteristics \rangle$. It is defined by the grammar:*

- $PCC \rightarrow \langle A, A \rangle$
- $A \rightarrow A \wedge A \mid \neg A \mid constraint$
- $constraint \rightarrow R(o_i.att_j, x_{ij})$

Where R is an algebraic relationship ($<$, $=$, $>$ etc.) between the attribute $o_i.att_j$ and the variable x_{ij}.

This grammar specifies that both obligations and characteristics are conjunctions of constraints on component o_q attributes.

4.4.4 Agreements

The objective of the collaboration between a requester and a supplier is to reach an agreement on the configuration of a product component o_q (this is the reason that we call our framework Product Level Agreement in analogy with Service Level Agreement [OVC14]). The requester and the supplier will exchange data until reaching this agreement.

Agreements are constructs that capture the product components configurations (i.e. $\langle Obligations, Characteristics \rangle$) as well as the partners that collaborate to find the appropriate configuration instance.

Definition 9. *Formally, an agreement on the component o_q is the tuple $\langle Requester, Supplier_q, o_q, PCC_q \rangle$. The agreement declaratively binds the requester and the supplier through the PCC of the component o_q.*

Agreements under-specify the relationship between the requester and the suppliers. Such a specification defines the minimal coupling between stakeholders without additional constraints.

An agreement on a particular component o_q is achieved if and only if an array V of values exists whose assignment to constraints' variables x_{ij} specified in the PCC_q make the *requester* satisfied (i.e., V is a model of the logic formula PCC_q). Formally: $\exists V = [v_{i1}, v_{i2}..., v_{iN}], \bigwedge_{j=1}^{N} x_{ij} = v_{ij} \models Obligations \wedge Characteristics$

4.4.5 Complex Agreements

Agreements are *atomic* because they concern a single component of the end product. Atomic agreements constitute the building blocks of the contract. Nevertheless, we wanted to control the dynamicity of the DMN by increasing the flexibility of the contract in a DMN. We thus introduced *complex agreements*. Complex agreements are specifications on components composed of lower level components. More specifically, for a given atomic agreement, when the supplier quits the network he can be replaced by several suppliers. In this situation, each new supplier will design a sub-component of the component designed by the supplier who left. Using complex agreements, we can split a component agreement into agreements of its sub-components.

Definition 10. *A complex agreement on a component o_k is a tuple $\langle Requester, Suppliers_k, o_k, C_k, PCC_k \rangle$, where:*

- *$Suppliers_k$ is the set of suppliers of the sub-components of the component o_k*
- *C_k is the set of components composing o_k. It ensures: $C_k \subseteq O$ and $\forall o_i \in C_k, (o_k, o_i) \in N$*
- *To express that this agreement is composed of other atomic or complex agreements: $\forall o_i \in C_k, \exists suppliers_i \subset Suppliers_k$:*
  - *$\exists o_j \in O : (o_i, o_j) \in N \wedge |suppliers_i| > 1 \Rightarrow \langle Requester, suppliers_i, o_i, C_i, PCC_i \rangle$ is a complex agreement*
  - *$\forall o_j \in O : (o_i, o_j) \notin N \wedge |suppliers_i| = 1 \Rightarrow \langle Requester, suppliers_i, o_i, PCC_i \rangle$ is an atomic agreement*
- *$PCC_k = \bigwedge_{i=1}^{|C_k|} PCC_i$ when no relationship between sub-agreements is specified.*

- $PCC_k = \mathcal{TR}\,(PCC_i, PCC_j)$ *when a temporal relationship is specified between sub-agreements of sub-components* o_i *and* o_j. $\mathcal{TR}$ *are defined in Table 4.4.*

For instance we wanted to provide requesters and suppliers with a visual language. We thus modeled agreements with boxes as depicted in Figure 4.3. Complex agreements are represented as containers of other agreements.

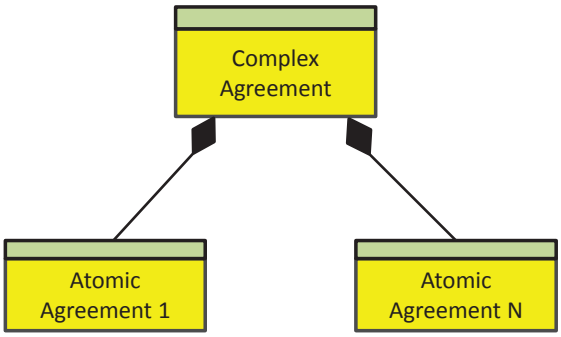


Figure 4.3: Graphical representation of atomic and complex agreements

4.4.6 Agreements Relationships

Modeling a contract by a set of agreements for each product component without interlinking them is possible. The agreements of this contract can be sought in parallel. However, sometimes there are constraints on the order in which agreements have to be achieved. For example in the design process of an aircraft [EME00], it is recommended that the design of the wing should take place before the design of the engine, and the design of the engine should take place before the design of the nacelle and so on. In the context of a collaborative design of these components, the previous constraints can be formulated as follows: (i) the agreement on the nacelle configuration between the *coordinator* and the nacelle designer can be achieved if the *coordinator* and the engine designer have already achieved an agreement on the engine configuration. (ii) Additionally, the agreement on the engine can only be achieved if the *coordinator* and the wing designer have already achieved an agreement on the wing.

In order to specify declaratively these temporal constraints, we enrich the framework to support temporal logic formulas [Fis11]. This is possible thanks to the logic-based formulation of product component constraints (PCCs). Although the need for formal models, including temporal logic, is clearly shown by several studies [TEvdHP11, SGN07], the knowledge required for their use remains a significant obstacle for their adoption particularly in an industrial context [EKK10]. Using patterns of temporal logic operators hides the complexity of the formalisms [YMH+06].

Pattern	Description
A LeadsTo B	*B* can be achieved only after that *A* has been achieved
A Response B	Whenever *A* is achieved, *B* has to be achieved
A Include B	*A* cannot be achieved without achieving *B*
A Mutual Exclusion B	If *A* is achieved *B* cannot be achieved and vice versa
N A	*A* cannot be achieved after N iterations

Table 4.4: Patterns of agreements relationships

To express the relationship between agreements, we use a set of meaningful patterns of temporal logic relationships formalized in [DAC98]. Table 4.4 summarizes these patterns. In this table *A* and *B* are two atomic or complex agreements. We believe that these patterns are sufficient to capture a large spectrum of situations that a contract modeler could face. Nevertheless, this set of patterns can be extended to capture other patterns of temporal constraints.

4.4.7 Contract on the final product

A contract is a (complex) agreement between the requester and all suppliers in the collaborative environment on the end product configuration.

Definition 11. *Formally, a contract is the tuple* $\langle Requester, Suppliers, root, O$,*End product configuration*$\rangle$.

The contract is designed following a bottom-up approach. The requester (that corresponds to the *coordinator* in the ECM standardized process), starts by defining agreements on the lowest level components. These agreements are then aggregated following the tree structure of the product breakdown until reaching the root that represents the end product.

4.4.8 Exception Handling

Using agreement-based contracts can lead to exceptions. These exceptions can arise during run-time and have to be addressed. One of the advantages of using agreement-based contracts is its natural way in handling exceptions. Indeed, in a contract, an exception occurs when a certain agreement cannot be reached. To handle exceptions, the requester specifies what should happen when an agreement for a particular component cannot be reached.

An agreement cannot be reached if its PCC formula is logically *unsatisfiable* (i.e., it has no model). That is, all value assignments to variables associated to component constraints do not satisfy the requester. This implies that its negation is *valid*.

Formally: $\forall V = [v_{i1}, v_{i2}..., v_{iN}], \bigwedge_{j=1}^{N} x_{ij} = v_{ij} \not\models Obligations \wedge Characteristics \Rightarrow \forall V = [v_{i1}, v_{i2}..., v_{iN}], \bigwedge_{j=1}^{N} x_{ij} = v_{ij} \models \neg(Obligations \wedge Characteristics)$

We use this property to define a new construct called *non-agreement* that is the contrary of an agreement. A non-agreement allows the requester to specify what should happen when its corresponding agreement cannot be reached.

Definition 12. *A non-agreement of an agreement α is a tuple of one attribute $\langle \neg\alpha.PCC \rangle$. Graphically, it is represented by a red box.*

4.4.9 Example

We apply the agreement based contract to a change request created by *coordinator1* on *FusGeo* under the responsibility of *participant1* and a change request on *FusAero* under the responsibility of *participant2*. Notice that we intentionally made a simplification by choosing only two product components to demonstrate the framework. Indeed, while it is not intended to portray a realistic ECR process, it is desirable not to camouflage the subject of this chapter by using a more complex example with too much technical detail.

To realize these changes an informal communication between the involved partners is no longer effective [OVC14]. Thus *coordinator1* creates an *Agreement* on the *FusGeo*. Then with the consent of *participant1*, *coordinator1* selects the properties of *FusGeo* that he will tailor during the collaboration and also selects the properties that he wants to receive from *participant1* when changes take place. *Coordinator1* does the same for the *FusAero* with the consent of *participant2*. Then he defines the temporal constraint of type **LeadTo** between *FusGeo* and *FusAero* as specified by [Sad12].

Figure 4.4 depicts a visual representation of the contract between *coordinator1* and *participant1* responsible of the *FuseGeo* and also with *participant2* responsible of the *FuseAero*. The Agreements defined in this contract are the following ones:

- $Ag_1 = \langle Coordiantor1, Participant1, FusGeo, PCC_1 = (R(NpaxFront, x_1) \wedge R(HFus, x_2) \wedge R(Naisles, x_3) \wedge R(Lfus, x_4), R(WettedArea, x_5)) \wedge R(FusMass, x_6) \rangle$
- $Ag_2 = \langle Coordiantor1, Participant2, FusAero, PCC_2 = (R(NpaxFront, y_1) \wedge R(HFus, y_2) \wedge R(Naisles, y_3) \wedge R(Lfus, y_4)), (R(WettedArea, y_5) \wedge R(FusMass, y_6)) \rangle$
- $Ag_3 = \langle Coordiantor1, \{Participant1, Participant2\}, Fuselage, PCC_1 \textbf{LeadsTo} PCC_2 \rangle$

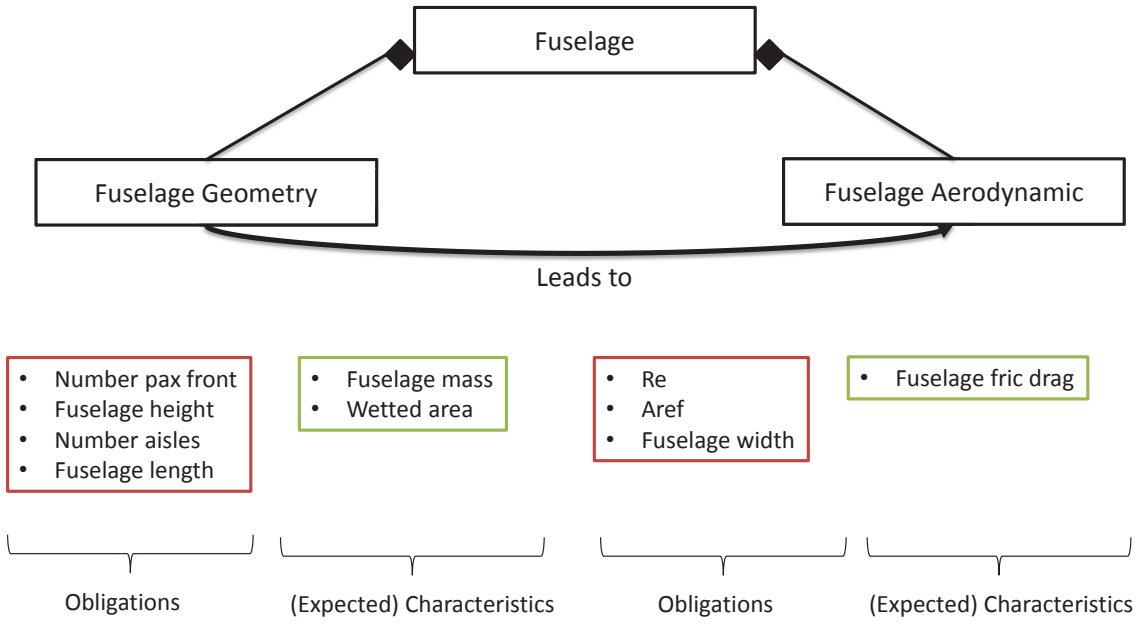


Figure 4.4: Contract on ECRs on the geometry and the aerodynamic of the Fuselage

At this stage, the contract between *coordiantor1* and *participant1*, *participant2* is built. The next step consists in generating the cross-organizational process that will support the data exchange between these partners in order to achieve the agreements defined in this contract while upholding their temporal constraints.

4.4.10 Summary

In this section we presented the formal framework that defines the concepts to help *coordinators* in an ECM process build contracts with *participants* in order to achieve an ECR. We also presented an example that shows that with this framework, *coordinators* focus on the objective that they want to achieve through the ECR rather than on how to achieve it by modeling the processes.

In the next section we detail how to automatically generate the underlying processes that will support *coordinators* and *participants* in detailing the ECR and satisfy what has been specified in the contract.

4.5 Contract Enactment

4.5.1 Motivations on Using Business Rules

Agreements are high-level constructs understandable by *coordinators/participants* involved in the DMN. When the contract design is completed, thanks to the actionable nature

of agreements and their formalization, the contract can be projected into the execution platform by generating automatically the cross-organizational process (COP) model. Indeed, when analyzing how the collaboration is conducted to handle a change request on a particular component, we can notice that the (COP) follows the same steps: (i) the requester sends an obligation instance to the supplier, (ii) the supplier receives the instance, (iii) the supplier carries out an internal process to study the feasibility of the design, (iv) the supplier replies the requester with a characteristic instance, (v) the requester analyzes the results, if he is satisfied then the agreement is reached, otherwise he assigns a new instance to the obligation and repeats the process. In systems engineering, this process is called the agreement process [HFoSE11].

A stakeholder could have concurrent engineering capabilities. These capabilities allow him to perform data exchange activities in parallel, while another stakeholder might be in lack of these capabilities and thus is obliged to run his activities in sequence. This difference is confirmed by the fact that *often there exist rules constraining the order in which messages may be sent* [YS97]. Indeed, each stakeholder has his own rules to which his process should be compliant [KRM+12]. Generally these rules are captured in terms of business rules. Business rules are statements that define or constrain some aspects of the business [Grooo]. If business rules are defined to their full extent, they will govern the entire execution of the processes [CM04]. In addition, business rules have the advantage to be very close to how designers think and talk [EP98].

From these observations, it is better to rely on business rules and to ask *coordinators* to model the collaboration contract using our framework and then generate the COP, rather than asking them to model the COP using a low level language such as BPM languages.

Since the contract model with the complete set of business rules constitute a comprehensive base of all information related to the collaboration, it is possible to use them to automatically generate the executable COP model.

4.5.2 Operations on Business Rules

Collaborative product design is concerned with a subset of business rules. Indeed, from the classification of the workflow patterns defined by [vdAtHKB03], we could determine what are the possible rules a stakeholder can define regarding the organization of data exchange activities in a collaborative environment. A subset of these constraints. summarized in Table 4.5.

These constraints have an associated set of axioms given in Table 4.6, where x_1, x_2, x_3

Constraint Type	Description
x_1 *Should Follow* x_2 SF(x_1,x_2)	Messages x_1 and x_2 should be sent/received in sequence, starting by x_2
x_1 *Should Precede* x_2 SP(x_1,x_2)	Messages x_1 and x_2 should be sent/received in sequence, starting by x_1
x_1 *Independent* x_2 I(x_1,x_2)	Messages x_1 and x_2 could be in sent/received in parallel
x_1 *Packaged With* x_2 P(x_1,x_2)	Message x_1 should be delivered with x_2

Table 4.5: Rule types on the order of messages

Axiom	Formalization
A1 (Transitivity of SP)	$\mathrm{SP}(x_1, x_2) \wedge \mathrm{SP}(x_2, x_3) \rightarrow SP(x_1, x_3)$
A2 (Transitivity of SF)	$\mathrm{SF}(x_1, x_2) \wedge \mathrm{SF}(x_2, x_3) \rightarrow SF(x_1, x_3)$
A3 (Transitivity of I)	$\mathrm{I}(x_1, x_2) \wedge \mathrm{I}(x_2, x_3) \rightarrow I(x_1, x_3)$
A4 (Transitivity of P)	$\mathrm{P}(x_1, x_2) \wedge \mathrm{P}(x_2, x_3) \rightarrow P(x_1, x_3)$
A5	$(\mathrm{SP}(x_1, x_2) \vee SF(x_1, x_2) \vee \mathrm{P}(x_1, x_2)) \wedge I(x_2, x_3) \rightarrow I(x_1, x_3)$
A6	$\mathrm{SP}(x_1, x_2) \wedge P(x_2, x_3) \rightarrow \mathrm{SP}(x_1, x_3)$
A7	$\mathrm{SF}(x_1, x_2) \wedge P(x_2, x_3) \rightarrow \mathrm{SF}(x_1, x_3)$

Table 4.6: Axioms related to rules on the order of messages

being messages.

An informal example of a data exchange activities rule that specifies that all send activities should be in sequence is: "*Since there is only one license of the simulation software, we can deliver one instance of a message per time*". Formally for the design of a given product component o_q: $\forall x_i, x_j \in Att(o_q) : SF(x_i, x_j) \vee SF(x_j, x_i)$. This rule captures the fact that the partner can only send one message instance per time and thus all messages will be delivered in sequence.

Partners' business rules reside in repositories $\{R_1, ..., R_n\}$. Although we could directly use the rules in these R_i to generate the COP model as depicted in Figure 4.2, it is better to *reduce* each R_i and then generate the COP model. In the following, we detail the steps for generating the COP model.

Each R_i could contain a large number of rules and thus traversing and analyzing all rules to generate the COP model could be time consuming because the generation algorithm should evaluate each rule. To avoid this, we can reduce the search space by performing two actions:

1. (**Projection**) Projecting R_i rules on a subset of attributes belonging to the obligations and the characteristics defined in the agreement. Accordingly, we keep only the rules involving these attributes. For this reason we define the Π operator on business rules repositories.

Definition 13. *The projection operator on the repository R_i gives the repository $R_i^{'}$: $\Pi_{Att \subseteq 2^{\{a_1,...,a_n\}}}(R_i) = R_i^{'}$ such that the rules in $R^{'}$ are defined by the predicates containing exclusively the attributes in Att.*

Lemma 1. *The workflow model involving messages $\in$ Att generated using the rules belonging to the repository R is equivalent to the workflow model generated using the rules belonging to the repository $R^{'}$.*

Proof. Suppose that there is no such equivalence. Then, there is at least a rule $r \in R - R'$ that impacts the order of attributes in Att. However, since any rule constrains only the attributes involved in its definition, then r cannot affect the order of attributes in Att. Thus, the first assumption leads to a contradiction. □

2. (**Reduction**) Eliminating rules that can be inferred from other rules in the repository using the axioms formalized in Table 4.6. The reason is that these inferable rules do not add any detail to the COP to be generated. Algorithm 1 performs projection and the reduction actions.

Algorithm 1 Repository Reduction

Require: Repository R, the set of Axioms AX, a set of involved attributes $Att \subseteq 2^{\{a_1,...,a_n\}}$
Ensure: A reduced Business rules repository
1: $\Pi_{Att}(R)$
2: **while** $\exists r \in R$ such that $\neg Marked(r)$ **do**
3: $r \leftarrow$ Pick a non-marked rule from the R
4: **if** $\exists \Sigma \subseteq R - \{r\} \wedge \exists ax \in AX$ such that $ax, \Sigma \models r$ **then**
5: Remove r from R
6: **else**
7: Mark r as has been visited
8: **end if**
9: **end while**

The algorithm visits rules residing in the repository R. Each time it finds that a rule can be inferred from a subset of rules following the axioms defined in Table 4.6 (line 4 of the algorithm), it removes it (line 5).

Once the repository has been reduced, the workflow generation algorithm uses the remaining rules to decide in which order the send/receive activities should be organized in the COP model.

Lemma 2. *The COP model generated by the original repository of rules R is equivalent to the COP model generated by the reduced repository $R^{'}$.*

Proof. The proof is conducted on each axiom of Table 4.6.

For axiom A_1 in Table 4.6, the COP fragment generated using the rules $\{SP(x_1, x_2), SP(x_2, x_3), SP(x_1, x_3)\}$ is equivalent to the COP fragment generated using the rules $\{SP(x_1, x_2), SP(x_2, x_3)\}$ as depicted in Figure 4.5.

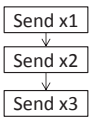


Figure 4.5: Equivalence between the generated COP fragments

The proof continues by making the same observation on the remaining axioms in Table 4.6.

□

Theorem 1. *The COP model generated using the rules of the repository R is equivalent to the COP generated from $R^{'}$ after applying the projection and the reduction operators on the repository R.*

Proof. From Lemma 1 and Lemma 2. □

Definition 14. *A repository R of business rules is defined to its full extent, if and only if every situation in the business process has a rule associated to it in R [EP98].*

Formally: $\forall x_i \in COP : |\Pi_{\{x_i\}}(R)| > 0$.

Lemma 3. *If a repository of rules R is defined to its full extent, then the reduced repository of R, that is $R^{'}$, is also defined to its full extent.*

Formally: $\forall x_i \in COP, |\Pi_{\{x_i\}}(R)| > 0 \Rightarrow |\Pi_{\{x_i\}}(R^{'})| > 0$

Proof. The proof is conducted on the type of axioms.

For the axiom A_1 in Table 4.6, we consider that $\{SP(x_1, x_2)$,SP$(x_2, x_3), SP(x_1, x_3)\} \subseteq R \Rightarrow \{SP(x_1, x_2)$,SP$(x_2, x_3)\} \subseteq R^{'} \wedge SP(x_1, x_3) \notin R^{'}$. Accordingly, $|\Pi_{\{x_1\}}(R)|, |\Pi_{\{x_2\}}(R)|, |\Pi_{\{x_3\}}(R)| > 0 \Rightarrow |\Pi_{\{x_1\}}(R^{'})|, |\Pi_{\{x_2\}}(R^{'})|, |\Pi_{\{x_3\}}(R^{'})| > 0$

The proof is the same for the remaining axioms in Table 4.6. □

Definition 15. *A process model is well formed iff each lane of the COP model has the three properties:*

1. *There is at least one start event*
2. *There is at least one end event*
3. *Every object is on a path from a start event to an end event.*

The corresponding formal definition of a well formed process diagram can be found in Definition 2 of [ODtHvdA06].

The workflow generation algorithm (Algorithm 2) generates for each agreement specified in the contract the corresponding process fragment that supports data exchanges for achieving agreement on the associated component. This algorithm uses the mapping rules associated to the agreement relationships patterns. We give the mapping rules to the three main patterns **Leads To**, **Response** and **Includes**. These mapping rules are depicted in Figures 4.6 4.7 4.8 respectively.

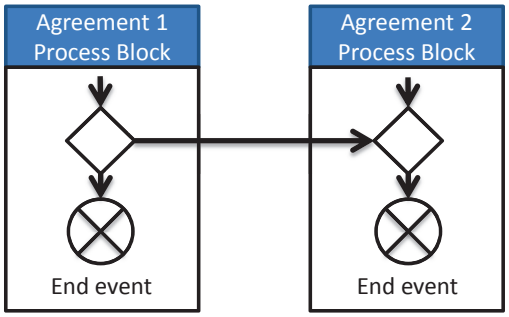


Figure 4.6: **LeadsTo** constraint mapping to XPDL

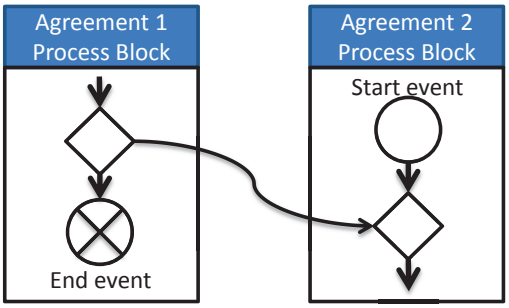


Figure 4.7: **Response** constraint mapping to XPDL

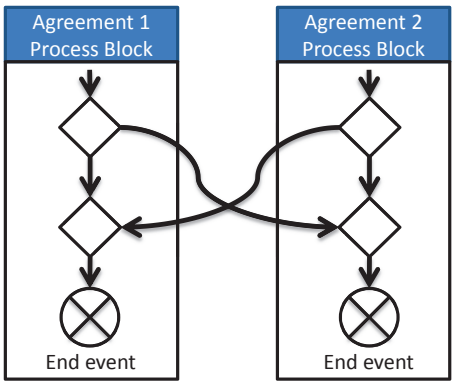


Figure 4.8: **Includes** constraint mapping to XPDL

The following theorem shows that a collaboration contract associated with a full extent repository of business rules generates a well-defined COP model:

Theorem 2. *In a collaboration contract C, if each rules repository R_i associated to partners is defined to its full extent, then the generated COP will be well-structured.*

Proof. From Theorem 1, we can reduce the repositories R_i to produce R_i'. The proof is performed by induction on the well-structuredness of each lane corresponding to every partner:

Algorithm 2 Workflow Generation

Require: The Contract C
Require: Stakeholders business rules repositories$R_1, ..., R_n$
Require: Product Model PM
Ensure: The cross-organizational process
1: Call Reduction Algorithm for BR_1,..., BR_n
2: Traverse the contract in a top-down way
3: **for all** Agreement $Ag \in C$ **do**
4: Use mappings in Figures 4.6, 4.7, 4.8 to generate the interconnections between Ag and other agreements
5: **if** Ag is atomic **then**
6: For the obligations specified in Ag, generate the send/receive activities for the requester and the supplier respectively
7: Use $R_{Ag.supplier}$ to define the order of activities for the supplier and the $R_{Ag.requester}$ for the requester
8: Repeat steps 6-7 for the agreement Ag characteristics
9: **end if**
10: **end for**

- If a partner P_i is involved in a single agreement, then the corresponding lane will have a start event and an end event and since its R_i is defined to its full extent then every object is on a path from the start event to the end event.
- If a partner P_i is involved in more than one agreement, then if there is a temporal constraint between agreements, the generation algorithm will interconnect the generated processes corresponding to each agreement. Otherwise the generation algorithm will ensure the well structuredness by generating *parallel splits* between processes corresponding to each agreement.

□

Theorem 3. *A collaboration contract C is honored (all obligations have been fulfilled) iff the COP has reached an end event corresponding to each agreement.*

Proof. We use equivalence proof method:

- $\rightarrow$ Suppose that there is a process fragment in the COP that has not reached its end event. This implies that at least an agreement has not been reached between two partners. This implies that the contract is not honored yet.
- $\leftarrow$ Suppose that the contract is not honored yet. This implies that there is at least one agreement which has not been reached yet and it is in progress. This implies that its end event has not been triggered yet.

□

Once the different process fragments corresponding to the atomic and complex agreements in the collaboration contract have been generated, the COP model can be deployed into the workflow engine. The workflow engine will run the workflow model to distribute the exchanged messages to the partners. Nevertheless, since a partner could have a different view on the component being designed [Rayo6], we need to allow him to keep this view (and thus his process) and ensure the data mediation and transformation transparently.

4.5.3 Example

We continue with the example of section 4.4.9. Once the contract has been specified in terms of *Agreements*, the workflow generation module (see Figure 4.2) generates the workflow that supports the achievement of the specified agreements. We made the assumption that all partners have a business rules repository and to simplify the example the rules specify that all activities are sequential.

With the mapping rule between the temporal logic pattern **LeadsTo** and workflow constructs depicted in Figure 4.6, the generation module will generate the workflow depicted in Figure 4.9 for the contract in Figure 4.4.

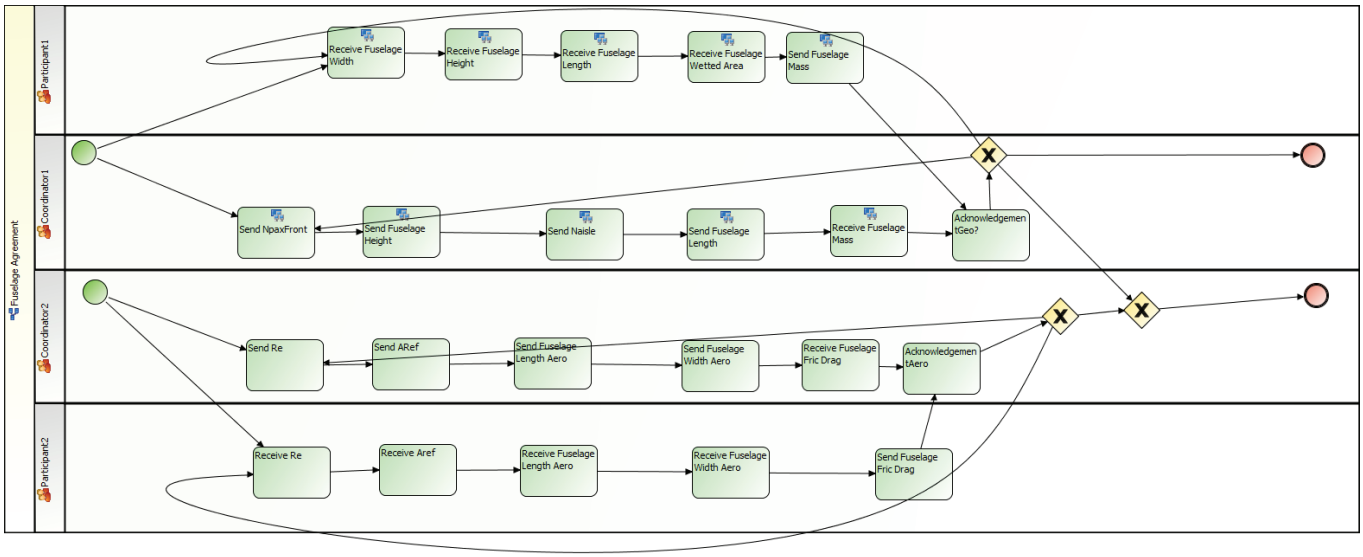


Figure 4.9: Generated process of the contract

Once this workflow is generated, it will be deployed into a workflow engine and will dispatch the messages for partners. Nevertheless, there is a mismatch in this workflow. For instance *participant1* expects to receive the *WFus* while *coordinator1* does not send this information. Hence a mediation solution should be used to resolve this heterogeneity.

4.5.4 Summary

In this section we presented the approach to generate the cross-organizational process that will support *coordinators* and *participants* in achieving an ECR from a contract specification while upholding the constraints set by business rules of each partner. To increase the efficiency of our process generation algorithm we defined a set of operations on business rules that help in reducing the number of business rules that the workflow generation algorithm should analyze. We formally proved that applying these operations on business rules will not impact the order of the activities of the process to be generated; it will just reduce the number of rules to be traversed.

When interconnecting the processes generated some mismatches could appear due to the difference of public process activities between partners. In the next section, we propose a novel mediation approach that efficiently resolves mismatches when the generated coordinators' and participants' processes will exchange data.

4.6 On-the-fly Mediation

The generated COP will support the interaction scenario between the *coordinator* and the *participant* working on an ECR. Since there is a possibility that some heterogeneity could exist between their processes, a mediation solution is required to resolve the possible mismatches. This mediation relies on a set of mapping functions as given in Table 4.3 so that when all messages required to compute an expected message are sent, the mediator applies the appropriate mapping function to generate the expected message. The mediation approach focuses only on behavioral aspects. Thus it considers that the names of the messages exchanged are standardized and the semantic compatibility is guaranteed. [BSBM04] has made the same assumption. This assumption is possible since much research has been done on adaptation where messages are enriched semantically through ontologies [BSBM04].

Previous works focused on defining the adaptation patterns of the possible mismatches that can occur when two processes communicate together. These patterns aim at helping the designer at design time [KNBSP14, BCG+05]. It is also possible to automate the discovery of the right pattern to apply when the mediator is to be generated automatically [EBMN13].

In our approach, we follow another path. We consider that for a particular domain, we can identify the messages that can be exchanged and the mapping functions between these messages if they exist. Then we formalize these mapping functions as marked au-

tomata so that we can develop an efficient on the fly mediation algorithm. The assumption of the identification of the messages for a particular domain and their relationship is feasible by making the analogy with the identification of the concepts used within a particular domain known to be an island of users [PTDL07]. This acceptable assumption makes our approach independent from the mismatch patterns identified previously since we are not sure of their completeness. Furthermore it helps us in develop a more efficient mediation algorithm adaptable for DMNs.

In the remaining of this section, in addition to specifying the generated public processes in a formal language, we introduce a new way to formalize the mapping functions in terms of marked automata so that we can find the right mapping function to apply to generate the expected message in a polynomial time.

4.6.1 Automaton Formal Definition

Conventional mediation approaches define mapping functions at design-time either (i) semi-automatically by involving a human in the loop or (ii) automatically by relying on the domain ontology and the mappings between its concepts [BCG+05].

During the run-time phase, automating conventional mediation algorithms so that they can find the right mapping rule to apply will lead to state space explosion. Indeed, consider that the workflow generator has generated a COP for the requester and the supplier as depicted in Figure 4.10. In this case, the conventional mediator algorithm should perform at most α iterations in order to determine what mapping function to use to generate the message y_1 from the sent messages $\{x_1..., x_n\}$ where:

$$\alpha = \sum_{k=1}^{N} \begin{pmatrix} N \\ k \end{pmatrix} = \sum_{k=1}^{N} \frac{N!}{k!(N-k)!} \tag{4.1}$$

This worst case is reached when the data required to deliver the message y_1 is dispatched in all messages sent: $\{x_1, ..., x_n\}$, thus to determine the mapping function to use to generate y_1, the mediator should verify if there is a mapping function between (x_1, y_1), then (x_2, y_1), then $(\{x_1, x_2\}, y_1)$ etc. until $(\{x_1, x_2...x_n\}, y_1)$. In this case a mapping function is found and y_1 can be generated.

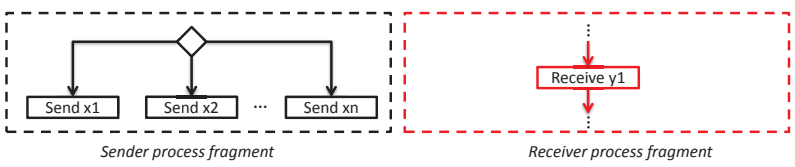


Figure 4.10: Generated fragments

Conventional mediation algorithms cause state space explosion because they search for the mapping function each time a message x_i arrives. The search consists in checking all combinations of messages that arrived to test whether a mapping function exists between a certain combination and the expected message y_1. This procedure requires α iterations. The algorithm is obliged to check all possible combinations of messages in order to ensure the *safety* of the mediation strategy.

To overcome the issue of state space explosion, instead of verifying whether the expected message y_1 could be derived from messages that already arrived $\{x_1, ...x_l\}$ ($l < N$), we split this verification into two steps (we call it a two-step approach):

1. determine whether the newly arrived message (x_l) can trigger the computation of the expected message y_1

2. if yes, then determine the mapping function. Otherwise wait for the next messages $\{x_{l+1}, ...x_n\}$

To implement the first step, we use an automaton structure (marked automaton) to represent the fact that there exists a mapping function between $\{x_1, ...x_n\}$ and y_1 as depicted in Figure 4.11.

Definition 16 (Marked Automaton). *$A^{\Phi} = [Q, C, \delta, q_0, \Phi]$ is a marked automaton iff Q is a non-empty finite set of states, apart from q_0 all states are ending state. C is a set of labels, $\delta \subseteq Q \times C \times Q$ is a transition relation such that every ending state is reachable from q_0 and from any other finite state via a direct transition $\in \delta$.*

This automaton has a starting state s_0 and the remaining states are specified to be ending states to show that messages can arrive in a random order. Additionally, the states of this automaton can be marked in order to differentiate between the messages that already arrived and the messages that have not arrived yet. In the example of Figure 4.10 and the marked automaton corresponding to the mapping function in Figure 4.11, when the message x_1 arrives:

1. the automaton performs a transition from the state s_0 to the state s_1

2. the state s_1 is marked to indicate that x_1 has arrived

3. since there is at least one ending state that has not been marked (in this case: $\{s_2, ..., s_n\}$) we need to wait for the next message because at this stage, we are sure that there is no mapping function between x_1 only and y_1

4. when all states have been marked, which indicates that all required messages to compute y_1 have arrived, we execute the step 2 that determines the derivation rule between $\{x_1, ...x_n\}$ and y_1.

When relying on the two-step approach, our mediation algorithm no longer needs to verify all combinations of available messages to determine the mapping function each time a new message arrives. In this case, the automaton progresses until all final states have been marked. When all final states have been marked, our algorithm can start searching for the mapping function (step 2) because this time we are sure that it will find it.

Numerically, we decrease the complexity of the mediation algorithm from α (see equation 1) to $N + 1$ operations of automaton progression.

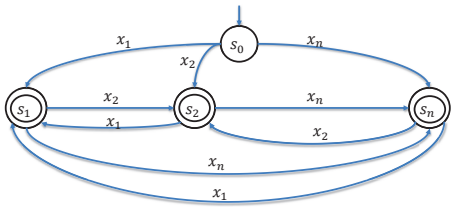


Figure 4.11: Automaton of the relationship between y_1 and $x_1...x_n$

Algorithm 3 details the main steps of our mediation approach.

Algorithm 3 Mediation on the Fly

Require: *PM*: Product Model, *RPP*:Receiver Public Process.
Ensure: the correct message required by the receiver.

```
1: Buffer ← ∅
2: while (SentMessage ≠ ACK ) ∧ (SentMessage ≠ Exception) do
3:    SentMessage ← capture the message sent by the sender
4:    Buffer.insert(SentMessage)
5:    Make the automata progress using the SentMessage
6:    Mark the state
7:    if all states have been marked then
8:       i ← determine the function to calculate the expected object data from the received messages
9:       MessageForReceiver ← f_i(Buffer)
10:      Send(MessageForReceiver)
11:      Buffer ← Buffer - RPP.ExpectedMessage
12:   end if
13: end while
```

Basically, this algorithm starts by verifying that the received message is neither an *Acknowledgment* to indicate that the requester is satisfied and the agreement has been reached, nor an exception. In this case the message received makes the automaton

closer to its ending and marks the associated state (lines 5 and 6). Then if all states have been marked (line 7: the end of step 1 of the two-steps approach), the algorithm determines the mapping function (line 8: step 2 of the two-steps approach).

Algorithms 4, 5 detail the behavior of the marked automaton when a message arrives. More specifically, the automaton is continously listening to messages that arrive. When a message arrives corresponds to its active state (line 1 of Algorithm 4) it notifies the observer. The observer can be another state or the mediator to notify it that all messages required to apply a mapping function are available.

Algorithm 4 Automaton State Transition: `processBufferEventReception`

Require: Buffer.lastInsertedEntry, Automaton A
Ensure: Mark the corresponding state
1: **if** `listening`($A.listeningState$,lastInsertedEntry) **then**
2: $s_i \leftarrow$ `notifyAppropriateObserver`(s_0.observers,lastInsertedEntry)
3: s_i.`processStateEventReception`(lastInsertedEntry)
4: **end if**

Algorithm 5 Automaton State Event Reception: `processStateEventReception`

Require: notification, $s_i \in A$
Ensure: Mark the corresponding state
1: mark(s_i)
2: **if** A.completed **then**
3: `notifyMediator`()
4: **else**
5: A.listeningState $\leftarrow s_i$
6: **for all** $s_j \in A \wedge \neg s_j.marked$ **do**
7: s_j.`setObservable`(s_i)
8: **end for**
9: **end if**

4.6.2 Application Example

We carry on with the example of section 4.5.3. Since *coordinator1* sends messages not expected by *participant1*, the mediation on-the-fly should intervene in order to resolve this mismatch. The *mediation on-the-fly* counts on the predefined mapping function between $WFus$ and $NpaxFront, Naisles$ defined in Table 4.3. Following our mediation approach, this mapping function will be formalized using a marked automaton as depicted in Figure 4.12.

1. Following the process of Figure 4.9, when *coordinator1* sends the message $NpaxFront$, this message will move the automaton of Figure 4.12 (a) from state s_0 to state s_1

and marks the state s_1. Since s_2 has not been marked yet, the mediator will not be notified.

2. When *coordinator1* sends the message *HFus* it will have no impact on the automaton and thus the mediator remains idle for this message. Here is where our optimization on searching mapping function occurs. Other approaches take into account this message and search for the mapping function even though this message is not involved in the computation the expected message.

3. When *coordinator1* sends the message *Naisles*, this message will move the automaton of Figure 4.12 (a) from state s_1 to state s_2 and marks the state s_2. Since all final states have been marked (instruction 7 of Algorithm 3, the expected message by the receiver (*WFus*) can be computed by the mediator.

For this example the on-the-fly mediation has performed only 4 iterations on the mapping functions base to find the right mapping function to apply, instead of 7 iterations in the case for the naive approach. When the mediator finds the right mapping function, it performs the computation of *WFus*, and forwards the result to *participant1* process. The latter receives the data and can carry on its execution.

4.6.3 Messages Computed Using a Composition of Mapping Functions

The automaton introduced previously allows the automatic detection of the possibility to compute the expected message from the messages that already arrived. However this is practical only when the messages that arrive are directly involved in the computation of the expected message (i.e. there exists a function that has as input the messages that arrived and as output the expected message). There are cases where the messages that arrive are involved in the computation of the expected message but only by using intermediate parameters as illustrated by the mapping functions that compute respectively *WFus* and *WettedArea* depicted in Figure 4.12 (b). In this figure, the expected message *WettedArea* in the workflow of Figure 4.9 cannot be computed directly from the messages that will be received. We first need to compute the intermediate parameter *WFus*, and then use this parameter to compute the expected message *WettedArea*. In the remaining of this section, we extend our mediation algorithm to address this issue for the general case.

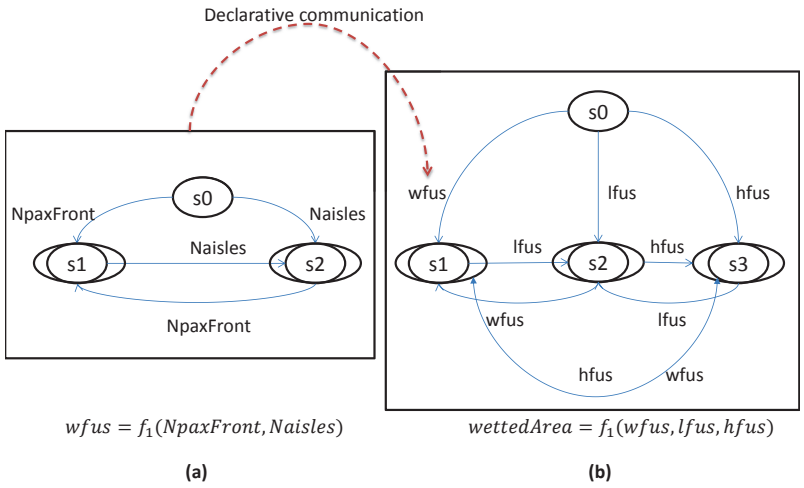


Figure 4.12: Example of composition of functions

4.6.3.1 Declarative Automata Communication

To address the issue of computing expected messages indirectly from the messages that arrived, we define a declarative communication means between automata in the repository of automata rules. Using this communication means, our mediation algorithm can detect that the messages that arrived can be used to compute the expected message even though they are not directly involved in the function that compute the expected message.

Definition 17. *A communication channel exists between two automata A_1 and A_2, if the output message of A_1 is involved in A_2.*

Figure 4.12 illustrates an example of a communication channel using $WFus$ between both automata. Thus, whenever $WFus$ is computed, the second automaton can perform the transition.

When the remaining messages arrive, Algorithm 6 performs a recursive traversal of the involved automata in order to compute the value for each involved parameter and finally compute the final value of the expected message.

To demonstrate the generality of our approach, we prove the following theorem:

Theorem 4. *Whatever the number of functions separating the computation of the expected message from the messages that arrive, our algorithm is able to compute the expected message.*

Proof. (By induction) First, suppose that the expected message y could be computed using the function $f(x_1, x_2, ..., x_m)_{m>1}$ (f could be the function id) and the messages $x_1, x_2, ...x_m$ have arrived. In this case, the automaton corresponding to f will trigger the computation of y.

Algorithm 6 Recursive traversal of automata repository to find all transformation function parameters `determineInputs`

Require: Automaton A
Ensure: Function f result
1: **for all** State $s \in A$ **do**
2: **if** $s.associatedParameter \in Buffer$ **then**
3: Add ($s.associatedParameter, f.inputs$)
4: **else**
5: Find automaton $B : B.output = s.associatedParamter \wedge B.s_0.marked$
6: Add (`determineInputs`(B),$f.inputs$)
7: **end if**
8: **end for**
9: Call f
10: return result

Now, suppose that $y = f_1 \circ f_2 \circ ... \circ f_n(x_1, x_2, ..., x_m)$ can be computed by Algorithm 6 and we prove that $z = f_1 \circ f_2 \circ ... \circ f_n \circ f_{n+1}(x_1, x_2, ..., x_m)$ can be computed by Algorithm 6 as well.

Since $y = f_1 \circ f_2 \circ ... \circ f_n(x_1, x_2, ..., x_m)$, then $z = f_{n+1}(y)$. In this case, Algorithm 6 will trigger the computation of y then the result of this computation will trigger the automaton corresponding to $z = f_{n+1}(y)$ function through their communication channel and calls the function $f_{n+1}(y)$ to compute z. □

Another important point to consider is that an expected message could be computed using different automata (i.e., multiple mapping functions exist to compute this message). In this case, the notion of declarative communication channel is helpful. The expected message is computed whenever all necessary data arrive for any mapping function.

4.6.4 Ensuring Correct Correlation of Exchanged Messages

Our solution to interoperability problem in DMNs was to propose a single mediation algorithm that is independent from the communicating partners. However, in a collaboration environment that involves multiple couples of partners, our algorithm may face ambiguity when delivering expected messages. Indeed, several partners may send different values of the same message targeting different other partners. In this case, our mediation algorithm should be able to differentiate between the incoming messages and deliver the computed message to the right partner.

To achieve a correct correlation of exchanged messages between partners, we extend the definition of two concepts: message and rule automaton.

Definition 18. *A message is identified by a triplet:* $\langle Name, Supplier, Requester \rangle$.

Definition 19. *An extended marked automaton is a marked automaton where:*

- *A transition is identified by the triplet:* $\langle Name, Supplier, Requester \rangle$. *A transition is triggered if and only if the identifier of the message that arrives corresponds to the transition identifier*
- *A state marking is defined by the couple:* $\langle Supplier, Requester \rangle$. *Thus a state could be associated to a set of marking where each mark is defined by the previous couple*
- *A communication channel is also defined by the triplet:* $\langle Name, Supplier, Requester \rangle$ *and is triggered whenever all equally identified states have been marked.*

Each time a new couple of partners is added to the network, all rules automata are extended by the corresponding transitions, markings and communication channels.

Now the mediation algorithm uses the extended marked automata of mapping functions in order to ensure correct correlation of the exchanged messages. We can assert that there will be no mediation ambiguity during the collaboration.

Lemma 4. *When two messages having the same name arrive to the mediator from two different partners, their derivation will be delivered to the right partners.*

Proof. We conduct the proof by contradiction:

Without loss of generality, suppose that the messages sent by P_1, P_2 to P_3, P_4 respectively, will trigger the same automaton to deliver the derived values to the right partners. When an ambiguity in delivering the right message to P_3, P_4 occurs, it might cause one of the following consequences:

1. Neither P_3 nor P_4 will receive the expected message
2. A partner will receive two message while the other will receive nothing
3. Derived messages will be altered and each partner will receive the wrong one.

Suppose that P_1 sends the message $\langle M, P_1, P_3 \rangle$ and P_2 sends the message $\langle M, P_2, P_4 \rangle$. Suppose also that the extended automaton of the function f is defined as illustrated in Figure 4.11. In the following, we prove that with the extended marked automaton definition, none of these 3 cases would occur:

1. When the automaton of the function f receives the message $\langle M, P_1, P_3 \rangle$, this will trigger the transition $\langle M, P_1, P_3 \rangle$ and thus the state s_1 will be marked. This marking will trigger the execution of the function and the delivery of the the message $\langle f(M), P_1, P_3 \rangle$ to P_3.

2. Since the transition has been triggered, then it cannot be triggered again since another iteration has not been executed yet

3. Derived message contains the identifier of the receiver, thus it is not possible to alter the computed messages.

□

4.6.5 General Properties of the Mediation Approach

In this section we show some properties of our on-the-fly mediation.

Definition 20 (Processes Similarity and Complementarity). *1. Two processes are said to be **equivalent** if they execute exactly the same strings of actions [Fokoo].*

2. *Two processes are said to be **similar** if they execute the same string of actions but without any specific order.*

3. *Two processes are said to be **semi-similar** if they execute strings of actions without any specific order and all data produced by one string can be derived from the other string.*

4. *Two processes are said to be **complementary** if the complement of one string is semi-similar to another string.*

Figure 4.13 illustrates each definition through an example.

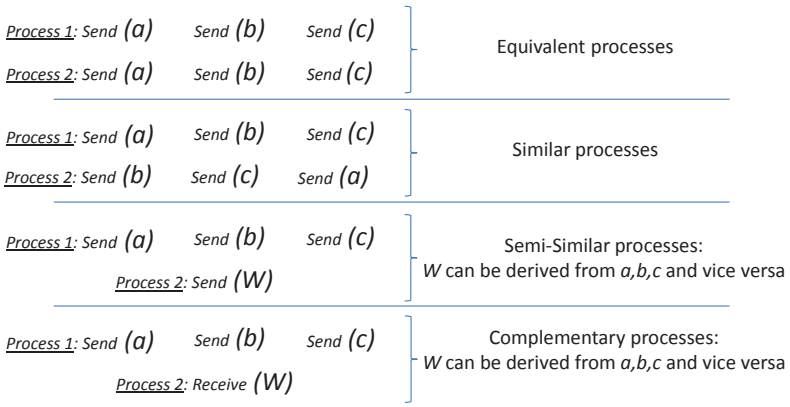


Figure 4.13: Processes Complementarity Illustration

Lemma 5. *For each agreement specification in the contract, the generated processes of the requester and supplier are complementary.*

Proof. From the complementarity definition, the contract formal definition and the workflow generation algorithm, we conclude that the processes of the requester and the supplier for the same agreement are complementary. □

Lemma 6. *A supplier process of an agreement in the contract receives an acknowledgment or an exception message ⇒ this process has already received all messages of the obligation.*

Proof. The requester sends an acknowledgment iff he has performed at least one iteration with the supplier and thus the supplier has received all messages. □

Theorem 5 (Safety of the mediation approach)**.** $SentMessage = (Acknowledgment \vee Exception) \Leftrightarrow$ *the processes of the supplier and the requester have reached a block that constitutes the end of the current agreement.*

Proof. We use equivalence proof method:

- $\Leftarrow$ by definition of *acknowledgment* and *exception* messages
- $\Rightarrow$ From Lemma 4 and Lemma 5.

□

Theorem 6 (Liveness of the mediation approach)**.** *The condition expressed in line 7 of Algorithm 3 will eventually be satisfied.*

Proof. Suppose that this condition cannot be satisfied. This could have two reasons:

1. All received messages cannot be used to compute the expected message. This is not possible since the models of the processes are complementary (Lemma 4)
2. The received message is either an *acknowledgment* or an *exception*. However, the *acknowledgment* or the *exception* messages will be received only when at least one iteration has been performed (Lemma 5). Since the processes are complementary, this condition will be satisfied once.

From these two points, we conclude that the condition will eventually be satisfied for all expected messages. □

4.7 Evaluation

To evaluate the developed framework, we start by establishing a set of objective metrics that allow us to compare our solution with the existing ones. Then we give experimental evaluations.

4.7.1 Evaluation of the Modeling Language Abstraction

We applied the goal-question-metric technique [BJ06] to define appropriate metrics to compare the abstraction of our contract specification framework and the run-time language that is XPDL. We selected two main issues to conduct the comparison; *effort*– how much of the designer's resources (e.g. time) are required to maintain the contract specification– and *productivity*– how productive is a designer who is using our contract construction concepts for contract specification.

To measure these qualities, we chose two size metrics to compare programs specified by both languages. The size metrics we utilize are number of concepts *NC* that are used in the model in order to express the same information. Size metrics are considered as relatively good predictors of maintenance effort even though they are not the sole predictor [LH93]. Considered from this angle *NC* provides an indicator of effort and productivity of designers using our framework to model the contract.

The evaluation methodology relies on the weight assigned to a concept:

Definition 21. *When a concept c appears in the model one time it receives the weight* $\omega(c) = 1$. *When c appears N time without adding new information, it receives the weight* $\omega(c) = \frac{1}{N}$. *When a concept does not appear despite its importance for the model, it receives the weight* $\omega(c) = 0$.

For our running example depicted in Figure 4.4, we have two important concepts for the *coordinator* that are the Product Components Properties (PCP), and the Product Components (PC).

In the XPDL model, each PCP could appear two times. One time in a send activity and one time in a receive activity. This is the case for the property *Fuselage_Height* in Figure 4.9. In this case, *Fuselage_Height* receives the weight $\frac{1}{2}$. The same method is used to compute the weight for all other PCP. Nevertheless, for the *coordinator*, it is better to remove the redundancy of properties. The reason is that if he sees a property, it means that by default it will be exchanged with the *participant*. Consequently, in our contract model, the property *Fuselage_Height* appears once and thus it is given the weight 1 as depicted in Figure 4.4. The same method is used to compute the weight of the other PCP.

For PCs, despite their importance for the *coordinator*, they do not appear at all in the XPDL model while they clearly appear in our contract model. Since our framework is more specialized, much of non-necessary information is hidden and is replaced by valuable information for the *coordinator*.

Table 4.7 details the computation of the added value of concepts in the model and their application to our example.

Concepts	**Calculation Method**	**XPDL Model** Figure 4.9	**Contract Model** Figure 4.4
Product Components Properties (PCP): PCP	$\sum_{i=0}^{N} \frac{1}{\lvert PCP_i \rvert}$	$\frac{1}{2} * 9 + 3$=7.5	(4+2+3+1) = 10
Product component: PC	$\sum_{i=0}^{N} \frac{1}{\lvert PC_i \rvert}$	0	3

Table 4.7: Comparison of model semantic richness for XPDL versus our framework

4.7.2 Evaluation of the on-the-fly Mediation

We define a metric to compare the impact of changes on the mediation strategy. We measure the impact of changes on the mapping functions established between the messages that are exchanged through the mediator. Basically, a mediation solution supports three mapping dimensions that are: *(i) syntactic*, *(ii) structural*, and *(iii) semantic*. Algorithm 3, addressed the structural mappings between the exchanged messages. In another work [KFFDSG14] we defined a set of mapping functions regarding syntactic and semantic mediation.

Using this metric, we will measure the impact of adding new (syntactic or semantic) mapping functions to the mediator and compare the impact of including these changes on conventional mediators and on our on-the-fly mediation.

First, consider that for a particular domain D (e.g. aircraft aerodynamics simulation), a software application A that covers the functionalities required by this domain can be defined by: $A = \langle O, S \rangle$ where:

- O is the ontology of concepts used by the application to exchange information with its environment
- S is the syntax of files used by the application in order to represent data exchanged with the application's environment

A domain is formally defined by: $D = \{A | A = \langle O, S \rangle\}$.

Consider that a partner P_1 using an application $A_1 \in D$ is to be replaced in the DMN by a partner P_2 using an application $A_2 \in D$. When this change request arrives, for a conventional mediator in order not to have a global impact on the DMN, mapping functions should have been defined between every couple of applications $(A_i, A_j) \in D \times D$ that covers this domain functionalities. This is the unique approach to avoid stopping the collaboration to include new mapping functions to the mediator. Following this

approach, the number of mapping couples that should be defined is equal to $\begin{pmatrix} |D| \\ 2 \end{pmatrix}$ since in the worst case one should define mapping functions between all possible couples of applications. It is clear that this number is very high and will require high costs.

For the on-the-fly mediation, we no longer need to define all these mappings, we only add mapping functions when they are required while keeping local the impact of changes on the collaboration.

Let us consider that we have $\begin{pmatrix} |D| \\ 2 \end{pmatrix}$ possible couples of mapping functions that can be defined to ensure syntactic and semantic interoperability. Let us consider also that it takes $T = (T_{def} + T_{dep})$ the time required to formalize the mapping functions and include them into the mediators database. Where T_{def} is the time required to define the mapping functions and T_{dep} is the time elapsed from stopping the collaboration until achieving the deployment of the new mapping functions. Finally let us consider that we have N components being designed in the collaborative environment (i.e., we have N atomic agreement in the contract). Formula 4.2 computes the impact of managing the replacement of partners in the DMN when a conventional mediator *(cm)* is deployed:

$$impact_{cm} = \begin{pmatrix} |D| \\ 2 \end{pmatrix} \times T \times N \tag{4.2}$$

For conventional mediators, a mediator can only ensure interoperability between a couple of partners. Thus for our collaborative platform, we will have several mediators deployed. This does not foster *reusability* of mapping functions. If we consider the dimension of updating the databases of mapping functions of the deployed mediators while keeping the partners in the DMN unchanged, the impact of changes on the conventional mediators is given by formula 4.3:

$$impact_{cm} = \begin{pmatrix} |D| \\ 2 \end{pmatrix} \times T \times N + m \times (\begin{pmatrix} |D| \\ 2 \end{pmatrix}) \times T_{dep} \times N \tag{4.3}$$

where m is the number of reusable mapping functions to update into each couple of applications.

For the on-the-fly mediation, there is a single database of all mapping functions. Thus every mapping function could be reused and does not need to be duplicated. Accordingly, the on-the-fly mediation is independent from the parameter m. Formula

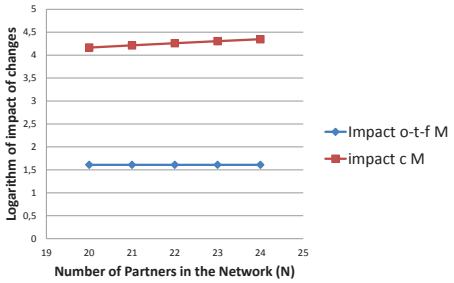


Figure 4.14: variating N

4 computes the mean impact[3] of updating the mapping functions database when an on-the-fly mediator is deployed:

$$impact_{o-t-fm} = \frac{(\begin{pmatrix} |D| \\ 2 \end{pmatrix} - 0)}{2} \times T \times 1 \tag{4.4}$$

where the number 1 indicates that only the component being designed is impacted by this change. The number 0 indicates that there is no impact when the change does not involve a new partner with a different application. Since formula 4.4 calculates the mean, we can derive two limit measures:

- The best case is when no change is required $impact_{o-t-fm} = 0 \times T \times 1 = 0$
- The worst case is when multiple changes are required $impact_{o-t-fm} = \begin{pmatrix} |D| \\ 2 \end{pmatrix} \times T \times 1 = \begin{pmatrix} |D| \\ 2 \end{pmatrix} \times T$

By comparing formulas 4.3 and 4.4, we conclude that the on-the-fly mediation is better in all cases.

Figures 4.14 4.15 4.16 depict the effects of varying N, m, and $|D|$ respectively on the impact of changes in both configurations. We conclude that our mediation approach is less impacted by changes that occur at run-time and more approapriate for DMNs

4.7.3 Evaluation of the on-the-fly mediation against aspect-oriented mediation

[KSPBC06] developed an aspect-oriented approach to achieve mediation between heterogeneous business processes. Our on-the-fly mediation and the aspect-oriented approach

[3] we calculate the mean since we do not know how many mapping couples should be defined

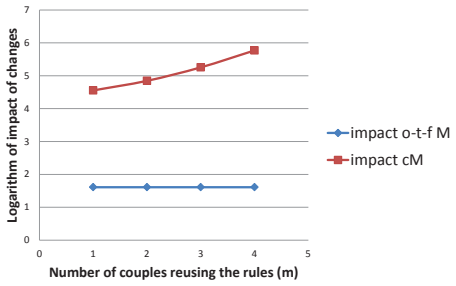


Figure 4.15: variating m

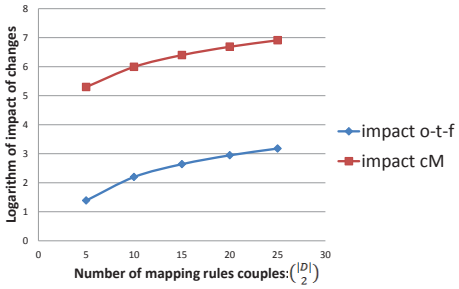


Figure 4.16: variating $|D|$

have in commonality their support for changing the communicating processes without changing the mapping functions in our case or the $\langle query, aspect \rangle$ in the aspect-oriented approach case. Nevertheless our approach outperforms the aspect oriented approach in an important situation that is the **Many-to-One Mismatch**.

A many-to-one mismatch occurs when the receiver business process waits for a message that can be computed from a set of messages sent by the sender. To resolve this mismatch using [KMNB+09] approach, one should write a couple $\langle query, aspect \rangle$ where in the aspect one should write successive *receive* actions in order to capture all sent messages from the sender. In this case, the aspect writer needs to define a specific order of the successive receive actions. This order creates a coupling between the sender business protocol and the receiver business protocol. Accordingly, in a DMN, each time the sender tailors the order of his send activities a new couple $\langle query, aspect \rangle$ should be defined to ensure the mediation.

In our approach, thanks to the links between all states of the marked automaton representing the mapping function, no order is specified at design time for the mapping function. Thus, when a change in the DMN configuration occurs, there is no need to adapt the automaton.

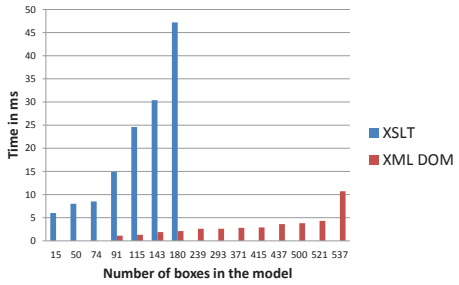


Figure 4.17: Comparison between XSLT+JS and XML DOM

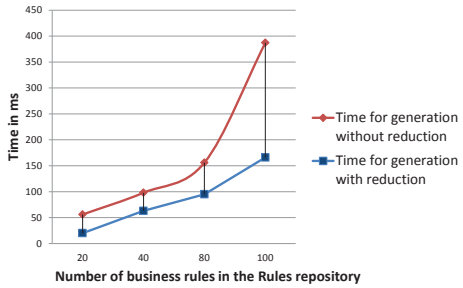


Figure 4.18: Repository reduction impact

4.7.4 User Interface Design

The reactivity of the user interface when *coordinators* and *participants* design the collaboration contracts is important to consider too. Due to the novelty of our technique of concurrent contract design and the number of involved constructs (a contract of an aircraft design may involve thousands of constructs) we evaluated the time that the interface takes to load the visual representation of the collaboration contract. We used two implementation approaches. One that transforms the XMI representation of the contract model into HTML using XSLT technology as was recommended by [SK03] and a home-built algorithm that exclusively uses JavaScript to perform the transformations. The comparison is given in Figure 4.17.

In this figure, the time required for the XSLT technique to display the contract model increases much more than the time required by our home-built algorithm. When we go beyond 180 boxes in the contract model, the XSLT algorithm crashes.

4.7.5 Workflow Generation Algorithm Performance

Figure 4.18 depicts the comparison between the two workflow generation techniques. From this analysis, we conclude that it is better to precede the workflow generation by applying the business rules repository reduction operator.

4.8 Conclusion

When running an engineering change management process in an inter-organizational cooperation, the involved parties specify a contract. This contract will serve to generate the cross-organizational process that will support them in fulfilling the obligations of the contract. Each party implements its part of the contract following its internal business rules or by reusing an existing process that could fulfill its obligations.

In this chapter we proposed a formal notion of contracts based on product breakdown structure and the temporal constraints that could exist on the design of the different components of the product. We defined the minimal set of concepts required by *coordinators* and *participants* to build a contract then we gave formal definition for these concepts. Since this contract needs to be supported by a cross-organizational process, we developed an efficient algorithm that uses the contract specification and each party business rules in order to generate the cross-organizational process. At run-time mismatches could appear between the fragments composing the cross-organizational process, we developed a novel mediation approach to resolve the possible mismatches.

The framework was developed while considering the dynamic aspect of the collaboration environment (DMN). We have evaluated how our mediation solution tackles this problem. In the next chapter, we aim at deepening the analysis on the management of dynamicity in the collaboration environment. We will define the required operations to perform the modifications on the contract and how to shield partners not concerned by the modifications from being impacted. Furthermore, we present an approach to help the recovery of modifications.

Chapter 5

Managing Evolutions for Product Level Agreements

Contents

5.1 Introduction

Product design is a collaborative effort that involves an Original Equipment Manufacturer (OEM) and several subcontractors. Detailed processes, methodologies and best practices guide the collaboration lifecycle. However these plans can only form a common starting point because unpredictable events always happen. Successful completion of collaboration projects depends on containing their inherent unpredictability, figuring out the best responses, and successfully making the necessary changes [OBRC10]. In product design collaborative environments, stakeholders are bound by a **collaboration contract** that determines their obligations with regard to the product being designed. Nevertheless, the emergence of Dynamic Manufacturing Networks (DMN) that put forth the evolution aspect in collaborative environments demonstrates the need for managing the evolution of collaboration contracts as well. Accordingly, to handle this evolution, it is necessary to include *flexibility in collaboration contracts* in order to respond to possible evolutions that could occur at run-time.

In this chapter we are interested in enriching the product based contracts framework developed in the previous chapter with management operations. These operations can be used by *coordinators* and *participants* in order to modify the specifications of *agreements* in contracts while shielding *coordinators* and *participants* from concerns related to the impact of changes on the underlying cross-organizational process. In addition, after several modifications on a contract, partners performing changes on this contract need to perform the least mental effort to figure out what is missing in this incomplete contract specification in order to include the missing information and re-launch the collaboration as fast as possible.

Throughout this chapter we lay down the foundations of management operations to be used by *coordinators* and *participants* in order to manage the evolution of the DMN. We define the operational semantics of these operations and we analyze their impacts on the underlying cross-organizational process. We limit this impact by developing an algorithm that prevents the execution of management operation from impacting partners not concerned by the change.

When multiple changes are incorporated in a contract by multiple partners, it can be challenging for them to keep in their mind or figure out what needs to be included in the contract to re-launch the collaboration. Accordingly, we enrich the framework with a feature that analyzes the current state of a contract, and then finds what information is missing in each *agreement* to become *committable* and generates the workflow that will support the associated partners to complete the *agreement* specification and makes it

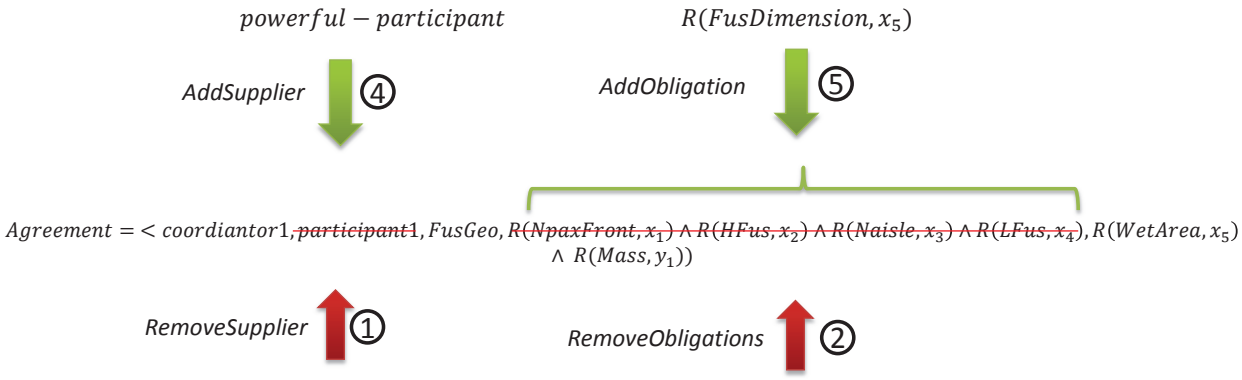


Figure 5.1: Example of modifying an agreement specification

committable. To achieve this, we extend the graph-planning algorithm to generate an executable workflow that supports partners in completing agreements following their own sequencing of actions.

5.2 Motivating Example

We carry on with the example of the *Fuselage* design involving *coordinator1*, *participant1*, and *participant2* that we detailed in the previous chapter. Let us suppose that *coordinator1* notices that *participant1* has capabilities issues in handling the change requests that he issues for the Fuselage geometry (*FusGeo*) (for example due to the time that *participant1* takes to perform simulation). In this case, *coordinator1* would like to replace *paritici-pant1* by another participant having more capabilities that we call *powerful-participant*. Furthermore, when replacing *participant1*, *coordinator1*, also, would like to change the obligations that he submits to *powerful-participant*. Figure 5.1 depicts the actions to be executed to realize this change.

To perform this change, *coordinator1* should follow these steps:

1. **Remove** *participant1* from the agreement corresponding to *FusGeo* in the contract specification
2. **Remove** the *obligations* specified in the *FusGeo* agreement
3. Look for *powerful-participant* in a repository (for example the partners blueprint developed within the IMAGINE project)
4. **Add** *powerful-participant* as a supplier in the agreement *FusGeo* in the contract
5. **Add** the new obligations for *powerful-participant*

6. When *coordinator1* is done, the cross-organizational process (COP) generation module generates the new COP involving *powerful-participant*.

To achieve this goal, *coordinator1* should be able to perform the following actions on the contract model:

- **Remove** the supplier form an agreement specification
- **Add** a supplier to a an agreement specification
- **Remove** obligations from an agreement specification
- **Add** obligations to an agreement specification

5.3 Challenges

When managing the contract, *coordinator1* faces the following challenges:

5.3.1 Applying High-level Management Operation

Coordiantor1 should be able to perform the management operations in a straightforward way. Thus, he needs to have access to the management operations through a user interface. Furthermore, *coordiantor1* could have some sequence of operations that he performs repetitively during the design process. For example the sequence ⟨ **Remove** Supplier, **Add** (Supplier), **Remove** (existing obligations), **Add** (new obligations) ⟩ might be used by *coordinator1* several times. In this case, he shall be able to specify this sequence as a new operation and use it.

5.3.2 Maintaining the COP executable

When *coordiantor1* replaces *pariticpant1* by *powerful-participant* the time interval to achieve this replacement could be significant. In this case the COP generation module should have enough intelligence not to generate COPs until it ensures that the COP it will generate remains executable and the *participants* not concerned by the changes that occurred can carry on their collaboration.

5.3.3 Ensuring Semi-automatic Recovery of Modified Contracts

In our example, we have seen that *coordiantor1* has tailored only one agreement by changing the requester and obligations. However, in real life scenarios, *coordiantor1* would need to change dozens of agreements specifications at the same time. In this

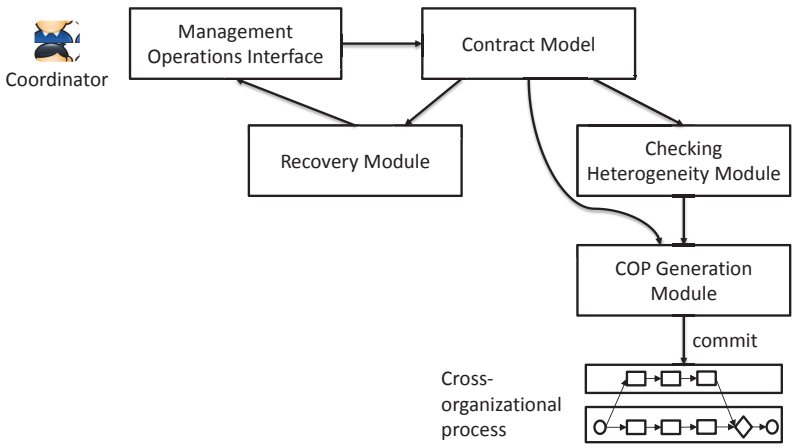


Figure 5.2: Overview of change management modules and their interactions

case, keeping in his mind all specifications that he changed (for example by removing the suppliers) and performing a recovery for each one could be painstaking and error-prone. Thus, we need to develop an approach that assists and guides *coordinator1* in performing the recovery of removals. This approach relies on a *recovery workflow* that will help resume the collaboration as fast as possible with the least mental effort from *coordiantor1*'s side.

5.3.4 Overview of the Proposed Approach

Figure 5.2 gives an overview on the on the modules to realize our approach and their interactions. Indeed, in order to address the identified challenges, we extend the framework developed in chapter 4 by a set of management operations that support *coordinators* in performing the desired changes on the contract specification at run-time. Furthermore, to lessen the impact of changes on other *participants* that are not concerned by the change, we develop a heterogeneity checking algorithm that answers the following question: if the COP generation module generates a new COP model, will it be executable by a workflow engine? The answer provided by this algorithm helps limit the impact of changes on other *participants* since we can determine whether we will deploy the new process or keep it on hold until other tailoring operations are executed which eventually will make it executable. Finally, we develop a recovery module that supports *coordinators* in recovering modified contracts by guiding them using a recovery workflow.

The reminder of this chapter is organized as follows: in section 5.4 we present the algorithm that assesses whether a change on an agreement specification in the contract has to be committed into the cross-organizational process or not. The result returned by this algorithm is used to limit as much as possible the impact of changes on other

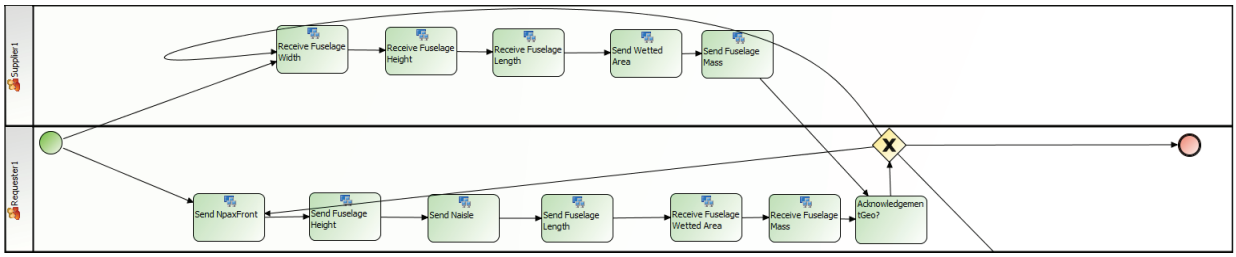


Figure 5.3: Original *FusGeo* Agreement Process

partners that are not concerned by the change that occurs. In section 5.5, we elaborate on the set of management operations, classify them and address concurrent calls issues.

We continue the contribution of this chapter by developing an extension of the graph-planning algorithm in sections 5.6 so that it generates an executable workflow. This (recovery) workflow will be used to figure out what information is missing in one or several agreements and generates the required actions to incorporate the missing information.

Section 5.8 evaluates the extended framework in comparison to existing works and section 5.9 concludes the chapter.

5.4 Checking Heterogeneity of Generated Processes

Using the framework developed in chapter 4, *coordinators* and *participants* can build a collaboration contract by using the product breakdown model. During the collaboration, *coordinators* might need to change the contract by adding/removing some of its building blocks. Nevertheless some of these changes can lead to the generation of a non-executable cross-organizational process and thus partners involved can no longer collaborate.

Example 8. *Suppose that coordiantor1 has removed participant1 from the agreement associated to FusGeo (the remove operation will be detailed later). Then, coordinator1 starts looking for powerful-participant. During this time, if the new contract specification (where the supplier of the FusGeo agreement is missing) is committed to the running COP (see Figure 5.3 vs Figure 5.4), then the generated process fragment corresponding to coordinator1 will send messages that have no recipient specified, because the latter has been removed. This will lead to structural mismatches in the COP and makes it non-executable by the workflow engine. Consequently, all other participants will no longer be able to exchange messages even though they are totally independent from the FusGeo design.*

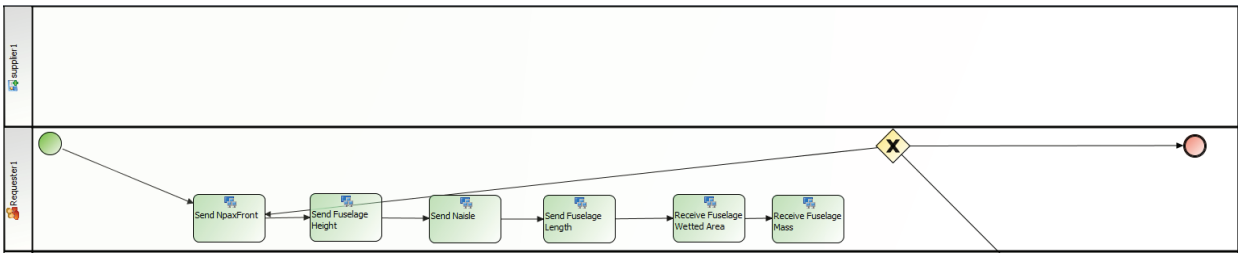


Figure 5.4: The new *FusGeo* Agreement Process after a commit

In this section we develop an algorithm that returns mismatch assessments between two communicating processes associated to an Agreement specification. Later on, these mismatch measurements will be used to decide whether to commit a change at the contract level into the running cross-organizational process or not.

Definition 22. *A cross-organizational process is executable if it can reach its final state.*

In addition to the syntactic correctness condition, a cross-organizational process composed of a set of couples $\langle P_{Req}, P_{Supp} \rangle$ is executable *if and only if* for each couple: P_{Req} and P_{Supp} are interoperable.

Proof.

- $\Rightarrow$ From *Definition* 1, since an executable process terminates, all expected messages will arrive. Thus both the sender and the receiver are interoperable.
- $\Leftarrow$ For each fragment $\langle P_{Req}, P_{Supp} \rangle$ of the cross-organizational process, each sent message will be consumed and reversely, each expected message will be sent. Thus the cross-organizational process will progress until reaching its final state.

□

Algorithm 7 assesses whether a couple of *Requester* and *Supplier* processes generated from an *Agreement* in a DMN contract model are interoperable or not (and thus, from proposition 1, executable or not). For this purpose it calculates the values of three parameters:

- α_{str} indicates if a *structural mismatch* exists between two communicating process. A structural mismatch occurs when there is a difference in the structure of messages exchanged.
- α_{syn} indicates if a *syntactic mismatch* exists between two communicating process. A syntactic mismatch occurs when one stakeholder uses a language (e.g. XML) to

serialize the messages it exchanges, while the other stakeholder uses a different language (e.g. EXPRESS[1]).

- α_{sem} indicates if a *semantic mismatch* exists between two communicating process. A semantic mismatch occurs when one stakeholder process uses a different ontology from the other.

Notice that even though a mismatch would have been detected between two processes, if Algorithm 7 finds out that it is possible to resolve it using a mediation solution, then it is not considered as a mismatch.

In order to write Algorithm 7 we need a formal representation of stakeholders' public processes P_{Req} and P_{Supp}. These processes can be specified using LTS (Labeled Transition Systems). Basically an LTS is a quadruplet $\langle S, Act, \mapsto, I \rangle$ where: *(i)* S is a set of states; *(ii)* Act is a set of actions; *(iii)* $\mapsto \subseteq S \times Act \times S$ is a transition relation; *(iv)* $I \subseteq S$ is a set of initial states.

For our convenience we define an extended LTS as a basic LTS with functions that specify what language an action of Act uses and what ontology it uses too. Thus an extended LTS is a tuple $\langle S, Act, \mapsto, I, M, O, L, ontology, language \rangle$ where:

- M, O, L are respectively the set of messages exchanged by the LTS; the set of ontologies to which a message concepts pertain; and the set of languages used to serialize the messages;

- $Act \subset \{send(m), receive(m)\}^+$ such that $m \in M$;

- $ontology : M \rightarrow O$ maps a message to the ontology it uses to describe its concepts;

- $language : M \rightarrow L$ maps a message to the language it uses in its serialization.

Basically, when an *Agreement* is tailored, it is passed to Algorithm 7. This algorithm analyzes both *Requester* and *Supplier* processes. It returns the values of α_{str} (line 43), α_{syn} (line 15) and α_{sem} (line 36). **If for a particular *Agreement*, the algorithm returns *true* for one of the three mismatches, the changes on this *Agreement* will not be committed to the COP model**.

Using this algorithm, we can control the commit of changes introduced by the management operations. We detail this feature in the next section.

[1] http://deslab.mit.edu/DesignLab/dicpm/step.html

Algorithm 7 Check Heterogeneities

Require: Agreement tuple Ag
Ensure: $\alpha_{str}, \alpha_{syn}, \alpha_{sem} \in \{true, false\}$

$\alpha_{str}, \alpha_{syn}, \alpha_{sem} \leftarrow false$
if $Ag.Requester \neq$ and $Ag.Supplier \neq$ and $Ag.PCC \neq$ and $Ag.o \neq$ **then**
  $LTS_r \leftarrow$ Generate Requester LTS from $Ag.PCC$
  $LTS_s \leftarrow$ Generate Supplier LTS from $Ag.PCC$
  $i \leftarrow 0$
  while $LTS_r.s_i \neq Final\_State$ **do**
    $m_1^{'} \leftarrow LTS_r$.expected message structure
    while $m_1^{'}$ is not complete **do**
      $m_1 \leftarrow$ sent message
      if ontology($m_1^{'}$) = ontology(m_1) **then**
        if concept(m_1) $\cap$ concept($m_1^{'}$) $\neq$ **then**
          if language(m_1) $\neq$ language($m_1^{'}$) **then**
            if mapping(language(m_1), language($m_1^{'}$)) = ø **then**
              Display in the dashboard the reason for non-commit
              return $\alpha_{syn} \leftarrow true$
            else
              transform_language(m_1,$m_1^{'}$)
            end if
          end if
          if concept(m_1) = concept($m_1^{'}$) **then**
            $m_1^{'} \leftarrow m_1$
          else
            if concept(m_1) $\subset$ concept($m_1^{'}$) **then**
              value($m_1^{'}$) $\leftarrow$ value(m_1)
              complete $m_1^{'}$ with buffer content if possible
            else
              value($m_1^{'}$) $\leftarrow$ value(concept (m_1) $\cap$ concept($m_1^{'}$))
              Buffer.add(concept(m_1) - concept($m_1^{'}$))
            end if
          end if
        else
          Buffer.add(m_1)
        end if
      else
        Display in the dashboard the reason for non-commit
        return $\alpha_{sem} \leftarrow true$
      end if
    end while
    $LTS_r.s_i$++
  end while
else
  Display in the dashboard the reason for non-commit
  return $\alpha_{str} \leftarrow true$
end if

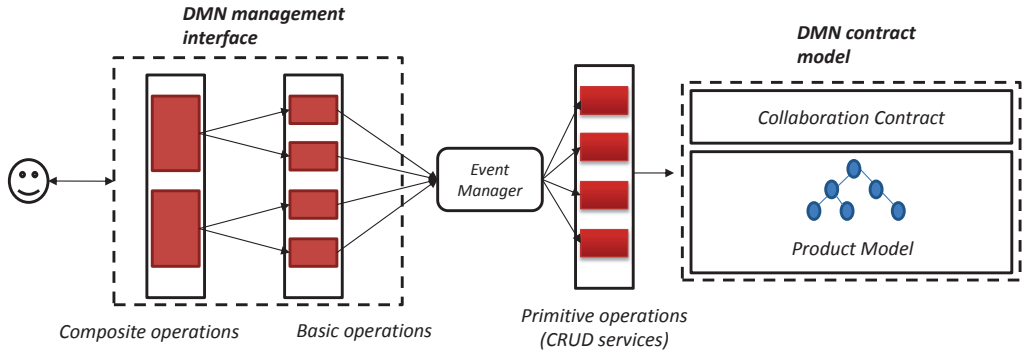


Figure 5.5: DMN management operations overview

5.5 Operations for DMN Management

An added-value of a DMN contract model is the ability to change its configuration in a controllable way. This is possible thanks, to the verifications performed by Algorithm 7. In this section we formally describe operations that tailor the DMN contract model following the DMN evolutions. Additionally, we demonstrate how their invocation maintains the DMN cross-organizational process executable by the workflow engine.

Basically, we define three types of operations on the top of the contract model as illustrated in Figure 5.5: (i) *Primitive operations* that act on the building blocks of the contract model. They are implemented as services upon this model. They are detailed in Tables 5.1, 5.2. (ii) *Basic operations* that have the same granularity as primitive operations. They can be invoked by partners through a visual interface and they use the primitive operations. (iii) *Composite operations* are successive calls to primitive operations. The objective of composite operations is to provide partners with more complex management operations.

We provide partners with a visual interface that displays basic and complex operations to manage the evolutions of the contract specification. The use of visual interfaces makes easier the perception of knowledge [HIL94, Har88] and thus the management task will be less complex. In addition, using a visual interface will help us resolve the problem of concurrent calls to management operations as we will see later.

5.5.1 Primitive Operations

Upon the contract model we define a set of CRUD (Create Read Update Delete) operations that manage its building blocks as detailed in Tables 5.1, 5.2. When invoking (programmatically) these operations it is necessary to consider their pre/post conditions. A primitive operation such as $RemoveComponent(c)$ can be described using operational

Operation	Precondition	Postcondition
$AddAgreement(ag)$	$ag \notin \mathcal{C}$	$ag \in \mathcal{C}$
$AddComponent(cp, ag)$	$ag \in \mathcal{C} \wedge cp \notin ag$	$ag \in \mathcal{C} \wedge cp \in ag$
$AddOConstraint(cn, Ag)$	cn is not a subformula of $Ag.PCC$	cn is a subformula of $Ag.PCC$
$AddCConstraint(cn, Ag)$	cn is not a subformula of $Ag.PCC$	cn is a subformula of $Ag.PCC$
$AddRelationship(Ag_1, Ag_2, r)$	r does not link Ag_1 and Ag_2	r links Ag_1 and Ag_2
$AddException(NonAg)$	$NonAg \notin \mathcal{C}$	$NonAg \in \mathcal{C}$
$AddRequester(p, Ag)$	$p \in PM.P \wedge Ag \in \mathcal{C} \wedge Ag.Requester = \emptyset$	$Ag.Requester = p$
$AddSupplier(p, Ag)$	$p \in PM.P \wedge Ag \in \mathcal{C} \wedge Ag.Supplier = \emptyset$	$Ag.Supplier = p$

Table 5.1: DMN management mperations

Operation	PostCondition	Precondition
$RemoveAgreement(Ag)$	$ag \notin \mathcal{C}$	$ag \in \mathcal{C}$
$RemoveComponent(cp, Ag)$	$ag \in \mathcal{C} \wedge cp \notin ag$	$ag \in \mathcal{C} \wedge cp \in ag$
$RemoveOConstraint(cn, Ag)$	cn is not a subformula of $Ag.PCC$	cn is a subformula of $Ag.PCC$
$RemoveCConstraint(cn, Ag)$	cn is not a subformula of $Ag.PCC$	cn is a subformula of $Ag.PCC$
$RemoveRelationship(Ag_1, Ag_2, r)$	r does not link Ag_1 and Ag_2	r links Ag_1 and Ag_2
$RemoveException(NonAg)$	$NonAg \notin \mathcal{C}$	$NonAg \in \mathcal{C}$
$RemoveRequester(p, Ag)$	$p \in PM.P \wedge Ag \in \mathcal{C} \wedge Ag.Requester = \emptyset$	$Ag.Requester = p$
$RemoveSupplier(p, Ag)$	$p \in PM.P \wedge Ag \in \mathcal{C} \wedge Ag.Supplier = \emptyset$	$Ag.Supplier = p$

Table 5.2: DMN management mperations

semantic notation as follows (the same applies for remaining operations. Notice that we use the same notation as [Win93]):

$$\frac{\langle exists(c), \sigma \rangle \rightarrow true}{\langle RemoveComponent(c), \sigma \rangle \rightarrow \sigma'} \tag{5.1}$$

This statement indicates that to remove the component c from the current state σ of the contract, this component should exist in the contract model.

Concurrent invocation of primitive operations to manage evolutions of the collaboration contract model could be problematic as illustrated by the following example:

Example 9. *Consider that coordinator1 wants to remove a particular agreement in the collaboration contract. In the meantime, suppose that the associated supplier, that is participant1, in this agreement arrived to the same conclusion and decided to remove the agreement too. When both partners call the RemoveAgreement operation simultaneously, although the first operation call will succeed, the second call will generate an exception. The second call no longer complies with the precondition of the operation RemoveAgreement that requires the presence of the agreement in the contract model.*

5.5.2 Basic Operations

In order to resolve the concurrent calls issue of primitive operations, we take advantage of the visual management interface and use an event-based approach to invoke primitive operations. Basic operations are duplications of primitive operations that partners invoke through the visual interface as depicted in Figure 5.5. When *coordinators/participants* invoke a basic operation (by pressing the corresponding button in the management interface), instead of calling the corresponding primitive operation, this call generates an event. This event is an intermediate means to call the corresponding primitive operation. Events have the advantage to be discrete in the sense that they can be organized in a queue structure. An event contains the information to be incorporated in the contract model, it is defined as a tuple with a single field $\langle tailoring\_element \rangle$. A module of the contract management called the *event manager* manages the calls to primitive operations. A primitive operation is executed when the payload of the event residing in the head of the queue satisfies its precondition. In this case the event manager invokes the primitive operation. Although the event manager eliminates the concurrent calls failures, it does not resolve all issues:

Example 10. *Suppose that coordinator1 and participant1 simultaneously call the operation RemoveAgreement. These calls will generate two events containing the same payload. When analyzing the incoming events, the event manager will call the RemoveAgreement operation. After this call, the agreement ag will not exist in the contract anymore. Accordingly, the second event payload will not match with the RemoveAgreement premises but, this time it matches with AddAgreement premises as given in statement 5.2:*

$$\frac{\langle exists(ag), \sigma \rangle \rightarrow false}{\langle AddAgreement(ag), \sigma \rangle \rightarrow \sigma'} \tag{5.2}$$

In this situation, the event manager will call the AddAgreement primitive operation, which was not the aim of coordinator1!

The problem faced by the event manager is due to the *non-exclusiveness* when matching the event content against the operations' premises.

Since *coordinators* and *participants* call the contract management operation through a visual interface, it is possible to incorporate the toggled button information in the event in order to eliminate the non-exclusiveness for the event manager. Thus, an event payload becomes the couple $\langle tailoring\_element, toggled\_button \rangle$. The new operational

semantics of the basic operation $RemoveAgreement(ag)$ is given in statement 5.3:

$$\frac{\langle exists(ag), \sigma\rangle \rightarrow true}{\langle if\, RemoveAgreementToggled \rightarrow RemoveAgreement(ag)\, fi, \sigma\rangle \rightarrow \sigma'} \tag{5.3}$$

Based on statement 5.3, the $RemoveAgreement(ag)$ primitive operation is called if and only if its premise is evaluated to true and the *Remove Agreement* button in the visual interface has been pressed.

Using this formalization of operational semantics, the event manager will no longer face non-exclusiveness situations.

Example 11. *Consider the $RemoveAgreement(ag)$ scenario again. Coordinator1 presses the Remove Agreement button in the visual interface which will generate the event to remove the agreement ag. In the meantime participant1 playing the role of supplier in the agreement ag presses the same button in his interface which will generate an equivalent event. Although the second event payload matches the premises of the $AddAgreement$ operation (since the agreement does not exist anymore) this operation will not be invoked because the $AddAgreementToggled$ condition is not satisfied as illustrated in statement 5.4.*

$$\frac{\langle exists(ag), \sigma\rangle \rightarrow false}{\langle if\, AddAgreementToggled \rightarrow AddAgreement(ag)\, fi, \sigma\rangle \rightarrow \sigma'} \tag{5.4}$$

5.5.3 Composite Operations

Basic operations aim to provide partners with satisfactory flexibility to manage contracts. Sometimes *coordinators* could have redundant needs for specific sequences of management operations. Hence, instead of calling each basic operation separately, it is better to provide *coordinators* a single high level operation, i.e. a composite operation. This composite operation is constituted of successive calls to primitive operation. The question that raises: *when building composite operations should we directly make invocations to primitive operations (without using the event manager) or invoking basic operations instead*?

Building composite operations by making direct invocations to primitive operations would lead to the same concurrent issue discussed previously. Hence we should use the second alternative.

Composing basic operations to generate more high level operations needs to ensure the exclusiveness condition for the basic operations invoked. In other terms, the buttons corresponding to basic operations should be pressed. This is a challenging issue since the aim of composite operations is to prevent partners from calling basic operations by themselves. It is possible to resolve this issue by automatically generating the events

corresponding to basic operations each time a complex operation is invoked. To create a complex operation a partner can simulate its execution by invoking basic operations that compose it. In the meantime an *event recorder* records the generated events. At the end of the simulation, the result is an automaton that saves the generated events and their sequence. Later on, when this composite operation is invoked, the previously generated automaton is executed by sending the events to the event manager which is equivalent to make calls to basic operations. Thus, we no longer face concurrent issues for composite operations.

5.5.4 Committing Changes

In the previous subsections we defined three types of operations to manage contracts in the context of a DMN. In this subsection, we determine which basic operations calls shall be followed by a commit through a set of lemmas. Then Algorithm 8 uses these lemmas to automate the commit operation.

Lemma 7. *Suppose that the cross-organizational process corresponding to a contract C is executable and suppose for the agreement* Ag_1 : $Ag_1.Requester \neq \emptyset$ *and* $Ag_1.Supplier \neq \emptyset$ *and* $Ag_1.object \neq \emptyset$. *In this case a call to* $AddOConstraint(cn_1, Ag_1)$ *from Table 5.1 on C will commit the changes to the underlying cross-organizational process.*

Proof. Referring to Algorithm 7, we prove that the $Ag_1.Requester$ process and $Ag_1.Supplier$ process remain interoperable after executing this operation. (i) The condition of syntactic interoperability is satisfied since adding a constraint to an existing agreement will generate new message exchange activities in the underlying processes. These activities will use the same language to serialize the exchanged messages. (ii) The condition of semantic interoperability is satisfied since the generated messages belong to existing ontologies (no new mapping needed). (iii) The condition of structural interoperability is satisfied since the process generation algorithm by default generates structurally interoperable processes as we have seen in the previous chapter.

Thus Algorithm 7 returns the value false for α_{str}, α_{syn}, α_{sem}. Accordingly the generated processes are interoperable, therefore the cross-organizational process remains executable. □

Lemma 8. *Suppose that for an agreement* Ag_1, $Ag_1.Supplier = \emptyset \wedge Ag_1.Requester = \emptyset$. *Committing* $AddRequester(q_1, Ag_1)$ *does not maintain the interoperability of the cross-organizational process.*

Proof. Referring to Algorithm 7, we prove that if a commit is performed, the process that results will not be executable. Indeed, if a commit is realized, the *Requester* q_1 will send messages while there will be no receiver. Thus the structural interoperability is not ensured ($\alpha_{str} = true$). Thus the cross-organizational process fragments are not interoperable and not executable (proposition 1). □

Theorem 7 states that even though a change in the contract does not commit, **it exists a finite sequence of operations that can be executed and will lead to a commit**.

Theorem 7. *Let op_0 be a basic operation and σ_0 the current state of the DMN contract. If $\langle op_0, \sigma_0 \rangle \rightarrow \sigma_1$ then $\exists op_1, op_2 ... op_n$ such that $\langle op_1, \sigma_1 \rangle \rightarrow \sigma_2$, $\langle op_2, \sigma_2 \rangle \rightarrow \sigma_3 ...$ $\langle op_n, \sigma_n \rangle \rightarrow \sigma_{n+1} \wedge \sigma_{n+1}$ is a state of the DMN contract model that generates an executable cross-organizational process.*

Proof. The proof proceeds by induction on the type of operation. We show by cases on the type of non-committable operations (proved in lemmas) the existence of a finite sequence of operations that ends up by an executable cross-organizational process. For example for the operation $AddRequester(q_1, Ag_1)$ presented in Lemma 8, after adding the requester, adding the supplier using the operation $AddSupplier(r_1, Ag_1)$ will lead to a commit because the processes of q_1 and r_1 will be interoperable. □

For each operation we use the same proof approach to determine whether its execution will commit the change or not. Algorithm 8 maintains the cross-organizational process executable each time a basic operation (or composite operation) is called. Basically, if an operation is proved to maintain the DMN cross-organizational process executable then it commits the change otherwise it does not.

Algorithm 8 Maintaining DMN cross-organizational process executable

Require: An invoked operation *op* to tailor the DMN
Ensure: no output
if *op* has been proved to maintain the DMN cross-organizational process executable **then**
Commit
end if

When the commit operation is invoked, it performs four basic actions:

1. Stop the running process instance while keeping the trace of this instance
2. Modify the schema of the cross-organizational process by incorporating the requested change and create a new version of the schema

3. Deploy the new process schema into the workflow engine by creating a new instance

4. At this stage, do not restart the instance from the beginning but resume it from the trace of the previous instance execution.

We give a formal definition to the **commit** operation:

Definition 23. *The precondition of the **commit** operation regarding a particular agreement is:* $Requester \neq \emptyset \wedge Supplier \neq \emptyset \wedge Component \neq \emptyset$.

*The postcondition of the **commit** operation on a particular agreement is the execution of the algorithm of workflow generation presented in the previous chapter.*

5.6 From Solution Plans To Executable Workflows

5.6.1 Approach Overview

In the previous sections we developed the formal foundations for a management framework that handles changes that could occur in the DMN at run-time. More specifically, we defined the operations that allow *coordinators* and *participants* to perform changes on the configuration of the contract.

In a DMN, a *coordinator* may decide to change multiple *agreements* in the contract. For example he may remove the *suppliers* in different agreements; he may also remove *obligations* and *characteristics* in other agreement with the perspective to replace them. To recover all these removals and resume the collaboration for the modified agreements, the *coordinator* needs to keep track of what has been removed and needs to be replaced. This mental effort could be painstaking and error-prone since it is possible that the *coordinator* forgets to add a *supplier* for a given agreement which could delay the commit of that agreement.

In order to assist *coordinators* in recovering what they have removed, we develop an approach that analyzes the current state of the contract and finds out what is missing for each agreement to become committable. Then, from this contract snapshot, the approach generates the operations that will guide *coordinators* in recovering the contract and resume the collaboration. Figure 5.6 gives an overview of the proposed approach.

The contract analysis module in Figure 5.6 is the core module of the recovery approach. To implement the functionality of this module, we use the planning-graph algorithm[NGT04]. Indeed, we have already defined the preconditions and post-conditions of management operations. The planning-graph algorithm will analyze the contract

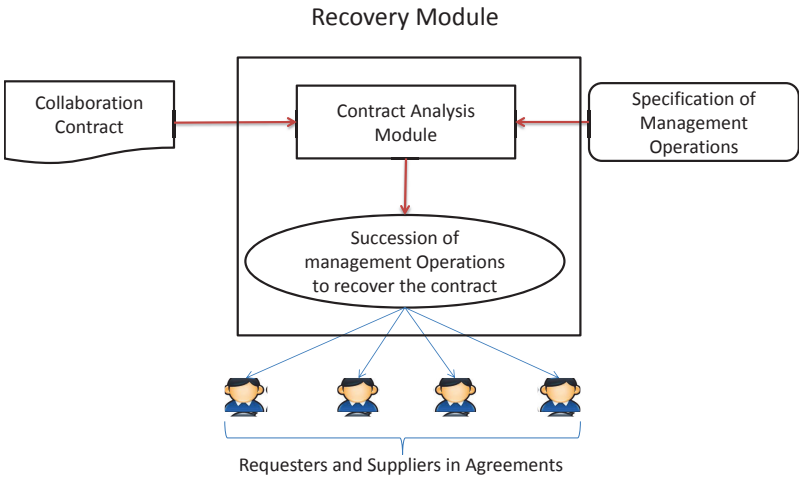


Figure 5.6: Overview of the recovery approach

specification and then from the definitions of operations, it can deduce what operations should be executed to make the contract committable. Nevertheless, the solution plan generated by the planning-graph algorithm is constituted of a sequence of sets of operations: $\Pi = \langle \pi_1, \pi_2, ..., \pi_n \rangle$. The operations in each π_i can be executed in any order. Hence, if we want to automate the execution of this sequence of sets of operations, we need to load them in a workflow engine. Nevertheless, a workflow engine cannot execute a *raw* set of operation. Each set of operations should be embedded into a construct (a workflow pattern WP) that interconnects the operations in this set. In this case, the workflow engine can parse this workflow pattern and then automate the flow of execution of the operations in the set.

Several WPs exist that can ensure a random execution of operations in a specific set π_i. For example: the **ParallelSplit** or the **SyncParSplit** [Bö7]. Thus to make a solution plan Π executable by a workflow engine, there is a need to automatically or semi-automatically decide on what pattern to use to encapsulate a particular set π_i. Our approach supports the semi-automatic selection of the appropriate WP.

5.6.2 Problem Position

The execution of the sets of operations generated by the planning-graph algorithm will resolve the problem but their sequencing is not executable by an engine. The engine needs more information on the parallelism, the sequencing or the randomness of operations in each set. The extension that we will bring to the planning-graph algorithm in the remaining of this chapter aims to enrich a solution plan Π with information related to the ordering of operations. This information can be deduced from the position of

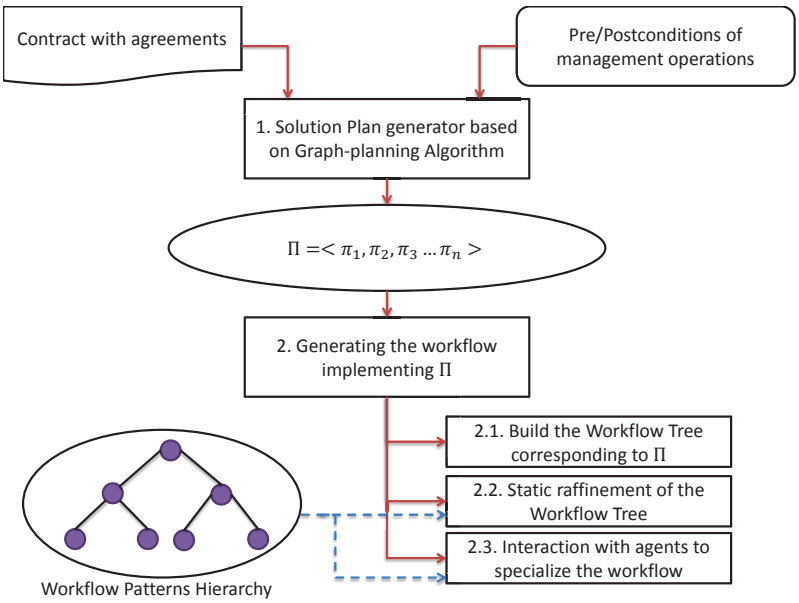


Figure 5.7: Overview of the extension of solution plans

the operations in the overall set but it may also come from the agents (partners) that will execute the operations. The result of the proposed approach is a recovery workflow that is executable by a workflow engine. It will make all agreements in the contract committable and thus helps resume the collaboration.

For our running example, if we submit the contract having the agreement depicted in Figure 5.1 (after removals), the planning-graph algorithm will analyze it and will generate the following two exclusive solution plans to :

1. $\Pi_1 = \langle \pi_1 = \{AddOConstraint, AddCConstraint, AddSupplier\} \rangle$

2. $\Pi_2 = \langle \pi_1 = \{RemoveAgreement\} \rangle$

Indeed, there are two cases for which an agreement can be committed. Either, it should contain all information (this is whet the solution plan Π_1 achieves). The other case is that the agreement is completely removed from the contract (this is what the solution plan Pi_2 achieves).

We develop a three-step approach to organize the solution plans operations into an executable workflow. Figure 5.7 gives an overview of the proposed three-step approach.

In the next subsections, we will elaborate on the three steps that generate an executable workflow from solution plans.

5.6.3 Step 1: Workflow Tree Generation

In this section, we develop the algorithm that generates the Workflow Tree (WT) from several solution plans for the same planning problem.

Definition 24. *The Workflow Tree (WT) (similar to the process tree defined in [GBno6]) is a tree structure generated from a set of solution plans* $\{\Pi_1, ...\Pi_m\}$ *and recursively defined by:*

- *Each leaf is an action (a management operation) of the solution plan* Π_j
- *Each action in the WT is defined by the tuple:* $\langle ID \in Activity, thread \in Thread, precond \in Precondition \rangle$
- *Each leaf is a child of the Block node*
- *Blocks corresponding to leaves of the same* π_i *have a common Block that is* π_i*'s Block*
- π_i*'s Block is connected to* π_{i+1}*'s Block by a* ***Sequence*** *WP*
- Π_j *sub-tree of a solution plan is connected to another solution plan* Π_{j+1} *of the same planning problem through Block construct*
- *Each node of the tree is annotated with the disjunction of its children preconditions*

To complete the construction of the WT, we need to run Algorithm 9 in order to organize the actions of each π_i to obtain the least number of commits.

Algorithm 9 Organize π_i's actions for the least number of commits

Require: the WT enriched with Blocks' preconditions$R$
Require: the commit action preconditions: precond(commit)
Ensure: a refined WT for commit
1: **for all** $\pi_i \in \Pi$ **do**
2: $actionsLeadingToCommit = \{a : a \in \pi_i \wedge \models precond(commit) \Rightarrow postcond(a)\}$
3: $Block_1 \leftarrow$ insert a common *Block* node for actions $\in actionsLeadingToCommit$
4: $Block_2 \leftarrow$ insert a common *Block* node for actions $\notin actionsLeadingToCommit$
5: insert a **Sequence** node between $Block_1$ and $Block_2$
6: **end for**

The objective of Algorithm 9 is to better refine the set of actions of a solution plan generated by the planning-graph algorithm. Indeed, the actions belonging to each set π_i can be executed in any order that will lead to the same final result. Nevertheless, to reduce the number of commits to the running workflow, we need to add more constraints on the order of actions belonging to the same π_i. This order should place the actions that do not lead to a commit in the beginning of the sequence and the actions that will lead to a commit at the end of the sequence.

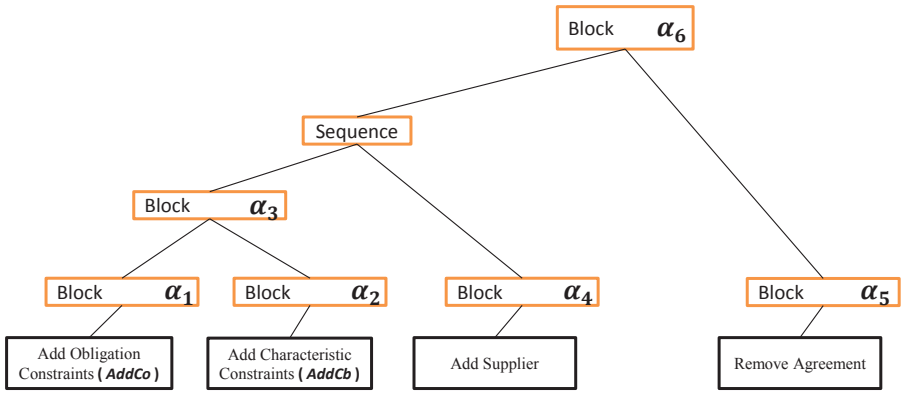


Figure 5.8: Workflow Tree Example

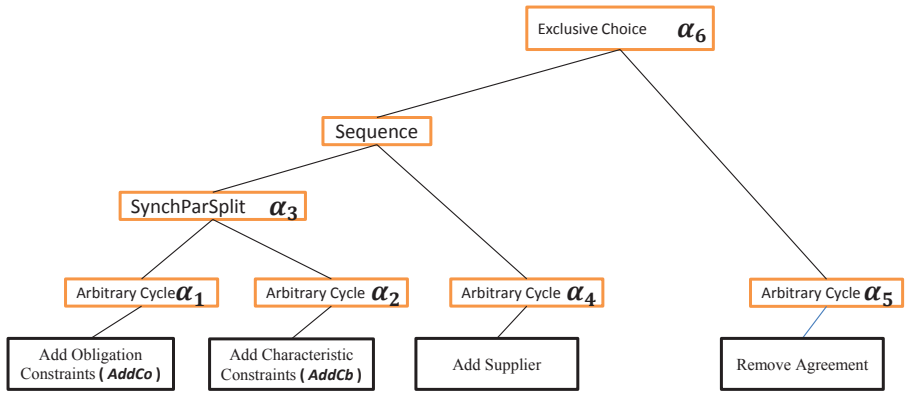


Figure 5.9: Workflow Tree after refinement

To find the actions that lead to a commit in a solution plan, we can use the following observation: the precondition of an action a that leads to a commit is a subformula of the precondition of the action **commit** (see Definition 23). Thus, the formula $precond(commit) \Rightarrow postcond(a)$ is valid (e.g. $\models A \wedge B \Rightarrow A$). Instruction 2 of the Algorithm 9 relies on this observation.

Figure 5.8 depicts the WT corresponding to the result of applying the WT generation and the Algorithm 9 on our running example. More specifically, Algorithm 9 creates a **Sequence** WP between the nodes α_3 and α_4 since the *AddSupplier* operation will lead to a commit.

5.6.4 Step 2: Refinement of *Block*s in the WT

The main algorithm of this Step (Algorithm 10) traverses recursively the WT generated in Step 1 (Figure 5.8) starting by its root to specialize the *Block* nodes interlinking the different actions in each π_i and between Π_j. It replaces the encountered *Block*s with the appropriate workflow patterns *WP*. The final result of the application of Algorithm 10

is depicted in Figure 5.9. This Algorithm uses other algorithms that we will detail subsequently.

Algorithm 10 Replace All Children by WP Patterns: `replaceChildrenBlocksByPatterns`

Require: Workflow Tree Root: root
Ensure: WT with instantiable patterns
1: **if** `NbrChildren`(root) = 0 **then**
2: return
3: **else**
4: SetOfChildren ← `obtainChildren`(root)
5: **for all** $\alpha_i \in$ SetOfChildren **do**
6: `replaceChildrenBlocksByPatterns`(α_i)
7: `determinePattern`(α_i) //Once all α_i's children have been replaced by appropriate WPs, the algorithm determines the pattern of the current node α_i
8: **end for**
9: **end if**

To determine the pattern that will replace each *Block* α_i in line 7 of Algorithm 10, we need to rely on a formal representation of WPs that allows the algorithm to automatically select the adequate WP for each encountered *Block*. We use the formalization of *WPs* in terms of Abstract State Machines (ASM) developed by [Bö7] to develop the selection algorithm.

Basically, our approach consists in extracting the ASM signature of a *Block* and then try to find the WP that has a signature equivalent to the *Block*'s signature using a formal comparison.

In the following subsection we detail the different steps of the algorithm of the procedure `determinePattern` .

5.6.4.1 Replacement of *Block* Nodes in the WT

From term evaluation definition (Definition 6 in Chapter 3), the *WT* leaves can be seen as values of terms belonging to the sort *Activity*. In order to find the pattern that replaces a given *Block* in the *WT*, Algorithm 11 generates the possible signatures that can be associated with the encountered term evaluation (line 5).

Line 5 of Algorithm 11 expects an array because a term value could be computed using different ASMs functions and thus it could belong to different ASMs sorts. Algorithm 12 aims to determine the functions that could generate the current term value as well as the sorts to which the current term value belongs. It also constructs the preconditions associated to the generated signature.

Example 12. *Referring to Figure 5.8, when applying Algorithm 12 on the term value AddCo, it*

Algorithm 11 Determine the right pattern and insert it into the WT: `determinePattern`

Require: *Block*: $\alpha_i \in WT$

Ensure: Replacement of a set of patterns in WT corresponding to the combination of *activities* children of α_i

if α_i isOfSort *Activity* **then**
  return
else
  for all a_j childOf α_i **do**
    $[\Sigma_0, ..., \Sigma_M] \leftarrow$ `generateSignature`(a_i)
  end for
  $\Sigma_{\alpha_i} \leftarrow$ `merge`$([\Sigma_0, ..., \Sigma_M])$
  var result : *ASM*
  for all *asmWP* $\in$ *WP_Tree* **do**
    appropriateOK $\leftarrow$ `checkAppropriateness`$(\Sigma_{\alpha_i}, asmWP)$
    if *appropriateOK* **then**
      if *result* $= null \vee$ `inheritsFrom`$(result, asmWP)$ **then**
        result $\leftarrow$ *asmWP*
      end if
    end if
  end for
  `tailor`$(result, \alpha_i)$
end if

Algorithm 12 Generate List of Sorts associated to a term value: `generateSignature`

Require: A term value: $Val_B(t)$

Ensure: The list of possible signatures of terms that could be associated to this value

Find all functions $\{f_1, f_2..., f_N\}$ st: $Val_B(t) \in range(f_i)_{i=1..N}$
if $\{f_1...f_N\} = \emptyset$ **then**
  Find S_a s.t. $Value(a) = Val_B(t)$
  $\Sigma_0 \leftarrow (S_0 = \{S_a\}, \Omega_0 = \emptyset)$
  `generatePrecondition`(Σ_0)
  return Σ_0
end if
for all $f_i(x_{i1}, x_{i2}..., x_{ip}) \in \{f_1, f_2..., f_N\}$ **do**
  Determine the values of each relevant x_{ij} from $Val_B(t)$ using f_i^{-1}
  for all $j = 1..p$ **do**
    $\langle S_{f_{ij}}, \Omega_{ij} \rangle \leftarrow$ `generateSignature`$(Val(x_{ij}))$
  end for
  $\Sigma_i \leftarrow (\cup_{j=1}^{p} S_{f_{ij}}, \{f_i\} \cup \cup_{j=1}^{p} \Omega_{ij})$
  `generatePrecondition`(Σ_i)
end for
Return $\{\Sigma_0, ..., \Sigma_N\}$

will generate the signature: $\Sigma_{AddCo} = (S = \{Activity = \{AddCo\}, Thread = \{coordiantor1\}\}, \Omega = \emptyset)$. *And* $\Sigma_{\alpha_1}.Precondition = (|Activity| = 1 \wedge |Thread| = 1)$.■

Determining the signature of a node α_i. Once all possible signatures of each child of a particular node α_i are generated, Algorithm 11 `merge`s them to build a signature associated to α_i (line 7). The merge is made by creating a new signature where the collection of sorts $\Sigma_{\alpha_i}.S$ is the union of collections of sorts of the generated signatures. The collection of ASM functions $\Sigma_{\alpha_i}.\Omega$ is the union of collections Ω of the generated signatures and the precondition of α_i is generated using the specificities of the cardinalities of the new collection $\Sigma_{\alpha_i}.S$ sets.

Example 13. 1. *The procedure* `merge` *generates the signature* Σ_{α_1} *for the Block* α_1 *from the signature of AddCo, where* $\Sigma_{\alpha_1} = (S = \{Activity = \{AddCo\}, Thread = \{coordiantor1\}\}, \Omega = \emptyset)$ *and* $\Sigma_{\alpha_1}.Precondition = (|Activity| = 1 \wedge |Thread| = 1)$. *The same for* Σ_{α_2} *with AddCb.*

2. *The procedure* `merge` *generates the signature* Σ_{α_3} *for the Block* α_3 *from the signatures of* α_1 *and* α_2, *where* $\Sigma_{\alpha_3} = (S = \{Activity = \{\alpha_1, \alpha_2\}, Thread = \{coordiantor1\}\}, \Omega = \emptyset)$ *and* $\Sigma_{\alpha_3}.Precondition = (|Activity| > 1 \wedge |Thread| = 1)$.■

Replacing the node α_i **with the appropriate pattern**. While traversing the tree of WP ASMs, Algorithm 11(line 9) uses the property of **state-term appropriateness** of Definition 5 in Chapter 3 to determine the right WP that can replace a particular *Block*. It uses the procedure `checkAppropriateness` described in Algorithm 13 to check whether the candidate WP ASM signature has equivalent sorts with the generated signature (line 3) and whether they have equivalent set of functions (line 4). In addition, it checks whether the preconditions of executing the WP ASM are satisfied by the preconditions of the generated signature (line 5). Since an ASM could use other ASMs as functions, `checkAppropriateness` should also check the appropriateness with these ASMs (lines 6,7,8). When Algorithm 11 finds the appropriate pattern, it inserts it into the WT (line 17).

Example 14. 1. *The procedure* `checkAppropriateness` *returns* ***true*** *when comparing* Σ_{α_1} *with the signature of the ASM of the pattern* ***Arbitrary Cycle***. *The reason is that this is the WP that uses a single activity run by a single thread (i.e.* $\Sigma_{ArbitraryCycle} = (S = \{Activity, Thread\}, \Omega = \emptyset)$ *and* $\Sigma_{ArbitraryCycle}.Precondition = (|Activity| = 1 \wedge |Thread| = 1)$*) [Bö7].*

2. *The procedure* `checkAppropriateness` *returns* ***true*** *when comparing* Σ_{α_3} *with the signature of the ASM of the pattern* ***SyncParSplit***. *This WP splits the process flow be-*

tween multiple activities ($|Activity| > 1$) while all activities are being executed by the same agent ($|Thread| = 1$) [Bö7].

■

Algorithm 13 Check appropriateness between two signatures: `checkAppropriateness`

Require: A term t signature Σ_t and a WP signature Σ_{wp}
Ensure: *True* if Σ_t and Σ_{WP} are appropriate. *False*, otherwise
1: Boolean: b_1, b_2, b_3, b_4
2: $b_4 \leftarrow True$
3: $b_1 \leftarrow (\forall S_i \in \Sigma_t.S, \exists S_j \in \Sigma_{wp}.S$ such that $S_i \equiv S_j)$
4: $b_2 \leftarrow (\forall f_i \in \Sigma_t.\Omega, \exists f_j \in \Sigma_{wp}.\Omega$ such that $f_i \equiv f_j)$
5: $b_3 \leftarrow (\Sigma_t.Precondition \wedge \Sigma_{wp}.Precondition)$
6: **for all** asm : `hasInput`(Σ_{wp}, asm) **do**
7: $b_4 \leftarrow b_4 \wedge$`checkAppropriateness`$(\Sigma_t, \Sigma_{asm})$
8: **end for**
9: return $b_1 \wedge b_2 \wedge b_3 \wedge b_4$

For performance considerations, Algorithm 11 relies on three heuristics to reduce the number of patterns to be visited in the WP tree. Indeed, Van der Aalst et al. [ATH12] claim that nearly 20% of process instances need more advanced patterns in comparison to the ones that have been identified. Thus the number of overall patterns can increase in the future. Accordingly, we need to address performance issues earlier. When traversing the WP signatures:

1. We do not check the **Selection patterns** sub-tree since there is no selection when executing the actions in a particular set π_i. The reason is that all actions of all sets of the solution plan Π should be executed to reach the goal.

2. We do not check the **Sequence patterns** sub-tree since there is no need to constrain actions of a set π_i to be executed in a particular sequence. The arbitrary order of actions execution can be ensured with the **ParallelSplit** patterns sub-tree.

3. We do not check the **Merge patterns** sub-tree. Indeed, by default each **ParallelSplit** pattern selected by Algorithm 11 must be associated with a **Merge** pattern in order to meet the definition of well-formed workflows (a formal definition of this notion is given in [ODtHvdA06]). Thus, the **Merge** patterns will be inserted automatically during the generation of the workflow model.

At this point, we have replaced all *Block* nodes in the WT using `replaceChildrenBlocks ByPatterns`. Nevertheless, the selected pattern could be not specialized enough to implement the exact control flow behavior required by the agent. In this case, to generate

the workflow, we need further information to select the most specialized pattern. This information can be provided by the agents who will exectue the actions of the workflow. Hence, we need to formally determine what information the agents should provide to achieve the selection. Step 3 computes the formal parameters that an agent (*coordinator*) should provide in order to generate the workflow that exactly fits his expectation in term of control flow.

5.6.5 Step 3: Pattern Specialization by Progressive Agent Interaction

In Step 1, we have generated the Workflow Tree WT from the solution plans Π_j. In Step 2, we have traversed the WT and have specialized the *Block* nodes using the specification of WP in term of ASM. The objective of Step 3 is to traverse the WT again in order to generate the executable workflow for the solution plans Π_j. When the algorithm of Step 3 encounters a node representing a WP, this pattern could be not specific enough since in Step 2 we did not have all necessary information to select the most specific pattern corresponding to the agent preference. Thus, the algorithm of Step 3 should include all possible children of this pattern in the Workflow model to that it generates. The decision of what pattern to instantiate will be taken at runtime while executing the workflow. In order to select the most specific pattern, we should obtain additional information on the behavior of the workflow model as desired by the agent. For this reason, there are three challenges that need to be addressed:

5.6.5.1 Challenge 1: Asking the Agent the Information to Select the Right Pattern

The information that the workflow should acquire from the agents to follow the right execution path consists in the additional parameters that differentiate between specific and generic patterns in the hierarchy of Figure 3.6. In this case the ASM formalization is very beneficial. The algorithm of Step 3 can formally compute the parameters that the workflow should ask from the agent to follow the path that leads the workflow control to the right WP. This requires: *(i)* defining the possible inheritance modes between two ASMs, and *(ii)* defining the *substraction* operation between two signatures Σ_1 and Σ_2 in order to obtain the required parameters.

Considering the ASM formalization of WP, there are three possibilities through which a pattern can inherit from another pattern:

1. The child pattern adds a set of sorts or a set of functions to the signature of the parent pattern

Inheritance mode identifier	Inheritance mode	Formalization
1	Additional sorts	$\|\Sigma_P.S\| < \|\Sigma_p.S\|$
2	Additional functions	$\|\Sigma_P.\Omega\| < \|\Sigma_p.\Omega\|$
3	Specialization of sorts	$\exists\Sigma_P.S_i \wedge \exists\Sigma_p.S_j : \Sigma_p.S_j \subset \Sigma_P.S_i$
4	Specialization of functions	$\exists f_i\Sigma_P.\Omega \wedge \exists f_j\Sigma_p.\Omega : f_j < f_i$

Table 5.3: Inheritance Modes: p inheritsFrom P

$$\Sigma_R.S = \begin{cases} \Sigma_1.S - \Sigma_2.S \\ \cup \\ \{S_k\}_{k\in K} \ s.t. \ \exists S_m \in \Sigma_1.S : S_m =^{def} S_k \wedge \exists S_l \in \Sigma_2.S : S_l =^{def} S_k \\ \wedge S_k.univ = S_m.univ \cap S_l.univ \wedge S_k \neq \emptyset \end{cases}$$
$$\Sigma_R.\Omega = \Sigma_1.\Omega - \Sigma_2.\Omega$$

Figure 5.10: Substraction of Signatures

2. The child pattern specializes a set of sorts or a set of functions from the signature of the parent pattern:

   (a) A function f' inherits from a function f if f' implements the behavior of f with additional instructions

   (b) A sort S' inherits from a sort S if S' universes are included in the universes of S

Table 5.3 formalizes the possible inheritance modes between a pattern and its children. Nevertheless, these modes can be combined and we could have a total of 15 inheritance modes.

Example 15. *The ASM **SyncParSplit** inherits from **ParallelSplit** because it specializes the sort Thread such that we always have: $Thread_{SyncParSplit} \subset Thread_{ParallelSplit}$ (inheritanceMode = 3).*■

Definition 25. *(Substraction of Signatures) Let Σ_1, Σ_2 be two signatures. The substraction of Σ_1 and Σ_2 denoted by $\Sigma_1 \ominus \Sigma_2 = \Sigma_R$ is formally defined by the specification in Figure 5.10. The next example illustrates this operation.*

Definition 26. *(Inheritance Value) The inheritance value between two WP signatures Σ_P and Σ_p s.t. p inherits from P consists of the universe of sorts $(\Sigma_p \ominus \Sigma_P).S$ and the functions of the set $(\Sigma_p \ominus \Sigma_P).\Omega$.*

Example 16. *Consider that:*

- $\Sigma_{Iterate} = \langle S = \{Activity\}, \Omega = \emptyset \rangle$
- $\Sigma_{ArbitraryCycle} = \langle S = \{Activity\}, \Omega = \{StopCriterion\} \rangle$

In this case, $\Sigma_R = \Sigma_{ArbitraryCycle} \ominus \Sigma_{Iterate} = \langle \emptyset, \Omega = \{StopCriterion\} \rangle$.

The *Substraction* operation helps us find what information the agent should provide in order to be able to instantiate the right pattern. Algorithm 14 uses the substraction operation to generate a more specific pattern during the traversal of the WT and the generation of the workflow.

Algorithm 14 Generate Specific Pattern: `generateSpecificPattern`

Require: An abstract pattern in the WT: P
Ensure: A workflow fragment to instantiate the pattern
1: **if** $|child(P)| = 0$ **then**
2: return; // the pattern has no children in the pattern hierarchy
3: **end if**
4: **for all** p subclassOf P **do**
5: $\Sigma_R = \Sigma_p \ominus \Sigma_P$
6: Create an Input Activity to ask for $\Sigma_R.S.Universes$ values
7: Create an Input Activity to ask for $\Sigma_R.\Omega$ functions
8: Assign the input activities to the right agents
9: `generateSpecificPattern`(p)
10: **end for**
11: Attach a *XOR* pattern After the input followed by the found patterns in addition to P as a default choice
12: **for all** transitions t_i out of *XOR* **do**
13: Attach to t_i the inheritance mode and the value of the signature Σ_R
14: **end for**

5.6.5.2 Challenge 2: Associating the *XOR* output flows with the Right Conditions.

As claimed above, the right pattern to be followed during the execution of the workflow is decided at run-time. For this reason Algorithm 14 attaches *XOR* patterns (line 11) in order to choose the right pattern at run-time. Moreover we need to associate the output flows of the *XOR*s with the right conditions on the inputs given by the agents:

Definition 27. *(Condition Couple) The condition associated to the XOR output flow is a couple* $\langle inheritanceMode, inheritanceValue \rangle$.

Associating the *XOR* output transitions to the couple $\langle inheritanceMode, inheritanceValue \rangle$ is sufficient to ensure the exclusion between the flows.

Proof. (By contradiction) Suppose we have two patterns p_1, p_2 using the same inheritance mode and the same inheritance values from a pattern P. Thus $\Sigma_{p1} \ominus \Sigma_P = \Sigma_{p2} \ominus \Sigma_P = \Sigma_R$ $\Rightarrow \Sigma_{p1} = \Sigma_R \cup \Sigma_P \wedge \Sigma_{p2} = \Sigma_R \cup \Sigma_P$

Since $\Sigma_R \cup \Sigma_P = \Sigma_R \cup \Sigma_P$ then $\Sigma_{p1} = \Sigma_{p2}$. Therefore, p_1 and p_2 are the same pattern. □

5.6.5.3 Challenge 3: Pattern Specialization for Parallel Control Flow Patterns

Algorithm 14 can be directly used for abstract patterns associated to a specific agent (i.e. Sequence Control Flow Patterns). However there is an entire class of patterns that is not associated to a particular agent (e.g. the parallel control flow patterns: **ParallelSplit**, **Merge** etc.). In this case, there should be a rule to allow Algorithm 14 to find the right agent for instruction in line 8.

To select the right agent we first start by formally defining a transition in a workflow:

Definition 28. *In a workflow model, a transition between two blocks is a tuple:* $\langle transitionID, from, to \rangle$ *where* $transitionID$ *is an attribute representing the identifier of the transition.* $from$ *is an attribute representing the source block of the transition, and to is an attribute representing the destination block of the transition.*

We rely on the rule defined by [DDDM11]. Indeed, Delias et al. claim that control flow patterns **ParallelSplit**[2] (and its children patterns) are handled as a form of **postactivity** processing in the attribute *from*. Accordingly, the agent who can specify what pattern to use is the agent that executes the activity in the *from* attribute of the transition specification. Additionally, Delias et al. also claim that control flow patterns *Merge*[3] (and its children patterns) are handled as a form of **preactivity** processing in the attribute *to*. Accordingly, the agent who can specify what pattern to use is the agent that executes the activity in the *to* attribute of the transition specification. Algorithm 15 determines for every **Parallel control flow block** the agent that could specify the inputs required to decide on what pattern to use:

5.7 Application Example

In this section we review how the complete workflow corresponding to the running example is generated. We limit our presentation for the solution plan Π_1 having the following set of actions (see definition of an action in Definition 24):

[2] they called them AND Split
[3] they called them AND Join

Algorithm 15 Determine the agent: `determineAgent`

Require: WT, Hierarchy of patterns, pattern: p
Ensure: Agent

```
1: if p is a ParallelSplit then
2:   if p.InputTransition.from is an Action then
3:     return p.InputTransition.from.Agent
4:   else
5:     if p.InputTransition.from is a Merge then
6:       Assign to p.InputTransition.from the agent of its corresponding ParallelSplit
7:       return p.InputTransition.from
8:     else
9:       return determineAgent(p.InputTransition.from)
10:    end if
11:  end if
12: else {The pattern p is a Merge}
13:   if p.InputTransition.to is an Action then
14:     return p.InputTransition.to.Agent
15:   else
16:     if p.InputTransition.to is a PrallelSplit then
17:       return the agent of the ParallelSplit corresponding to p
18:     else
19:       return determineAgent(p.InputTransition.to)
20:     end if
21:   end if
22: end if
```

- $\langle AddCo, coordiantor1, Co \notin Agreement \rangle$
- $\langle AddCb, coordiantor1, Cb \notin Agreement \rangle$
- $\langle AddSupplier, coordiantor1, Supplier \notin Agreement \rangle$

5.7.1 Execution of the `determinePattern` Algorithm 11 to Replace *Block*s

This algorithm starts by determining the signatures of the ASMs corresponding to the actions *AddCo*, *AddCb*:

- $\Sigma_{AddCo} = (\left\{ \begin{array}{c} Activity = \{AddCo\} \\ Thread = \{coordiantor1\} \\ Preconditions = \{Co \notin Agreement\} \end{array} \right\}, \Omega = \emptyset)$
- $\Sigma_{AddCb} = (\left\{ \begin{array}{c} Activity = \{AddCb\} \\ Thread = \{coordiantor1\} \\ Preconditions = \{Cb \notin Agreement\} \end{array} \right\}, \Omega = \emptyset)$

Once the signature of *AddCo* (respectively of *AddCb*) is generated, the `determinePattern` can replace the *Block* α_1 (respectively α_2). In this case, the `determinePattern` executes the instructions starting from line 9. Since $|childOf(\alpha_1)| = 1$ and with the assistance of the identified heuristics, the resulting signature of α_1 is:

$$Array(\Sigma_{\alpha_1}) = (\left\{ \begin{array}{c} Activity = \{AddCo\} \\ Thread = \{coordiantor1\} \\ Preconditions = \{Co \notin Agreement\} \end{array} \right\}, \Omega = \emptyset) \tag{5.5}$$

Respectively the signature of α_2:

$$Array(\Sigma_{\alpha_2}) = (\left\{ \begin{array}{c} Activity = \{AddCb\} \\ Thread = \{coordiantor1\} \\ Preconditions = \{Cb \notin Agreement\} \end{array} \right\}, \Omega = \emptyset) \tag{5.6}$$

Checking the appropriateness of signatures $\Sigma_{\alpha_1}, \Sigma_{\alpha_2}$ with the workflow patterns in the WP hierarchy, we obtain the *ArbitraryCycle* pattern. Thus, the new WT is illustrated in Figure 5.11:

We execute the same algorithm (`determinePattern`) on α_3, we first obtain the following array of signatures:

$$Array(\Sigma_{\alpha_3}) =$$

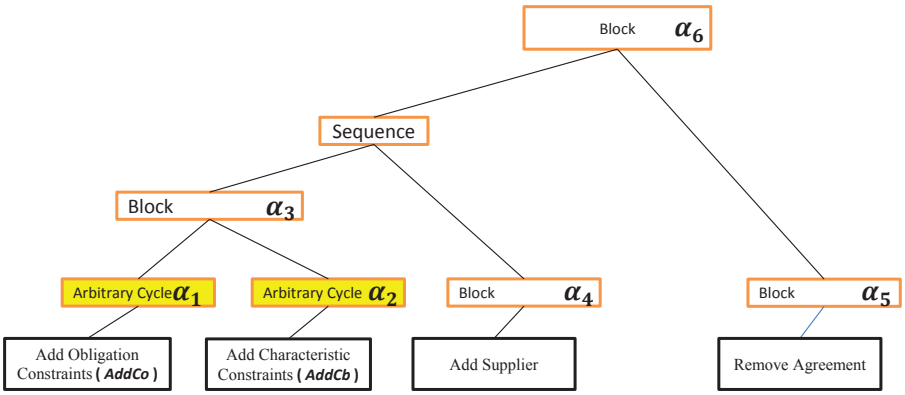


Figure 5.11: New version of the WT

$$(\Sigma_{\alpha_{3_1}} = (\left\{\begin{array}{c} Activity = \{\alpha_1, \alpha_2\} \\ Thread = \{coordiantor1\} \\ Preconditions = \{Co \notin Agreement \vee Cb \notin Agreement\} \end{array}\right\}, \Omega = \{TriggerExec(t,a)\}),$$

$$\Sigma_{\alpha_{3_2}} = (\left\{\begin{array}{c} Activity = \{\alpha_1, \alpha_2\} \\ Thread = \{coordiantor1\} \\ Preconditions = \{Co \notin Agreement \vee Cb \notin Agreement\} \end{array}\right\}, \Omega = \{TriggerExec(a)\})) \quad (5.7)$$

Again, using the **state-term appropriateness** relationship between signatures, $\Sigma_{\alpha_{3_1}}$ will not be appropriate with the **ParallelSplit** pattern. The reason is that this pattern requires as many agents as the number of actions. However $\Sigma_{\alpha_{3_1}}$ has two actions that should be executed in parallel by the same agent *coordinator1* (the function $TriggerExec(t,a)$) which is impossible. Thus, this signature is excluded.

The algorithm selects $\Sigma_{\alpha_{3_2}}$ since its signature is *appropriate* with the signature of **SyncParSplit** pattern. It replaces the *Block* of α_3 with **SyncParSplit** pattern in the WT.

Notice that algorithms developed in the literature do not perform such deep analysis, they would simply have chosen the pattern **ParallelSplit** since it is the most generic pattern even though, as we see here, it is not the appropriate one.

The final WT after the completion of Algorithm 11 is depicted in the Figure 5.9.

5.7.2 Generation of the Workflow Model

Once the WT has been generated, Algorithm 14 will generate the workflow model corresponding to this WT. For nodes α_1, α_2 we have replaced them with the **ArbitraryCycle** pattern because this was the most specific pattern appropriate with the generated signatures. Since **ArbitraryCycle** has children, then in order to further specialize the selected pattern, we generate the corresponding workflow segment and attach it to the *XOR* construct in order to select the most appropriate specialized pattern at run-time.

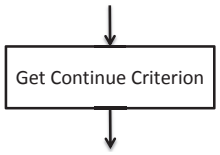


Figure 5.12: Action to obtain the value of the parameter to follow the most appropriate pattern at run-time

Algorithm 14 calculates the following differences between signatures when traversing the WP hierarchy of Figure 3.6:

$$\Sigma_{R_1} = \Sigma_{StructuredLoopPostTest} \ominus \Sigma_{ArbitraryCycle} = (S = act, \Omega = \emptyset) \tag{5.8}$$

The inheritance mode of Σ_{R_1} is 3 of Table 5.3. For the mode 3, the reason is that the **ArbitraryCycle** can be associated to several actions while the **StructuredLoopPostTest** is associated to only 1 action.

$$\Sigma_{R_2} = \Sigma_{StructuredLoopPreTest} \ominus \Sigma_{ArbitraryCycle} = (S = act, \Omega = \{ContinueCriterion\}) \tag{5.9}$$

The inheritance mode of Σ_{R_2} is 3 and 4. The reason for the mode 4 is that the **StructuredLoopPreTest** has an additional function: **ContinueCriterion** in its signature.

From Σ_{R_1} and Σ_{R_2}, we can compute what parameters the agent (*coordiantor1*) should provide in order to determine what pattern to follow during the execution of the workflow. Accordingly, we generate the input action as depicted in Figure 5.12 in the final workflow.

At this stage, the workflow has the necessary information to instantiate the right pattern at run-time. From here we include a *XOR* pattern in order to select the right pattern to follow depending on the data entered by the agent. We associate to each flow the adequate condition. The output for this example is depicted in Figures 5.13 5.14 5.15.

Once the actions are encapsulated in the right patterns, we generate the fragments for the remaining *Blocks* using the same algorithm. Considering the pattern **SyncParSplit**. This pattern is a leaf in the hierarchy tree. Thus, there is no need to specialize it. Nevertheless since we must generate well-defined workflows, we need to associate to each **SyncParSplit** a merge. Accordingly we need to insert input activities to obtain information in order to select the **Merge** pattern that fits the agent needs. In this example the agent that provides this information is automatically determined by the

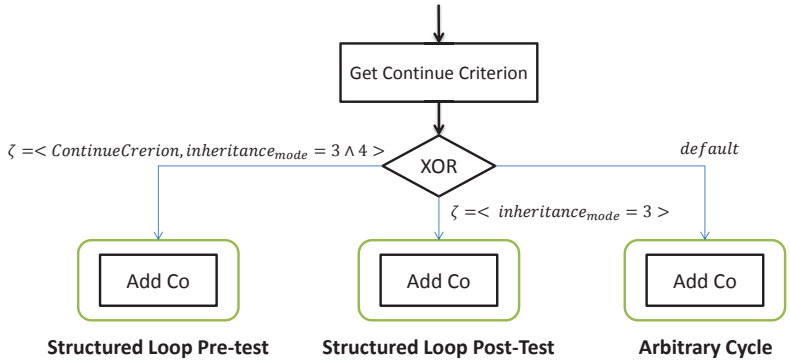


Figure 5.13: Add Co Block

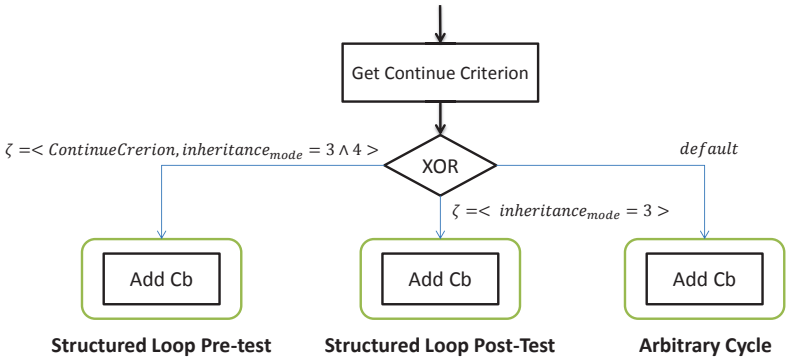


Figure 5.14: Add Cb Block

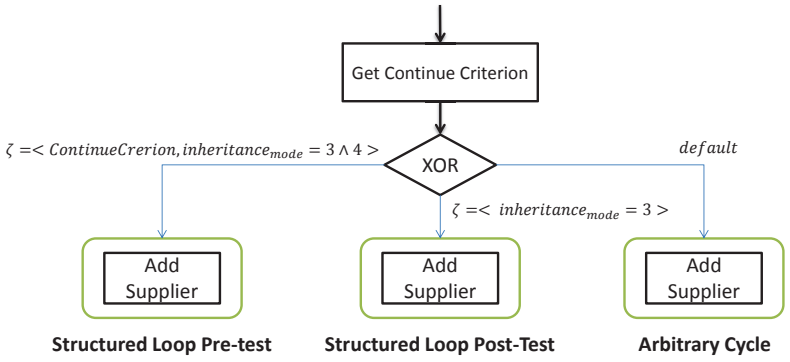


Figure 5.15: Add Supplier Block

`determineAgent` algorithm that is *coordiantor1*.

Without loss of generality we focus on a specific **Merge** sub-tree namely the **Structured N-out-of-M** in order to generate the possible merges corresponding to the agent needs.

There are four possible merges that can be used to merge the flows of the **SyncPar-Split**. The **Merge**, **Structured-N-out-of-M**, **Generalized AND-Join**, and **Discriminator**. The signature of these patterns is a follows [Bö7]:

- $\Sigma_{Merge} = (S = \{Activity\}, \Omega = \{exec, MergeEv_0, PROCEED, RESET\})$
- $\Sigma_{Structured-N-out-of-M} = (S = \{Activity\}, \Omega = \{exec, MergeEv_1, PROCEED, RESET\})$
- $\Sigma_{GeneralizedAND-Join} = (S = \{Activity\}, \Omega = \{exec, MergeEv_2, PROCEED, RESET\})$
- $\Sigma_{Discriminator} = (S = \{Activity\}, \Omega = \{exec, MergeEv_3, PROCEED, RESET\})$

Since **Structured-N-out-of-M** inherits from **Merge**, the workflow needs to exclude between these two patterns at run-time. Thus the algorithm calculates the substraction of their signatures:

$$\Sigma_{Structured-N-out-of-M} \ominus \Sigma_{Merge} = (S = \emptyset, \Omega = \{MergeEv_1\}) \tag{5.10}$$

Algorithm 14 inserts input activities in the workflow in order to obtain the values of $MergeEv_1$ from *coordinator1*. Nevertheless, $\Sigma_{Structured-N-out-of-M}$ has other inheriting patterns. The algorithm considers them too by calculating the following substractions and inserting the corresponding input activities in the workflow:

$$\Sigma_{GeneralizedAND-Join} \ominus \Sigma_{Structured-N-out-of-M} = (S = \emptyset, \Omega = \{MergeEv_2\}) \tag{5.11}$$

$$\Sigma_{Discriminator} \ominus \Sigma_{Structured-N-out-of-M} = (S = \emptyset, \Omega = \{MergeEv_3\}) \tag{5.12}$$

Figure 5.16 shows the interconnection between patterns and the input activities in the workflow.

The previous examples explain how each fragment of the process can be generated. Figure 5.17 depicts the model of the complete workflow generated by our algorithms.

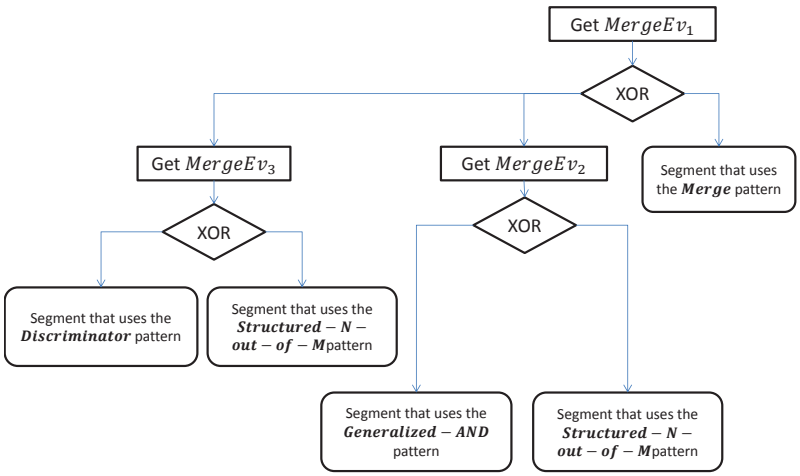


Figure 5.16: Merge patterns and their interconnections

5.8 Evaluation

5.8.1 Evaluation of the Number of Input activities

We establish a measurement to estimate the benefits of running the step 2. Indeed, if we start the generation of the workflow from WT without replacing abstract *Block* nodes, we should ask the agent to enter the inputs of all 43 patterns in order to decide what pattern to use. The step 2 of our algorithm reduces the number of required inputs from the agent. Figure 5.18 gives a comparison of the number of inputs required when step 2 is executed, and the number of inputs required when step 2 is not executed. We observe that step 2 contributes significantly in reducing the number of required inputs from the agent. This increases the automation level of the workflow generation.

5.8.2 Comparison with Existing Work Regarding the Size of the Generated Workflow

We have compared on three cases the number of constructs of the executable workflow generated by our algorithms and the number of constructs that could be generated using existing algorithms such as the one developed by [HVN+12]. Figure 5.19 depicts this comparison. Although the number of constructs of our workflow evolves polynomially in comparison to the linear evolution of the number of constructs of [HVN+12]'s algorithm. This proves that much workflow semantic is not captured when using the previous work algorithms.

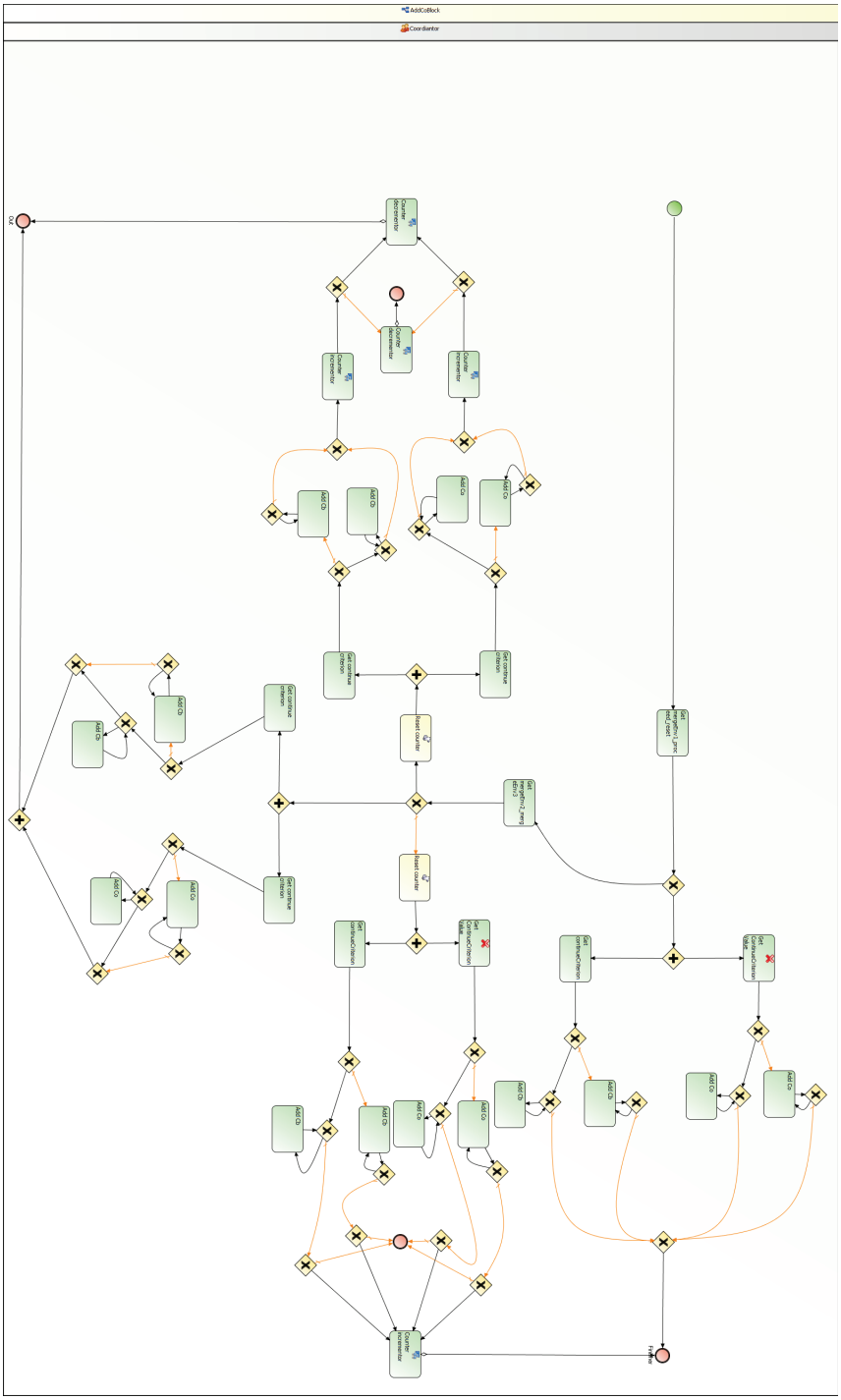


Figure 5.17: The generated recovery workflow

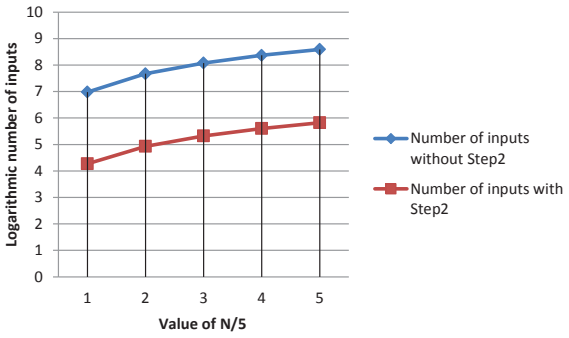


Figure 5.18: Step 2 evaluation

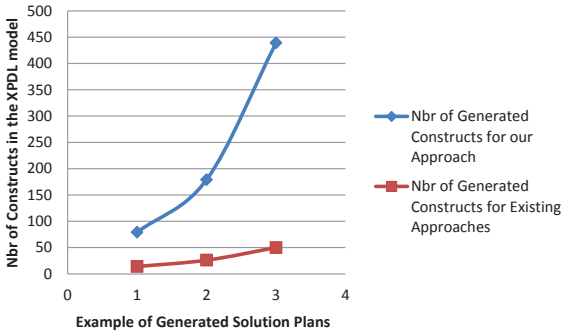


Figure 5.19: Evolution of the number of XPDL constructs in the final workflow model

5.9 Conclusion

In this chapter we enriched the product-based contract framework with a set of management operations. These operations allow partners to tailor the configuration of the collaboration contract at run-time. Furthermore partners can compose existing operations to create more complex operations.

When a management operation is executed its effect, if committed to the running cross-organizational process, can create mismatches in this process model. Therefore, we developed an algorithm that checks the existence of mismatches after the execution of a management operation. If a mismatch exists, the commit will not be realized until other management operations are executed which leads eventually to the removal of the found mismatches. We proved that if a commit of an operation would create a mismatch in the running cross-organizational process, there is always a suite of finite operations that make all changes committable. Nevertheless, this suite of operations is not always easy to find especially if there are multiple agreements in the contract that. To help partners manage the contract, we developed an approach that generates a recovery workflow. Basically, this approach analyzes the current (non-committable) state of the contract and generates a recovery workflow that supports partners in incorporating the

changes that lead to the commit of all agreements in the contract. We extended the planning-graph algorithm [NGT04] to generate this executable recovery workflow.

Until now we provided *coordinators* in the DMN with a framework to model the collaboration by constructing a flexible collaboration contract. We enriched this framework with management operations that shield partners not associated to modifications in the DMN from their impacts. To complete our framework, we enrich it by providing *coordinators* with an efficient monitoring capability that is the subject of the next chapter.

Chapter 6

Patterns for Monitoring Product Level Agreements

Contents

6.1 Introduction

Continuous monitoring of a business process is defined as the set of methodologies and tools to collect and disseminate relevant information about the process execution to interested stakeholders simultaneously with, or within a reasonably short period after the occurrence of relevant events in the process [CV11].

During collaborative product design, business entities in an extended enterprise need information about the activities taking place in the business landscape (at internal units, partner organizations, or third parties) so that they can react and/or adapt to them. Thus, business entities need mechanisms that ensure the collection of relevant data from the environment [CAV12].

Monitoring is a well-known strategy that aims to inform business entities (subscribers) interested in watching the unfolding of the collaboration between other business entities (publishers). The publish/subscribe interaction paradigm provides subscribers with the ability to express their interest in an event or a pattern of events [EFGK03]. Several approaches were developed to perform monitoring using the publish/subscribe paradigm and a thorough survey can be found in [EFGK03] and [Tar12]. More specifically, a number of monitoring systems aim to filter the incoming events depending on a specification set by the subscriber. They consider that a single event contains all required attributes by the subscriber and it would be sufficient to efficiently deliver this event to the subscriber [ASS+99]. Although this approach is practical, it remains limited since attributes may arrive at different rates and not in a single block. Other systems consider that events could come at different rates and address the problem of events correlation by waiting that all event attributes arrive before notifying the subscriber [BMB+00]. However, these systems consider that there is no processing to be performed on the incoming events and thus they deliver them to the subscriber as they arrive. In the next subsection we provide a motivating example that illustrates the importance of processing events during monitoring and the shortcomings of the current approaches.

6.2 Running Example

In this example we carry on with the scenario of the *Fuselage* design and we extend it to the *Aircraft* design. We consider that now the *coordinator* identified by **D** is the aircraft architect and he wants to monitor how the aircraft structure will evolve. The collaborative design of the aircraft structure involves the engineer **(A)** and the (*Wing+Empennage*)

designer **(B)**, while the collaborative design of the *Fuselage* will involve **A** and the *Fuselage* designer **(C)**. The aircraft architect **(D)** may need to analyze how the aerodynamic property of the whole aircraft structure (*Wing* + *Empennage* + *Fuselage*) evolves at real-time. In this case **D** shall be able to subscribe to the exchanged messages between **A** and **B**) and the messages exchanged between (**A** and **C**). Additionally, he shall be able to *define on-the-fly processing* by combining incoming messages exchanged during the collaboration to obtain a high-level characteristic of the whole *aircraft aerodynamic property*.

For instance, **A** will iteratively exchange different values of (*resistivity* (x_1) and *curvature* (x_2)) with **B** and (*material* (x_3) and *length* (x_4)) with **C** until achieving a satisfactory configuration of the *Wing+Empennage* as well as the *Fuselage*. **D** would like to run a composition of simulation services on the exchanged messages as depicted in Figure 6.1 to monitor the whole aircraft aerodynamic property. Using EPL (Event Processing Language)[1] languages and their associated mechanism to call external services (User Defined Functions), **D** could write the following query:

```
   Insert into Aircraft_Aerodynamic_Evolution
Select s8(s1(x1,x3),s7(s4(s2(x1,x3)),s6(s5(s3(x4,x1),x3)))) as AAE
From x1^topic,x2^topic,x3^topic,x4^topic
Where AAE > 50
```

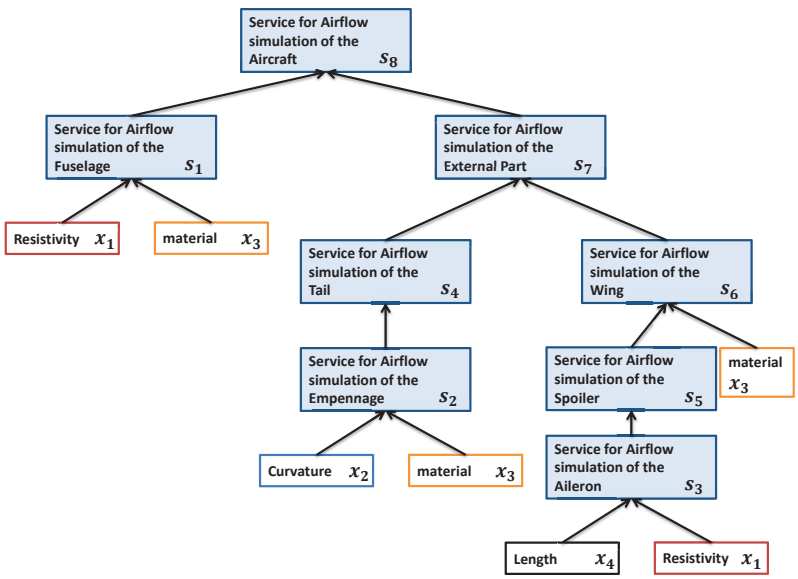


Figure 6.1: A tree representation of services composition

[1] http://esper.codehaus.org/

6.3 Challenges

6.3.1 Providing an Efficient Execution Plan

The advantage of writing the previous query to monitor the evolution of the aerodynamic property of the whole aircraft structure is that the query hides composition details from the aircraft architect **D**. Nevertheless, if we consider that the rate of occurrence of x_1, x_3, x_4 $>>>$ x_2, the execution plan corresponding to the monitoring query runs the services $s_1...s_8$, specified in the query, only when the first occurrence of x_2 occurs even though some services are totally independent from x_2: (s_1, s_3, s_5, s_6). This is what happens when we run this query using monitoring software like Esper.

The shortcoming of the (naive) execution plan of this query is that it waits that all events that are expected occur to start the processing of the composition. In this case the execution engine performs a *post-order* traversal of the tree in Figure 6.1 and calls the associated services. During the processing of these services for a particular set of values of $x_1..x_4$, other instances may arrive. In this case the execution engine is obliged to block them and not include them in $x_1^{topic}...x_4^{topic}$ until finishing the current execution. Once it is done, the execution engine is obliged to include and process the new instances of $x_1...x_4$ sequentially one by one. This leads to: *(i)* the execution of the same service several times for the same inputs and *(ii)* sequential processing of messages. Accordingly, the query processing time increases drastically.

6.3.2 Overview of the Proposed Approach

In the present contribution, we leverage the processing of events by a composition of services. We use Colored Petri Nets (CPN) [Jen94] to define a set of patterns that will be used to decompose a service composition specification that processes the incoming events. The decomposition produces a set of sub-compositions that can be executed in parallel. Consequently, we maximize the utilization of the monitor resources and deliver the final result to the subscriber as fast as possible.

The contributions described in this chapter can be summarized as follows:

- We define a set of CPN patterns that will be used to decompose a service composition specification in order to execute its sub-parts in parallel
- We effectively correlate the instances of the separated messages when aggregating the decomposition, and we formally prove that this aggregation is correct when using our model;

- We formally prove that if a sub-composition is executed once for a set of instances of messages, it will not be re-executed again for the same instances. This is highly beneficial since some services could take long time to return results and it would be better to avoid re-executing them for the same inputs.
- We extend our contract framework developed in the previous chapters in order to include this monitoring capability
- We compare the performance of our prototype with existing software that is Esper.

This chapter is organized as follows. Section 6.4 gives some preliminaries on Colored Petri Nets (CPN) since it is the formalism used in this chapter. Section 6.5 details the patterns used to decompose a service composition into parallel executable threads with their formalization in CPN. Section 6.6 elaborates on how the CPN is generated from the service composition tree and proves important properties of the generated CPN. Section 6.8 presents the integration of the developed monitoring approach with the product-based contract discussed in the two previous chapters. Section 6.9 conducts experiments on the monitoring mechanism and gives a quantitative comparison it with Esper. Finally section 6.10 concludes the chapter.

6.4 Preliminaries on Colored Petri Nets

Petri nets, on one hand, provide the primitives for the description of the synchronization of concurrent processes. Programming languages, on the other hand, provide the primitives for the definition of data types and the manipulation of their data values. A Colored Petri Net (CPN)[Jen94] aims at combining the capabilities of Petri nets and programming languages in order to create a more powerful language. This language takes into account data types and their values when describing the synchronization of concurrent processes. The idea of CPNs is to extend ordinary Petri nets with:

- Global declarations (Types also called color sets, values, variables and functions)
- Types on places
- Token elements
- Guards on transitions

Formally a CPN is a tuple $CPN = (\Sigma, P, T, A, N, C, G, E, I)$ where:

1. Σ is a finite set of non-empty **types**, also called color sets

2. P is a finite set of **places**
3. T is a finite set of **transitions**
4. A is a finite set of arcs such that : $P \cap T = P \cap A = \emptyset$
5. N is a **node** function. It is defined from A into $P \times T \cup T \times P$
6. C is a **color** function. It is defined from P to Σ
7. G is a **guard** function. It is defined from T into expression such that:
   - $\forall t \in T : [Type(G(t)) = Boolean \wedge Type(Var(G(t))) \subseteq \Sigma]$
8. E is an **arc** expression. It is defined from A into expressions such that:
   - $\forall a \in A : [Type(C(p)) \wedge Type(Var(E(a))) \subseteq \Sigma]$ where p is the place of $N(a)$
9. I is an initialization function. It is defined from P into closed expressions such that:
   - $\forall p \in P : [Type(I(p)) = C(p)]$

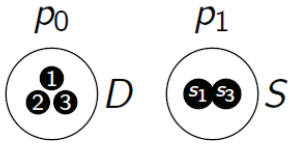
6.4.1 Global Declaration

A CPN is associated to a global declaration. In the global declaration we define all types used in the net. In addition we can define variables and operations on the types. The following listing gives an example of a global declaration for a CPN.

```
   val n = 5;
color D = index d with 1..n declare ms;
color PR = product D * D declare mult;
fun diff(x,y) = (x<>y);
color MES = subset PR by diff declare ms;
var s,r :  D;
```

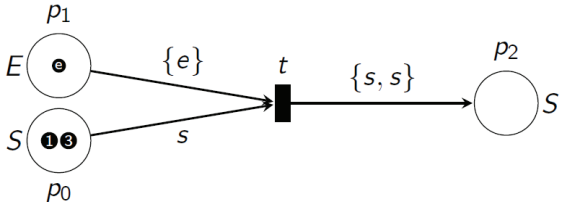
6.4.2 Types on Places

In a CPN, each place has an associated type that determines the kind of data tokens that the place may contain. Figure 6.2 shows two places: tokens in the first place are of type D and tokens in the second place are of type S.



$$D = \{1, 2, 3, 4\} \quad S = \{s_1, s_2, s_3\}$$

Figure 6.2: Places and their types



$$S = \{1, 2, 3\} \quad E = \{e\}$$
```
var s : S;
```

Figure 6.3: CPN before transition triggering

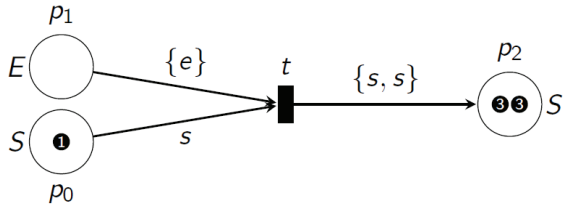
6.4.3 Token Element

A token element is a pair (p, c) where $p \in P$ and $c \in C(p)$

6.4.4 Arc Expressions

In a CPN, all arcs are augmented with an arc expression. An arc is always associated with exactly one place p. The type of an arc must be a multiset of $C(p)$. The arc expressions determine the number of tokens to remove from the input places and add to the output places. Complex arc expressions are allowed, e.g. *case* x *of* $p \Rightarrow \{e\} | \, q$.

Figure 6.3 gives an example of a CPN with arc expressions and Figure 6.4 gives a snapshot of this CPN when the transition is triggered.



$$S = \{1, 2, 3\} \quad E = \{e\}$$
```
var s : S;
```

Figure 6.4: CPN after transition triggering that upheld the arcs expressions

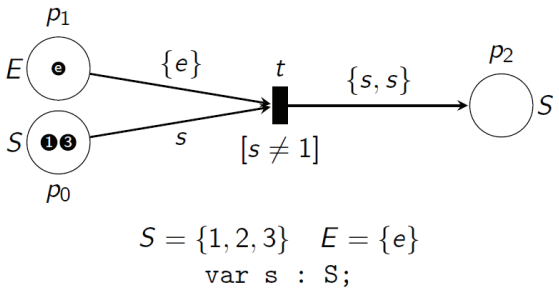


Figure 6.5: Guard associated to a transition

6.4.5 Guards on Transitions

A transition in a CPN can be associated to a guard that prevents its triggering for certain token values. Figure 6.5 gives an example of a guard associated to the transition t that prevents its triggering when the token element in S is bound to the value 1.

In the next section we elaborate on our CPN patterns to decompose a service composition.

6.5 CPN Patterns for Decomposing a Service Composition

To develop our monitoring system, we need to detect whether an event contributes to the processing of the monitoring query even partially. During the collaboration between (**A** and **B**) and (**A** and **C**), incoming events $x_1...x_4$ are inserted into their corresponding topics: $x_1^{topic}...x_4^{topic}$. If an incoming event can be used to call some services, then the system call these services and saves the intermediate results into their own topics (i.e. $s_1^{topic}..s_7^{topic}$).

To achieve this, we use CPN constructs to represent the decomposition of the services composition into a set of patterns that can be executed in parallel. CPNs have been chosen because, as we have seen, they have a graphical representation, they have a precise semantic that can be translated to or directly implemented in other languages e.g. Esper EPL queries, or CEA queries [BMB+00].

The principle of our approach consists in considering the incoming events (e.g. $x_1..x_4$) as token elements to be inserted into the CPN places (e.g. $x_1^{topic}..x_4^{topic}$). The CPN arcs can be annotated with service calls that wait for the required events to occur prior to their invocation.

6.5.1 Adaptation of CPN Definition

To model our patterns using the CPN formalism, we need to introduce some changes into the definition of some of its building blocks.

6.5.1.1 Extending the definition of CPN Places

We will adapt the default CPN definition. Indeed, in our case, places will contain the incoming instances of the arguments $x_1..x_4$ used to call services. Places will also contain intermediate results that are outputs of some services and constitute inputs for other services. To identify the token elements that contributed in computing the token elements of a given place p, each place p is associated to a *placeType*. A *placeType*[] is an array of *Booleans*. In this array, if the token element of type x_i has been involved in deriving the token elements of this place, then $p.placeType[i] = true$. For example:

- The *placeType* of the place containing instances of the message x_1 is the array: [**true**, $false$, $false$, $false$].
- The *placeType* of the place containing instances of the output of s_1 is the array: [**true**, $false$, **true**, $false$] since x_1 and x_3 instances are the inputs to compute the outputs of s_1 (see Figure 6.1).

6.5.1.2 Extending the definition of Token Elements

We add an identifier to token elements pertaining to places. Indeed, the default definition of a CPN defines a token element as a couple (p, c). Nevertheless, in a single place p, we could have multiple token elements with the same color. Thus a place becomes a multi-set of token elements. To avoid this (because we need to differentiate between token elements even though they have the same color), we add a third element that is the token element identifier in order to differentiate between all token elements. Thus a token element becomes defined by the tuple $< p, c, ID >$. Since a token element belongs to its place, then it must either be the input associated to this place ($x_1..x_4$) or it must be derived from the inputs associated to this place. Accordingly, we define the identifier of a token element ID as an array of integers where:

- $placeType[i] = true \Rightarrow ID[i] \geq 0$
- $placeType[i] = false \Rightarrow ID[i] = -1$

The reason for this adaptation is that an input x_i in a services composition can appear multiple times in a single tree (e.g. in Figure 6.1, x_3 that is an input for s_4 and s_6).

Since services corresponding to sub-trees of the original composition may be invoked in parallel (e.g. s_4 and s_6), their results will be combined to run another service (s_7). Thus we must keep track of the token elements that result from running s_4 and s_6. The token element identifier allows us to combine exclusively token elements that are calculated from the same common inputs.

More formally, during the execution of the service s_7 for two input places s_4^{topic} and s_6^{topic}, our approach will exclusively combine token elements $te_i \in s_4^{topic}$ and $te_j \in s_6^{topic}$ if $te_i.ID_{x_3} = te_j.ID_{x_3}$.

6.5.2 Patterns for Splitting Service Compositions

In this section, we define CPN patterns that will be used to decompose a service composition specification. These patterns aim to parallelize as much as possible the execution of the composition. Observing the decomposition tree in Figure 6.1, we notice that there are five types of sub-trees:

- Input leaf (constant or variable input)
- Single operator service (e.g. s_4)
- Duplication of an input (e.g. x_3 that is the input of s_1 and s_2)
- Two operator service (e.g. s_1)
- N operator service
- The preconditions/postconditions related to services execution.

From these different types of sub-trees we derive their corresponding patterns expressed in CPN formalism.

6.5.2.1 Input Pattern

A leaf in the composition tree is represented by a place in the CPN. When a new event occurs, we create a new token element that has as a color this event payload and that has a single element array *id* that has not been assigned before. When a token element is consumed by a service, it should be returned to its initial place. This pattern is formalized in Figure 6.6. This pattern is used to represent leaves that are required by one or N operator service ($N >= 2$).

6.5.2.2 Single operator Pattern

A sub-tree in the composition tree representing a single operator service is represented by the CPN formalized in Figure 6.7. The place A contains the instances of the input parameter of the service s. When a new instance comes, if it upholds the transition guard that represents the precondition of the service s, then the service s is called and a new token element is created and inserted into the place B. This token element is represented by the tuple: $\langle B, s(\alpha.color), \alpha.ID \rangle$. Notice that the token element generated by s keeps the same ID of the input token element since it no longer exists in A.

Removing the input token element has no effect on other services that wait for the same event, because they have their own copies of the event in other places (c.f. the duplication pattern).

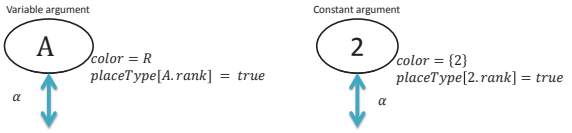


Figure 6.6: Input pattern

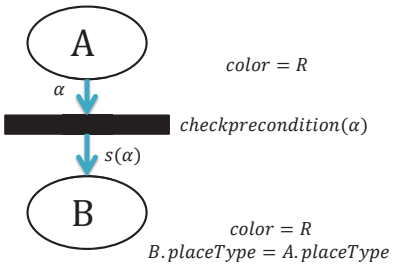


Figure 6.7: Single service input pattern

Algorithm 16 details the steps of this pattern.

Algorithm 16 Single Input Pattern Implementation

Ensure: Correct data in the Output Place

```
1:  while TRUE do
2:    Inputs ← InputPlace.getPlaceTokens()
3:    for all te_i ∈ Inputs do
4:      if checkPrecondition(te_i) then
5:        te_o.ID ← tei.ID
6:        te_o.color ← s(te_i.color)
7:        outputPlace.addToken(te_o)
8:      end if
9:      remove te_i from Inputs
10:   end for
11: end while
```

6.5.2.3 Duplication pattern

An input argument can appear in different sub-trees (e.g. x_1 is an input in the sub-tree of s_1 and also in the sub-tree of s_3). Since services in two separate sub-trees can

be run in parallel, we need to give to each sub-tree an instance of the required input. Nevertheless, we should pay attention to maintaining the coherency of the final result by combining only the instances having the same IDs in subsequent calculations as we will see in section 6.6. This pattern, formalized in Figure 6.8, ensures an effective parallelism in running parallel services. Indeed, even though a service makes a delay in releasing an input token element, the other services requiring the same token element will not be impacted since they have their own copies of that token element. Additionally, this pattern is very helpful to avoid re-invoking a service that appears in different spots of the tree and that requires the same inputs. In this case, the service will be called once and the result will be shared by all subsequent services expecting it.

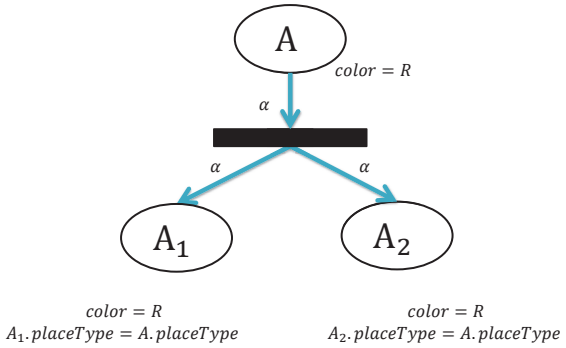


Figure 6.8: Duplication pattern

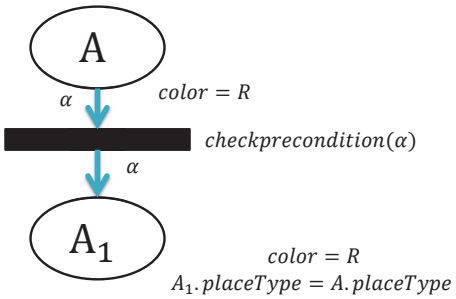


Figure 6.9: Pre/postcondition pattern

Algorithm 17 details the processing steps of this pattern.

Algorithm 17 Duplication Pattern Implementation

Ensure: Correct data in the Output Place

```
1: while TRUE do
2:    Inputs ← InputPlace.getPlaceTokens()
3:    for all outputPlaces_i do
4:       te_o.id ← te_i.id
5:       te_o.color ← te_i.color
6:       outputPlace_i.add(te_o)
7:       remove te_i from Inputs
8:    end for
9: end while
```

6.5.2.4 Two operator pattern

The two operator pattern is formalized in Figure 6.10. The places above the transition (A_1 and A_2) have input and output arcs. The input arcs aim to keep all token elements

in these places and not remove them. Indeed, if a new token element arrives to the place A_1, it must be combined with all token elements that already arrived at the place A_2 when calling the service s and vice versa. The transition t is associated with a guard specifying that if two token elements have been combined before, we no longer need to combine them again ($\neg combined(\alpha_1, \alpha_2)$). This guard avoids recalling a service for the same inputs. Moreover, if there is a precondition associated to the service that concerns **its two inputs**, then, it will be specified in the transition $checkprecondition(\alpha_1, \alpha_2)$. The output arc from the transition t is augmented with a complex expression[2]. In addition to the satisfaction of the service precondition, there are two cases when two token elements can be combined:

1. When the colors of the token elements $\alpha_1 \in A_1, \alpha_2 \in A_2$ have been computed from at least one common input. Formally: $\exists I = \{i_1, .., i_n\}$ such that the following two conditions are satisfied:

   (a) $A_1.placeType[i_1] = A_2.placeType[i_1] = true$

   (b) ...

   (c) $A_1.placeType[i_n] = A_2.placeType[i_n] = true$

   AND

   (a) $\alpha_1.ID[i_1] = \alpha_2.ID[i_1]$

   (b) ...

   (c) $\alpha_1.ID[i_n] = \alpha_2.ID[i_n]$

   AND

   (a) $\forall j \notin I : A_1.placeType[j] \neq A_2.placeType[j]$

2. When the color of the token elements $\alpha_1 \in A_1, \alpha_2 \in A_2$ have been computed from two disjoint sets of inputs. Formally: $\forall i : A_1.placeType[i] \wedge A_2.placeType[i] = false$.

These conditions are fundamental to ensure a correct correlation of intermediate results.

The result of calling the service s is a new token element α in the place B. The place $B.placeType$ captures that the token elements that reside in B come from the token elements of A_1 and A_2.

[2]The syntax and semantics of such expressions are given in [Jen94]

The token element α has the color $s(\alpha_1.color, \alpha_2.color)$ that is the result of calling the service s. In addition the identifier of this token element is the identifiers of the token element α_1 associated to the identifier of the token element α_2 (see lines 8 to 18 of Algorithm 18) .

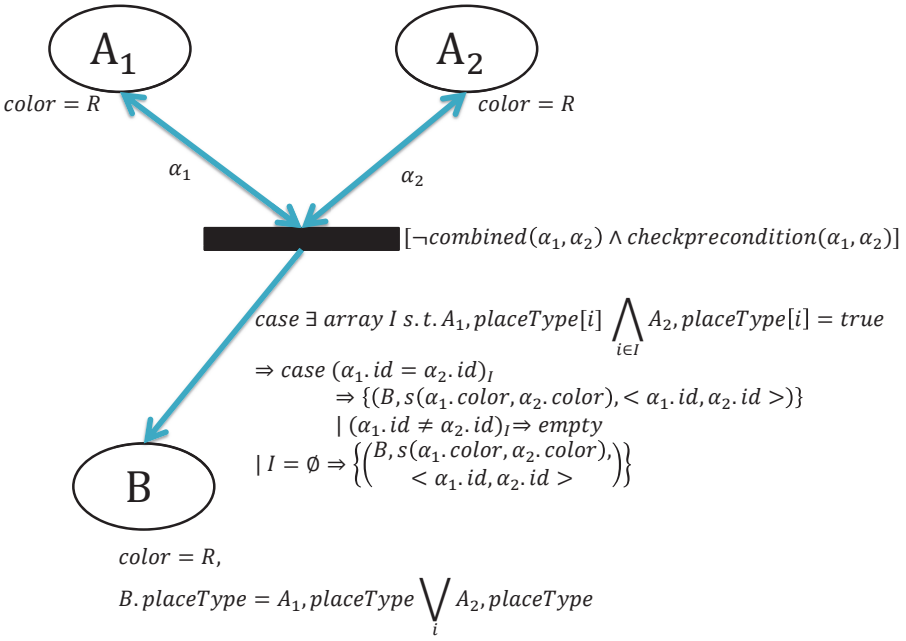


Figure 6.10: Two-operator pattern

6.5.2.5 N operator pattern

It is similar to the pattern 2 but it has $N > 2$ inputs. It is depicted in Figure 6.11.

This thread runs Algorithm 18

6.5.2.6 Precondition/Postcondition pattern

A N-operator service ($N \geq 2$) may have preconditions/postconditions on its inputs. For example they can be specified in OWL-S[3] ontology when providing a semantic description of the service. When the constraints specified in these pre/postconditions concern only one input, we should use a particular derivation of the single service pattern to ensure that an input satisfies the precondition prior to the invocation of the associated service. This pattern is formalized in Figure 6.9 and detailed in Algorithm 19.

The reason of using an independent pattern and not directly including the precondition into the guard of the transition that precedes the service call is related to performance. Indeed, if we included the constraints associated to the pre/postconditions

[3]www.w3.org/Submission/OWL-S/âĂŐ

Algorithm 18 Two Operator Pattern Implementation

Ensure: Correct data in the OutputPlace

```
1: while TRUE do
2:   leftInput ← leftInputPlace.getPlaceTokens()
3:   rightInput ← rightInputPlace.getPlaceTokens()
4:   for all tel ∈ leftInput do
5:     for all ter ∈ rightInput do
6:       if ∃I = {i_1, .., i_n} : (⋀_{i∈I} leftInputPlace.placeType[i] =
         rightInputPlace.placeType[i] = true) ∧ (⋀_{i∈I} tel.ID[i] = ter.ID[i]) ∧
         haveNotBeenCombined(ter, tel) then
7:         initialize a new token element: teo such that all teo.ID[] = −1
8:         for all i ∈ I do
9:           teo.ID[i] ← ter.ID[i]
10:        end for
11:        for all j ∉ I do
12:          if ter.ID[j] ≠ −1 then
13:            teo.ID[j] ← ter.ID[j]
14:          end if
15:          if tel.ID[j] ≠ −1 then
16:            teo.ID[j] ← tel.ID[j]
17:          end if
18:        end for
19:        Create a color of teo by calling service
20:        combinedTokens ← combinedTokens ∪ {⟨tel.ID, ter.ID⟩}
21:        outputPlace.addToken(teo)
22:      end if
23:    end for
24:  end for
25: end while
```

Algorithm 19 Pre/Postcondition Pattern Implementation

Require: **Precondition Pattern**

Ensure: Correct data in the OutputPlace

```
1: while TRUE do
2:   Inputs ← InputPlace.getPlaceTokens()
3:   for all tei ∈ Inputs do
4:     remove tei from Inputs
5:     if checkPrecondition(tei) then
6:       add tei to OutputPlace
7:     end if
8:   end for
9: end while
```

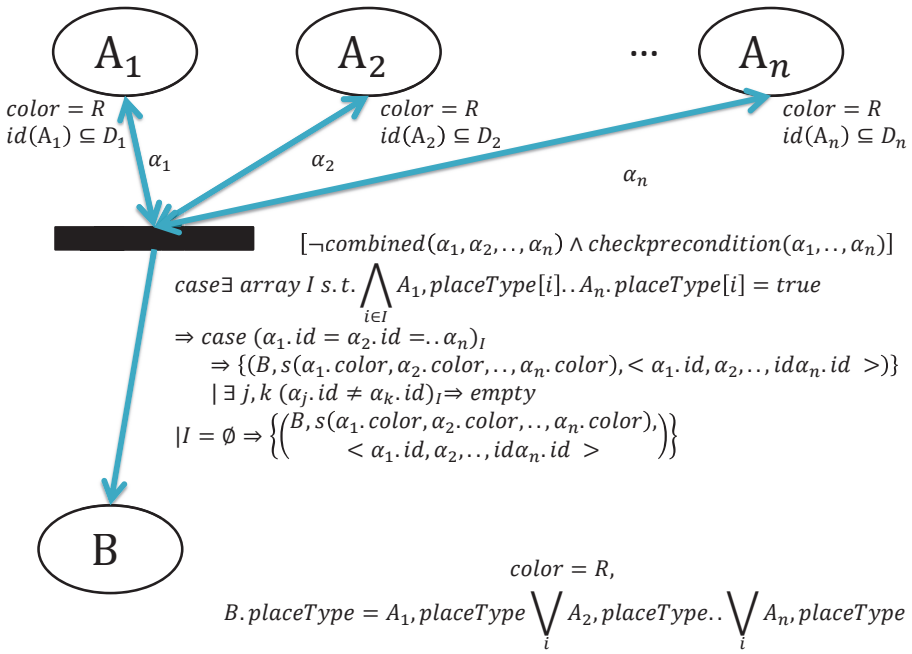


Figure 6.11: N-operator pattern

in the guard of the transition, we would keep the token elements in the input places and verify them each time a new token for another input arrives. This involves additional, non-necessary, processing that can be avoided by initially removing the token elements prior to starting the processing of the service pattern. This fact is expressed in the following lemma.

Lemma 9. *The behavior of the decomposition using an N-operator pattern ($N \geq 2$) of a service s that associates a guard on a single place to the transition (left part of Figure 6.12) is equivalent to the N-operator pattern of the service s preceded by a pre/postcondition pattern having the same guard (right part of Figure 6.12).*

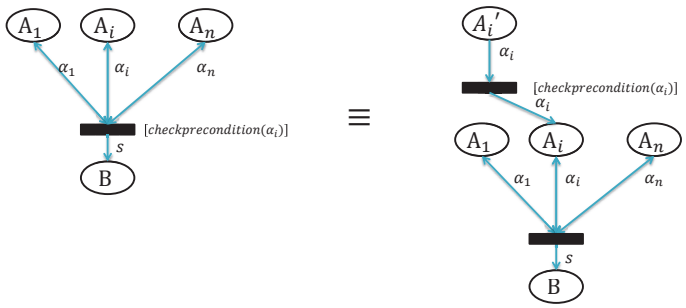


Figure 6.12: pre/postcondition effect on N-operator pattern

Equivalence proof.

- $\rightarrow$ In the left part of Figure 6.12, α_i instances that do not uphold the guard associated to the transition will never trigger this transition. This implies

that removing these instances will have no impact on future executions of the service s

- $\leftarrow$ In the right part of Figure 6.12, α_i instances that do not uphold the guard associated to the first transition will never be checked again. This implies that it will never trigger the second transition.

□

Notice that when the pre/postcondition concerns more than one input (e.g. the sum of all arguments should be > 0), in this case we can attach it to the guard of the transition. Indeed, we cannot remove any instance in the input places because there could be future instances that can be combined with this instance and satisfy the precondition.

6.6 CPN Generation and Execution

6.6.1 CPN generation

In this section, we elaborate on the algorithm that builds the CPN corresponding to a tree representation of a monitoring specification. Indeed, this algorithm (Algorithm 20) traverses the tree structure recursively in an in-order way. Depending on the node that it encounters, it calls the appropriate function that builds the CPN pattern corresponding to this node. Thus in our algorithm lines $8-9$ correspond to the case where the algorithm encounters a leaf node and it calls the function that instantiates the argument pattern and attaches it to the final CPN. Lines $10-11$ and $12-13$ generate the patterns corresponding to the cases where the encountered node is respectively a single operator service (e.g. s_4) and an $N \geq 2$ operator service (e.g. s_1). Lines $6-7$ correspond to generating the duplication pattern when a node (or a sub-tree) has already been generated.

In the following, we prove two important properties:

Theorem 8. *Executing the CPN generated by Algorithm 20, there will be no ambiguity when calling a service that has common arguments in its left hand side and right hand side.*

Proof. Suppose we have a tree representing a monitoring specification with two sub-trees X and Y as depicted in Figure 6.15.

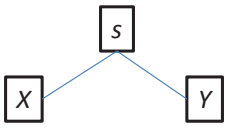


Figure 6.13: Service tree representation

Where X and Y are two sub-trees containing a common argument z: $|X \cap Y| = 1$. Suppose that there have been two occurrences of z: z_1, z_2. Thus in the CPN, there will be four token elements: $\langle X, z_1, id_1 \rangle$, $\langle X, z_2, id_2 \rangle$, $\langle Y, z_1, id_1 \rangle$, $\langle Y, z_2, id_2 \rangle$. When achieving the processing of X there will be two token elements having in their id: id_1 and id_2 respectively. The same for Y. In our CPN, we included in the transitions a guard that specifies that only tokens having common ids will be combined by s. Accordingly X_{id1} will be combined only with Y_{id1} and X_{id2} will be combined only with Y_{id2}.

For two sub-trees, we can have more than one common input. Indeed, we can have common called services. Thus we use proof by recurrence to prove that there is no ambiguity in calculating arbitrary complex monitoring formulas.

Suppose that there is no ambiguity when processing a service with two sub-trees having more than one common input: $|X \cap Y| = n$. We prove that there will be no ambiguity in processing services with two sub-trees such that: $|X \cap Y| = n + 1$. From the hypothesis, Figurefig:restricted illustrates what is valid for this case. Since $X_{idn+1} \cap (Y_{id1} \cup Y_{id2} \cup ... \cup Y_{idn}) = \emptyset$ and $Y_{idn+1} \cap (X_{id1} \cup X_{id2} \cup ... \cup X_{idn}) = \emptyset$. Then X_{idn+1} will be exclusively combined with Y_{idn+1}. Thus there will be no ambiguity. □

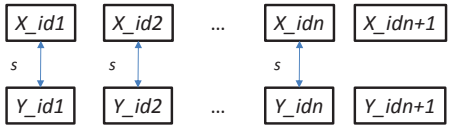


Figure 6.14: Restricted combination of token elements

The following corollary is a consequence of Theorem 1:

Corollary 1. *Running the CPN generated by Algorithm 20, there will be no ambiguity when calling a service that has common services expecting the same inputs in its left hand side and right hand side sub-trees.*

Theorem 9. *When executing the generated CPN by Algorithm 20, once a service representing a sub-tree has been called for a set of values of its arguments, it will not be re-called again for the same values.*

Proof. The service $s_2(X, Y)$ has given the value v_0 of some occurrence of X and Y (Figures 6.15 6.16). Suppose that several instances of the argument Z came. In this case, the two operator service pattern will prevent from recalculating $s_2(X, Y)$ for the same values of X and Y and only processes $s_1(s_2(X, Y), Z)$ for each occurrence of Z. □

The same approach is used to prove that other patterns do not imply systematic service re-invocation.

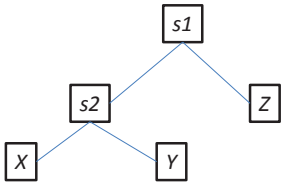


Figure 6.15: Services tree

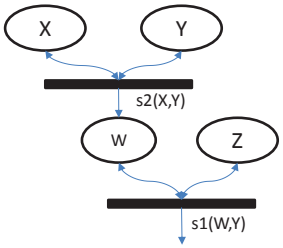


Figure 6.16: CPN pattern of services

Algorithm 20 Recursive construction of the CPN: `Postorder`

Require: The tree root R
Ensure: The CPN block corresponding to the tree

```
1: if R == null then
2:     return
3: end if
4: Inordre(R.left)
5: Switch Pattern(R)
6: Case : Duplication
7: Call : add duplication pattern (R)
8: Case : Argument
9: Call : add an argument pattern (R)
10: Case : Single operator service
11: Call : add a single service pattern (R)
12: Case : N operator service
13: Call : add N operator service pattern(R)
14: Postorder(R.right)
```

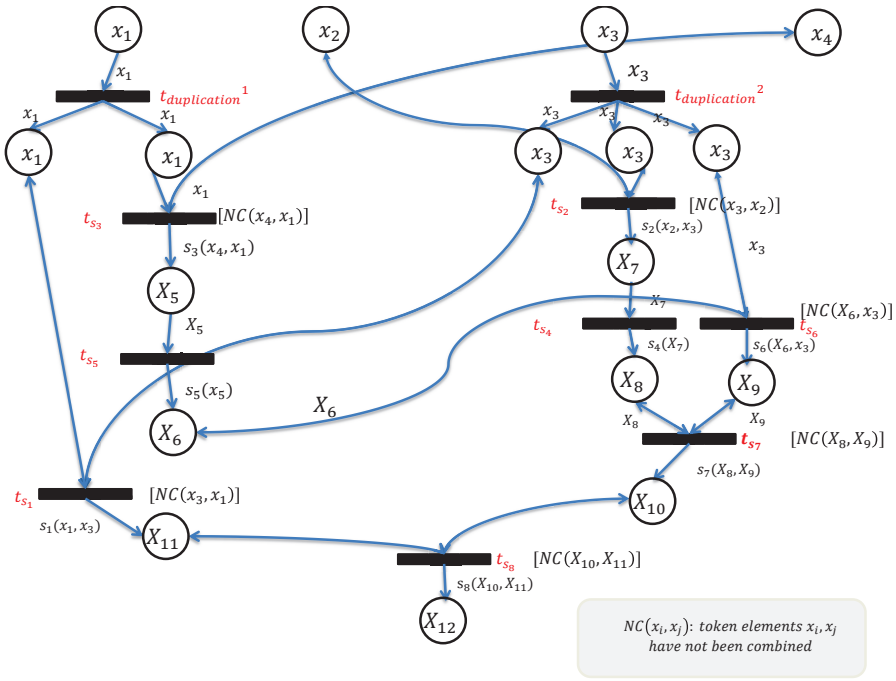


Figure 6.17: CPN generated for the service tree

Figure 6.17 depicts the CPN generated for the service tree in Figure 6.1.

6.6.2 CPN Execution

Once the patterns have been generated, each pattern will be associated to a thread that calls its corresponding service whenever its required inputs are available. These threads are running in parallel and the previous formalization allows keeping the coherence between the inputs, the intermediate results and the final result. Each thread is continuously listening to its corresponding topic and it reacts whenever this topic has been updated.

6.7 Managing the Number of Running Threads

Using the framework as it is, we create as many threads as the number of extracted patterns associated to the decomposition of the service composition. Moreover, each thread has an infinitely running loop (*while (true) do*) that allows it to capture all incoming events and process them. Nevertheless, the number of threads can quickly increase and thus, for scalability purposes, we need a mechanism to optimize the number of running threads through dynamic wake-up and dynamic sleep of threads. Indeed, depending on the pattern type, a thread will start if and only if an event or a set of events occur.

In addition, a thread could be set to a sleep state when certain conditions are satisfied because there is no longer a need for the service associated to this thread.

Using the *Observer design pattern*, we were able to remove the threads loops and therefore, the thread will run if and only if it will produce new results. Otherwise it remains in the idle state. Nevertheless, the Observer design pattern is specified in terms of UML class diagram. This specification is not formal enough. We need a more formal specification for the means of communication between threads.

In this section, we extend our monitoring framework with the capability to dynamically manage the number of running threads. The threads management framework uses Event Condition Actions (ECA) rules in order to manage the running threads.

6.7.1 Event Condition Actions for Threads Management

An *event* in ECA rules is something that occurs at a specific instance of time (e.g. updating the property of a product component) and may change the state of a system. Conditions are specifications associated to the pattern in order to set a filter and receive only pertinent information of what is occurring.

ECA rules can be used to manage the running threads for a monitoring query. Indeed, we need to identify the relevant events for which a particular thread should be started and the relevant events for which a thread should be killed.

For our framework there are several events that can be captured. Nevertheless just a subset of these events is relevant for the management of threads corresponding to a monitoring query.

For a particular *agreement* in the collaboration contract, there will be as many iterations as necessary until reaching an agreement and terminating the process involving a supplier and a requester for this agreement. We call an iteration an agreement attempt. Since an agreement attempt takes time to be achieved, we need to define two predicates that allow a monitor to specify his conditions on a particular agreement. We will have: $Attempt\_started(ag)$ to specify that an attempt has been started. $Attempt\_achieved(ag)$ to specify that an agreement iteration has been performed. Additionally the predicate $Attempt\_positive(ag)$ specifies whether an agreement has been reached or not yet (i.e. the process involving requester and a supplier has terminated).

- The event in an ECA specifies when a rule has to be executed. Events are not time consuming and do not include any activity to be performed by an actor. Thus $Happens(e)$ is a predicate to indicate that the event e has occurred. Since an event e is generated due to a message exchange during the process execution for a

A_1	$\exists e Happens(e) \Rightarrow Attemp\_achieved(ag) \wedge belongs(e, ag) \wedge \neg Attempt\_positive(ag)$
A_2	$\exists e, Started\_checking(e, c) \Rightarrow Happens(e)$
A_3	$\forall c, Checking\_achieved(c) \Rightarrow Started\_checking(c)$
A_4	$\forall ag, Attempt\_achieved(ag) \Rightarrow Attempt\_started(ag) \wedge \neg Attempt_positive(ag)$
A_5	$\forall a, \exists c, Started(a) \Rightarrow Checking\_achieved(c) \wedge Precondition(c, a) \wedge Checking\_positive(c)$
A_6	$\forall a, Performed(a) \Rightarrow Started(a)$

Table 6.1: Axioms of Threads Management

particular agreement ag, we define the predicate $belongs(e, ag)$

- The condition in an ECA rule indicates a condition to be checked before any action is triggered (i.e. before any information is transmitted to the monitor). The predicate $Started\_checking(e, c)$ indicates that the occurrence of the event e starts the checking of the condition c. The predicate $Checking\_achieved(c)$ indicates that the condition c has been achieved. The predicate $Checking\_positive(c)$ indicates that the checking of the condition c is $true$.

- Once the required events have occurred and the conditions on the values that they carry have been verified. The action component states what has to be done depending on the result of the evaluation of the condition component. In analogy with the condition verification of an ECA, the action is time consuming too, but its termination is characterized by raising one or more process relevant events. To characterize these actions, we define the following predicates: $Started(a)$, $Performed(a)$.

The specification of requirements in the condition expression should be compliant with the standard axioms of ECA rules. These axioms are shown in table 6.1.

6.7.2 ECA Rule for Starting a Two-Operator Pattern Thread (TH)

- Event: e_1, e_2, $\{e_1, e_2\}$
- Condition: $Happen(e_1) \wedge Happens(e_2) \wedge \neg Started(TH)$
- Action: $Start(TH)$

Informally, this rule specifies that the thread Th associated to a Two-operator pattern can start if and only if both events expected in the input places have happened (not necessarily at the same time) and the thread has not been started before.

6.7.3 ECA Rule for Killing a Two-Operator Pattern Thread

- Event: e_1, e_2, $\{e_1, e_2\}$

- Condition: $Attempt\_positive(e_1) \wedge Attempt\_positive(e_2) \wedge Started(TH) \wedge \forall \alpha_1 \in A_1, \forall \alpha_2 \in A_2 : combined(\alpha_1, \alpha_2) \wedge check\_precond(\alpha_1, \alpha_2)$
- Action: $Kill(TH)$

Informally, this rule specifies that the thread TH associated to a Two-operator pattern is killed when expected events will no longer be produced by their respective processes because agreements have been achieved and there is no need to continue the execution of this thread. Nevertheless, notice that all data produced by this thread is available for the subsequent services in the composition.

6.7.4 ECA Rule for starting a Single-operator Pattern and Precondition Pattern Threads

- Event: e_1
- Condition: $Happen(e_1) \wedge \neg Started(TH)$
- Action: $Start(TH)$

Informally, this rule specifies that the thread TH associated to a single-operator pattern can start if and only if the input event required by this thread has occurred.

6.7.5 ECA Rule for Killing a Single-operator Pattern and Precondition Pattern Threads

- Event: e_1
- Condition: $Attempt\_positive(e_1)(TH)$
- Action: $Kill(TH)$

Informally, this rule specifies that the thread TH associated to a Single-operator pattern is killed when the expected event will no longer be produced by the process associated to the agreement.

6.7.6 ECA Rule for Starting a Duplication Pattern Thread

- Event: e_1
- Condition: $Happen(e_1) \wedge \neg Started(TH)$
- Action: $Start(TH)$

6.7.7 ECA Rule for Killing a Duplication Pattern Thread

- Event: e_1
- Condition: $Attempt\_positive(e_1)(TH)$
- Action: $Kill(TH)$

The management of the running threads is performed by a rule engine that executes the ECA rules.

6.8 Integration of the Monitoring with the Product-based Contract Framework

At this stage we have developed the formal framework of the optimized monitoring capability to be used by *coordinators* in order to monitor the unfolding of the design of different product components. In this section, we extend the Contract framework in order to incorporate this monitoring capability and give the coordinator the means to specify what they want to monitor in the agreements specifications.

In chapter 4 we have defined an agreement as a tuple: $\langle Requester, Supplier, Component, Obligations \wedge Characteristics \rangle$. We extend this tuple by adding the Monitors attribute. Hence, a monitorable agreement becomes: $\langle Requester, Supplier, Component, Obligation \wedge Characteristics, Monitors \rangle$ where:

Monitors is a set of couples $\langle Subscriber, Messages \rangle$ such that each *Subscriber* is a partner in the DMN that is interested in obtaining copies of messages exchanged between the *Requester* and the *Supplier* such that this partner can specify to obtain copies of attributes belonging to *Obligations* and/or *Characteristics*. Formally:

$\forall m\ Messages, \exists m' \in Obligations \cup Characteristics : equivalent(m, m')$

Once all messages to be monitored are selected, a query is associated to the minimal complex agreement common to all agreements to which the monitor has been specified. Figure 6.18 depicts an example of the agreements corresponding to our example. Once the queries have been specified, the COP is generated. The monitoring queries have no impact on the generated workflow. Thus they can be also specified during run-time without impacting the running workflow. Nevertheless, we need to use the administration layer of the workflow engine in order to capture the exchanged messages and deliver copies of these messages to the monitoring query processor. The latter executes the service composition and then provides the final result to **D**.

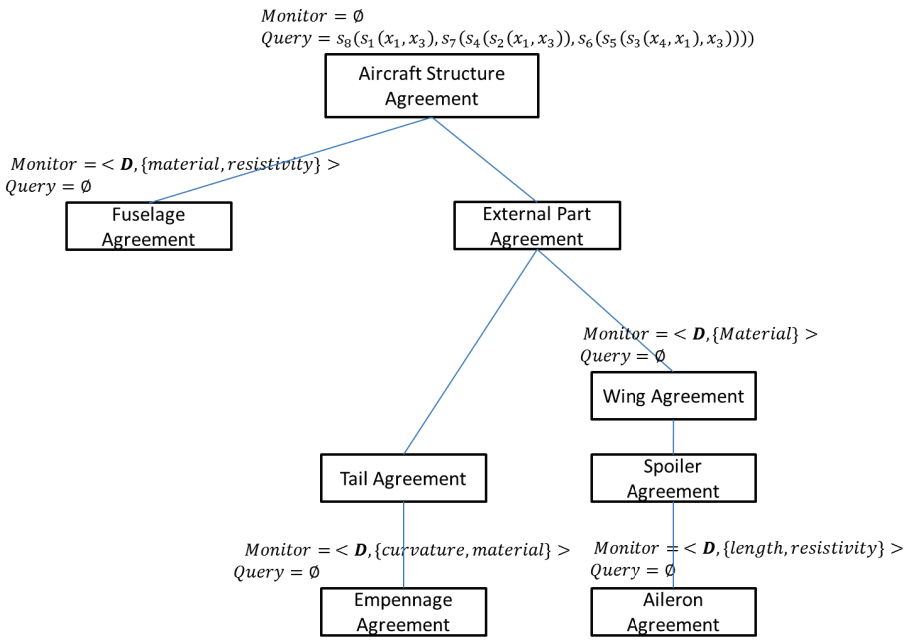


Figure 6.18: Specifying the monitoring query in the collaboration contract

6.9 Evaluation

6.9.1 Evaluation Approach

In this section, we present the prototype that implements our framework. This prototype leverages our collaborative platform for dynamic collaborative networks [IMA11] with monitoring capabilities. It constitutes the continuation of a comprehensive framework [KFB+13] that we are developing in the context of the European project IMAGINE [IMA11]. Basically, this prototype provides the engineer of **D** a user interface that he uses to declaratively specify the monitoring query while calling User Defined Functions (UDF) implemented as external services. Our system uses the messages that will be exchanged between the participants **A, B, C**. Once **D** has submitted the query, the messages will be watched by the monitoring component deployed in the collaborative platform. The monitor generates an execution plan corresponding to the monitoring specification. The execution plan uses our algorithm (based on CPN patterns) to route the combination of messages that satisfy the engineer specification. The execution plan uses our algorithm (based on CPN patterns) to route the combination of messages that satisfy the engineer specification. Figure 6.19 illustrates the queries corresponding to the decomposition.

Table 6.9.1 shows a comparison between the execution of our algorithm that uses the CPN patterns and the naive approach as implemented by Esper. In this comparison we

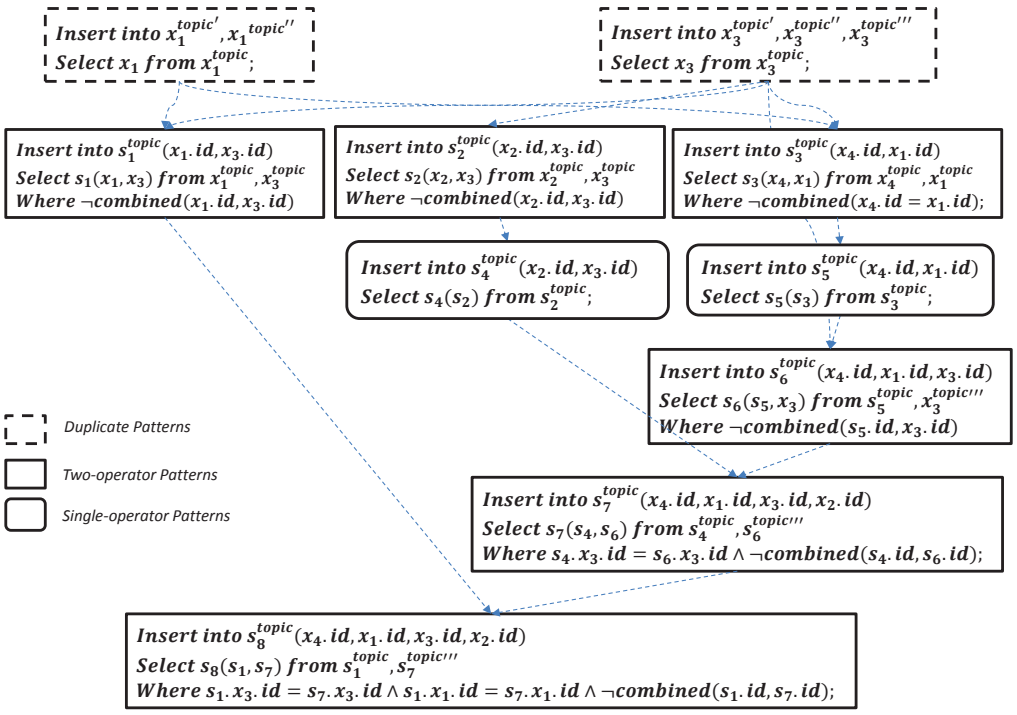


Figure 6.19: . Output of the CPN generation Algorithm for the running example

assumed that an instance of x_2 arrives after 30 seconds and the average time of a service execution is 10 seconds. Moreover, we assumed that a second instance of x_1 arrives at time 40s:

Notice that at time slot 70s, the service s_7 runs for the second time without re-running s_2 and s_4 since there are no new instances of x_2.

Our approach takes much less time to send the results to the monitor. At time slots 60 and 80, we have already sent two notification, while the naive approach sends the same notifications at time slots 100 and 200 respectively. Another possible approach to process the received data consists in creating an XPDL model (or any other model that composes web services: BPEL etc.) of the service composition and then run it to handle the incoming messages in parallel. Figure 6.20 depicts a model of this process. Although this approach runs independent services in parallel, it is inefficient regarding the number of running instances of the process composing the services. Indeed, in this approach, for each new instance of any argument, we should create an instance of the XPDL model and deploy it into the process execution engine in order to ensure correct correlation of incoming messages and intermediate results. Moreover an arriving argument instance should be submitted to all other process instances so that they can carry on their execution. This involves the creation of multiple copies of this argument

Time slot	Instances of arguments	Executed threads for CPN approach	Executed threads for the naive approach
0	x_1, x_3, x_4	$\emptyset$	$\emptyset$
10	$\emptyset$	s_3, s_1	$\emptyset$
20	$\emptyset$	s_5	$\emptyset$
30	x_2	s_2, s_6	s_1
40	x_1	s_4, s_3, s_1	s_2, $\emptyset$
50	$\emptyset$	s_7, s_5	s_4 $\emptyset$
60	$\emptyset$	s_8, s_6	s_4, $\emptyset$
70	$\emptyset$	$\emptyset$, s_7	s_5, $\emptyset$
80	$\emptyset$	s_8	s_6
90	$\emptyset$	$\emptyset$	s_7
100	$\emptyset$	$\emptyset$	s_8
200	$\emptyset$	$\emptyset$	s_8

Table 6.2: Comparison between the CPN approach and the naive approach

instance which increases the memory consumption of the approach. Accordingly, our approach performs well regarding memory consumption thanks to the patterns that can be combined together. We also have a constant number of threads running in parallel. Our approach performs well regarding memory consumption thanks to the patterns that can be combined together.

Figure 6.21 makes a logarithmic comparison between the XPDL approach and our framework regarding the number of created copies of each incoming event. Figure 6.22 makes a logarithmic comparison between the number of running threads in our approach and the number of process instances that are running in the XPDL approach.

6.9.2 Discussion

From these preliminary results, we can conclude that our approach provides better performance in case where two conditions are satisfied:

1. There is a difference between the rates of the arrival of messages. This is the case in workflows where actors that generate messages are software applications and humans. Indeed humans, generally provide data in a low throughput in comparison to software applications that could generate messages at a very high rate.

2. The services take time to accomplish their processing and give results. Thus running them in parallel is more advantageous. This is the case of simulation services.

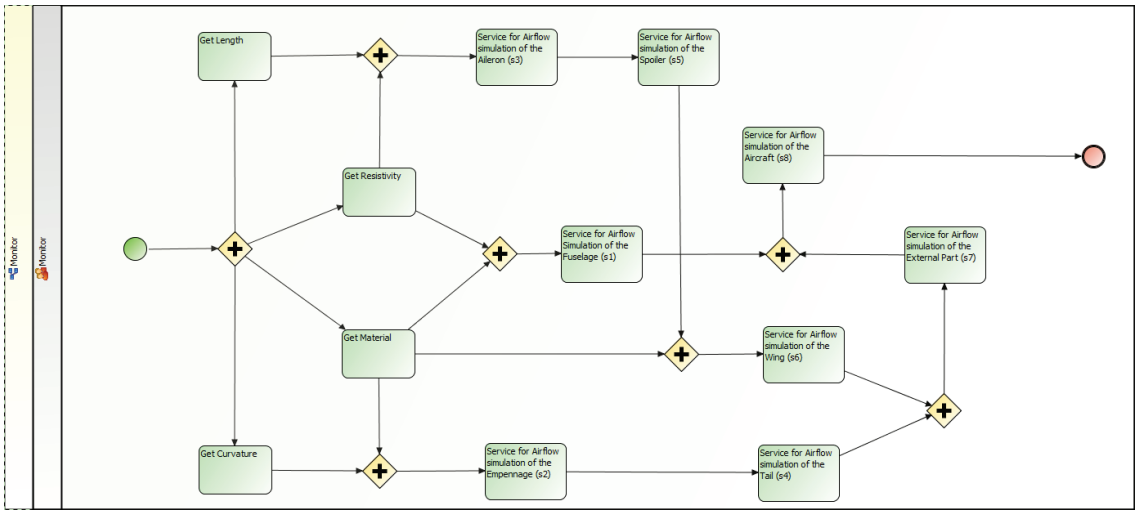


Figure 6.20: Process composing services for the monitor

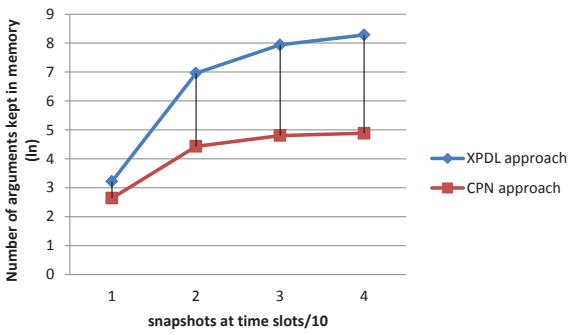


Figure 6.21: memory consumption

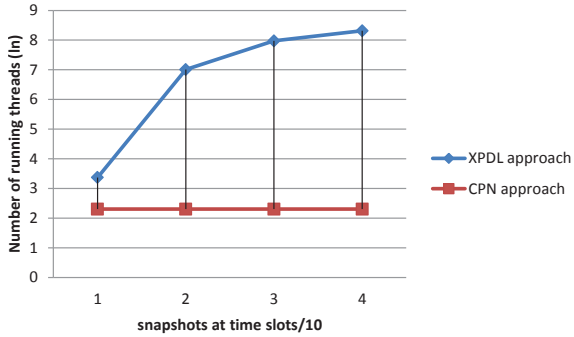


Figure 6.22: number of threads

Existing approaches provide better performance when the messages come at the same rate (e.g. a single event containing all data like CEA) and the set of services take less time to execute in comparison to the rate of messages. In this case our approach is less efficient due to the additional processing introduced by the CPN constructs. Nevertheless, this case is less likely to occur in our domain.

6.10 Conclusion

In this chapter we enriched the product-based contract framework with an efficient monitoring capability. Unlike existing (academic or industrial) monitoring tools, our approach aims to decouple the invocation of services involved in the processing of incoming data. To achieve this we defined a set of patterns of service composition in monitoring queries. We backed these patterns with formal foundations using an adapted version of Colored Petri Nets.

The advantages of using the monitoring approach proposed are (i) the non re-invocation of the same service for the same inputs and (ii) the correct correlation of incoming messages as well as intermediate results. We formally proved these two claims. To further increase the performance of our approach, we defined a wake-up/killing mechanism that manages the threads calling the services of the query. Basically, if a thread does not expect data anymore, it no longer needs to be in a wake state. In addition, if data useful for a particular thread arrive, this thread is started. We formalized and implemented this thread management mechanism using Event Condition Action (ECA) rules.

We made a preliminary evaluation of our monitoring approach and we compared it with ESPER tool. The first results are positive since our approach showed better performance. Additionally we compared our approach to another way to specify the composition of services in the query by creating a more complex service involving all primitive services. The results show that our approach performs better regarding memory consumption as well as the number of running threads.

The work presented in this chapter can also be put in the context of the workflow patterns initiative. Indeed, much research is being conducted on the workflow patterns. Other researchers are contributing to the workflow change patterns. The work presented in this chapter can be an initiator to addressing new challenges related to patterns to monitor workflows.

Chapter 7

Prototype Implementation

Contents

7.1 Introduction

This chapter presents details regarding the implementation of the product-based contracts framework developed in the previous chapters. Since we have already presented the performance evaluation of the mediation, change planning, and monitoring techniques, in this chapter we will focus on the way we concretely realized these techniques in our working prototype.

Our prototype relies on several existing software modules that provide generic functionality (e.g. workflow engine, PLM System). These software modules are generally backed by standardized interfaces. Since

Thus, in the second part of this chapter we provide a formal approach that tests and reports on the compliance between software modules and their corresponding standards. The objective of this report is to help developers select the most compliant software to a particular standard.

This chapter is organized as follows. Section 7.2 provides the architecture of the prototype of the PLM Hub and highlights the different modules that implement our framework. In its subsections we detail each software module and the techniques used for its implementation. Section 7.3 presents our conformance testing approach. More specifically, it presents its formal foundations and then gives an example by testing and reporting the conformance of the workflow engine of our prototype with the WfMC[1] standard. Finally section 7.4 concludes this chapter.

7.2 Platform Architecture

We have implemented a collaborative platform (PLM Hub) that allows *coordinators* and *participants* to log in and create a collaboration contract model using an intuitive user interface. Then, they can start the collaboration while being supported by a cross-organizational process that guides their dialog. The overall architecture of our platform is depicted in Figure 7.1.

The architecture is organized into three layers. The first layer contains the systems pertaining to the partners in the collaboration environment. More specifically it contains the PLM system of partners. A PLM System through its standardized interface called PLM Services[2] allows the *coordinator* to access his Product breakdown and selects the product component on which he wants to change properties. This layer also contains

[1] www.wfmc.org/
[2] http://www.omg.org/spec/PLM/

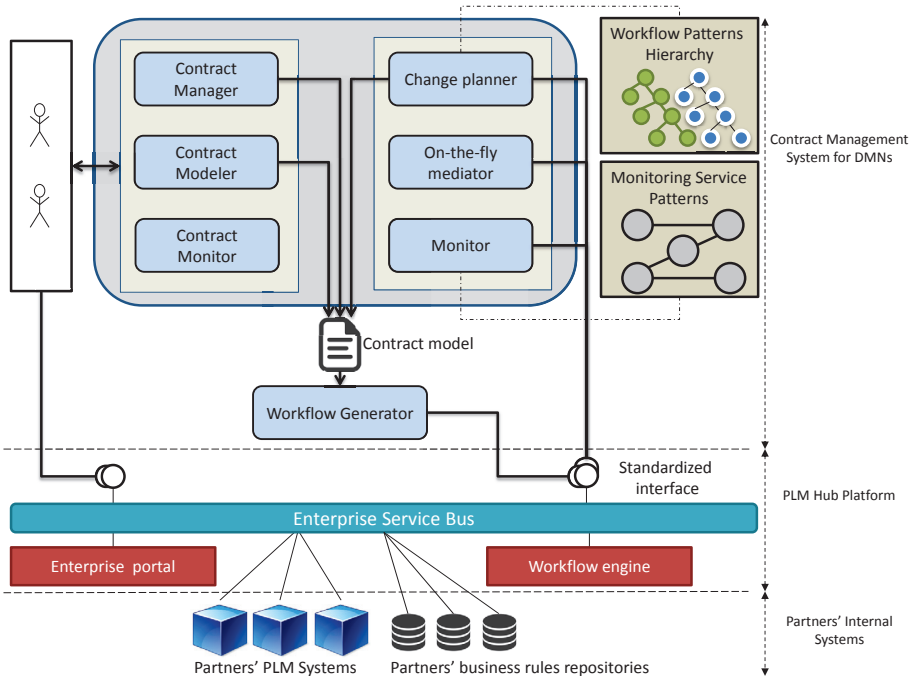


Figure 7.1: Platform Architecture

the business rules repository of each partner where all enterprise rules are stored. In particular, it contains rules related to the organization of data exchange activities.

The second layer contains the generic software modules of the collaborative platform. It contains the workflow engine: *Together Workflow Server* TWS[3], the enterprise portal *Liferay*[4] along with other modules developed in the context of the IMAGINE project. IMAGINE modules provide functionality specific to the manufacturing phase of the product which is not covered in this thesis. All modules of this layer are interconnected through an enterprise service bus OpenESB[5].

The workflow engine executes the cross-organizational process in order to distribute the data submitted by partners to other partners. The language of the process model accepted by this workflow engine is XPDL 2.0 that is visually rendered into BPMN 2.0[6].

The enterprise portal allows partners to access the contract design and management interface. In addition, it allows each partner to access his tasks list that displays him the data received and that have to be processed to carry the execution of the cross-organizational process. We used Liferay since it is a reference in implementing enterprise portals[7].

[3]http://www.together.at/prod/workflow/tws
[4]https://www.liferay.com
[5]http://www.open-esb.net/
[6]www.omg.org/bpmn/Documents/XPDL$_B$PMN.pdf
[7]https://www.liferay.com/about-us/awards/gartnermq-portals

The software modules at this layer expose a web service management interface. These interfaces are connected to an enterprise service bus to make their access uniform by the software modules of the upper layer.

The upper layer includes a Contract Management System for DMNs (CMSD) and a Graphical User Interface. *Coordinators* and *participants* access the system via a GUI developed using the Vaadin[8] framework. We used Vaadin to develop the graphical modeling interface for contracts because it has libraries that facilitate this work. In addition GUIs developed using Vaadin can be easily integrated with Liferay. Through this interface, *coordinators* model contracts and assign *participants* to the right *agreements* and then the workflow of the cross-organizational process is generated.

The CMSD is composed of several user interface modules: *Contract Modeler*, *Contract Manager*, and *Contract Monitor* as well as applicative modules: *Workflow Generator*, *On-the-fly Mediator*, *Change Planner* and *Monitor*.

7.2.1 Contract Modeler

The Contract Modeler helps *coordinators* specify agreements on components of the product. For this purpose we have constructed a meta-model using Eclipse EMF[9] that captures the concepts of our product-based contract framework. *Coordinators* use this meta-model in order to build instances (models) of collaboration contracts. Figure 7.2 illustrates a snapshot of the Contract Modeler interface. In this figure, we can see the contract between the *coordinator* and the *Fuselage* components *participants*.

Basically there are two portlets: The Contract Modeler and Management on the left (used at design-time) and the worklist handler that lists to each partner the message that arrived or to be submitted (used at running-time). In the Contract modeler there is the product breakdown structure (e.g. the aircraft breakdown). This structure is extracted from the *coordinator*'s PLM system. To perform this extraction, we call the PLM Services.

In our running example, the *coordinator* who wants to perform changes on the *Fuselage* will use this interface to define the models that will be changed *Fuselage Geometry* and *Fuselage Aerodynamic*). Then in the bottom panel, he defines the attributes on which the constraints will be defined at run-time. In addition, he defines the requester of this study and who will perform it (the supplier).

Once all details of the contract have been specified a file that captures the agreements of the contract as well as their temporal constraints is generated after pushing the `Export` button.

[8]https://vaadin.com
[9]www.eclipse.org/emf/

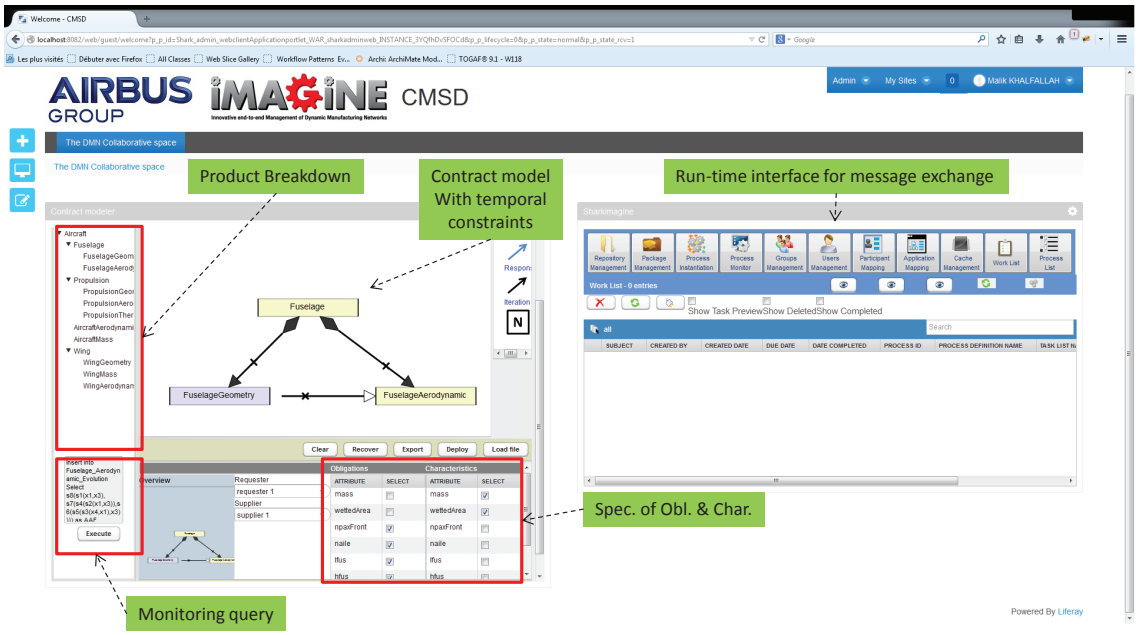


Figure 7.2: Contract Modeling Interface

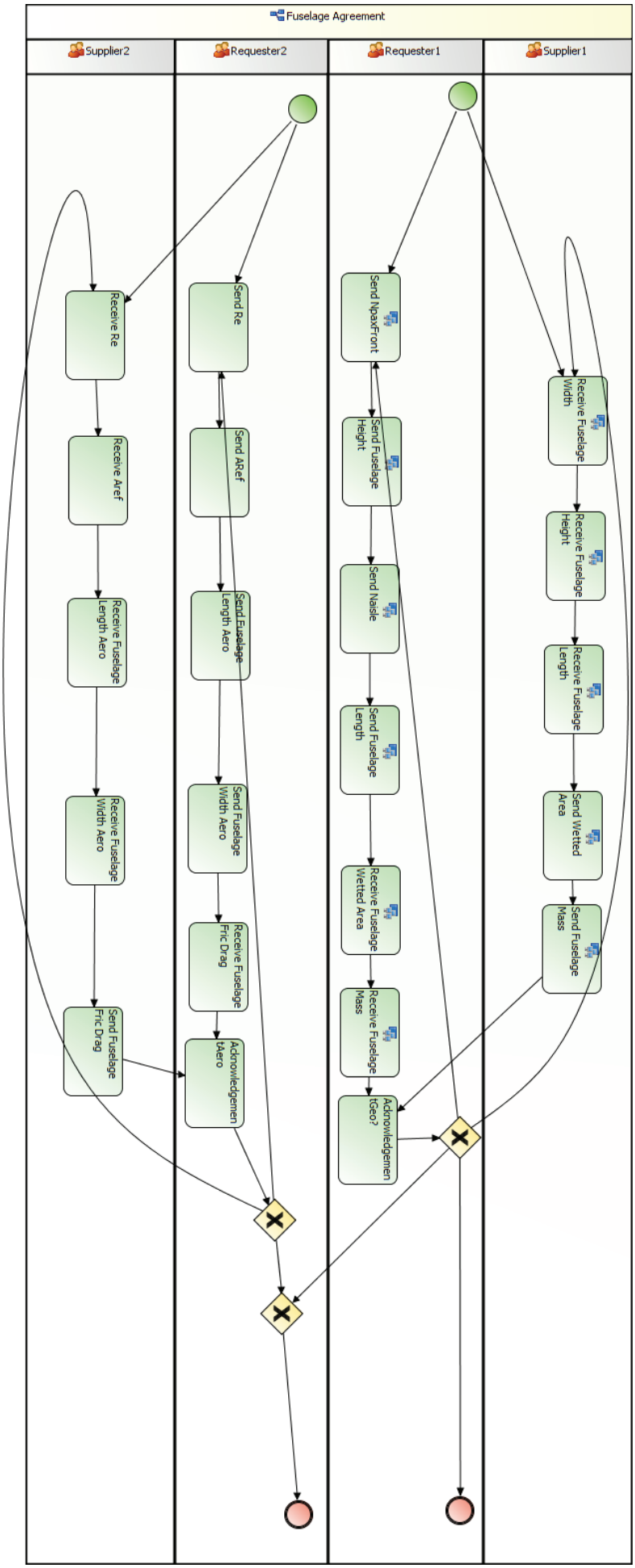
To specify temporal constraints, since we used Eclipse (EMF[10]) to build the meta-model of our contract framework, we can use Object Constraint Language (OCL[11]) to specify the temporal constraints between agreements. Originally, OCL does not support temporal constraints; nevertheless several extensions have been performed on this language to include capabilities to express temporal constraints. We have used the extension made by [KT13] since they have implemented the temporal constraint patterns that we used to express temporal constraints between agreements.

Once the contract model is ready, the *coordinator* clicks on `Deploy`. This action will generate the workflow model of the cross-organizational process and deploys it into the workflow engine as depicted in Figure 7.3.

7.2.2 Workflow Generator

The Workflow Generator consumes the serialized contract model in order to generate the model of the cross-organizational process in XPDL. It also accesses the business rules repositories belonging to the partners of the collaboration to find how the activities of this process should be organized. Once the XPDL model is generated it deploys it into TWS. To achieve this deployment the Workflow Generator uses the management interface provided by TWS. This interface provides services to deploy, instantiate, abort,

[10]http://www.eclipse.org/modeling/emf/
[11]www.omg.org/spec/OCL/


Fuselage Agreement
Supplier2
Requester2
Requester1
Supplier1
Receive Re
Receive Aref
Receive Fuselage Length Aero
Receive Fuselage Width Aero
Send Fuselage Fric Drag
Send Re
Send ARef
Send Fuselage Length Aero
Send Fuselage Width Aero
Receive Fuselage Fric Drag
Acknowledgemen tAero
Send NpaxFront
Send Fuselage Height
Send Naisle
Send Fuselage Length
Receive Fuselage Wetted Area
Receive Fuselage Mass
Acknowledgemen tGeo?
Receive Fuselage Width
Receive Fuselage Height
Receive Fuselage Length
Send Wetted Area
Send Fuselage Mass

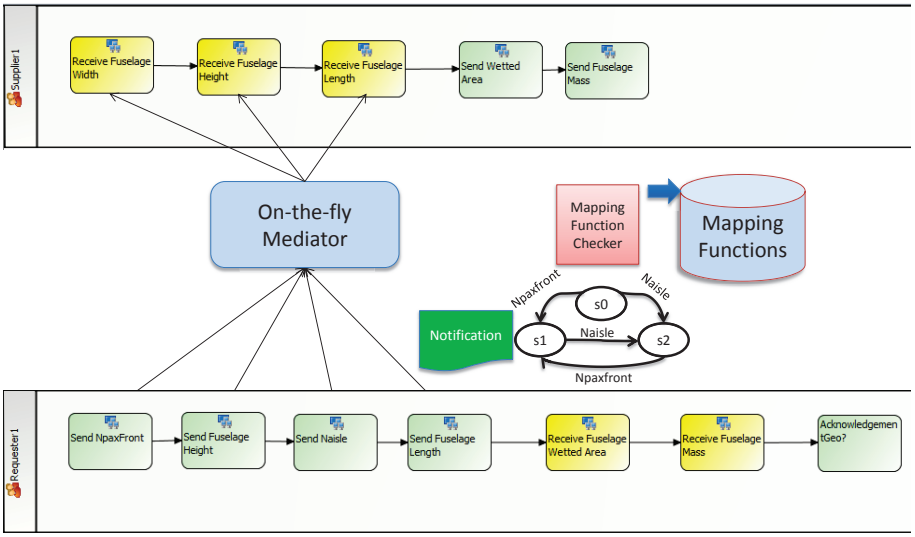


Figure 7.4: The place of the mediator between two communicating lanes

resume and terminate the execution of processes. Other management services exist as well.

7.2.3 On-the-fly Mediation

In our prototype, when data exchange activities of the cross-organizational process are generated, by default they make calls to the on-the-fly mediation module as depicted in Figure 7.4. The on-the-fly mediation is deployed in the CMSD as a Java application. Data exchange activities call this application by submitting the message name, its value and eventually a file associated to the message. To realize this call, each activity is configured to use the tool agent *Runtime Application*. A tool agent specifies what kind of application a particular activity in a workflow will call (e.g. standalone application, web service).

The on-the-fly mediator always knows what messages are expected by a process. The reason is that the expected message's receive activity is in the state *opened* while all other activities of the same process are in the state *not_opened*. To identify this activity, the on-the-fly mediator uses the service `listActivityInstances`.

When the on-the-fly mediator is called by a send activity, it informs the automaton of the mapping function expecting this message. This will lead to a state transition in that automaton. When all states of the automaton are marked by the arriving messages, the mediator is notified of the applicability of the mapping function that generates the expected message.

When the on-the-fly mediator is notified of the applicability of a mapping function, it starts the receive activity of the expected message and submits the result to

that activity. To realize this starting, the on-the-fly mediator calls the TWS service `changeActivityInstanceState`.

7.2.4 Contract Manager

Workflow management systems are used to configure and control structured business processes from which well-defined workflow models and instances can be derived [vdAW05, Joe99]. However, the existing process definition frameworks make it difficult to support (i) dynamic evolution (i.e. modifying process definitions during execution) following unexpected or developmental change in the business process being modeled [MB00]; and (ii) deviations from the prescribed process model at run-time [RRD04b, EKR95].

The Contract Manager helps *coordinators* and *participants* implement changes on contracts that are already in the run-time phase. It modifies the contract model and submits it to the Workflow Generator. In this case, the Workflow Generator has another important task that is checking whether a contract can be committed to the running cross-organizational process or not.

7.2.4.1 Changing the contract

When agreements of a contract are modified and these modifications cannot be committed then, the involved partners in these agreements must not carry the execution of their processes. In other terms, the corresponding lanes in the cross-organizational process should be inhibited until other modifications arrive that trigger the commit of all changes.

To realize this inhibiting of process lanes at run-time while letting other lanes of the same process active, we followed this procedure:

1. Remove the mapping between the role associated to the lane and the actor playing this role
2. The remaining activities in this lane will be automatically assigned to a default actor of the workflow engine (admin). Of course this actor cannot execute these activities. Thus these activities will remain pending until the lane will be associated to a new partner (i.e. activated)
3. Once the lane becomes active again, we traverse all pending activities and assign them to the newly added partner. To realize this assignment, we use the method `WAPI.reassignWorkItem(session, admin, newActor, processId)`
4. At this stage the newly added partner can execute the pending activities.

7.2.4.2 Resuming the collaboration

When a contract becomes committable after one or more changes, the process that will be generated and deployed should not start its execution from scratch. Instead, we need to recover the state of activities that have already been terminated and start the new deployed process from where the previous version has been stopped. The following steps allow us to implement this switching:

1. Get the state of the currently running process using the operation `getProcessInstance` of the WAPI service

2. Get the activities instantiated in this process instance using the operation `listActivityInstances` of the WAPI service

3. Terminate the current process instance using the operation `terminateProcessInstance` of WAPI. Then remove this process from the workflow engine cache

4. Using the new version of the process, copy the state of the activities that have been saved in step 2 into their equivalent activities in the new process instance using the operation `assignActivityInstanceProperty`. Then resume the instance of the new process to continue the execution.

7.2.5 Change Planner

Partners could tailor multiple agreements in a contract model; this could make these agreements not committable and thus delay their fulfillment. In order to help partners recover the modified agreement and incorporate the missing information, they can ask the Change Planner module. The Change Planner implements our extension of the planning graph algorithm. It generates the recovery workflow and deploys it into the workflow engine. Partners will receive the tasks to perform on their task list in order to recover the agreements and make the contract committable.

7.2.5.1 Generation of the solution plan

To generate the solution plan for our example presented in Chapter 5, we have used the implementation of the planning-graph algorithm: PL-Plan[12]. PL-Plan generates the solution plan when given a set of actions, their preconditions, their post-conditions and a goal. Then, in order to implement our approach that extends the planning graph

[12]http://www.philippe-fournier-viger.com/plplan/

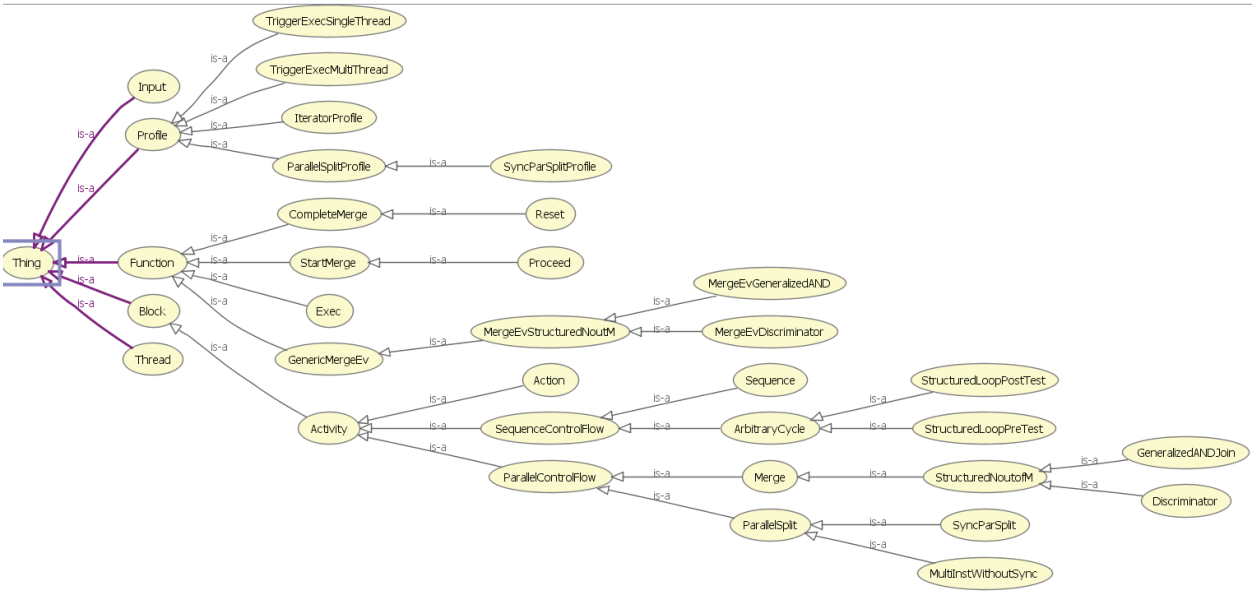


Figure 7.5: An excerpt of the hierarchy of the workflow patterns (in Protege)

algorithm and generates the executable workflow, we formalized the workflow patterns using OWL-S[13] and we implemented the discovery algorithms.

7.2.5.2 Formalization and Discovery of the Workflow Patterns

Formally, workflow patterns can be represented using Abstract State Machines (ASM) [Bö7]. In order to implement the ASM formalization of workflow patterns, existing ASM tools can be used. Nevertheless, these tools are not widely used and suffer from maintainability issues. In order to make our implementation maintainable, we have decided to implement the ASM formalization of workflow patterns using the *Profile* ontology of OWL-S. Indeed, this ontology can capture the signature and preconditions of each workflow pattern. Figure 7.5 depicts the ontology that formalizes a subset of workflow patterns as classified in [Bö7]. For example the Profile ontology of the **Iterate** pattern is given in Listing 7.1 (*IterateProfile*). It shows that the **Iterate** has two inputs of types *Activity* and *Thread* respectively. Listing 7.2 shows the precondition of this pattern that we formalized in Jena[14]. Indeed, we used Jena rules to capture the precondition of each pattern. In our example, the precondition specifies that the **Iterate** pattern can have only one activity as input and this activity is executed by one agent (thread).

Listing 7.1: A portion of the profile ontology of the Iterator pattern

[13]http://www.w3.org/Submission/OWL-S/
[14]https://jena.apache.org

```
<!-- http://www.airbus.com/45520709/ontologies/2014/2/workflowpatterns2#
    iterateprofile -->

    <NamedIndividual rdf:about="&workflowpatterns2;#IterateProfile">
        <rdf:type rdf:resource="&workflowpatterns2;#IterateProfile"/>
        <workflowpatterns2:hasActivityprofile rdf:resource="&
            workflowpatterns2;#defaultAction"/>
        <workflowpatterns2:hasThreadprofile rdf:resource="&
            workflowpatterns2;#defaultThread"/>
        <profile:hasInput rdf:resource="&workflowpatterns2;#input1-d"/>
        <profile:hasInput rdf:resource="&workflowpatterns2;#input2-s"/>
    </NamedIndividual>

   <!-- http://www.airbus.com/45520709/ontologies/2014/2/workflowpatterns2#
       input1-d -->

    <NamedIndividual rdf:about="&workflowpatterns2;#input1-d">
        <rdf:type rdf:resource="&process;#Input"/>
        <process:parameterType rdf:resource="&workflowpatterns2;#Activity"/
            >
    </NamedIndividual>

    <!-- http://www.airbus.com/45520709/ontologies/2014/2/workflowpatterns2
        #input2-s -->

    <NamedIndividual rdf:about="&workflowpatterns2;#input2-s">
        <rdf:type rdf:resource="&process;#Input"/>
        <process:parameterType rdf:resource="&workflowpatterns2;#Thread"/>
    </NamedIndividual>
```

Listing 7.2: The precondition of the **Iterator** pattern

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix owl: <http://www.w3.org/2002/07/owl#>
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
@prefix wp: <http://www.airbus.com/45520709/ontologies/2014/2/
    workflowpatterns2#>
[iteratorProfilePrecond:      (wp:iterateProfile wp:hasActivityProfile ?x1)
                (wp:iterateProfile wp:hasActivityProfile ?x2)
                equal(?x1,?x2)
                (wp:iterateProfile wp:hasThreadProfile ?y1)
                (wp:iterateProfile wp:hasThreadProfile ?y2)
                equal(?y1,?y2)
                ->
```

```
13                (wp:iterateProfile wp:preconditionProfileSatisfied true)
14
15 ]
```

For a specific *Block* node in the workflow tree that needs to be replaced by a workflow pattern, we need to compare its ASM signature (i.e. its Profile ontology generated during the workflow tree traversal) to the ASM signatures of the workflow patterns in the hierarchy of Figure 7.5. To implement the corresponding algorithm of this feature that is `checkAppropriateness` algorithm, we used a match making approach. Indeed, to compare two *Profile* ontologies, we need to make the following comparisons:

1. Check that both *Profiles* use the same ontologies
2. Check that the number of inputs of the candidate *Block* is equal to the number of inputs of the pattern
3. Check that the classes of the input parameters of the candidate *Block* and the workflow patterns are equivalent
4. Check that the conjunction of the precondition of the candidate *Block* and the precondition of the workflow pattern is valid as depicted in Listing 7.3. To make this checking, we used the `Reasoner` class of Jena.

This mechanism allows us to find the appropriate pattern for each *Block* in the workflow tree.

Listing 7.3: The conjunction of the Block precondition and the Iterator Pattern precondition

```
1 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 @prefix owl: <http://www.w3.org/2002/07/owl#>
3 @prefix xsd: <http://www.w3.org/2001/XMLSchema#>
4 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5 @prefix wp: <http://www.airbus.com/45520709/ontologies/2014/2/
      workflowpatterns2#>
6 [checkAppropriatenessRuleAlpha2iterateProfileAddCbPrecond:
7                 (wp:AddCb rdf:type ?a)
8                 (wp:OEM rdf:type ?t)
9                 (wp:iterateProfileAddCb wp:hasActivityProfile ?x1)
10                (wp:iterateProfileAddCb wp:hasActivityProfile ?x2)
11                equal(?x1,wp:AddCb)
12                equal(?x2,wp:AddCb)
13                equal(?x1,?x2)
14                (wp:iterateProfileAddCb wp:hasThreadProfile ?y1)
```

```
15                  (wp:iterateProfileAddCb wp:hasThreadProfile ?y2)
16                  equal(?y1,wp:OEM)
17                  equal(?y2,wp:OEM)
18                  equal(?y1,?y2)
19                  ->
20                  (wp:iterateProfileAddCb wp:preconditionProfileSatisfied true
                        )
21
22 ]
```

7.2.6 Contract Monitor

The Contract Monitor gives coordinators the interface to write their monitoring queries in Event Processing Language EPL as depicted Figure 7.2. It analyzes this query and asks the Monitor module to execute it. Using the monitoring patterns identified previously, the Monitor creates a structure that supports an efficient execution of the monitoring query.

We have realized a preliminary implementation of the mechanism of generating a Colored Petri Net (CPN) from a tree of services expressed in a monitoring query. We have implemented a simple model of CPNs in Java and we created the CPN corresponding to our example as an instance of this model.

Places of the CPN have an independent class. This class extends the `Observable` class of the Java Development Kit JDK. This means that other classes can subscribe and be notified of what is happening in this class. In addition, the class uses a locking mechanism since a single Place can be updated by multiple threads at the same time. To realize this locking we used the `ReentrantLock` class.

We implemented classes for the monitoring patterns identified in chapter 6. These classes implement two interfaces:

- `Observer` so that the pattern can be notified when the input places associated to it are updated by new token elements
- `Runnable` because each pattern will be executed as an independent thread

The current implementation status does not generate the parallel EPL queries that optimize the execution of a monitoring query by Esper. Nevertheless, the remaining task consists in transforming the CPN instance to EPL queries which normally should be straightforward to realize.

7.3 Selecting the most Conformant Software to a Standard

Until now, we have seen the main software modules developed for the CMSD system. Nevertheless, developing collaborative platforms requires integrating other software components. One of our tasks in the European project IMAGINE as part of this thesis consists in realizing this integration. Among these software components are those located at the first and second layer of the PLM Hub in Figure 4.2, (e.g. the workflow engine, the PLM systems).

Although, the development of the CMSD system components is made for prototyping purposes and not for immediate industrial use, the components of the PLM Hub should have enough robustness and be conformant to their respective standards. Accordingly, these components cannot be selected arbitrarily. Indeed, the ASD SSG (Aerospace and Defense Strategic Standardization Group[15]) is a consortium between European manufacturers including the Airbus Group that aim to support effective governance at European level of International and European standards. Thus, in our context, we should follow the recommendations of this organism in terms of conformance to standards prior to selecting the components of the PLM Hub.

Ideally, every implementation of a standard should be identical and thus completely interoperable with any other implementation. However, this is far from reality. Standards, when incorporated into products, tools, and services undergo customizations and extensions because every vendor wants to create a unique selling point as a competitive advantage [LMSW08]. Accordingly, we need formal metrics that allow us to assess whether a software implementation is compliant with a standard specification or not.

Up until the present time, several compliance checking approaches have been proposed. In the software engineering domain, formal approaches were developed that consist in testing whether a software component implements a particular function specified in the standard. This is what is called function coverage [FHP02]. Other approaches perform syntactic coverage in order to test whether an implementation upholds the defined syntax in the standard specification [S.96].

The common shortcoming to all these approaches is that they focus only on one dimension when checking the conformance of a software application with the standard specification.

In the remaining of this chapter we give details on a formal approach that allows us to select the most conformant software application regarding a standard. Indeed shifting from a standard specification could be done on several dimensions (e.g. syntactic,

[15]http://www.asd-ssg.org/

semantic, and functional). Accordingly we consider that there are three dimensions on which the conformance could be initially calculated. Namely: the syntactic difference Δ_{syn} between standard operations and software operations. The semantic difference Δ_{sem} between standard operations parameters and software operations parameters. The functional difference Δ_{fun} between standard operations functionality and software operations functionality. When we determine the value of the difference for each dimension, then we can compute the total conformance of software implementation regarding a standard specification and upon this total conformance decide what software to use. Since these dimensions are independent from each other, we can use vector calculus to compute the total conformance value.

7.3.1 Semantic Difference

Calculating the semantic difference of two operations consists in measuring the semantic difference between the inputs/outputs of an operation specified in the standard and an operation implemented in the software having the same purpose.

To measure the semantic difference, we enriched semantically the parameters of these operations such that we can compute the semantic value of each parameter. To achieve this we associated inputs and outputs to an ontology formalized in OWL. Once the semantic values are obtained, we calculate their difference. For two input parameters p_t and p_s associated respectively to two operations o_t and o_s, we compute the their semantic difference as follows:

$$\begin{cases} \bullet class(p_t) < class(p_s) \Rightarrow \\ \Delta_{sem-input}(pt, ps) = 1 + (NbrProp(p_t) - NbrProp(p_s)) \\ \bullet class(p_t) > class(p_s) \Rightarrow \\ \Delta_{sem-input}(pt, ps) = -1 + (NbrProp(p_t) - NbrProp(p_s)) \\ \bullet class(p_t) \equiv class(p_s) \Rightarrow \\ \Delta_{sem-input}(pt, ps) = NbrProp(p_t) - NbrProp(p_s) = 0 \end{cases} \tag{7.1}$$

where $class(p_t) < class(p_s)$ means that the class $class(p_t)$ subsumes the class $class(p_s)$, and $class(p_t) \equiv class(p_s)$ means that both classes are equivalent. The same computing method is applicable for the output parameters.

The overall $\Delta_{sem-input}$ between two operations o_t and o_s is computed from the $\Delta_{sem-input}(p_t, p_s)$ of all couples of parameters (p_t, p_s) where each couple belongs to a separate dimension. Thus: $\Delta_{sem-input} = \sqrt{\sum \Delta_{sem-input}(p_t, p_s)^2}$.

Since the semantic value of the inputs and the semantic of the outputs are indepen-

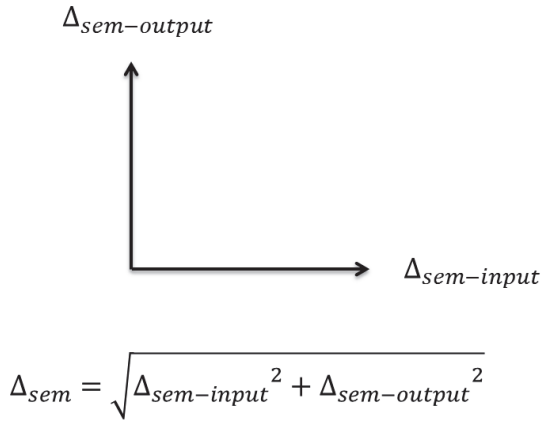


Figure 7.6: Calculation of the semantic delta

dent, then they constitute two orthogonal vectors as depicted in Figure 7.6. Thus the total semantic value is given by: $\Delta_{sem} = \sqrt{\Delta^2_{sem-input} + \Delta^2_{sem-output}}$

7.3.2 Syntactic Difference

The syntactic difference between a standard operation and its counterpart in the software implementation measures the distance between the naming of the operation in the standard and its naming in the software implementation. The syntactic difference concerns the naming of the input parameters of the operation, the naming of output parameters of the operation and the naming of the operation itself. Obviously, the dimensions remain extensible for example to include the naming of the package to which the operation belongs.

The syntactic difference is measured by comparing the similarity of two strings (s_1, s_2): $\Delta_{syn}(s1, s2) = 1 - \frac{|s_1 \cap s_2|}{|s_1|}$

Since the syntactic differences of the inputs, outputs and operation names are independent, then they constitute orthogonal vectors as depicted in Figure 7.7. The total value of the syntactic difference between two operations is given by: $\Delta_{syn} = \sqrt{\Delta^2_{syn-input} + \Delta^2_{syn-output} + \Delta^2_{syn-name} + \Delta^2_{path}}$. We can derive the following syntactic differences between two operations:

- $\Delta_{syn} = 0$ when a function and the associated parameters have exactly the same notation as specified in the standard

- $\Delta_{syn} \neq 0$ when a function or its parameters have differences in their notation in comparison to the standard specification.

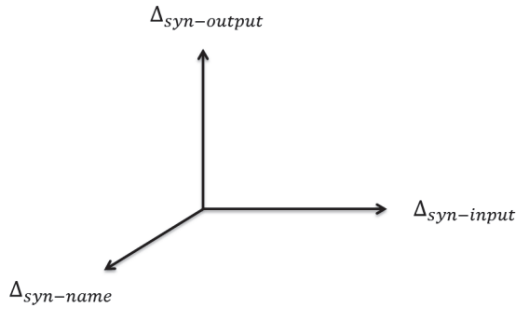


Figure 7.7: Calculation of the syntactic delta

7.3.3 Functional Difference

The functionality provided by an operation depends on the assessment of the post-condition that results upon terminating the execution of this operation. We assume that the post-conditions of operations are captured by the outputs of their execution. Accordingly, we can make the following remarks regarding the functional difference measurement:

- $\Delta_{fun} < 0 \Rightarrow \Delta_{sem-output} < 0$: if an operation has less functionality in comparison to another operation, then it provides less information after its execution
- $\Delta_{fun} < 0 \Rightarrow \Delta_{sem-output} > 0$: if an operation has more functionality in comparison to another operation, then it provides more information after its execution
- $\Delta_{fun} < 0 \Rightarrow \Delta_{sem-output} = 0$: if two operations have equivalent functionality then they will provide the same information

Although the $\Delta_{sem-output}$ is involved in calculating the Δ_{fun}, it is not the unique parameter. Indeed, when looking up a standard specification and a software implementation, we can find three types of operations:

1. **Core operations (CO)**: are the operations that have an added business value. For example in a workflow engine an operation to obtain a process instance has an added business value;
2. **Repetitive Core operations (RCO)**: they extend core operations by providing a collection of data that is provided by the core operation. For example in a workflow engine an operation to obtain all running instances of a process;

The cross-aspect operation **is required** to execute the core operation	• $implement(o_t, cao) \wedge specify(o_s, cao) \Rightarrow deviation = 0$ • $implement(o_t, cao) \wedge \neg specify(o_s, cao) \Rightarrow deviation = 0$ • $\neg implement(o_t, cao) \wedge specify(o_s, cao) \Rightarrow deviation = 0$ • $\neg implement(o_t, cao) \wedge \neg specify(o_s, cao) \Rightarrow deviation = 0$
The cross-aspect operation **is not required** to execute the core operation	• $implement(o_t, cao) \wedge specify(o_s, cao) \Rightarrow deviation = 0$ • $implement(o_t, cao) \wedge \neg specify(o_s, cao) \Rightarrow deviation = 1$ • $\neg implement(o_t, cao) \wedge specify(o_s, cao) \Rightarrow deviation = -1$ • $\neg implement(o_t, cao) \wedge \neg specify(o_s, cao) \Rightarrow deviation = 0$

Table 7.1: Deviation introduced by cross-aspect operations

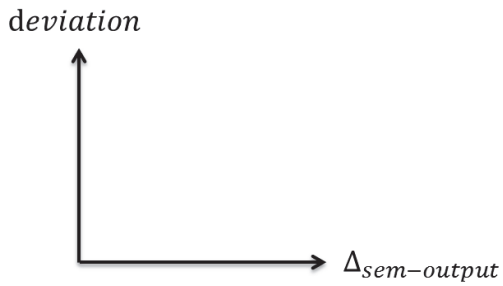
3. **Cross-aspect operation**: are the operations that do not have an added business value but sometimes they are required to run core operations. For example in a workflow engine, one needs to connect to the workflow engine by providing his credentials prior to obtaining the right to obtain the process instance.

This classification of operations implies that:

1. We should include a new parameter called deviation induced by cross-aspect operations when calculating the difference between the software implementation of a core operation and its counterpart in the standard specification. Table 7.1 summarizes the possible situations. In this table two predicates are used to characterize each situation:

   - $implement(o_t, cao)$: means that the software operation implements internally the cross-aspect operation. For example the operation `getProcessInstance` implements internally the `connect` operation. In this case the predicate $implement(getProcessInstance, Connect) = TRUE$
   - $specify(o_s, cao)$: means that the standard operation specifies that it uses the cross-aspect operation internally. For example in WfMC, it is specified that `WMFetchProcessInstance` does not use internally the `WMConnect` operation. Thus $specify(WMFetchProcessInstance, WMConnect) = FALSE$

   In this case, the deviation induced by the cross-aspect operation is independent from the information provided by the core operation. Thus they constitute two independent vectors as depicted in Figure 7.8. The value of the functional difference



$$\Delta_{fun} = \overset{+}{-}\sqrt{deviation^2 + \Delta_{sem-output}{}^2}$$

Figure 7.8: Calculation of the functional delta

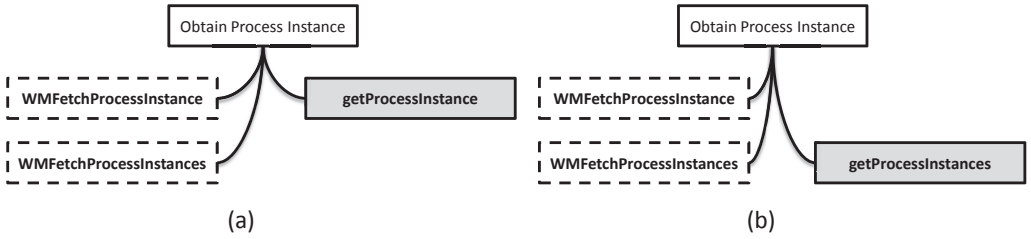


Figure 7.9: An example of difference between the standard specification and software implementation

is given by the equation:

$$\Delta_{fun}(st, sf) = \begin{cases} \sqrt{\Delta^2_{sem-output} + deviation^2} & \text{for} \\ \qquad\qquad \Delta_{sem-output} > 0 & \\ -\sqrt{\Delta^2_{sem-output} + deviation^2} & \text{for} \\ \qquad\qquad \Delta_{sem-output} < 0 & \end{cases} \tag{7.2}$$

2. We should include the cases when there are differences in implementing or specifying the core operations and the repetitive core operations. For example for the functionality **obtain process instance**, we could have the following cases depicted in Figure 7.9.

   In Figure 7.9 on the left hand side, the software implements the core operation but not the repetitive core operation and vice versa for (b). Since the core operation and the repetitive core operation cover each other. In other terms the result obtained by the core operation can be computed from the result obtained by the repetitive core operation with further processing. Accordingly, we need to take into account

The Standard specifies:	**The Software implements:**	Δ_{total}
CO	CO	f_0
CO	RCO	f_1
$\emptyset$	CO	f_2
$\emptyset$	RCO	f_3
CO		f_4
RCO	$\emptyset$	f_5
RCO	RCO	f_0
RCO	CO	f_6
CO, RCO	CO	$-\sqrt{f_0^2 + (\lvert f_4 \rvert - f_6\rvert)^2}$
CO, RCO	RCO	$\sqrt{f_0^2 + (\lvert f_4 \rvert - f_1\rvert)^2}$
CO	CO, RCO	$\sqrt{f_0^2 + f_3^2}$
RCO	CO, RCO	$\sqrt{f_0^2 + f_2^2}$
CO, RCO	$\emptyset$	$-\sqrt{f_4^2 + f_5^2}$
$\emptyset$	CO, RCO	$\sqrt{f_2^2 + f_3^2}$
CO, RCO	CO, RCO	$\sqrt{2} * f_0$

Table 7.2: All possible cases to compute Δ_{total}

this coverage when calculating the Δ_{total} for all possible cases. There are totally 15 cases summarized in Table 7.2:

In Table 7.2 the Δ_{total} is equal to Δ_{fun} in some cases because the operations have the same core operation but belong to different categories (CO or RCO). Thus their functional difference is measured from the difference of the outputs that they provide. In the other cases we include the other dimensions when computing Δ_{total} because the two operations belong to the same category. In these cases $\Delta_{total} = f_0 = \sqrt{\Delta_{fun}^2 + \Delta_{sem-input}2 + \Delta_{syn}2}$. Since from a functional point of view the functions are equivalent, then we need to include deviations induced by the semantics of the inputs and the syntax. For the other cases, since the standard is functionally different from the implementation then there is no need to include the semantic of the inputs and the syntax.

In the remaining cases (lines 8 to 14 of Table 7.2) both categories are mixed and thus we compute the difference between operations of each categories and then we combine them to obtain the final result.

7.3.4 Conformance Testing Approach

We have implemented a software environment that is used to compute the conformance between the standard operations and the software operations and to report the results.

To illustrate our approach, we will assess the conformance of the administration interface of the workflow engine TWS to the workflow systems standard that is the WfMC. Since the WfMC specification is paper based (informal), we should construct a model of these interfaces as well as for the interfaces of TWS Workflow engine. Then the conformance assessor will consume these models and then generate a graphic that depicts the conformance for each service.

To construct these models, we used Archimate[16] as a modeling language to capture the operations implemented by a software application and the operations specified in the standard. There are three reasons of using Archimate:

1. Archimate is a modeling language that offers a set of concepts dedicated to capturing standard specifications as well as software modules and their operations;

2. Archimate was already in use within the Airbus Group Innovations, for example to formalize the specification the systems engineering standard ISO 15288, SCORE, ISA-95;

3. Archimate offers a set of roles that we can use to separate between the task of capturing the standard specification that should be done by the standard expert in the organization and the task of capturing the software implementation that should be performed by the software engineer. Thus we used Archimate in the perspective to follow the methodology already in place.

In the Archimate models of the standard specification and the software implementation, the inputs and outputs of their operations are captured through Archimate data objects. The standard and the software application functions are then enriched with semantics using OWL-S to calculate their Δ_{sem}. OWL-S library provides us with the capability to check the relationship between two classes (`hasSuperClass`, `hasSubClass`, `hasEquivalentClass`) as well as the capability to calculate the number of properties of each class (`listProperties`).

To build the Archimate model, on one hand, the software expert should take the role of **Application Architect**. As its designation indicates, this role is appropriate for the software expert since it allows him to interlink the software functionalities with the domain functionality ontology. Moreover, this role restricts the software expert from

[16] www.opengroup.org/subjectareas/enterprise/archimate

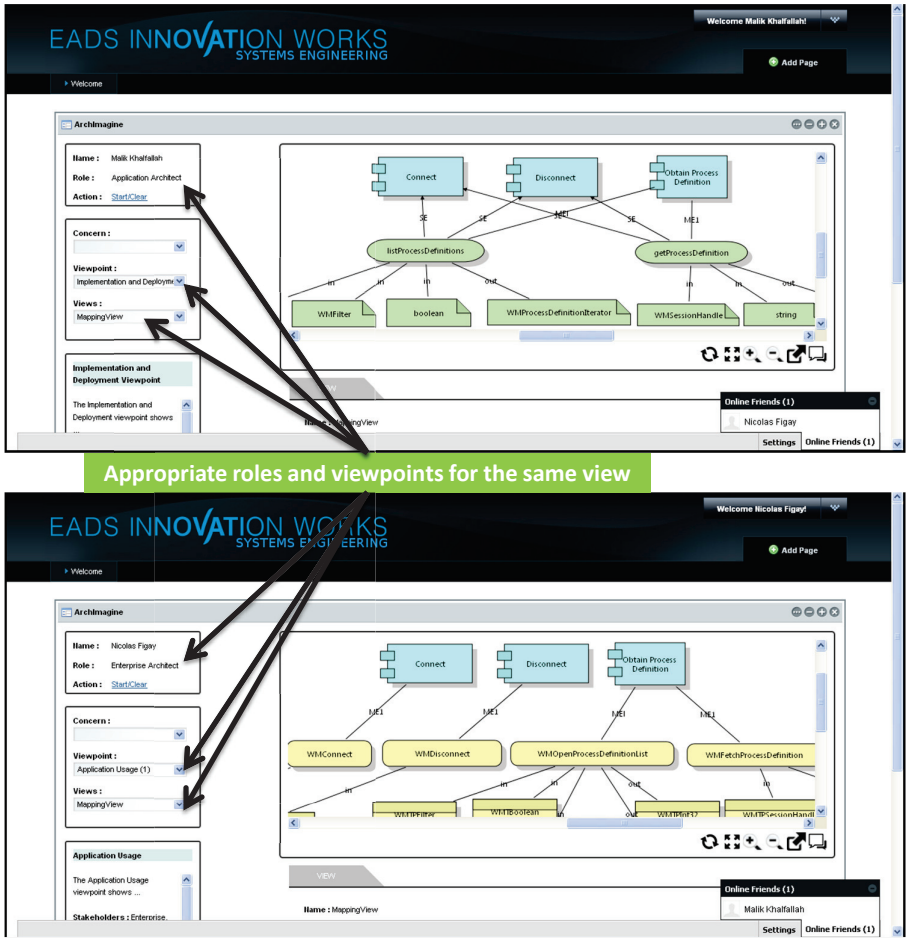


Figure 7.10: A software environment to capture the standard and the software operations in Archimate

accessing the mapping of functionalities between the standard specification and the domain functionalities.

On the other hand, the standard expert takes the role **Enterprise Architect**. This role is appropriate for the standard expert since it allows him to interconnect the standard functionalities with the domain identified functionalities. Moreover, this role restricts the standard expert from accessing the mapping of functionalities between the software interface and the domain functionalities while sharing the same Archimate model.

Figure 7.10 illustrates the interfaces displayed to the standard expert and to the software expert after they have logged in and received their corresponding roles.

7.3.5 Compliance of TWS with the WfMC

We tested the current implementation of the workflow engine TWS through the web services interface provided by the software. Although TWS claims that it is fully compliant

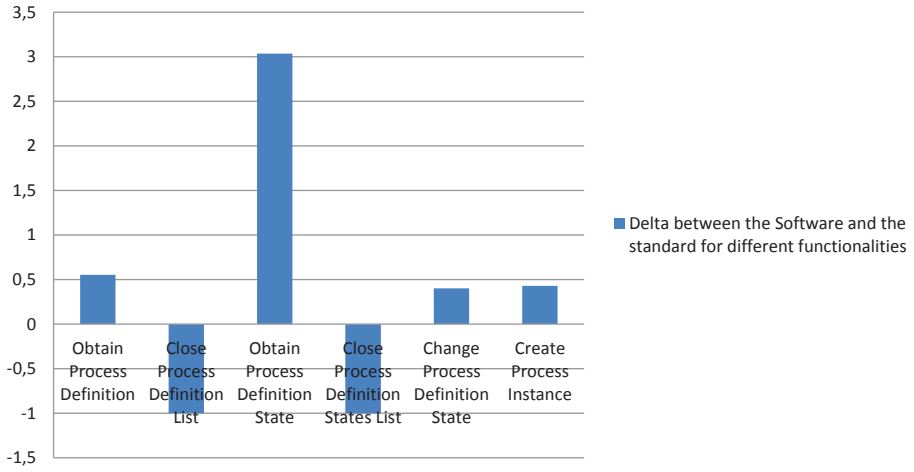


Figure 7.11: Compliance of TWS implementation with the WfMC standard for some operations

with the standard[17] we have found some gaps in its implementation and our measurements give an overview on how much each function of TWS is far from the WfMC specification by considering all dimensions as depicted in Figure 7.11.

From Figure 7.11, we notice that there is a difference between what is specified in the WfMC standard and what is implemented in the workflow engine TWS (other differences exist but we left them due to space limitation). Indeed, since each process should provide the capability to **Obtain Process Definition**, the WfMC has specified two functions: `WMOpenProcessDefinitionList` and `WMFetchProcessDefinition`, while TWS has implemented `listProcessDefinition` and `getProcessDefinition` (following the Java naming convention[18]). For this functionality the difference between TWS and the standard is of syntactic nature. For the functionality **Close Process Definition List**, the difference between the implementation and the standard specification equals -1 (calculated using f_4 of Table 7.2) because TWS does not implement this functionality at all while it is specified in the WfMC standard. The functionality **Obtain Process Definition State** is specified in the standard through a *RCO* `WMOpenProcessDefinitionStatesList` and a *CO* `WMFetchProcessDefinitionState`. This functionality is implemented in TWS through a single *RCO* function `listProcessDefinitionState`. Hence, this case corresponds to *CO,RCO* | *RCO* of Table 7.2.

The overall difference between a software application and the standard that it claims it implements is the sum of differences regarding all operations: $TotalDifference = \sum_{i=0}^{N} |\Delta_i|$. This overall measure provides us a formal basis to select the closest implementation to the standard.

[17] https://sourceforge.net/projects/sharkwf/files/shark/6.0-1/tws-6.0-1.doc.pdf pp.14
[18] http://www.oracle.com/technetwork/java/codeconv-138413.html

7.3.6 Discussion

From this preliminary evaluation of the conformance of TWS with the WfMC standard, we can conclude that the difference is mainly syntactic. Nevertheless, we should not neglect this difference. The reason is that vendor-specific terminology for workflow constructs led to an inconsistent vocabulary of workflow terms [S.96]. In order to counter this trend, the first goal of the WfMC was to establish a common terminology and glossary and naming convention [zM04][19]

In addition to the compliance to a standard, we can consider other dimensions for our evaluation. For example, we can include the dimension of support of workflow patterns when assessing TWS. Van der aalst et al. have already performed this assessment. We can use their evaluation by including a new dimension in addition to the standard dimensions.

7.4 Conclusion

In this chapter, we gave an overview of the system that we have implemented to prototype our conceptual framework. We presented the software modules that we have developed for our prototype. Since our prototype is a part of a larger platform (that covers the design phase as well as the manufacturing phase of the product) we developed standards conformance testing approach. This approach assesses the conformance of software modules of this platform regarding their respective standards. This assessment helped us choose the most conformant software components as a backbone for our prototype but also to realize a better integration with the remaining modules.

Our implementation is part of the Airbus Group Innovations deliverable in the project IMAGINE [IMA14]. Furthermore, we are working to enhance it by performing further evaluations to produce an industrial platform in the future. This work is being conducted in the project SIP at SystemX[20].

[19] WfMC Terminology and Glossary: http://www.aiai.ed.ac.uk/project/wfmc/ARCHIVE/DOCS/glossary/glossary.html
[20] http://www.irt-systemx.fr/systemx-lance-le-projet-sip-standards-interoperabilite-plm/?lang=fr

Chapter 8

Conclusion and Perspectives

Contents

8.1 Conclusion

In this chapter, we summarize the results of our dissertation and discuss future research directions for on-the-fly mediation, modifications during run-time and monitoring for cross-organizational processes.

8.1.1 Reminder of the Thesis Context

Engineering Change Management (ECM) is a standardized process that manages change requests on components of a certain product designed within an enterprise or in the context of a cross-organizational collaboration. In a cross-organizational collaboration, an important phase of the ECM process is the *Specification and the Decision on Change*. During this phase the *coordinator* and the *participant* who are working on a particular component of the product. They define the interaction scenario that specifies what messages they should exchange iteratively in order to reach an agreement on the configuration of this component. Thus, they need an adequate modeling language to model this interaction scenario. In addition, when considering the dynamicity of the collaboration environment that we referred to as Dynamic Manufacturing Networks (DMN), they need to manage the modifications that could occur at run-time. Finally, the product architect would need to monitor the unfolding of this interaction scenario and eventually in conjunction with other interaction scenarios in order to take the right decision at the right-time.

8.1.2 Summary of Contributions

In this dissertation, we noticed the need for a high-level language that helps *coordinators* focus on the objectives of the collaboration when specifying contracts representing interaction scenarios with *participants* for a set of product components. This language prevents *coordinators* from addressing technical problems related to the deployment of the cross-organizational process and the development of mediation solutions that are tasks out of their skills. We gave formal foundations to this language using temporal logic and demonstrated how a contract specification can be coupled with the partners' business rules to generate the cross-organizational process.

On-the-fly Mediation. We have proposed a new mediation approach called *on-the-fly mediation*. The on-the-fly mediation consists in considering that there is one generic mediation algorithm. This algorithm is embedded in a software module that captures the messages sent in a cross-organizational process and generates the expected message using one or more mapping functions. These mapping functions are formalized in terms

of *marked automata* so that the mediation algorithm finds the right mapping function to apply in a linear time. In comparison with existing approaches, our algorithm outperforms them since the time that existing approach would take (in the worst case) to find the right mapping function can be exponential.

Management Operations for DMNs. We have provided *coordinators* and *participants* a management framework to perform modifications on the collaboration contract at run-time while shielding all other partners not concerned by a modification from being impacted. To achieve this, we have developed an algorithm that keeps modifications on the contract on hold until this algorithm formally ensures that these modifications will lead to a DMN state where all partners can collaborate correctly.

Furthermore, when multiple modifications (removals) are performed on a contract, we have used the planning-graph algorithm to generate a set of actions that will incorporate missing elements in the contract. Once these elements are incorporated, the underlying cross-organizational process can be generated. We have extended this algorithm in order to embed the generated raw set of actions into an executable recovery workflow. This recovery workflow will guide the right partners to recover the contract and re-launch the collaboration.

Efficient Monitoring. We have provided *coordinators* a means to specify monitoring queries. These queries capture and perform processing on the exchanged messages and then deliver the resultant high-level information to the *coordinator*. The processing consists in calling external services (e.g. aerodynamic simulation services). We have formalized a set of patterns that generate an efficient execution plan for the monitoring query involving a service composition. We proved that using our execution plan, a service is never executed twice for the same inputs. Furthermore, we proved that independent services can be executed in parallel. We compared the efficiency of our plan with industrial monitoring software (Esper) and we obtained positive results.

Implementation and Conformance Testing. We have implemented the proposed collaboration contract framework and integrated it into a comprehensive collaborative platform that is the implementation of the Aeronautic Interoperability Framework [Fig09]. While performing this integration, we noticed that it was necessary that all generic software modules (e.g. the workflow engine) should be compliant with their standards specifications. To ensure that we select the most appropriate one we developed a set of quantitative metrics that test this conformance and report a diagram that allowed us to take the right decision.

8.1.3 Directions for Future Research

Our work on declarative modeling of collaboration using agreements, on-the-fly mediation, generating executable solution plans and patterns for monitoring have generated several questions on their extensibility:

Arguments and Design Process Analysis. Fulfilling each agreement of the contract involving partners needs one or more iterations of the underlying process. The decision of performing a new iteration can be recorded. At the end of the collaboration, there might be a need to analyze all decisions as well as their relationships. The final report is called a debate report of arguments [PC11]. Our framework can be used to capture the argument of terminating an agreement process as well as their relationships. Since it is a report, those relationships should be past temporal relationships. We can construct new Past Temporal Logic patterns to capture these relationships which will enrich the framework developed by [PC11] since they have used simple relationships between arguments.

Re-Extension of the Graph-Plan Algorithm. Our extension of the Planning Graph algorithm can be further extended to take into account other workflow patterns. Indeed, in chapter 5 we only considered the control flow patterns. Nevertheless, other pattern categories exist: Workflow Data Patterns, Workflow Resource Patterns. Including these patterns makes our extension more comprehensive and generates workflows that better cover agents' needs.

Considering other Service Composition Patterns. When defining the monitoring patterns to generate more efficient execution plans for a monitoring query we considered that the service composition in the query has a tree structure. Nevertheless, the composition can be more complex especially when using declarative service composition languages [vdAP06]. A future work regarding this contribution consists in extracting further patterns when the service composition is more complex than a service tree. Moreover, we need to make further performance evaluation of our approach using more services with different message rates in a cloud platform.

Interaction of Standards. In the last chapter we proposed a formal approach that defines quantitative metrics to assess the conformance of a software application regarding a standard specification. This approach allowed us to test and decide on what software to use for our integration with the European project IMAGINE platform. The resultant platform will thus be constituted of several software applications where each software application is conformant to its respective standard. Our framework can be extended to evaluate the compliance of a set of interacting software applications that each claims that

it implements a standard and assess whether these interactions do not lead to coherency issues due to emergent behavior.

Bibliography

[AAB+05] Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Çetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag Maskey, Alex Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stanley B. Zdonik. The design of the borealis stream processing engine. In *CIDR*, pages 277–289, 2005. 67, 68

[ABP09a] Vasilios Andrikopoulos, Salima Benbernou, and Mike P. Papazoglou. Evolving services from a contractual perspective. In *CAiSE*, pages 290–304, 2009. 31

[ABP09b] Vasilios Andrikopoulos, Salima Benbernou, and Mike P. Papazoglou. Evolving services from a contractual perspective. In *CAiSE*, pages 290–304, 2009. 78

[ACc+03] Daniel J. Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Aurora: A new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, August 2003. 67

[AG99] Tariq A. Aldowaisan and Lotfi K. Gaafar. Business process reengineering: an approach for process mapping. *Omega*, 27(5):515 – 524, 1999. 45

[AGS+14] SAMER ABDUL GHAFUR, Parisa Ghodous, Behzad Shariat, Eliane Perna, and Farzad Khosrowshahi. Semantic Interoperability of Knowledge in Feature-based CAD Models. *Computer Aided Design*, 56:45–57, November 2014. 44

[ARCR14] Renaud Angles, Philippe Ramadour, Corine Cauvet, and Sophie Rodier. Adaptation dynamique de processus métier. application au circuit du

médicament à l'ap-hm. *Ingénierie des Systèmes d'Information*, 19(2):35–60, 2014. 56

[ASS+99] Marcos K. Aguilera, Robert E. Strom, Daniel C. Sturman, Mark Astley, and Tushar D. Chandra. Matching events in a content-based subscription system. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '99, pages 53–61, New York, NY, USA, 1999. ACM. 153

[ATH12] W. M. Aalst and A. H. Ter Hofstede. Workflow patterns put into context. *Softw. Syst. Model.*, 11(3):319–323, July 2012. 135

[Bö7] Egon Börger. Modeling workflow patterns from first principles. In *Proceedings of the 26th International Conference on Conceptual Modeling*, ER'07, pages 1–20, Berlin, Heidelberg, 2007. Springer-Verlag. xiii, 60, 61, 128, 132, 134, 135, 145, 190

[Bao13] Aymen Baouab. *Gouvernance et supervision décentralisée des chorégraphies inter-organisationnelles*. These, Université de Lorraine, June 2013. xiii, 65, 66

[BBC05] Andrea Bracciali, Antonio Brogi, and Carlos Canal. A formal approach to component adaptation. *J. Syst. Softw.*, 74(1):45–54, January 2005. 51

[BCG+05] Boualem Benatallah, Fabio Casati, Daniela Grigori, Hamid R. Motahari Nezhad, and Farouk Toumani. Developing adapters for web services integration. In *CAiSE*, pages 415–429, 2005. 5, 31, 34, 50, 51, 53, 93, 94

[BCGG10] L. Baresi, M. Caporuscio, C. Ghezzi, and S. Guinea. Model-driven management of services. In *Web Services (ECOWS), 2010 IEEE 8th European Conference on*, pages 147–154, Dec 2010. 68

[BFPG12] Aymen Baouab, Walid Fdhila, Olivier Perrin, and Claude Godart. Towards decentralized monitoring of supply chains. In *Web Services (ICWS), 2012 IEEE 19th International Conference on*, pages 600–607, June 2012. 67, 68

[BGH+07] Kamal Bhattacharya, Cagdas Gerede, Richard Hull, Rong Liu, and Jianwen Su. Towards formal analysis of artifact-centric business process models. In *Proceedings of the 5th International Conference on Business Process*

Management, BPM'07, pages 288–304, Berlin, Heidelberg, 2007. Springer-Verlag. 48

[BJ94] C. Bussler and S. Jablonski. Implementing agent coordination for workflow management systems using active database systems. In *Research Issues in Data Engineering, 1994. Active Database Systems. Proceedings Fourth International Workshop on*, pages 53–59, Feb 1994. 46

[BJ06] Patrik Berander and Per Jönsson. A goal question metric based approach for efficient measurement framework definition. In *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, ISESE '06, pages 316–325, New York, NY, USA, 2006. ACM. 104

[BMB+00] Jean Bacon, Ken Moody, John Bates, Richard Hayton, Chaoying Ma, Andrew McNeil, Oliver Seidel, and Mark Spiteri. Generic support for distributed applications. *Computer*, 33(3):68–76, March 2000. 153, 159

[BOAT13a] Saïda Boukhedouma, Mourad Oussalah, Zaia Alimazighi, and Dalila Tamzalit. Adaptation patterns for service based inter-organizational workflows. In *IEEE 7th International Conference on Research Challenges in Information Science, RCIS 2013, Paris, France, May 29-31, 2013*, pages 1–10, 2013. 56

[BOAT13b] Saïda Boukhedouma, Mourad Oussalah, Zaia Alimazighi, and Dalila Tamzalit. Flexible loosely coupled inter-organizational workflows using SOA. In *ACS International Conference on Computer Systems and Applications, AICCSA 2013, Ifrane, Morocco, May 27-30, 2013*, pages 1–8, 2013. 56

[BP06] Antonio Brogi and Razvan Popescu. Automated generation of bpel adapters. In *Proceedings of the 4th International Conference on Service-Oriented Computing*, ICSOC'06, pages 27–39, Berlin, Heidelberg, 2006. Springer-Verlag. 52

[BS03] E. Borger and Robert F. Stark. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003. 61

[BSBM04] Lucas Bordeaux, Gwen Salaün, Daniela Berardi, and Massimo Mecella. When are two web services compatible? In *TES*, pages 15–28, 2004. 50, 93

[CAV12] Marco Comuzzi, Samuil Angelov, and Jochem Vonk. Patterns to enable mass-customized business process monitoring. In *Proceedings of the 24th International Conference on Advanced Information Systems Engineering*, CAiSE'12, pages 445–459, Berlin, Heidelberg, 2012. Springer-Verlag. 67, 68, 153

[CCPP98] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. *Data Knowl. Eng.*, 24(3):211–238, January 1998. 54

[CD10] David Chen and Nicolas Daclin. *Framework for Enterprise Interoperability*, pages 77–88. ISTE, 2010. 29

[CH09] David Cohn and Richard Hull. Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Data Eng. Bull.*, 32(3):3–9, 2009. 48

[CM04] Anis Charfi and Mira Mezini. Hybrid web service composition: business processes meet business rules. In *ICSOC*, pages 30–38, 2004. 73, 86

[CMS06] Emilia Cimpian, Adrian Mocan, and Michael Stollberg. Mediation enabled semantic web services usage. In *ASWC*, pages 459–473, 2006. 29

[CS03] Amit K. Chopra and Munindar P. Singh. Nonmonotonic commitment machines. In *Workshop on Agent Communication Languages*, pages 183–200, 2003. 47

[CV11] Marco Comuzzi and Irene T. P. Vanderfeesten. Product-based workflow design for monitoring of collaborative business processes. In *Proceedings of the 23rd International Conference on Advanced Information Systems Engineering*, CAiSE'11, pages 154–168, Berlin, Heidelberg, 2011. Springer-Verlag. 45, 153

[DAC98] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Property specification patterns for finite-state verification. In *Proceedings of the Second Workshop on Formal Methods in Software Practice*, FMSP '98, pages 7–15, New York, NY, USA, 1998. ACM. 46, 47, 83

[DDDM11] P. Delias, A Doulamis, N. Doulamis, and N. Matsatsinis. Optimizing resource conflicts in workflow management systems. *Knowledge and Data Engineering, IEEE Transactions on*, 23(3):417–432, March 2011. 139

[DHV11] Elio Damaggio, Richard Hull, and Roman Vaculín. On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles. In *Proceedings of the 9th International Conference on Business Process Management*, BPM'11, pages 396–412, Berlin, Heidelberg, 2011. Springer-Verlag. 48

[DKLW07] Gero Decker, Oliver Kopp, Frank Leymann, and Mathias Weske. Bpel4chor: Extending bpel for modeling choreographies. In *ICWS*, pages 296–303, 2007. 73

[DKRR98] Hasan Davulcu, Michael Kifer, C. R. Ramakrishnan, and I. V. Ramakrishnan. Logic based modeling and analysis of workflows. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '98, pages 25–33, New York, NY, USA, 1998. ACM. 46

[DME+13] Codé Diop, Emna Mezghani, Ernesto Exposito, Christophe Chassot, and Khalil Drira. Qos-driven autonomic abilities through a multi-homed transport protocol. In *27th IEEE International Conference on Advanced Information Networking and Applications, AINA 2013, Barcelona, Spain, March 25-28, 2013*, pages 645–652, 2013. 44

[DSW06] Marlon Dumas, Murray Spork, and Kenneth Wang. Adapt or perish: Algebra and visual notation for service interface adaptation. In *Business Process Management*, pages 65–80, 2006. 50, 51

[DvdAtH05] Marlon Dumas, Wil M. van der Aalst, and Arthur H. ter Hofstede. *Process-aware Information Systems: Bridging People and Software Through Process Technology*. John Wiley & Sons, Inc., New York, NY, USA, 2005. 28, 61, 62

[EBMN13] Maryam Eslamichalandar, Kamel Barkaoui, and Hamid Reza Motahari-Nezhad. Dynamic adapter reconfiguration in the context of business protocol evolution. In *CSE*, pages 301–308, 2013. 51, 93

[EFGK03] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003. 67, 153

[EKK10] Marwane El Kharbili and Tobias Keil. Bringing agility to business process management: Rules deployment in an soa. In Walter Binder and

Schahram Dustdar, editors, *Emerging Web Services Technology Volume III*, Whitestein Series in Software Agent Technologies and Autonomic Computing, pages 157–170. Birkhauser Basel, 2010. 82

[EKR95] Clarence Ellis, Karim Keddara, and Grzegorz Rozenberg. Dynamic change within workflow systems. In *Proceedings of Conference on Organizational Computing Systems*, COCS '95, pages 10–21, New York, NY, USA, 1995. ACM. 188

[EME00] Murman E., Walton M., and Rebentisch E. Challenges in the better, faster, cheaper era of aeronautical design, engineering and manufacturing. *Aeronautical Journal*, 104(3):481–489, 2000. 2, 82

[EP98] Hans-Erik Eriksson and Magnus Penker. *Business Modeling With UML: Business Patterns at Work*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1998. 86, 89

[FB02] Dieter Fensel and Christoph Bussler. The web service modeling framework wsmf. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002. 29

[FHP02] E. Farchi, A Hartman, and S.S. Pinter. Using a model-based test generator to test for standard conformance. *IBM Systems Journal*, 41(1):89–110, 2002. 194

[Fig09] Nicolas Figay. *Interoperability of Technical Enterprise Application*. These de doctorat en informatique, Universite Lyon 1, December 2009. 32, 33, 207

[Fis11] Michael Fisher. *An Introduction to Practical Formal Methods Using Temporal Logic*. Wiley, 2011. 82

[Fok00] Wan Fokkink. *Introduction to Process Algebra*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 2000. 102

[FRMR12] W. Fdhila, S. Rinderle-Ma, and M. Reichert. Change propagation in collaborative processes scenarios. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on*, pages 452–461, Oct 2012. 55, 59

[FTG+14] Nicolas Figay, David Tchoffa, Parisa Ghodous, Ernesto Exposito, and Abderrahman El Mhamedi. Dynamic manufacturing network, plm hub and

business standards testbed. In Kai Mertins, Frederick BÃl'naben, Raul Poler, and Jean-Paul Bourrieres, editors, *Enterprise Interoperability VI*, volume 7 of *Proceedings of the I-ESA Conferences*, pages 453–463. Springer International Publishing, 2014. viii, 14, 35

[GBno6] Luciano García-Bañuelos. An asml executable model for ws-bpel with orthogonal transactional behavior. In *Proceedings of the 4th International Conference on Business Process Management*, BPM'06, pages 401–406, Berlin, Heidelberg, 2006. Springer-Verlag. 130

[GDo7] R. Garbade and W.R. Dolezal. Dmuatairbus evolution of the digital mock-up (dmu) at airbus to the centre of aircraft development. In Frank-Lothar Krause, editor, *The Future of Product Development*, pages 3–12. Springer Berlin Heidelberg, 2007. 30

[GKMW11] Sam Guinea, Gabor Kecskemeti, Annapaola Marconi, and Branimir Wetzstein. Multi-layered monitoring and adaptation. In *Proceedings of the 9th International Conference on Service-Oriented Computing*, ICSOC'11, pages 359–373, Berlin, Heidelberg, 2011. Springer-Verlag. 68

[GMSo6] Guido Governatori, Zoran Milosevic, and Shazia Wasim Sadiq. Compliance checking between business processes and business contracts. In *EDOC*, pages 221–232, 2006. 49

[Govo5] Guido Governatori. Representing business contracts in ruleml. *Int. J. Cooperative Inf. Syst.*, 14(2-3):181–216, 2005. 49

[Grooo] Business R. Group. Defining Business Rules ˜ What Are They Really? Technical report, 2000. 86

[Har88] David Harel. On visual formalisms. *Commun. ACM*, 31(5):514–530, May 1988. 121

[HDDM+11] Richard Hull, Elio Damaggio, Riccardo De Masellis, Fabiana Fournier, Manmohan Gupta, Fenno Terry Heath, III, Stacy Hobson, Mark Linehan, Sridhar Maradugu, Anil Nigam, Piwadee Noi Sukaviriya, and Roman Vaculin. Business artifacts with guard-stage-milestone lifecycles: Managing artifact interactions with conditions and events. In *Proceedings of the 5th ACM International Conference on Distributed Event-based System*, DEBS '11, pages 51–62, New York, NY, USA, 2011. ACM. 48, 53

[HDF+11] Richard Hull, Elio Damaggio, Fabiana Fournier, Manmohan Gupta, Fenno Terry Heath, III, Stacy Hobson, Mark Linehan, Sridhar Maradugu, Anil Nigam, Piyawadee Sukaviriya, and Roman Vaculin. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *Proceedings of the 7th International Conference on Web Services and Formal Methods*, WS-FM'10, pages 1–24, Berlin, Heidelberg, 2011. Springer-Verlag. 48

[HFoSE11] C. Haskins, K. Forsberg, and International Council on Systems Engineering. *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities; INCOSE-TP-2003-002-03.2.1*. INCOSE, 2011. 86

[HIL94] Eben M. Haber, Yannis E. Ioannidis, and Miron Livny. Foundations of visual metaphors for schema display. *J. Intell. Inf. Syst.*, 3(3-4):263–298, July 1994. 121

[HMS11a] Thomas T. Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Designing a cross-organizational case management system using dynamic condition response graphs. In *EDOC*, pages 161–170, 2011. 47, 77

[HMS11b] Thomas T. Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Nested dynamic condition response graphs. In *FSEN*, pages 343–350, 2011. 47

[HSS+14] Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik, and Robert Grimm. A catalog of stream processing optimizations. *ACM Comput. Surv.*, 46(4):46:1–46:34, March 2014. 67

[HVN+12] Ourania Hatzi, D. Vrakas, Mara Nikolaidou, N. Bassiliades, Dimosthenis Anagnostopoulos, and L. Vlahavas. An integrated approach to automated semantic web service composition through planning. *Services Computing, IEEE Transactions on*, 5(3):319–332, Third 2012. 62, 64, 146

[IMA11] IMAGINE. Innovative end-to-end Management of Dynamic Manufacturing Networks (IMAGINE Project). Technical report, 2011. [Online; http://www.imagine-futurefactory.eu/]. 3, 30, 176

[IMA14] IMAGINE. Imagine deliverable 4.1 - aerospace living lab. Technical report, 2014. [Online; http://www.imagine-futurefactory.eu/]. 204

[Jen94] Kurt Jensen. An introduction to the theoretical aspects of coloured petri nets. In *A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium*, pages 230–272, London, UK, UK, 1994. Springer-Verlag. 155, 156, 164

[JGPG12] Ricardo Jardim-Goncalves, Keith Popplewell, and Antonio Grilo. Sustainable interoperability: The future of internet based industrial enterprises. *Comput. Ind.*, 63(8):731–738, October 2012. 29

[Joe99] Gregor Joeris. Defining flexible workflow execution behaviors. In *University of Ulm*, pages 49–55, 1999. 188

[KBFG14] Malik Khalfallah, Mahmoud Barhamgi, Nicolas Figay, and Parisa Ghodous. An Architecture for a Centralized Mediation In Dynamic Networks. In *21th IEEE International Conference on Web Services ICWS 2014*, July 2014. 8, 39, 53

[KEP00] Gerhard Knolmayer, Rainer Endl, and Marcel Pfahrer. Modeling processes and workflows by business rules. In *Business Process Management, Models, Techniques, and Empirical Studies*, pages 16–29, London, UK, UK, 2000. Springer-Verlag. 46, 76

[KFB+13] Malik Khalfallah, Nicolas Figay, Mahmoud Barhamgi, Catarina Ferreira Da Silva, and Parisa Ghodous. Towards combining declarative specification with on-the-fly mediation. In *IEEE SCC*, pages 691–698, 2013. 8, 39, 53, 176

[KFBG13a] Malik Khalfallah, Nicolas Figay, Mahmoud Barhamgi, and Parisa Ghodous. Controlling the evolution of product-based collaboration contracts. In *IEEE SCC*, pages 713–720, 2013. 9, 40

[KFBG13b] Malik Khalfallah, Nicolas Figay, Mahmoud Barhamgi, and Parisa Ghodous. Product-based business processes interoperability. In *SAC*, pages 1472–1473, 2013. 8, 39, 53

[KFBG14a] Malik Khalfallah, Nicolas Figay, Mahmoud Barhamgi, and Parisa Ghodous. Model Driven Conformance Testing for Standardized Services. In *11th IEEE International Conference on Services Computing (SCC 2014)*, July 2014. 10, 40

[KFBG14b] Malik Khalfallah, Nicolas Figay, Mahmoud Barhamgi, and Parisa Ghodous. Patterns for Monitoring Parallel Processes. In *11th IEEE International Conference on Services Computing (SCC 2014)*, July 2014. 9, 40, 68

[KFFDSG14] Malik Khalfallah, Nicolas Figay, Catarina Ferreira Da Silva, and Parisa Ghodous. A cloud-based platform to ensure interoperability in aerospace industry. *Journal of Intelligent Manufacturing*, pages 1–11, 2014. 105

[KKR12] Jens Kolb, Klaus Kammerer, and Manfred Reichert. Updatable process views for user-centered adaption of large process models. In *Proceedings of the 10th International Conference on Service-Oriented Computing*, ICSOC'12, pages 484–498, Berlin, Heidelberg, 2012. Springer-Verlag. 57

[KLC13] A Kashlev, Shiyong Lu, and A Chebotko. Coercion approach to the shimming problem in scientific workflows. In *Services Computing (SCC), 2013 IEEE International Conference on*, pages 416–423, June 2013. 52

[KMNB+09] W. Kongdenfha, H.R. Motahari-Nezhad, B. Benatallah, F. Casati, and R. Saint-Paul. Mismatch patterns and adaptation aspects: A foundation for rapid development of web service adapters. *Services Computing, IEEE Transactions on*, 2(2):94–107, April 2009. 51, 53, 108

[KMSF09] Mick Kerrigan, Adrian Mocan, Elena Paslaru Bontas Simperl, and Dieter Fensel. Modeling semantic web services with the web service modeling toolkit. *J. Network Syst. Manage.*, 17(3):326–342, 2009. 29

[KNBSP14] Woralak Kongdenfha, Hamid R. Motahari Nezhad, Boualem Benatallah, and Regis Saint-Paul. Web service adaptation: Mismatch patterns and semi-automated approach to mismatch identification and adapter development. In *Web Services Foundations*, pages 245–272. 2014. 53, 93

[KR13] Jens Kolb and Manfred Reichert. Data flow abstractions and adaptations through updatable process views. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1447–1453, New York, NY, USA, 2013. ACM. 57

[KRM+12] David Knuplesch, Manfred Reichert, Jürgen Mangler, Stefanie Rinderle-Ma, and Walid Fdhila. Towards compliance of cross-organizational processes and their changes - research challenges and state of research. In *Business Process Management Workshops*, pages 649–661, 2012. 31, 86

[KSPBC06] Woralak Kongdenfha, RÃľgis Saint-Paul, Boualem Benatallah, and Fabio Casati. An aspect-oriented framework for service adaptation. In Asit Dan and Winfried Lamersdorf, editors, *Service-Oriented Computing âĂŞ ICSOC 2006*, volume 4294 of *Lecture Notes in Computer Science*, pages 15–26. Springer Berlin Heidelberg, 2006. 107

[KT13] Bilal Kanso and Safouan Taha. Temporal constraint support for ocl. In Krzysztof Czarnecki and Gorel Hedin, editors, *Software Language Engineering*, volume 7745 of *Lecture Notes in Computer Science*, pages 83–103. Springer Berlin Heidelberg, 2013. 185

[LFB+12] Jae-Hyun Lee, Steven J. Fenves, Conrad Bock, Hyo-Won Suh, Rachuri Sudarsan, Xenia Fiorentini, and Ram D. Sriram. A semantic product modeling framework and its application to behavior evaluation. *IEEE T. Automation Science and Engineering*, 9(1):110–123, 2012. 78

[LG12] Lam-Son Lê and Aditya Ghose. Contracts and goals = roles? In *Proceedings of the 31st International Conference on Conceptual Modeling*, ER'12, pages 252–266, Berlin, Heidelberg, 2012. Springer-Verlag. 49

[LH93] Wei Li and Sallie Henry. Object-oriented metrics that predict maintainability. *J. Syst. Softw.*, 23(2):111–122, November 1993. 104

[Lim09] Moises Lima Dutra. *An Ontology-Based Approach to Manage Conflicts in Collaborative Design*. ThÃĺse de doctorat en informatique, UniversitÃľ Claude Bernard Lyon 1, November 2009. 44

[LMSW08] Grace A. Lewis, Edwin J. Morris, Soumya Simanta, and Lutz Wrage. Why standards are not enough to guarantee end-to-end interoperability. In *ICCBSS*, pages 164–173, 2008. 29, 61, 62, 194

[MB00] T. Murata and A Borgida. Handling of irregularities in human centered systems: a unified framework for data and processes. *Software Engineering, IEEE Transactions on*, 26(10):959–977, Oct 2000. 188

[MHD14] Emna Mezghani, Riadh Ben Halima, and Khalil Drira. DRAAS: dynamically reconfigurable architecture for autonomic services. In *Web Services Foundations*, pages 483–505. 2014. 44

[ML13] Andrea Marrella and Yves Lesperance. Synthesizing a library of process templates through partial-order planning algorithms. In *14th International*

Working Conference on Business Process Modeling, Development and Support (BPMDS 2013), in conjunction with CAiSE 2013, 2013. 63, 64

[MMK02] M.D.R.-Moreno and P. Kearney. Integrating {AI} planning techniques with workflow management system. *Knowledge-Based Systems*, 15(5):285 – 291, 2002. 63

[MMR11] A Marrella, M. Mecella, and A Russo. Featuring automatic adaptivity through workflow enactment and planning. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th International Conference on*, pages 372–381, Oct 2011. 63, 64

[MNBM+07] Hamid Reza Motahari Nezhad, Boualem Benatallah, Axel Martens, Francisco Curbera, and Fabio Casati. Semi-automated adaptation of service interactions. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 993–1002, New York, NY, USA, 2007. ACM. 51

[MR09] Christophe Merlo and Mickael Romain. Towards PLM interoperability between aeronautical partners. In Chris McMahon Bath University, editor, *Product Lifecycle Management PLM'09 Supporting the extended enterprise*, pages 163–172, ISBN 0–907776–49–3, Bath, Royaume-Uni, July 2009. 20, 27

[MTS13] Felipe Meneguzzi, Pankaj R. Telang, and Munindar P. Singh. A first-order formalization of commitments and goals for planning. In Marie desJardins and Michael L. Littman, editors, *AAAI*. AAAI Press, 2013. 47

[NC03] A Nigam and N.S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003. 48

[NGT04] Dana Nau, Malik Ghallab, and Paolo Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. 9, 46, 59, 62, 127, 149

[NVSM07] Meenakshi Nagarajan, Kunal Verma, Amit P. Sheth, and John A. Miller. Ontology driven data mediation in web services. *Int. J. Web Service Res.*, 4(4):104–126, 2007. 29

[OBRC10] Daniel V. Oppenheim, Saeed Bagheri, Krishna Ratakonda, and Yi-Min Chee. Coordinating distributed operations. In *ICSOC Workshops*, pages 213–224, 2010. 73, 113

[ODtHvdA06] Chun Ouyang, Marlon Dumas, Arthur H. M. ter Hofstede, and Wil M. P. van der Aalst. From bpmn process models to bpel web services. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '06, pages 285–292, Washington, DC, USA, 2006. IEEE Computer Society. 90, 135

[OMG11] OMG. Product lifecycle management services. Technical report, mai 2011. [Online; http://www.omg.org/spec/PLM/]. 2, 20, 22

[OVC14] Daniel V. Oppenheim, Lav R. Varshney, and Yi-Min Chee. Work as a service. In *Advanced Web Services*, pages 409–430. 2014. 80, 84

[PC11] T. Polacsek and L. Cholvy. A framework to report and to analyse a debate. In *Computer Supported Cooperative Work in Design (CSCWD), 2011 15th International Conference on*, pages 84–90, June 2011. 208

[PTDL07] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, November 2007. 94

[PvdA06] M. Pesic and W. M. P. van der Aalst. A declarative approach for flexible business processes management. In *Proceedings of the 2006 International Conference on Business Process Management Workshops*, BPM'06, pages 169–180, Berlin, Heidelberg, 2006. Springer-Verlag. 46

[Ray06] Daniel P Raymer. *Aircraft design: A conceptual approach*. AIAA education series. American Institute of Aeronautics and Astronautics, Reston, Va., 4. ed. edition, 2006. 16, 17, 92

[RD98] Manfred Reichert and Peter Dadam. Adept flex - supporting dynamic changes of workflows without loosing control. *Journal of Intelligent Information Systems*, 10:93–129, 1998. 54

[RLvdA03] Hajo A. Reijers, Selma Limam, and Wil M. P. van der Aalst. Product-based workflow design. *J. of Management Information Systems*, 20(1):229–262, 2003. 45

[RMBCO07] Maria Dolores R-Moreno, Daniel Borrajo, Amedeo Cesta, and Angelo Oddi. Integrating planning and scheduling in workflow domains. *Expert Systems with Applications*, 33(2):389 – 406, 2007. 63

[RRD03] Manfred Reichert, Stefanie Rinderle, and Peter Dadam. Adept workflow management system: Flexible support for enterprise-wide business processes. In *Proceedings of the 2003 International Conference on Business Process Management*, BPM'03, pages 370–379, Berlin, Heidelberg, 2003. Springer-Verlag. 54

[RRD04a] Stefanie Rinderle, Manfred Reichert, and Peter Dadam. Correctness criteria for dynamic changes in workflow systems: A survey. *Data Knowl. Eng.*, 50(1):9–34, July 2004. 54

[RRD04b] Stefanie Rinderle, Manfred Reichert, and Peter Dadam. Correctness criteria for dynamic changes in workflow systems: A survey. *Data Knowl. Eng.*, 50(1):9–34, July 2004. 188

[RRKD05] Manfred Reichert, Stefanie Rinderle, Ulrich Kreher, and Peter Dadam. Adaptive process management with adept2. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 1113–1114, Washington, DC, USA, 2005. IEEE Computer Society. 54, 59

[RW12] Manfred Reichert and Barbara Weber. *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies*. Springer, 2012. 54

[RWR06] Stefanie Rinderle, Andreas Wombacher, and Manfred Reichert. Evolution of process choreographies in dychor. In *Proceedings of the 2006 Confederated International Conference on On the Move to Meaningful Internet Systems: CoopIS, DOA, GADA, and ODBASE - Volume Part I*, ODBASE'06/OTM'06, pages 273–290, Berlin, Heidelberg, 2006. Springer-Verlag. 57, 59

[S.96] Joosten S. Workpad: a conceptual framework for process analysis and design. *ACM Transactions of Office Information Systems*, 1996. 194, 204

[Sad12] M H Sadraey. *Aircraft Design: A Systems Engineering Approach*. Aerospace Series. Wiley, 2012. xiii, 16, 17, 18, 19, 84

[SAS10] SASIG. Vda engineering change management document. Technical report, september 2010. xiii, 21, 22, 23, 24, 25, 27

[SCMF06] Michael Stollberg, Emilia Cimpian, Adrian Mocan, and Dieter Fensel. A semantic web mediation architecture. In *CSWWS*, pages 3–22, 2006. 29

[SEG09] R. Seguel, R. Eshuis, and P. Grefen. Constructing minimal protocol adaptors for service composition. In *Proceedings of the 4th Workshop on Emerging Web Services Technology*, WEWST '09, pages 29–38, New York, NY, USA, 2009. ACM. 52

[SEG10] R. Seguel, R. Eshuis, and P. Grefen. Generating minimal protocol adaptors for loosely coupled services. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 417–424, July 2010. 52

[SGN07] Shazia Wasim Sadiq, Guido Governatori, and Kioumars Namiri. Modeling control objectives for business process compliance. In *BPM*, pages 149–164, 2007. 82

[Sin00] Munindar P. Singh. A social semantics for agent communication languages. In *Issues in Agent Communication*, pages 31–45, 2000. 47

[SK03] Shane Sendall and Wojtek Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE Softw.*, 20(5):42–45, September 2003. 109

[SKT02] Pinar Senkul, Michael Kifer, and Ismail H. Toroslu. A logical framework for scheduling workflows under resource allocation constraints. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 694–705. VLDB Endowment, 2002. 46

[SMG+06] Catarina Ferreira Da Silva, Lionel Médini, Samer Abdul Ghafour, Patrick Hoffmann, Parisa Ghodous, and Celson Lima. Semantic interoperability of heterogeneous semantic resources. *Electr. Notes Theor. Comput. Sci.*, 150(2):71–85, 2006. 44

[SMTA95] Munindar P. Singh, Greg Meredith, Christine Tomlinson, and Paul C. Attie. An event algebra for specifying and scheduling workflows. In *Workflows,âĂİ Proceedings 4th International Conference on Database System for Advance Application*, pages 53–60, 1995. 46

[Spi00] Marc Spielmann. Model checking abstract state machines and beyond. In *Proceedings of the International Workshop on Abstract State Machines, The-*

ory and Applications, ASM '00, pages 323–340, London, UK, UK, 2000. Springer-Verlag. 60, 61

[TABFB09] Y. Taher, A Ait-Bachir, M.-C. Fauvet, and D. Benslimane. Diagnosing incompatibilities in web service interactions for automatic generation of adapters. In *Advanced Information Networking and Applications, 2009. AINA '09. International Conference on*, pages 652–659, May 2009. 51

[Tar12] Sasu Tarkoma. *Publish / Subscribe Systems: Design and Principles*. Wiley Publishing, 1st edition, 2012. xiii, 65, 153

[TBSR06] M. Tarek, C. Boutrous-Saab, and S. Rampacek. Verifying correctness of web services choreography. In *Web Services, 2006. ECOWS '06. 4th European Conference on*, pages 306–318, Dec 2006. 55

[TEvdHP11] Oktay Türetken, Amal Elgammal, Willem-Jan van den Heuvel, and Mike P. Papazoglou. Enforcing compliance on business processes through the use of patterns. In *ECIS*, 2011. 82

[TKS14] Pankaj R. Telang, Anup K. Kalia, and Munindar P. Singh. Engineering service engagements via commitments. *IEEE Internet Computing*, 18(3):46–54, 2014. 47, 77

[TLM06] G. Thimm, S. G. Lee, and Y.-S. Ma. Towards unified modelling of product life-cycles. *Comput. Ind.*, 57(4):331–341, May 2006. 20

[TPPvdH11] Yéhia Taher, Michael Parkin, Mike P. Papazoglou, and Willem-Jan van den Heuvel. Adaptation of web service interactions using complex event processing patterns. In *Proceedings of the 9th International Conference on Service-Oriented Computing*, ICSOC'11, pages 601–609, Berlin, Heidelberg, 2011. Springer-Verlag. 51

[TS09] Pankaj R. Telang and Munindar P. Singh. Business modeling via commitments. In *SOCASE*, pages 111–125, 2009. 3, 6, 35, 37, 47, 73

[TS12a] Pankaj R. Telang and Munindar P. Singh. Comma: A commitment-based business modeling methodology and its empirical evaluation. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '12, pages 1073–1080, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems. 47

[TS12b] P.R. Telang and M.P. Singh. Specifying and verifying cross-organizational business models: An agent-oriented approach. *Services Computing, IEEE Transactions on*, 5(3):305–318, Third 2012. 47

[VBKB+14] N. R. T. P. Van Beest, E. Kaldeli, P. Bulanov, J. C. Wortmann, and A. Lazovik. Automated runtime repair of business processes. *Inf. Syst.*, 39:45–79, January 2014. 63

[vdA99] W.M.P van der Aalst. On the automatic generation of workflow processes based on product structures. *Computers in Industry*, 39(2):97 – 111, 1999. 45, 79

[vdALM+10] Wil M. P. van der Aalst, Niels Lohmann, Peter Massuthe, Christian Stahl, and Karsten Wolf. Multiparty contracts: Agreeing and implementing interorganizational processes. *Comput. J.*, 53(1):90–106, 2010. 53, 56, 73

[vdAP06] Wil M. P. van der Aalst and Maja Pesic. Decserflow: Towards a truly declarative service flow language. In *The Role of Business Processes in Service Oriented Architectures*, 2006. 7, 37, 46, 73, 77, 208

[vdAPS09] Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113, 2009. 46

[vdARL01] Wil M. P. van der Aalst, Hajo A. Reijers, and Selma Limam. Product-driven workflow design. In *CSCWD*, pages 397–402, 2001. 45, 79

[vdAtHKB03] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003. 86

[vdAW05] Wil M. P. van der Aalst and Mathias Weske. Case handling: A new paradigm for business process support. *Data Knowl. Eng.*, 53(2):129–162, May 2005. 188

[VHH+11] R. Vaculin, R. Hull, T. Heath, C. Cochran, A Nigam, and P. Sukaviriya. Declarative business artifact centric modeling of decision and knowledge intensive business processes. In *Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International*, pages 151–160, Aug 2011. 48

[VO11] Lav R. Varshney and Daniel V. Oppenheim. On cross-enterprise collaboration. In *BPM*, pages 29–37, 2011. 3, 31

[VRvdA08] Irene T. P. Vanderfeesten, Hajo A. Reijers, and Wil M. P. van der Aalst. Product based workflow support: Dynamic workflow execution. In *CAiSE*, pages 571–574, 2008. 45

[Wes10] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer Publishing Company, Incorporated, 1st edition, 2010. 28, 35

[WHR11] Bochao Wang, A Haller, and F. Rosenberg. Generating workflow models from owl-s service descriptions with a partial-order plan construction. In *Web Services (ICWS), 2011 IEEE International Conference on*, pages 714–715, July 2011. 62, 64

[Wie92] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992. 3, 7

[Win93] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, Cambridge, MA, USA, 1993. 122

[WLH04] Michael Winikoff, Wei Liu, and James Harland. Enhancing commitment machines. In *DALT*, pages 198–220, 2004. 47

[WPTR13] Barbara Weber, Jakob Pinggera, Victoria Torres, and Manfred Reichert. Change patterns in use: A critical evaluation. In *BMMDS/EMMSAD*, pages 261–276, 2013. 56

[WRR13] Barbara Weber, Stefanie Rinderle, and Manfred Reichert. Change patterns and change support features in process-aware information systems. In *Seminal Contributions to Information Systems Engineering*, pages 381–395. 2013. xiii, 7, 56, 57

[YMH+06] Jian Yu, Tan Phan Manh, Jun Han, Yan Jin, Yanbo Han, and Jianwu Wang. Pattern based property specification and verification for service composition. In *WISE*, pages 156–168, 2006. 82

[YS97] Daniel M. Yellin and Robert E. Strom. Protocol specifications and component adaptors. *ACM Trans. Program. Lang. Syst.*, 19(2):292–333, 1997. 7, 76, 86

[ZBDtH06] Johannes Maria Zaha, Alistair P. Barros, Marlon Dumas, and Arthur H. M. ter Hofstede. Let's dance: A language for service behavior modeling. In *OTM Conferences (1)*, pages 145–162, 2006. 6, 53, 73

[ZGS+13] Zhangbing Zhou, Walid Gaaloul, Lei Shu, Samir Tata, and Sami Bhiri. Assessing the replaceability of service protocols in mediated service interactions. *Future Gener. Comput. Syst.*, 29(1):287–299, January 2013. 50

[zM04] M. zur Muehlen. *Workflow-based Process Controlling: Foundation, Design, and Implementation of Workflow-driven Process Information Systems.*, volume 6 of *Advances in Information Systems and Management Science*. Logos, Berlin, 2004. 204

[ZY08] Xianrong Zheng and Yuhong Yan. An efficient syntactic web service composition algorithm based on the planning graph model. In *Web Services, 2008. ICWS '08. IEEE International Conference on*, pages 691–699, Sept 2008. 63, 64

Publications

International journals with reviewing committee

1. A cloud-based platform to ensure interoperability in aerospace industry. **M. Khalfallah**, NFI Figay, C Ferreira Da Silva, P Ghodous. Journal of Intelligent Manufacturing ():1-11, Springer US, ISSN 1572-8145. 2014.

2. Provenance-aware Monitoring System. **M. Khalfallah**, NFI Figay, M. Barhamgi, P Ghodous. To be submitted

International conferences with reviewing committee

3. Patterns for Monitoring Parallel Processes. **M. Khalfallah**, NFI Figay, M. Barhamgi, P Ghodous. In 11th IEEE International Conference on Services Computing (SCC 2014), Alaska. 2014.

4. Model Driven Conformance Testing for Standardized Services. **M. Khalfallah**, NFI Figay, M. Barhamgi, P Ghodous. In 11th IEEE International Conference on Services Computing (SCC 2014), Alaska. 2014.

5. An Architecture for a Centralized Mediation In Dynamic Networks. **M. Khalfallah**, M. Barhamgi, NFI Figay, P Ghodous. In 21th IEEE International Conference on Web Services ICWS 2014, Alaska. 2014.

6. Controlling the Evolution of Product-based Collaboration Contracts. **M. Khalfallah**, NFI Figay, M. Barhamgi, P Ghodous. In IEEE 10th International Conference on Services Computing (SCC), Santa Clara Marriott, California, USA (Center of Silicon Valley) . pp. 713-720. 2013.

7. Towards Combining Declarative Specification with On-the-fly Mediation. **M. Khalfallah**, M. Barhamgi, NFI Figay, C Ferreira Da Silva, P Ghodous. In IEEE 10th

International Conference on Services Computing (SCC), Santa Clara Marriott, California, USA . pp. 691-698. 2013.

8. Product-Based Business Processes Interoperability. **M. Khalfallah**, M. Barhamgi, NFI Figay, P Ghodous. In Symposium On Applied Computing, Coimbra, Portugal. 2013.

9. Cross-organizational Business Processes Modeling Using Design-by-Contract Approach. **M. Khalfallah**, NFI Figay, P Ghodous, C Ferreira Da Silva. In International IFIP Working Conference on Enterprise Interoperability , Enschede, The Netherlands. 2013.

10. A Novel Approach to Ensure Interoperability Based on a Cloud Infrastructure . **M. Khalfallah**, M. Barhamgi, NFI Figay, P Ghodous. In 19th ISPE International Conference on Concurrent Engineering CE2012, Springer ed. Trier (Germany). pp. 1143-1154. 2012.