Towards a formal framework for product level agreements

Malik Khalfallah* and Nicolas Figay
Airbus Group Innovations, Suresnes, France
Email: malik.khalfallah@airbus.com
Email: nicolas.figay@airbus.com
*Corresponding author

Mahmoud Barhamgi and Parisa Ghodous
Lyon 1 University, Lyon, France
Email: mahmoud.barhamgi@liris.cnrs.fr
Email: parisa.ghodous@liris.cnrs.fr

Abstract: In the European project IMAGINE we address the problem of cross-organisational processes interoperability in dynamic collaboration environments for product detailed design. By dynamicity, we mean that partners can quit the collaboration and be replaced by other partners at run-time. We first propose to capture the objective of the collaboration, that is reaching agreements between the involved partners on the configurations of the final product components. Successfully capturing agreements contributes to minimising the coupling between partners which reduces the impact of changes in the collaboration environment. Then, from the specification of agreements, we generate the cross-organisational process that automates message exchanges between partners. Nevertheless, the fragments of this cross-organisational process can face interoperability issues during run-time. This issue arises because each process fragment belonging to a partner should be generated while upholding business rules of this partner. Accordingly, we propose an automatic mediation approach that is well-adapted for dynamic collaboration environments. We present the implementation and the evaluation of our framework and an industrial case study is followed to illustrate how our framework can be used in practice.

Keywords: collaboration contracts; cross-organisational processes; COPs; mediation; temporal logic; automata.


Biographical notes: Malik Khalfallah made his research at Airbus Group Innovations and received his PhD in Computer Science from the Claude Bernard University Lyon 1 in 2014. He published several papers in IEEE SCC, IEEE ICWS, ACM SAC as well as numerous Elsevier and Springer journals. He is pursuing his research at Airbus Group.

Nicolas Figay is an expert in the area of interoperability for PLM. At EADS, he has been involved for years in research, standardisation and industrial projects dealing with Product data exchange and sharing, system engineering, PLM and interoperability.

Mahmoud Barhamgi is an Assistant Professor at the University of Claude Bernard Lyon 1. He received his PhD in Computer Science from the Claude Bernard University Lyon 1 in October 2010. His research interests include web services, and web data integration. He has published several papers on web services in international conferences and journals including the PVLDB, WWW, and ICWS, SCC, and IEEE Transactions on Services Computing.

Parisa Ghodous is currently a full Professor in Computer Science Department of University of Lyon 1. She is a member of LIRIS UMR 5205 (Laboratory of Computer Graphics, Images and Information Systems). Her research expertise is in the following areas: interoperability, web semantic, web services, collaborative modelling, product data exchange and modelling and standards. She is in editorial boards of CERA, ICAE and IJAM journals and in the committees of many relevant international associations such as concurrent engineering, ISPE, and interoperability.
1 Introduction

Forces of globalisation and the ever growing need for differentiation and innovation are moving work from a co-located to a distributed environment. Organisations form collaborations to achieve a goal that none could achieve individually (Oppenheim et al., 2010). This inter-organisational cooperation between enterprises is realised by specifying an abstract description of the overall inter-organisational process that is the choreography.

In the context of collaboration environments for product design, the engineering change management (ECM) process (SASIG, 2010) has been defined as the standard to follow when designing the different components of the final product. The ECM involves two standardised roles: coordinator and participant (SASIG, 2010) who collaborate by exchanging messages iteratively in order to achieve the right configuration of the final product components (PCs). The coordinator specifies the configuration of a particular component by setting constraints on this component attributes and the participant provides characteristics of this component after performing engineering studies. To model message exchanges between these two roles, choreography modelling languages can be used.

The choreography serves as a common contract between the parties involved in the overall process (van der Aalst et al., 2010). Examples of choreography languages are WSCDL (http://www.w3.org/TR/ws-cdl-10), Let us dance (Zaha et al., 2006), BPEL4Chor (Decker et al., 2007). Nevertheless, those specific modelling languages do not provide a sufficient level of abstraction to capture the intention behind the interactions (Telang and Singh, 2009). Moreover, contracts built with these modelling languages increase the coupling of stakeholders’ processes because they capture many details and thus, they over-specify contracts (van der Aalst and Pesic, 2006).

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 modelled 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.

The coupling between partners that results from these specifications makes the changes difficult to achieve especially in dynamic collaboration environments. This is the case of dynamic manufacturing networks (DMNs) where partners can be replaced at run-time. Handling this dynamicity requires less coupling in order to minimise the impact of changes on other partners in the same collaboration environment.

In this paper, we are interested in developing a modelling framework that facilitate the work of coordinators when defining and running contracts with participants. The proposed modelling approach underspecifies the collaboration contract by avoiding the inclusion of too much detail. Accordingly, our modelling approach decreases the coupling between partners and facilitates the dynamicity in the context of a DMN. Nevertheless, 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. Since partners will uphold their internal (business) rules when defining the processes (Charfi and Mezini, 2004), mismatches between the communicating processes will definitely appear and thus a mediation solution is mandatory to resolve those mismatches.

To develop our framework, in Section 2, we give the case study used throughout this paper. Section 3 details the scientific challenges addressed in this paper. Section 4 defines the main high-level concepts that are used by coordinators and participants to build collaboration contracts. Section 5 elaborates on the approach to generate the cross-organisational process (COP) from a contract specification to support coordinators and participants in fulfilling the contractual obligations. Since mismatches could appear in the COP, Section 6 elaborates on a novel mediation strategy to resolve the mismatches. Section 7 presents the implementation details and the evaluation of our framework. Finally, Section 9 concludes the paper.

2 Running example

Consider that in a collaboration environment for aircraft design, the coordinator wants to modify the properties of the geometry model of the fuselage (FusGeo) and the aerodynamic model of the fuselage (FusAero) as depicted in Figure 1. Since the FusGeo and the FusAero are managed by two different participants belonging to an external organisation, the coordinator needs to create a contract with this organisation involving both participants to perform the modifications on the fuselage (= FusGeo + FusAero). In this contract, the coordinator specifies the attributes of FusGeo and FusAero that he wants to modify and the characteristics that he wants to obtain after the modifications in order to analyse them.

Tables 1 and 2 summarise the fuselage attributes that the coordinator sends to FusGeo and FusAero participants respectively and the characteristics that he expects from them.
Towards a formal framework for product level agreements

Figure 1 Collaboration involving coordinator and participants on FusGeo and FusAero (see online version for colours)

To design the process supporting these change requests on the FusGeo and the FusAero, the coordinator can use business process modelling (BPM) languages (e.g., BPMN, http://www.bpmn.org/). 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. Furthermore, the coordinator defines another independent process model to support message exchanges regarding the FusAero as well.

Nevertheless, in engineering, it is known that there is a temporal constraint between the design of the FusGeo and the design of the FusAero, the coordinator needs to keep this constraint in his mind. In other words, prior to validating the modifications made on the FusAero, he knows that he should first validate the modifications on the FusGeo.

Table 1 Exchanged FusGeo properties between the coordinator and the participant

<table>
<thead>
<tr>
<th>Properties of the FusGeo to be changed</th>
<th>FusGeo characteristics desired by the coordinator after changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of paxes in the front (NPaxFront)</td>
<td>Wetted area</td>
</tr>
<tr>
<td>Fuselage height (HFus)</td>
<td>Fuselage mass</td>
</tr>
<tr>
<td>Number of aisles (Naisles)</td>
<td></td>
</tr>
<tr>
<td>Fuselage length (LFus)</td>
<td></td>
</tr>
</tbody>
</table>

Table 2 Exchanged FusAero properties between the coordinator and the participant

<table>
<thead>
<tr>
<th>Properties of the FusAero to be changed</th>
<th>FusAero characteristics desired by the coordinator after changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AiR density (Re)</td>
<td>Fuselage Fric drag</td>
</tr>
<tr>
<td>Air speed reference (Aref)</td>
<td></td>
</tr>
<tr>
<td>Fuselage length (LFus)</td>
<td></td>
</tr>
<tr>
<td>Fuselage width (WFus)</td>
<td></td>
</tr>
</tbody>
</table>

Table 3 Some mapping functions in the context of Fuselage design

<table>
<thead>
<tr>
<th>Mapping functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W_{fus} = f(NPaxFront, Naisle) )</td>
</tr>
<tr>
<td>( WettedArea = f(WFus, HFus, LFus) )</td>
</tr>
</tbody>
</table>

Finally, there is a set of existing mapping functions between the Fuselage attributes that are summarised in Table 3.

3 Challenges

When the coordinator models the processes to support message exchanges, he will face the following key challenges.

3.1 Manual creation of processes

Process modelling can be difficult and painstaking in particular for a coordinator who typically has no experience with process modelling 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 modelling the processes, the coordinator changes his focus from what he wants to achieve to how to achieve it).

3.2 Finding and applying the right rules

Yellin and Strom (1997) noticed that process activities are organised 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 organise the 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 (Knolmayer et al., 2000), their large number, however, could make this task error-prone.

3.3 Defining mediators

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

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 modelling language to model contracts between coordinators and participants where the coordinator does not need to use low-level concepts while giving him the possibility to formally express the objectives of the collaboration. In addition, it shows the need for a dynamicity-aware mediation mechanism that resolves the possible mismatches that could appear when coordinators’ and participants’ processes communicate.

4 High-level concepts for contract design

4.1 Overview of the proposed approach

We adopt the creation of a high-level modelling framework (van der Aalst and Pesic, 2006; Telang et al., 2014; Hildebrandt et al., 2011a). Specifically, as Figure 2 shows, the coordinator uses this framework to define a contract with participants that will receive and handle change requests sent by the coordinator 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 generates the COP that will support the partners in performing their collaborative work. This procedure generates for each partner the process that fits with his business rules stored in business rules repositories. It uses a set of operations (projection, reduction) in order to optimise the COP 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.

Contracts can be considered as a means to leverage the decoupling between stakeholders in a collaborative environment (Andrikopoulos et al., 2009). 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 also that all expected data will be sent.

![Figure 2](online version for colours)
In this section, we define the constructs to build contracts.

1. We formally define the product model that captures the end product breakdown structure and that is used by all partners in their work.

2. Partners collaborate in order to reach agreements on the configuration of PCs (Lee et al., 2012).

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

### 4.2 Product breakdown model

The product breakdown model is a tree-like structure with the end product as root and PCs as nodes (van der Aalst et al., 2011). Standards exist that represent the product breakdown model, for example, the product breakdown for support (http://www.plcs-resources.org). 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 van der Aalst (1999). A product breakdown model is defined as a tuple \( \langle \text{root}, \text{O}, \text{N} \rangle \) where:

- **root** represents the end product.
- **O** is a set of objects representing the components of the end product. Each element \( o_i \in \text{O} \cup \{\text{root}\} \) has a set of attributes \( \{\text{att}_1, \text{att}_2, \ldots, \text{att}_n\} \).
- **N** is 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.

### 4.3 Product component configuration

The coordinator defines a set of constraints on the properties of the end PCs. These constraints specify the expected configuration of each component. Given a PC \( o_i \), there are two types of constraints associated to it:

- **Obligations** are parameterised constraints that the participant should consider before designing the component.
- **Characteristics** are parameterised constraints that the coordinator 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 conjunction of the obligations and the characteristics regarding a component defines the **product component configuration**.

**Definition 4.1:** A product component configuration (PCC) of a component \( o_i \) is the tuple \((\text{Obligations}, \text{Characteristics})\). It is defined by the grammar:

- \( \text{PCC} \rightarrow (A, A) \)
- \( A \rightarrow A \land A \lor \neg A \)
- \( \text{constraint} \rightarrow \theta(o_i, \text{att}_j, v_j) \)

where \( \theta \) is an algebraic relationship (<, =, >, etc.) between the attribute \( o_i, \text{att}_j \) and the variable \( v_j \), and \( A \) is a first order logic predicate.

This grammar specifies that both obligations and characteristics are conjunctions of constraints on component \( o_i \) attributes.

### 4.4 Agreements

The objective of the collaboration between the coordinator and a participant is to reach an agreement on the configuration of a PC \( o_q \) [that is the reason why we call our framework product level agreement in analogy with service level agreement (Oppenheim et al., 2014)]. The coordinator and the participant will exchange data until reaching this agreement.

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

**Definition 4.2:** Formally, an agreement on a component \( o_q \) (\( q \) is an integer) is the tuple \((\text{Coordinator}, \text{Participant}_q, o_q, \text{PCC}_q)\). The agreement declaratively binds the coordinator and the participant through the PCC of the component \( o_q \).

Agreements under-specify the relationship between the coordinator and the participants. Such a specification defines the minimal coupling between partners 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 \( x_j \) specified in the PCC make the coordinator satisfied (i.e., \( V \) is a model of the logic formula PCC\(_q\)). Formally: \( \exists V = [v_{i_1}, v_{i_2}, \ldots, v_{i_N}] \), \( \bigwedge_{j=1}^{N} x_j = v_j \models \text{Obligations} \land \text{Characteristics} \).

### 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 participant quits the collaboration, he can be replaced by several participants. In this situation, each new participant will design a sub-component of the component designed by the participant who left. Using complex agreements, we can split a component agreement into agreements of its sub-components.

Definition 4.3: A complex agreement on a component \( o_k \) is a tuple \( \langle \text{Coordinator}, \{ \text{Participants}_i \}, o_k, C_k, \text{PCC}_k \rangle \), where:

- \( \text{Participants}_k \) is the set of suppliers of the subcomponents of the component \( o_k \).
- \( C_k \) is the set of components composing \( o_k \). It ensures: \( C_k \subseteq O \) and \( \forall o_i \in C_k, (o_i, o_k) \in N \).
- To express that this agreement is composed of other atomic or complex agreements: \( \forall o_i \in C_k, \exists \text{participants} \subseteq \text{Participants}_i \);
  \[ a \ \exists o_i \in O : (o_i, o_k) \in N \land |\text{participants}| > 1 \Rightarrow \langle \text{Coordinator}, \text{participants}, o_i, C_i, \text{PCC}_i \rangle \text{ is a complex agreement} \]
  \[ b \ \forall o_i \in O : (o_i, o_k) \notin N \land |\text{participants}| = 1 \Rightarrow \langle \text{Coordinator}, \text{participants}, o_i, \text{PCC} \rangle \text{ is an atomic agreement} \]
- \( \text{PCC}_k = \bigwedge_{i=1}^{\exists o_i} \text{PCC}_i \) when no relationship between sub-agreements is specified.
- \( \text{PCC}_k = \text{TR}(\text{PCC}_1, \text{PCC}_j) \) when a temporal relationship is specified between sub-agreements of sub-components \( o_i \) and \( o_j \).

For instance, we wanted to provide coordinators and participants with a visual language. We thus modelled agreements with boxes as depicted in Figure 3. Complex agreements are represented as containers of other agreements.

Figure 3 Graphical representation of atomic and complex agreements (see online version for colours)

4.6 Agreements relationships

Modelling a contract by a set of agreements for each PC 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 (Murman et al., 2000), it is recommended that the design of the wing should take place before the design of the engine. Similarly, 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:

1. The agreement on the nacelle between the coordinator and the nacelle participant can be achieved if the coordinator and the engine participant have already achieved an agreement on the engine configuration.
2. Additionally, the agreement on the engine can only be achieved if the coordinator and the wing participant have already achieved an agreement on the wing.

In order to specify declaratively these temporal constraints, we enrich our framework to support temporal logic formulas (Fisher, 2011). This is possible thanks to the logic-based formulation of product component constraints. Although the need for formal models, including temporal logic, is clearly shown by several studies (Türetken et al., 2011; Sadiq et al., 2007), the knowledge required for their use remains a significant obstacle for their adoption particularly in an industrial context (El Kharbili and Keil, 2010). Using patterns of temporal logic operators hides the complexity of the formalisms (Yu et al., 2006).

To express the relationship between agreements, we use a set of meaningful patterns of temporal logic relationships formalised in Dwyer et al. (1998). Table 4 summarises 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 modeller could face. Nevertheless, this set of patterns can be extended to capture other patterns of temporal constraints.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A LeadsTo B</td>
<td>B can be achieved only after A has been achieved</td>
</tr>
<tr>
<td>A Response B</td>
<td>Whenever A is achieved, B has to be achieved</td>
</tr>
<tr>
<td>A Include B</td>
<td>A cannot be achieved without achieving B</td>
</tr>
<tr>
<td>A Mutual Exclusion B</td>
<td>If A is achieved B cannot be achieved and vice versa</td>
</tr>
<tr>
<td>N A</td>
<td>A cannot be achieved after N iterations</td>
</tr>
</tbody>
</table>
4.7 Contract on the final product

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

Definition 4.4: The End product configuration is the PCC defined by the PCCs of all subcomponents of the root and connected TR of Table 4.

Definition 4.5: Formally, a contract is an agreement that is formally defined by the tuple \(\langle \text{Coordinator}, \text{Participants}, \text{root}, O, \text{End product configuration} \rangle\).

The contract is designed following a bottom-up approach. The coordinator 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.

4.8 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.

To realise these changes an informal communication between the involved partners is no longer effective (Oppenheim et al., 2014). 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 LeadsTo between FusGeo and FusAero.

Figure 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:

- \(Ag_1 = \langle \text{Coordinator1}, \text{Participant1}, \text{FusGeo}, \text{PCC}_1 = (R(NpaxFront, x_1) \land R(HFus, x_2) \land R(Nailes, x_3) \land R(Lfus, x_4) \land R(WettedArea, x_5) \land R(FusMass, x_6)) \rangle\)
- \(Ag_2 = \langle \text{Coordinator1}, \text{Participant2}, \text{FusAero}, \text{PCC}_2 = (R(NpaxFront, y_1) \land R(HFus, y_2) \land R(Nailes, y_3) \land R(Lfus, y_4) \land R(WettedArea, y_5) \land R(FusMass, y_6)) \rangle\)
- \(Ag_3 = \langle \text{Coordinator1}, \{\text{Participant1}, \text{Participant2}\}, \text{Fuselage}, \text{PCC}_1 \text{LeadsTo} \text{PCC}_2 \rangle\).

At this stage, the contract between coordinator1 and participant1, participant2 is built. The next step consists of generating the COP that will support the data exchange between these partners in order to achieve the agreements defined in this contract while upholding their temporal constraints.

Figure 4 Contract on ECRs on the geometry and the aerodynamic of the Fuselage (see online version for colours)
4.9 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 capture the objective of their collaboration regarding the design of a PCs.

In the next section, we detail how to automatically generate the underlying processes that will support coordinators and participants in achieving the objectives that have been specified in the contract.

5 Contract enactment

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 formalisation, the contract can be projected into the execution platform by generating automatically the COP model. Indeed, when analysing how the collaboration is conducted to handle a change request on a particular component, we can notice that the COP follows the same steps:

1. the coordinator sends obligation instances to the participant
2. the participant receives the instances
3. the participant carries out an internal process to realise the requested modification on the PC
4. the participant replies the coordinator with characteristic instances
5. the coordinator analyses the results, if he is satisfied then the agreement is reached, otherwise he assigns new instances to the obligation and repeats the process.

In the field of systems engineering, this process is called the agreement process (Haskins and Forsberg, 2011).

A partner could have concurrent engineering capabilities. These capabilities allow him to perform data exchange activities in parallel, while another partner 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 (Yellin and Strom, 1997). Indeed, each partner has his own rules to which his process should be compliant (Knuplesch et al., 2012). Generally, these rules are captured in terms of business rules. Business rules are statements that define or constrain some aspects of the business (Group, 2000). If business rules are defined to their full extent, they will govern the entire execution of the processes (Charfi and Mezini, 2004). In addition, business rules have the advantage to be very close to how designers think and talk (Eriksson and Penker, 1998).

Table 5 Rule types on the order of messages

<table>
<thead>
<tr>
<th>Constraint type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x₁ Should Follow x₂</td>
<td>Messages x₁ and x₂ should be sent/received in sequence, starting by x₂</td>
</tr>
<tr>
<td>SF(x₁, x₂)</td>
<td>Messages x₁ and x₂ should be sent/received in sequence, starting by x₁</td>
</tr>
<tr>
<td>x₁ Should Precede x₂</td>
<td>Messages x₁ and x₂ should be sent/received in parallel</td>
</tr>
<tr>
<td>SP(x₁, x₂)</td>
<td>Messages x₁ and x₂ could be in parallel</td>
</tr>
<tr>
<td>I(x₁, x₂)</td>
<td>Message x₁ should be delivered with x₂</td>
</tr>
<tr>
<td>P(x₁, x₂)</td>
<td>Messages x₁ and x₂ should be sent/received in parallel</td>
</tr>
</tbody>
</table>

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.

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 van der Aalst et al. (2003), we could determine what are the possible rules a partner can define regarding the organisation of data exchange activities in a collaborative environment. A subset of these constraints summarised in Table 5.

These constraints have an associated set of axioms given in Table 6, where x₁, x₂, x₃ being messages.

An informal example of a rule on the order of activities is that all send activities should be organised in sequence: “Since there is only one licence of the simulation software, we can deliver one instance of a message per time”.

Formally for the design of a given PC oₐ : ∀x₁, x₂ ∈ Att(oₐ); SF(x₁, x₂) ∨ SF(x₂, x₁). 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.

Table 6 Axioms related to rules on the order of messages

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Formalisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 (transitivity of SP)</td>
<td>SP(x₁, x₂) ∧ SP(x₂, x₃) → SP(x₁, x₃)</td>
</tr>
<tr>
<td>A2 (transitivity of SF)</td>
<td>SF(x₁, x₂) ∧ SF(x₂, x₃) → SF(x₁, x₃)</td>
</tr>
<tr>
<td>A3 (transitivity of I)</td>
<td>I(x₁, x₂) ∧ I(x₂, x₃) → I(x₁, x₃)</td>
</tr>
<tr>
<td>A4 (transitivity of P)</td>
<td>P(x₁, x₂) ∧ P(x₂, x₃) → P(x₁, x₃)</td>
</tr>
<tr>
<td>A5</td>
<td>(SP(x₁, x₂) ∨ SF(x₁, x₂) ∨ P(x₁, x₂)) ∧ I(x₁, x₂) → I(x₁, x₃)</td>
</tr>
<tr>
<td>A6</td>
<td>SP(x₁, x₂) ∧ P(x₂, x₃) → SP(x₁, x₃)</td>
</tr>
<tr>
<td>A7</td>
<td>SP(x₁, x₂) ∧ P(x₂, x₃) → SF(x₁, x₃)</td>
</tr>
</tbody>
</table>

Partners’ business rules reside in repositories {R₁, ..., Rₙ}. Although we could directly use the rules in these Rᵢ to generate the COP model as depicted in Figure 2, it is better to reduce each Rᵢ 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 analysing all rules to generate the COP model could be time consuming because the generation algorithm should evaluate each rule. To avoid repetitive and non-necessary evaluations, 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 $\Pi$ operator on business rules repositories.

   **Definition 5.1:** The projection operator on the repository $R_i$ gives the repository $R_i' : \Pi R_i \subseteq R_i$ such that the rules in $R_i'$ are defined by the predicates containing exclusively the attributes in $Att$.

   **Lemma 1:** The workflow model involving messages $\in Att$ whose send and receive activities are 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. $\square$

2. **Reduction** Eliminating rules that can be inferred from other rules in the repository using the axioms formalised in Table 6. The reason is that these inferable rules do not add any detail to the COP to be generated.

   **Algorithm 1:** Repository reduction reduceRepository

   **Require:** Repository $R$, the set of Axioms $AX$, a set of involved attributes $Att \subseteq 2^{\{a_1, \ldots, a_n\}}$

   **Ensure:** A reduced Business rules repository

   1: $\Pi_{red}(R)$
   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 $\exists \Sigma \subseteq R - \{r\} \land \exists ax \in AX$ 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

Algorithm 1 performs projection and the reduction actions. First, the algorithm projects the rule repository on the involved messages (line 1). Then, 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 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 organised 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 6.

For axiom $A_1$ in Table 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 5.

The proof continues by making the same observation on the remaining axioms in Table 6. $\square$

**Figure 5** Equivalence between the generated COP fragments

<table>
<thead>
<tr>
<th>Send x1</th>
<th>Send x2</th>
<th>Send x3</th>
</tr>
</thead>
</table>

**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. $\square$

**Definition 5.2:** 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 $R$ (Eriksson and Penker, 1998).

Formally: $\forall x_i \in COP, |\Pi_{full}(R_i)| \geq 0$.

**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: $\forall x_i \in COP, |\Pi_{full}(R_i)| \geq 0 \Rightarrow |\Pi_{full}(R'_i)| > 0$.

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

For the axiom $A_1$ in Table 6, we consider that $\{SP(x_1, x_2), SP(x_2, x_3), SP(x_1, x_3)\} \subseteq R \Rightarrow \{SP(x_1, x_2), SP(x_2, x_3)\} \subseteq R' \land \neg SP(x_1, x_3) \notin R'$. Accordingly, $|\Pi_{full}(R_i)| > 0 \Rightarrow |\Pi_{full}(R'_i)| > 0$.

The proof is the same for the remaining axioms in Table 6. $\square$

**Definition 5.3:** A process model is well formed if and only if 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
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 Ouyang et al. (2006).

The workflow generation algorithm (Algorithm 2) generates for each agreement specified in the contract the corresponding process fragment that supports data exchanges to achieve that agreement. 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 6, 7 and 8. The 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:

- 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 honoured (all agreement have been achieved) 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 honoured yet.

- $\leftarrow$ Suppose that the contract is not honoured 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.
Algorithm 2  Workflow generation \( \text{generateWorkflow} \)

\[ Require: \] The Contract \( C \)
\[ Require: \] Stakeholders business rules repositories \( R_1, …, R_n \)
\[ Require: \] Product Breakdown Model \( PM \)
\[ Ensure: \] The cross-organisational process (COP)

1: Call \( \text{reduceRepository} \) on \( R_1, …, R_n \)
2: Traverse the contract in a top-down way
3: for all Agreement \( A_g \in C \) do
4: Use mappings in Figures 6, 7, 8 to generate the interconnections between \( A_g \) and other agreements
5: if \( A_g \) is atomic then
6: For the obligations specified in \( A_g \), 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 \( A_g \) characteristics
9: end if
10: end for

5.3 Example

We continue with the example of Section 4.8. Once the contract has been specified in terms of agreements, the workflow generation module (see Figure 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 \( \text{LeadsTo} \) and workflow constructs depicted in Figure 6, the generation module will generate the workflow depicted in Figure 9 for the contract in Figure 4.

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, \( \text{participant1} \) expects to receive the \( WFus \) while \( \text{coordinator1} \) does not send this information. Hence, a mediation solution should be used to resolve this heterogeneity.

5.4 Summary

In this section, we presented the approach to generate the COP that will support \( \text{coordinators} \) and \( \text{participants} \) in achieving the design of PCs 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 reduce the number of business rules that the workflow generation algorithm should analyse. 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 generated processes some mismatches could appear due to the difference of process activities between partners. In the next section, we propose a novel mediation approach that efficiently resolves mismatches when the generated \( \text{coordinators’} \) and \( \text{participants’} \) processes will exchange data.

6 On-the-fly mediation

The generated COP will support the interaction between the \( \text{coordinator} \) and the \( \text{participant} \) working on a particular PC. 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 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 behavioural aspects. Thus, it considers that the names of the exchanged messages are standardised and the semantic compatibility is guaranteed. Bordeaux et al. (2004) have made the same assumption. This assumption is possible since much research has been done on adaptation where messages are enriched semantically through ontologies (Bordeaux et al., 2004).

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 (Benatallah et al., 2005). It is also possible to automate the discovery of the right pattern to apply when the mediator is to be generated automatically (Eslamichalandar et al., 2013).

In our approach, we aim at defining the mapping functions between exchanged messages (c.f. Table 3) in a way such that we can develop an automated mediation algorithm that can find the right mapping function to apply to derive the expected message from sent messages in an optimal time. This is challenging because existing automated approaches (Kongdenfha et al., 209, 2014) face state space explosions when performing this search process.

In the remainder of this section, we introduce marked automata to formalise the mapping functions so that we can find the right mapping function to apply to generate the expected message in linear time.

6.1 Automaton formal definition

Conventional mediation approaches define mapping functions at design-time either:
1 semi-automatically by involving a human in the loop
2 automatically by relying on the domain ontology and the mappings between its concepts (Benatallah et al., 2005).

During the run-time phase, automating conventional mediation algorithms so that they can find the right mapping
function to apply will lead to state space explosion. Indeed, consider that the workflow generator has generated a COP involving the coordinator and the participant as depicted in Figure 10. In this case, the conventional mediation algorithm (Kongdenfha et al., 2009, 2014) should perform, in the worst case, $\alpha$ iterations in order to determine what mapping function to use to generate the message $y_1$ from the sent messages $\{x_1, \ldots, x_n\}$ where:

$$\alpha = \sum_{k=1}^{N} \frac{N!}{k!(N-k)!}$$

and $N$ is the number of sent messages.

This worst case is reached when the data required to deliver the message $y_1$ is dispatched in all messages sent: $\{x_1, \ldots, x_n\}$, thus to determine the mapping function to apply 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 $(\{x_1, x_2, \ldots, x_n\}, y_1)$. In this last case a mapping function is found and $y_1$ can be generated.

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 $\alpha$ 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 $\{x_1, \ldots, x_l\}$ ($l < N$), 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 $\{x_{l+1}, \ldots, x_n\}$.

To implement the first step, we use an automaton structure (marked automaton) to represent the fact that there exists a mapping function between $\{x_1, \ldots, x_n\}$ and $y_1$ as depicted in Figure 11.

**Marked automaton**: $A^\Phi = [Q, C, \delta, q_0, \Phi]$ is a marked automaton if $Q$ 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. Consider the example of Figure 10 and the marked automaton corresponding to the mapping function between $y_1$ and \{x_{1,}, \ldots, x_n\} in Figure 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, \ldots, 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 $y_1$. When $x_2$ arrives, $s_2$ is marked but the mapping function remains not applicable. The advantage of our algorithm is that it will not check whether \( (x_1, x_2) \) can be used to derive $y_1$.
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,}, \ldots, 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 our 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 $\alpha$ [see equation (1)] to $N + 1$ of our mediation approach.

Basically, this algorithm starts by verifying that the received message is neither an Acknowledgements 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).

Algorithm 3  Mediation on the fly onTheFlyMediation

Require: PM: product model, RPP: receiver public process.
Ensure: the correct message required by the receiver.

1: Buffer ← 0
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(Buffer)
10:    send(MessageForReceiver)
11:   Buffer ← Buffer – RPP.ExpectedMessage
12: end if
13: end while

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 ← notifyAppropriateObserver(s_0.observers, lastInsertedEntry)
3:   s_i.processStateEventReception(lastInsertedEntry)
4: end if

Algorithms 4 and 5 detail the behaviour of the marked automaton when a message arrives. More specifically, the automaton is continuously 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 5 Automaton state event reception: processStateEventReception

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

6.2 Application example

We carry on with the example of Section 5.3 and the process of Figure 9. 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 3. Following our mediation approach, this mapping function will be formalised using a marked automaton as depicted in Figure 12 (a).

1 Following the process of Figure 9, when coordinator1 sends the message NpaxFront, this message will move the automaton of Figure 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 WFus it will have no impact on the automaton and thus the mediator remains idle for this message. Here is where our optimisation 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 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 four iterations on the mapping functions base to find the right mapping function to apply, instead of seven 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.

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 12(b). In this figure, the expected message WettedArea in the workflow of Figure 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 remainder of this section, we extend our mediation algorithm to address this issue for the general case.

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 6.2: 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 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.
**Figure 12** Example of composition of functions (see online version for colours)

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

```
Algorithm 6 determineInputs

Require: Automaton A
Ensure: Function f result

1: for all State s ∈ A do
2:    if s.associatedParameter ∈ then
3:        Add (s.associatedParameter, f.inputs)
4:    else
5:        Find automaton B: B.output = s.associatedParameter ∧ B.s0.marked
6:        Add (determineInputs(B), f.inputs)
7:    end if
8: end for
9: Call f
10: return result
```

**Proof:** (By induction)

First, suppose that the expected message $y$ could be computed using the function $f(x_1, x_2, ..., x_m)$ (if 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 $y$.

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

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

### 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 6.3:** A message is identified by a triplet: $(Name, Coordinator, Participant)$.

**Definition 6.4:** An extended marked automaton is a marked automaton where:

- A transition is identified by the triplet: $(Name, Coordinator, Participant)$. 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: $(Coordinator, Participant)$. 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: \(\langle \text{Name}, \text{Coordinator}, \text{Participant} \rangle\) and is triggered whenever all equally identified states have been marked.

Each time a new couple of partners is added to the network, all mapping functions 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 11. In the following, we prove that with the extended marked automaton definition, none of these three 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 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. □

### 6.5 General properties of the mediation approach

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

**Processes similarity and complementarity:**

1. Two processes are said to be **equivalent** if they execute exactly the same strings of actions (Fokkink, 2000).
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 13 illustrates each definition through an example.

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

![Processes complementarity illustration](see online version for colours)

| Process 1: Send (a) Send (b) Send (c) | Equivalent processes |
| Process 2: Send (a) Send (b) Send (c) |
| Process 1: Send (a) Send (b) Send (c) |
| Process 2: Send (b) Send (c) Send (a) |
| Process 1: Send (a) Send (b) Send (c) |
| Process 2: Send (W) |
| Process 1: Send (a) Send (b) Send (c) |
| Process 2: Receive (W) |

Semi-Similar processes:

\(W\) can be derived from \(a, b, c\) and vice versa

Complementary processes:

\(W\) can be derived from \(a, b, c\) and vice versa
7 Implementation and evaluation

7.1 Implementation

We have implemented our framework by developing a collaborative modelling environment to build contracts using the portal Liferay (http://www.liferay.com/) as depicted in Figure 4. To run the generated processes from contract models (Figure 9) we have used the workflow engine Shark (sourceforge.net/projects/sharkwf/).

Shark proposes a WfMC (http://www.wfmc.org) compliant management interface that can be used to deploy and manage processes. To develop our mediation algorithm, we have used this interface to capture the exchanged messages (line 3 of Algorithm 3) and to submit the expected message to its receiver (line 9).

To develop our two-steps approach, we have realised a home-built implementation of marked automata. We used the observer design pattern to implement the notification mechanism between the mediator and the all automata corresponding to mapping functions (line 3 of Algorithm 5).

We provide a video that shows how our mediation mechanism works for the running example. An Eclipse project excerpt of the source code of our application is also available (http://www.eads-iw.net/web/thesis-malik/demonstrator).

7.2 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.

7.2.1 Evaluation of the abstraction level of the proposed language

We applied the goal-question-metric technique (Berander and Jönsson, 2006) to define appropriate metrics to compare the abstraction of our contract specification framework and the runtime 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 choose two size metrics to compare programs specified by both languages. The size metrics we utilise 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 (Li and Henry, 1993). 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 7.1: 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, we have two important concepts for the coordinator that are the product components properties (PCP), and the PCs.

In the XPDL model, each PCP could appear two times, once in a send activity and once in a receive activity. This is the case for the property Fuselage_Height in Figure 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. 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 specialised, much of non-necessary information is hidden and is replaced by valuable information for the coordinator.

Table 7 details the computation of the added value of concepts in the model and their application to our example.

7.2.2 Evaluation of on-the-fly mediation

The main advantage of using the on-the-fly mediation approach is the possibility to change mapping functions at run-time with minimum impact on the collaboration. The impact of changing the mapping functions for both conventional mediation and on-the-fly mediation depends on three variables:

1. the number of couples of software applications that are being used during the collaboration \(|D|\)
2. the number of mediation applications deployed in the collaboration platform \(m\)
3. the number of partners participating to the collaboration.

The detailed formulas to compute the impacts on both conventional mediators (impact \(c\) \(M\)) and on-the-fly mediation (impact \(o-t-f\) \(M\)) are given in Section 4.7 of Khalfallah (2014).

Figures 14, 15 and 16 summarise the impacts of varying these variables on each mediation strategy.

From these figures, we can observe that the on-the-fly mediation approach is not impacted by both variables the number of partners \(N\) and the number of mapping functions. This is possible thanks to the existence of a single mediation approach that is not impacted by adding/removing partners from the DMN. In addition, thanks to the possibility to add/remove mediation rules without disturbing the collaboration.

The impact of adding/removing couples of applications represented by the variable \(|D|\) is less important on the on-the-fly mediation strategy than on the conventional mediation.

8 Related work

8.1 High-level languages for business modelling

Imperative models such as BPMN, Petri nets, UML sequence and activity diagrams are only good in describing the operational way to fulfil the constraints, leaving the constraints implicit. Accordingly, alternative approaches studied by different research groups is the use of declarative process models (Bussler and Jablonski, 1994; Davulcu et al., 1998; Senkul et al., 2002; Singh et al., 1995; Pesic and van der Aalst, 2006).

Table 7 Comparison of model semantic richness for XPDL versus our framework

<table>
<thead>
<tr>
<th>Concepts</th>
<th>Calculation method</th>
<th>XPDL model Figure 9</th>
<th>Contract model Figure 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product components properties (PCP): PCP</td>
<td>(\sum_{i=1}^{N} \frac{1}{</td>
<td>PCP</td>
<td>})</td>
</tr>
<tr>
<td>Product component: PC</td>
<td>(\sum_{i=1}^{N} \frac{1}{</td>
<td>PC</td>
<td>})</td>
</tr>
</tbody>
</table>

Figure 14 Varying \(N\) (see online version for colours)
Van der Aalst et al. (van der Aalst and Pesic, 2006; Pesic and van der Aalst, 2006; Pesic et al., 2007) pointed out the limits of imperative languages including their lack to support changes. They have proposed a declarative framework to model processes. They used the temporal logic patterns defined by Dwyer et al. (1998) to define constraints between the execution of activities in a process. Despite the advantages that they bring with this framework, this framework has a major drawback. Indeed, since the process specification will end up being a temporal logic formula, the valuation of this formula into an executable process domain (BPMN for example) will induce ambiguity. Indeed when relying on Dwyer et al. (1998) patterns, authors specify that LeadsTo(A, B) (where A, B are activities) and interpret it as B can start only when A has finished. However, Knolmayer et al. (2000) claim that an action (or activity) is characterised by the fact of being time consuming. Thus, the interpretation made by the authors is only one among two possible interpretations. 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 to be future of the instant when A has started and thus in all these instants B can start while respecting the specification.

In our framework, the contract specification is temporal logic formula where operands are constraints and thus they are not time consuming as activities. We formally defined the mapping between the declarative specifications and the execution language that is XPDL. Thus, our framework will not face this ambiguity.

Hildebrandt et al. (2011b) have formalised Dwyer et al. (1998) patterns not in temporal logic but using a special class of graphs that is dynamic condition response graphs. They introduce a new approach to model cross-organisation processes for case-management systems. They still relies on a declarative approach to avoid using imperative languages. Basically, they model the COP by:
In their prototype architecture, they developed a new workflow engine to run the model, while in our proposal we rely on existing workflow engines that are WfMC standard compliant.

8.2 Mediators

Mediation (or adaptation) can be derived into two types: interface level mediation that addresses transformation issues related to types of messages (Bordeaux et al., 2004). Such a static compatibility is essential to check, process level mediation is more challenging however. Mismatches between processes (Benatallah et al., 2005; Zhou et al., 2013; Dumas et al., 2006) 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.

Several works have been carried out in order to resolve the main process mismatch patterns that can occur when two processes interact. A survey is given here (Munusamy et al., 2011).

Benatallah et al. (2005) developed a set of patterns that capture the possible mismatches that can occur between two communicating business processes. They assume that analysts will analyse the two processes that will communicate and then identify the mismatches and the corresponding patterns to resolve them. Since an analyst is always required, this approach is not adapted for DMNs.

Taher et al. (2009, 2011) developed a language based on labelled transition systems to formalise the mismatch patterns. The patterns are the same as those that have been defined by Benatallah et al. (2005).

Dumas et al. (2006) 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, automatically discovering the applicability of the GATHER operator [that corresponds to the many-to-one pattern of Kongdenhfa et al. (2009)] is challenging because it could lead to the state space explosion as explained in Section 6.

Bracciali et al. (2005) 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 behaviour 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 common issue of these approaches is that they focus on identifying transformation patterns of the exchanged messages between two processes. Nevertheless, identifying mismatches patterns is necessary but not sufficient for dynamic collaboration environments where mediation patterns to apply to resolve mismatches should be found automatically. Efforts have been carried out to achieve this automatic discovery but, as we have seen in Section 6, they have scalability issues.

9 Conclusions

In this paper, we addressed the problem of business processes interoperability in dynamic collaboration environments for products design. We tackled this problem during the design-time phase and the run-time phase. In the design-time phase, we provided involved partners a modelling language that allows them to formally specify the objectives of their collaboration by defining the collaboration contract. This modelling language uses high-level concepts that facilitate the modelling task while decreasing the coupling between partners which is necessary in dynamic collaboration environments. From the contract specification, we efficiently generated the COP that supports partners in achieving the contract. At run-time, mismatches could appear between the fragments composing the COP. Accordingly, we developed a novel dynamicity-aware mediation approach to resolve the possible mismatches. We compared our mediation approach with the naive one provided in the literature and we proved that it is more efficient. We evaluated the abstraction of our contract modelling language and the evaluation results are encouraging. We provided an illustration of our working prototype through a real-world running example. In the future, our aim is to test our prototype using an industrial scenario from aeronautic industry. Additionally, we aim at enriching our framework with management operations in order to efficiently handle changes that could occur at run-time in the collaboration environment.

References


Notes

1 DMN is a concept that emerged in the European project IMAGINE, http://www.imagine-futurefactory.eu/.