STANDARDS COMPLIANT PLATFORM FOR PRODUCT DESIGN IN DYNAMIC ENVIRONMENTS
Malik Khalfallah, Nicolas Figay, Mahmoud Barhamgi, Parisa Ghodous
AIRBUS Group Innovations, Lyon 1 University
{malik.khalfallah,nicolas.figay}@airbus.com
{mahmoud.barhamgi,parisa.ghodous}@liris.cnrs.fr

Abstract
The literature on product-based workflow design considers only static collaborative environments where the product model is static and does not change. However the possible evolution of the product model and thus its impact on the collaborative environment are not considered despite the importance of this evolution for innovative product design. As a result the uncontrolled changes can easily break the cross-organizational process that links the different stakeholders in the collaborative environment. In this paper we present a framework that builds the collaboration contract from the product model and supports a controlled evolution of this contract. We leverage this framework by a set of management operations (services) that enable the contract evolution and investigate how a tailoring operation will not impact the executable cross-organizational process unless this process is maintained executable by the workflow engine. These operations are provided as services by a collaborative platform. Nevertheless, software applications that are provided through services following the Software as a service (SaaS) paradigm generally need to be compliant to some standards. In order to implement our management operations, we need to select the software application that provides the most conformant service interface to a given standard. Accordingly, we develop a formal framework that tests the compliance and reports quantitative results that help experts take the right decision. Unlike existing work that focuses on a single dimension when checking the compliance of a software application with the corresponding standard (for example the functional dimension exclusively, or the syntactic dimension exclusively), in this work we consider multiple dimensions at the same time. This provides more comprehensive results.

Keywords: Dynamic Manufacturing Networks, Concurrency, Product Design Contracts, Standards, Conformance, Vector Calculus

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, figure out the best responses, and successfully make the necessary changes (Oppenheim, Bagheri, Ratakonda, & Chee, 2010). In product design collaborative environments, stakeholders are bound by a collaboration contract that determines their obligations w.r.t the product being designed. Nevertheless, we have seen recently the emergence of Dynamic Manufacturing Networks (DMN) (IMAGINE, 2012) that put forth the evolution aspect in collaborative environments. Accordingly, to handle this evolution, it is necessary to have a flexible collaboration contract in order to respond to possible evolutions that could occur during run-time.

1.1. RUNNING EXAMPLE OF DMN
Consider the scenario of designing a new aircraft by an OEM. The aircraft architect starts by defining the first draft of the aircraft breakdown model that is closer to meet the customer requirements. Since this OEM is operating in the context of an extended enterprise, parts of this aircraft model will be designed by external subcontractors and then assembled in the shared aircraft digital model. Due to the complexity of the product being designed, there is a high probability that the first draft of the aircraft breakdown model will evolve during the collaboration until reaching an optimal model that meets all customer requirements. During this evolution, aircraft designers may need to add/remove components, change their configuration, and call for new subcontractors or replace existing ones. All these evolutions need to be manageable in order to allow aircraft designers to focus on technical issues.

1.2. EXISTING APPROACHES FOR DMN MANAGEMENT
To manage evolutions in a DMN we need to build a model of the collaboration contract that clearly defines stakeholders’ obligations. Existing cross-organizational process models (BPMN, WS-CDL) or product-based workflow design approaches (Wil M P van der Aalst, Reijers, & Limam, 2001) could be used for this

http://hipore.com/ijsc
purpose. They could be used to build a cross-organizational process model that describes the exchanged data between stakeholders and define what each stakeholder should deliver and what each stakeholder should expect during the collaboration. Nevertheless the resulting models will fail in optimally supporting evolutions. Such languages are imperative in nature and are not flexible enough to be tailored in a transparent way (W M P van der Aalst & Pesic, 2006). Although the flexibility issue of cross-organizational processes modeling languages has been addressed by several declarative frameworks such as Decserflow (W M P van der Aalst & Pesic, 2006) or Let's dance (Zaha & Barros, 2006), these languages still use low level abstractions (Singh, 2011). Low level abstractions are not appropriate for modeling DMNs collaboration contracts because engineers involved in the management of the DMN should execute several fine grained operations in order to achieve a meaningful business change.

To overcome the raised issues, we developed in a previous research (M Khalfallah, Figay, Barhamgi, & Ghodous, 2013) a declarative framework that uses high-level abstractions to model the collaboration contract in the context of a DMN. In this framework stakeholders use the first draft of the shared product model as a basis to define their collaboration contract. Once each stakeholder’s obligations regarding the shared product model are determined, an executable cross-organizational process is automatically derived that will support the collaboration between stakeholders.

1.3. THE CONTRIBUTION

In this paper we enhance this framework by defining high-level operations to manage the evolution of the DMN collaboration contract. Specifically we make the following contributions:

- An approach to automatically decide whether to commit modifications in the collaboration contract into the running cross-organizational process or not.
- A formal definition of high-level management operations that improve the management of DMN collaboration contract.
- A conformance testing approach to test whether the underlying implementation of these operations upholds standards’ specifications.

In summary: in Section 2 we provide an overview of our declarative framework that defines flexible collaboration contracts. In Sections 3 and 4 we elaborate on DMN collaboration contract management operations and how their execution does not tailor the running cross-organizational process unless it is maintained executable. In Section 5 we elaborate on the approach to select the most standardized software module. In Section 6 we review the related work. Finally in Section 7 we conclude the paper and discuss the future work.

2. A META-MODEL FOR COLLABORATION CONTRACTS IN DMNs

To make the paper self-contained, we briefly present the meta-model to build collaboration contracts, however a more detailed presentation can be found in (M Khalfallah et al., 2013).

2.1. META-MODEL OBJECTIVES

To overcome the imperative nature of existing cross-organizational process modeling languages, and the use of low-level concepts in existing declarative languages to model business processes, we developed a declarative framework that uses the product model as a baseline to define the collaboration contract. Basically in this contract, each stakeholder in the collaborative environment will be assigned obligations that it should fulfill w.r.t. the product components. Then the contract model is used to generate an executable cross-organizational process that supports the collaboration among stakeholders.

2.2. CONTRACT BUILDING

Figure 1 provides an overview of the concepts used to build a collaboration contract. Basically, a contract is composed of a set of Agreements, each Agreement concerns a specific Component of the product. It defines for this Component the parameterized Constraints that the component Supplier should consider when designing the digital model of the component (i.e. Obligations) and the parameterized Constraints that the component Requester should consider when assessing the delivered digital model of the component (i.e. Benefits). A Product Model is composed of different Views (Physical, System, and Functional) and each View is composed of a set of Components that have Attributes.

Cross-organizational Process Generation.

For a specific Agreement instance in the contract, when all its associated classes are specified (see Figure 1), it is possible to use it to generate a fragment of the cross-organizational process. Each fragment is a couple of processes \(<P_{req}, P_{Supp}>\) that will exchange messages to achieve Obligations and Benefits for the Supplier and the Requester respectively. The transition from a fragment corresponding to an Agreement to another fragment is achieved by Event Condition Action (ECA) rules.

3. CHECKING HETEROGENEITY OF GENERATED PROCESSES

Using the presented meta-model, stakeholders can build a collaboration contract by relying on the shared product model.

http://hipore.com/ijsc

45
During the collaboration, stakeholders' DMN managers might need to change the contract by adding, removing, binding or unbinding its building blocks. Nevertheless some of these changes can generate a non-executable cross-organizational process and thus involved stakeholders can no longer collaborate.

Example.
Suppose that the OEM DMN manager decided to add a new component to the shared aircraft model with a set of constraints regarding its design. However he has not found the subcontractor that will design it yet. Accordingly the Agreement corresponding to this component in the collaboration contract will be incomplete because the supplier field is missing. If this contract evolution is committed into the running cross-organizational process, then the OEM public process will send messages that have no corresponding recipient. This leads to a structural mismatch and makes the whole cross-organizational process not executable.

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

Definition 1.
A cross-organizational process is executable if it can reach its final state.

Proposition 1.
In addition to the syntactic correctness condition, a cross-organizational process composed of a set of couples

< PR, PS > is executable if and only if for each couple: PR and PS are interoperable.

Proof.
• ⇒ From Definition 1, since an executable process terminates, all expected messages will arrive. Thus both the sender and the receiver are interoperable.
• ⇐ For each fragment < PR, PS > 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 1 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_srt indicates if a structural mismatch exists between two communicating process. A structural mismatch occurs when there is a difference in the exchanged messages structure.
• 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).
• 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 1 find 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 1 we need a formal representation of stakeholders' public processes.

---

4 DMN Manager is a role that has the rights to tailor the collaboration contract. Each stakeholder has an instance of this role.
These processes can be specified using LTS (Labeled Transition Systems). Basically an LTS is a quadruplet \(< S, Act, \rightarrow, I >\) where:

(i) \( S \) is a set of states;
(ii) \( Act \) is a set of actions;
(iii) \( \rightarrow \subseteq S \times Act \times S \) 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 \(< S, Act, \rightarrow, I, M, O, L, ontology, language >\) 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 1. This algorithm analyzes both Requester and Supplier processes. It returns the values of \( \alpha_{str} \) (line 43), \( \alpha_{syn} \) (line 15) and \( \alpha_{sem} \) (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.

4. 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 1. In this section we formally describe operations that tailor the DMN contract model as the needs change. 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 2: (i) Primitive operations that manage the building blocks of the contract model. They are implemented as services upon this mode. (ii) Basic operations that have the same granularity as primitive operations. They can be invoked by the DMN manager through a visual interface. (iii) Composite operations are successive calls to primitive operations. The objective of composite operations is to have more complex management operations.

We provide DMN managers a visual interface that displays basic and complex operations to manage evolutions in the DMN. The use of visual interfaces makes easier the perception of knowledge (Haber, Ioannidis, & Livny, 1994) and thus the management task will be less complex.

4.1. PRIMITIVE OPERATIONS

Upon the contract model there is a set of CRUD (Create Read Update Delete) operations that manage its building blocks as detailed in Table 1. When (programmatically) invoking these operations it is necessary to consider their pre/post conditions. A primitive operation such as RemoveView\((v)\) can be described using operational semantic notation as follows (the same applies for remaining operations. Notice that we use the same notation as (Winskel, 1993)):

\[
\begin{align*}
(exists(v), \sigma) & \rightarrow true \\
(RemoveView(v), \sigma) & \rightarrow \sigma'
\end{align*}
\]

This statement indicates that to remove the view \( v \) from the current state \( \sigma \) of the contract, this view 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.**

Consider that the OEM DMN manager wants to remove the functional view \( v_1 \) in the shared aircraft model. In the meantime, suppose that the simulation performer subcontractor arrived to the same conclusion and decided to remove the view \( v_1 \) too. When both stakeholders call the RemoveView\((v_1)\) 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 RemoveView\((v_1)\) that requires the presence of \( v_1 \) in the shared aircraft model.

4.2. 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. Basic operations are duplications of primitive operations that the DMN manager invokes through the visual interface as depicted in Figure 2. When a DMN manager invokes 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 < tailoring_element >.
A software component 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 service that implements the primitive operation. Although the event manager eliminates the concurrent calls failures, it does not resolve all issues regarding the concurrent calls of a primitive operation:

**Example.**

Suppose that DMN managers of the OEM and the simulation performer subcontractor simultaneously call the operation RemoveView($v_1$). These calls will generate two events containing the same payload. When analyzing the incoming events, the event manager will call the RemoveView operation. After this call the view $v_1$ will not exist anymore. Accordingly, the second event payload will not match with the RemoveView premises but, this time; it matches with AddView premises as given in statement 1:

$$(\exists \text{exists}(v), \sigma) \rightarrow false$$

$$(\text{AddView}(v), \sigma) \rightarrow \sigma'$$ (2)

In this situation, the event manager will call the AddView operation, which was not the aim of any DMN manager!

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

Since DMN managers call the DMN 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 becomes the couple $<$ tailoring_element, toggled_button $>$. The new operational semantics of the basic operation RemoveView($v$) is given in statement 3:

$$(\exists \text{exists}(v), \sigma) \rightarrow \text{true}$$

$$(\text{ifRemoveViewToggled} \rightarrow \text{RemoveView}(v) f i, \sigma) \rightarrow \sigma'$$ (3)

Based on statement 3, the RemoveView($v_1$) primitive operation is called if and only if its premise is evaluated to true and the Remove View button in the visual interface has been pressed.

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

**Example.**

Indeed, consider the RemoveView($v_1$) scenario again. The OEM DMN manager presses the Remove View button in the visual interface which will generate the event to remove the view $v_1$. In the meantime a Simulation subcontractor DMN manager presses the same button in his interface which will generate the same event. Although the second event payload matches the premises of the AddView operation (since the view does not exist anymore) this
operation will not be invoked because the AddViewToggled condition is not satisfied as illustrated in statement 4.  

\[
\langle \text{exists}(v), \sigma \rangle \rightarrow \text{false} \\
\langle \text{if AddViewToggled} \rightarrow \text{AddView}(v), \sigma \rangle \rightarrow \sigma'
\]  

TABLE 1. MANAGEMENT OPERATIONS

<table>
<thead>
<tr>
<th>Operation</th>
<th>Precondition/Postcondition</th>
<th>Postcondition/Precondition</th>
<th>*Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddView(v)</td>
<td>( v \in PM.V )</td>
<td>( v \in PM.V )</td>
<td>RemoveView(v)</td>
</tr>
<tr>
<td>AddComponent(cp,v)</td>
<td>( v \in PM.V \land cp \notin v )</td>
<td>( v \in PM.V \land cp \in v )</td>
<td>RemoveComponent(cp,v)</td>
</tr>
<tr>
<td>AddPartner(p)</td>
<td>( p \in P )</td>
<td>( p \in P )</td>
<td>RemovePartner(p)</td>
</tr>
<tr>
<td>AddConstraint(cn,Ag)</td>
<td>( \text{cn} \not\subseteq \text{subformula} \text{Ag}.\text{PCC} )</td>
<td>( \text{cn} \text{subformula} \text{Ag}.\text{PCC} )</td>
<td>RemoveConstraint(cn,Ag)</td>
</tr>
<tr>
<td>AddRel(Ag1,Ag2,r)</td>
<td>( r ) \text{does not link} \text{Ag1} \text{and} \text{Ag2} )</td>
<td>( r \text{links} \text{Ag1} \text{and} \text{Ag2} )</td>
<td>RemoveRel(Ag1,Ag2,r)</td>
</tr>
<tr>
<td>AddException(NonAg)</td>
<td>( \text{NonAg} \in \text{C} )</td>
<td>( \text{NonAg} \in \text{C} )</td>
<td>RemoveException(NonAg)</td>
</tr>
<tr>
<td>BindRequester(p,Ag)</td>
<td>( p \in PM.P \land Ag \in \text{C} \land Ag.\text{Requester} = \emptyset )</td>
<td>( Ag.\text{Requester} = p )</td>
<td>UnbindRequester(p,Ag)</td>
</tr>
<tr>
<td>BindSupplier(p,Ag)</td>
<td>( p \in PM.P \land Ag \in \text{C} \land Ag.\text{Supplier} = \emptyset )</td>
<td>( Ag.\text{Supplier} = p )</td>
<td>UnbindSupplier(p,Ag)</td>
</tr>
</tbody>
</table>

4.3. COMPOSITE OPERATIONS

Basic operations aim to provide stakeholders with a satisfactory flexibility to manage the DMN. Sometimes DMN managers have redundant needs for complex management operations. For example: adding, in a single block, a new view with all product components and a set of new stakeholders that design these components. Instead of calling each basic operation separately and keeping track of the current stage which can be error prone, it is better to provide the DMN manager a single high level operation i.e. a composite operation. This composite operation is constituted of successive calls to primitive operations. The question that rises: when building composite operations should we directly make invocations to primitive operations (without using the event manager) or invoking basic operations?

Building composite operations by making direct invocations to primitive operations will lead to the same issue discussed in sub section 4.2 when two composite operations are called concurrently. Hence we should use the second alternative.

Composing basic operations to generate more high level operations needs to ensure the exclusiveness condition for the invoked basic operations. 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 DMN managers 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 DMN manager 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.

4.4. COMMITTING CHANGES

In the previous subsections we defined three types of operations to manage the DMN. Here, we determine which calls to basic operations shall be followed by a commit through a set of lemmas. These lemmas can be used later on to automate the commit operation. Due to space limitation, we only give two examples of lemmas.

**Lemma 1.**

Suppose that the cross-organizational process is executable and suppose that for the agreement \( Ag_1, Ag_1.\text{Requester} \neq \emptyset \) and \( Ag_1, Ag_1.\text{Supplier} \neq \emptyset \), \( Ag_1, Ag_1.\text{object} \neq \emptyset \). In this case a call to \( \text{AddConstraint(cn,Ag)} \) will commit the changes.

**Proof.**

Referring to algorithm 1 we prove that the \( Ag_1, Ag_1.\text{Requester} \) process and \( Ag_1, Ag_1.\text{Supplier} \) process are interoperable. (i) The condition of syntactic interoperability is satisfied since adding a constraint to an existing agreement will generate new exchanged messages but represented with already used languages. (ii) The condition of semantic interoperability is satisfied since the generated messages concepts belong to existing ontologies (no new mapping is needed). (iii) The condition of structural interoperability is satisfied since the process generation algorithm by default generates structurally interoperable processes (Malik Khalfallah, Figay, Barhamgi, & Ghodous, 2013).

Thus Algorithm 1 returns the value false for \( \alpha_{str}, \alpha_{syn} \) and \( \alpha_{sem} \). Accordingly the generated processes are interoperable; therefore the cross-organizational process remains executable.

**Lemma 2.**

Suppose that for an agreement \( Ag_1, Ag_1.\text{Supplier} = \emptyset \). Committing \( \text{BindRequester}(q_1,Ag_1) \) does not maintain the interoperability of the cross-organizational process.

**Proof.**
Referring to algorithm 1, we prove that if a commit is performed the process that results will not be executable. Indeed, if a commit is realized, the Requester \( q_3 \) will send messages while there will be no receiver. Thus the structural interoperability is not ensured \( s_{str} = true \). Consequently, the cross-organizational process fragments are not interoperable and not executable (proposition 1).

Theorem 1.

Let \( \alpha p_0 \) be a basic operation and \( \sigma_0 \) the current state of the DMN contract. If \( < \alpha p_0, \sigma_0 > \rightarrow \sigma_1 \) then \( \exists \alpha p_1, \alpha p_2... \alpha p_n \) such that \( < \alpha p_1, \sigma_1 > \rightarrow \sigma_2, < \alpha p_2, \sigma_2 > \rightarrow \sigma_3... < \alpha p_n, \sigma_n > \rightarrow \sigma_{n+1} \land \sigma_{n+1} \) is a state of the DMN contract model that generates an executable cross-organizational process.

\[\text{Proof.}\]

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.

4.5. MANAGEMENT INTERFACE OF THE DMN COLLABORATIVE PLATFORM

At this stage we have defined the management operations of our contact framework in DMNs. The implementation of these operations requires interaction with the management interface of the workflow engine. An example of this interaction consists in stopping, modifying and resuming the execution of a business process.

On one hand, a standard of workflow engines interface has already been developed that is the Workflow Management Coalition (WFMC).

On the other hand, to realize our contract framework and its associated management operations, the underlying software applications cannot be selected arbitrarily. Indeed, the ASD SSG (Aerospace and Defense Strategic Standardization Group\(^5\)) 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 collaborative platform including the workflow engine.

5 http://www.asd-ssg.org/

50

50


http://hipore.com/ijsc

http://www.asd-ssg.org/

5 http://www.asd-ssg.org/

4.5. MANAGEMENT INTERFACE OF THE DMN COLLABORATIVE PLATFORM

At this stage we have defined the management operations of our contact framework in DMNs. The implementation of these operations requires interaction with the management interface of the workflow engine. An example of this interaction consists in stopping, modifying and resuming the execution of a business process.

On one hand, a standard of workflow engines interface has already been developed that is the Workflow Management Coalition (WFMC).

On the other hand, to realize our contract framework and its associated management operations, the underlying software applications cannot be selected arbitrarily. Indeed, the ASD SSG (Aerospace and Defense Strategic Standardization Group\(^5\)) 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 collaborative platform including the workflow engine.

5. SELECTING THE MOST CONFORMANT SOFTWARE TO A STANDARD

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 (Lewis, Morris, Simanta, & Wrage, 2008). 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 domain of business processes. The proposed approaches attempt to check whether a business process upholds the standard specifications and possibly warn the business expert of the exceptions (O Turetken, Elgammal, & Heuvel, 2011). 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 (Farchi, Hartman, & Pinter, 2002). Other approaches perform syntactic coverage in order to test whether an implementation upholds the defined syntax in the software implementation (Nakamura, Satoh, & Chung, 2007).

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 paper, we develop a formal framework that establishes quantitative measures used to calculate the compliance of a software application interface (services) with the specifications in the standard documents. The set of measures corresponds to the possible dimensions that could exist but remains extensible to include other dimensions. We use vector spaces to capture the different independent dimensions and then we use vector calculus to calculate the total deviation between software implementation and a standard specification. We have implemented a software environment that is used to perform this verification and to report the results.

5.1. 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 \( p_o \) associated respectively to two operations \( o_t \) and \( o_o \), we compute their semantic difference as follows:

\[
\begin{align*}
\text{class}(p_t) < \text{class}(p_o) & \Rightarrow \Delta_{\text{sem}}\text{-input}(p_t,p_o) = 1 + (\text{NbrProp}(p_t) - \text{NbrProp}(p_o)) \\
\text{class}(p_t) > \text{class}(p_o) & \Rightarrow \Delta_{\text{sem}}\text{-input}(p_t,p_o) = -1 + (\text{NbrProp}(p_t) - \text{NbrProp}(p_o))
\end{align*}
\]

\(^5\) http://www.asd-ssg.org/
5.2. 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 \((s_1, s_2)\): 
\[
\Delta_{\text{syn}}(s_1, s_2) = 1 - \frac{|s_1 \cap s_2|}{|s_1|}
\]

Since the syntactic differences of the inputs, outputs and operation names are independent, then they constitute orthogonal vectors as depicted in Figure 4. The total value of the syntactic difference between two operations is given by:

\[
\Delta_{\text{syn}} = \sqrt{\Delta_{\text{syn-input}}^2 + \Delta_{\text{syn-output}}^2 + \Delta_{\text{syn-name}}^2}
\]

FIGURE 3 CALCULATING THE SEMANTIC DELTA

5.3. 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:

- \(\Delta_{\text{fun}} < 0 \Rightarrow \Delta_{\text{sem-output}} < 0\): if an operation has less functionality in comparison to another operation, then it provides less information after its execution.
- \(\Delta_{\text{fun}} < 0 \Rightarrow \Delta_{\text{sem-output}} > 0\): if an operation has more functionality in comparison to another operation, then it provides more information after its execution.
- \(\Delta_{\text{fun}} < 0 \Rightarrow \Delta_{\text{sem-output}} = 0\): if two operations have equivalent functionality then they will provide the same information.

Although the \(\Delta_{\text{sem-output}}\) is involved in calculating the \(\Delta_{\text{fun}}\), it is not the unique parameter. Indeed, when looking up a standard specification and a software implementation, we can find three types of operations:

- **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;
- **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;
- **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.

http://hipore.com/ijsc
engine by providing his credentials prior to obtaining the right to obtain the process instance.

This classification of operations implies that we should include a new parameter called deviation. It is induced by cross-aspect operations when calculating the difference between the software implementation of a core operation and its counterpart in the standard specification. Figure 5 summarizes the possible situations. In this table two predicates are used to characterize each situation:

1. implement($o_c, cao$): 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.

2. specify($o_c, cao$): 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 6. The value of the functional difference is given by the equation:

$$\Delta_{\text{fun}}(st, sf) = \begin{cases} 
\sqrt{\Delta_{\text{sem-output}}^2 + \text{deviation}^2} & \text{for } \Delta_{\text{sem-output}} > 0 \\
-\sqrt{\Delta_{\text{sem-output}}^2 + \text{deviation}^2} & \text{for } \Delta_{\text{sem-output}} < 0 
\end{cases}$$

FIGURE 5 POSSIBLE DEVIATIONS

3. 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.

In Figure 7 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 this coverage when calculating the $\Delta_{\text{total}}$ for all possible cases. There are totally 15 cases summarized in Figure 8:

FIGURE 7 EXAMPLE

In Figure 8 the $\Delta_{\text{total}}$ is equal to $\Delta_{\text{fun}}$ 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 $\Delta_{total}$ because the two operations belong to the same category. In these cases:

$$\Delta_{total} = f_0 = \sqrt{\Delta^2_{fun} + \Delta^2_{sem\text{-}input} + \Delta^2_{syn}}.$$ 

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 Figure 8) 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.

5.4. 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\(^6\) 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:

- Archimate is a modeling language that offers a set of concepts dedicated to capturing standard specifications as well as software modules and their operations;
- 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;
- 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 $\Delta_{sem}$. OWL-S library provides us with the capability to check the relationship between two classes ($\text{hasSuperClass, hasSubClass, hasEquivalentClass}$) as well as the capability to calculate the number of properties of each class ($\text{listProperties}$).

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. In addition, this role restricts the software expert from 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. In addition, 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 9 illustrates the interfaces displayed to the standard expert and to the software expert after they have logged in and received their corresponding roles.

5.5. 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 with the standard\(^7\) 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 10.

From Figure 10, 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: $\text{WOpenProcessDefinitionList}$ and $\text{WMFetchProcessDefinition}$, while TWS has implemented $\text{listProcessDefinition}$ and $\text{getProcessDefinition}$ (following the Java naming convention\(^8\)). 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 $s_0$ of Figure 8).

The reason is that 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 $\text{RCO}$

\(^6\) www.opengroup.org/subjectareas/enterprise/archimate

\(^7\) https://sourceforge.net/projects/sharkwf/files/shark/6.0-1/tws-6.0-1.doc.pdf pp.14

\(^8\) http://www.oracle.com/technetwork/java/codeconv-138413.html
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 Figure 8.

The overall difference between a software application and the standard that it claims it implements is the sum of differences regarding all operations: Total Difference = \( \sum_{i=0}^{n} |\Delta_i| \). This overall measure provides us a formal basis to select the closest implementation to the standard.

5.6. 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 (zur Muehlen, 2004). In order to counter this trend, the first goal of the WfMC was to establish a common terminology and glossary and naming convention\(^9\).

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\(^10\) 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.

6. RELATED WORK

This paper deals with two research challenges. The first challenge concerns the management of changes in collaboration environments. The second challenge concerns implementing the framework for managing changes. More specifically, it concerns checking the conformance of the software modules implementing the framework with their respective standards.

6.1. MANAGING DYNAMIC COLLABORATION ENVIRONMENTS

The problem of evolution of collaboration environments has been addressed for Service oriented architectures. The question was: when a service provider upgrades its service, how to shield service consumers from this evolution? Indeed, introducing the dynamic aspect in collaboration environments will necessarily introduce between stakeholders’ public processes. New mismatches would appear that have not been forecasted before.


Nevertheless, these solutions fail to resolve the problem in the context of dynamic collaboration environments. The reason is that each time a new heterogeneity appears after performing a change, it is necessary to generate a new mediator and deploy it into the collaboration platform. This is a penalizing approach for stakeholders that are not concerned by the changes that occurred. It is necessary to stop and restart the collaboration platform in order to deploy the new mediator. In our previous research (M Khalfallah et al., 2013) we addressed this problem by developing an on-the-fly mediation approach. This approach has the advantage to handle the heterogeneities even in dynamic environments without the need to restart the collaboration platform.

6.2. CONFORMANCE TESTING APPROACHES

Farachi et al. (Farchi et al., 2002) developed a formal approach to test the conformance of software against a standard specification. They considered the preconditions and postconditions of the data model impacted by executing the software functions. They used finite state machines in order to model the preconditions and postconditions. Although their approach has formal basis and can be automated, the authors clearly state that the skills required for the use of their technique are not usually found in existing organizations.

Schonenberger et al. (Schöenberger, Schwabl, & Wirtz, 2011) developed a manual approach to assess the coverage of WS-Sec-Policy specifications by Oracle’s Metro WS-stack and Apache’s Axis2 WS-stack. In their approach they send SOAP messages to the software application that claims being conformant with the WS-Sec-Policy standard, then they used a network analysis tool Wireshark\(^11\) to capture the exchanged messages and analyze them. Although our framework relies on a human in the loop as well, nevertheless the conformance analysis is performed automatically and not manually.

Van der aalst et al. (Wil M P van der Aalst, Dumas, Ouyang, Rozinat, & Verbeek, 2008) used petri nets to check the conformance of the behavior of a service regarding its specification. Again, they considered only one dimension in their evaluation.

Hu et al. (Hu, Martin, Hwang, & Xie, 2007) developed an approach to check that policies specified with the standard XACML are conformant with RBAC policy specification. The authors described informally how this checking should proceed without giving quantitative metrics that would provide well established basis to assess how much a developed policy is conformant to RBAC policy rules.

Nakamura et al. (Nakamura et al., 2007) developed a framework to syntactically validate security specifications


\(^10\) http://www.workflowpatterns.com

\(^11\) www.wireshark.org
with the standard WS-SecurityPolicy. They claim that validating the conformance of the syntactic dimension only is realistic and sufficient in many situations. Later on, they worked on a framework to check the conformance of the semantic dimension of security policies. However, they do not address the problem of combining both dimensions.

There is an extensive work on conformance checking of business processes with standards and regulations. Authors in (Elgammal, Turetken, & Van Den Heuvel, 2012) (Oktay Turetken, Elgammal, & Heuvel, 2012) defined a compliance conceptual model for the definition of compliance concepts and used a set of temporal logic patterns to facilitate the specification of compliance rules.

Knuplesch et al. (Knuplesch, Reichert, Ly, Kumar, & Rinderle-Ma, 2013) developed a visual compliance language to specify compliance rules of business with regard to regulations and standards while considering not only the activities and their sequence but also the compliance of resources, time and data. The language can be used at design-time when creating the process from scratch but not to verify the conformance of existing process with standards.
7. Conclusion

In this paper, we developed a model-driven approach to control evolutions of collaboration contracts at the Business layer by defining a set of basic operations. Then we formally proved that each change at the Business layer will not be committed to the running process unless it maintains this process executable. Ultimately, our framework simplifies the control of collaboration contract evolution and provides a solution through which an ecosystem targeted to design a (new) product can successfully evolve overtime in a very manageable way.

Nonetheless, to implement the platform that realizes this framework, the underlying software modules need to be compliant to their community standard if it exists. This is a very strong policy inside AIRBUS Group to be vendor independent regarding its software. Accordingly, we developed a complementary approach that allows us to select a software module for our platform that is the most conformant. This approach returns quantitative measures that support a confident choice.

We presented an application example of this approach and the associated software environment used to perform the conformance testing and reporting.

Our framework can be extended to evaluate the compliance of a set of interacting software applications regarding their respective standards. Then it checks whether these interactions do not lead to coherency issues due to emergent behavior.

8. References


http://hipore.com/ijsc


Authors

Malik Khalfallah is a PhD candidate at AIRBUS Group Innovations and Lyon University. His research interests include processes interoperability, data provenance, cloud computing and service oriented architecture. He has published several papers in IEEE SCC, IEEE ICWS, ACM SAC.

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

Mahmoud Barhamgi is an Associate Professor at Lyon 1 University. His research interests include Data Integration, Query Processing, Information and Data Management and Data Privacy. He has published several papers in VLDB, EDBT, IEEE SCC, IEEE TSC, and IEEE TDKE.

Parisa Ghodous is full professor in computer science department of University of Lyon I. She is member of LIRIS UMR 5205. Her research expertise is in the following areas: Interoperability, Web semantic, Web services, Collaborative modeling, Product data exchange and modeling 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, Interoperability.

http://hipore.com/ijsc