Flexibility of Collaborative Processes using Versions and Adaptation Patterns

Imen Ben Said, Mohamed Amine Chaâbane, Rafiq Bouaziz, Éric Andonoff

Flexibility of Collaborative Processes using Versions and Adaptation Patterns

I. BEN SAID, MA. CHAABANE, R. BOUAZIZ
MIRACL / Université de Sfax
Route de l'aéroport, BP 1088, 3018 Sfax, Tunisia
{Imen.Bensaid, MA.Chaabane, Raf.Bouaziz}@fsegs.rnu.tn

E. ANDONOFF
Laboratoire IRIT / Université Toulouse 1
2 rue doyen Gabriel Marty
31042 Toulouse Cedex, France
andonoff@univ-tlse1.fr

***Abstract*— Process aware Information Systems (PaIS) have an ever-increasing importance in Enterprise Information Systems for supporting both their intra and inter-organizational processes. However PaIS still have important issues to address, including flexibility of inter-organizational (*i.e.*, collaborative) processes before their definitive acceptance and their use in companies. This paper addresses this issue advocating (i) the modelling of collaborative processes as BPMN collaboration, (ii) the use of the version notion to deal with collaborative process flexibility and to make process instance migration easier, and (iii) the introduction of 6 new adaptation patterns which are high-level operations for collaborative process schema update.**

Keywords*— *Collaborative Processes; Flexibility; BPMN; Version; Adaptation Pattern

I. INTRODUCTION

Process flexibility is an important issue that Process aware Information Systems (PaIS) have to address before their definitive acceptance and use in companies [1]. Indeed the strong competition in which companies are involved often lead them to adapt their processes to face new operational, organizational, or customer requirements, new regulation laws, or to benefit from new collaboration opportunities. This challenge is all the more important as the economic environment in which companies operate is more and more dynamic, open and competitive, and requires flexibility capabilities [2]. Thus the economic success of companies is closely related to their ability to implement changes happening in their environment and to adapt their processes accordingly.

Flexibility has been deeply investigated, mainly in the context of intra-organizational (or internal) processes. Several taxonomies to characterize process flexibility have been proposed in literature. The more suitable one which will serve as a support for related work analysis is given in [3]. This taxonomy differentiates between two times for process flexibility: flexibility at *design-time*, which refers to foreseeable changes which can be taken into account in modelled process schemas, and flexibility at *run time*, which refers to unforeseeable changes occurring during process execution. In addition, this taxonomy identifies four needs of flexibility [3]:

- *Variability*, for representing a process differently, depending on the context. Each process schema is represented as a variant: variants share the same core process whereas the activity execution differs from variant to variant [4–5]. Note that parameters causing process variability are mostly known a priori.
- *Adaptation*, for handling occasional situations or exceptions which have not been necessarily foreseen in the process schema [6–7].
- *Evolution*, for handling changes in processes, which require occasional or permanent modifications in their schemas [8–12].
- *Looseness*, for handling knowledge intensive processes whose schemas are not known a priori and which correspond to non-repeatable, unpredictable, and emergent processes. Such processes require loose specifications [13–14].

However, process flexibility is still an open issue in the context of inter-organizational processes, *i.e.* Collaborative Processes (CPs). In CPs, flexibility may be related to the availability of involved processes or to the collaboration schema. Research efforts about CP flexibility mainly address process availability in the context of dynamic inter-organizational processes. Dynamic inter-organizational processes refer to processes where the different partners involved are not necessarily known at design-time, or can evolve at run-time (*e.g.*, they become unavailable or their quality of service decreases significantly, etc.) [15]. The provided solutions support finding new partners offering requested services, along with negotiation, contracting and service execution (*e.g.*, [14]). On the other hand, flexibility of schema collaboration has rather been neglected and the following research question has to be addressed: how to change collaborations according to process partner schema changes and which operations can be performed for collaboration schema changes?

This paper deals with this research question, adopting the following approach: (i) BPMN, which is known as the standard notation for process modelling, will serve for collaborative process modelling within collaboration diagrams, (ii) the version notion, which has been recognized

as a powerful mechanism for process flexibility in intra-organizational context [10–14], will be used to model versions of collaborations, and (iii) specific adaptation patterns will be introduced to define high-level primitives for collaborative process schema creation and update.

More precisely, the paper contributions are: (i) BPMN4VC (BPMN for Versioning Collaborations), an extension of BPMN meta-model for modelling versions of collaborative processes within BPMN collaboration diagrams, (ii) a set of basic operations for collaborative process version management, along with a state chart defining when these operations are available, and (iii) a set of 6 adaptation patterns, which are high-level primitives making collaborative process creation and update easier to perform. The intertwining of both versioning operations and adaptation patterns is also an interesting contribution to address the flexibility of collaborative processes.

The remainder of the paper is organized as follows. Section 2 gives an overview of related works, considering process flexibility both in an intra-organizational and in an inter-organizational context. It also positions this paper with our previous contributions. Section 3 presents BPMV4VC, the provided BPMN extension for modelling and handling versions of collaborations. This section also introduces the provided operations for collaborative process version management, along with the state chart defining when these operations are available. Section 4 presents the provided collaborative process adaptation patterns; it also illustrates how to intertwine the performing of both versioning operations and collaborative process adaptation patterns. Finally Section 5 concludes the paper, summarizing our contributions and giving some directions for future work.

II. Related Works

Process flexibility has been highly investigated in the context of intra-organizational processes since the end of the 1990s. In addition to the typologies proposed to feature process flexibility and to evaluate the ability of PaIS and process models to support process flexibility *(e.g.*, [3]), different contributions have been made to deal with this issue, and they follow several approaches: activity-driven approach [4–13, 16], constraint-driven approach [17–18], data-driven approach [19–20], case-driven approach (case handling) [21], and more recently, social-driven approach [22]. In this paper, we rather focus on activity-driven process flexibility as activity-oriented models are used in the majority of (service-oriented) PaIS.

Regarding flexibility of activity-driven intra-organizational processes, we can mention works supporting the modelling of process variants [4–5, 16, 23–24]. These contributions address process flexibility and more precisely *process variability* (according to the typology of [3]). A variant is an adjustment at run-time of a base process schema according to the context. We differentiate between behavioural approaches, which define the base process schema as a superset of variants and derive a specific process variant by hiding and blocking process components of the base process schema, and structural approaches which derive a process variant by applying a set of changes to a base process schema [25]. Specific notations and systems such as C-EPC [4], Provop [5] and vBPMN [23] support process variants.

We can also mention works supporting the modelling of process versions for capturing process changes over time. In the version-based approach, the significant changes on process schema results in the definition of new process versions. Several works, including ADEPT [26], advocate this approach to deal with process flexibility [8–12, 26]. This is probably the most comprehensive version-based contribution. These works are interesting since the notion of version has been recognized as a key notion to deal with process flexibility and more precisely, with *process variability*, *process evolution* and *process adaptation* (when adaptation can be defined a priori, at design-time) [27]. However, these works have two main drawbacks. First, they mainly focus on the behavioural dimension of processes, leaving aside their organizational and informational dimensions. However, these dimensions have also to be considered when dealing with process flexibility, since flexibility may be related to the resources involved during process execution or to the information being managed during process execution. Secondly, each of these contributions introduce specific notations, which are not standards and are unlikely to be used by process designers who are in charge of modelling variability of processes. Therefore, we have introduced BPMN4V (BPMN for Versions), an extension of BPMN to support intra-organizational process version modelling, considering both behavioural (what, how), organizational (who) and informational (when) dimensions of processes [28]. In [29], we have extended BPMN4V to model the contextual dimension of processes in order to capture the situations in which processes are executed and thus to define why version of processes are used instead another according to the context.

Further addressing activity-driven intra-organizational process flexibility, we can mention works in relation to change patterns [23, 30]. [30] is a fundamental contribution which introduces a set of 14 *adaptation patterns* to structurally change process schemas and a set of 4 *change support features patterns* to allow process actors to add information regarding unspecified parts of process schemas at run-time. These patterns correspond to high-level primitives making process schema update easier. Figure 1 gives an overview of the proposed adaptation patterns of [30]. For instance, adaptation patterns AP1 and AP2 allow for the insertion (AP1) and deletion (AP2) of process fragments in a given process schema. In the same way, moving and replacing fragments is supported by adaptation patterns AP3 (Move Process Fragment), AP4 (Replace Process Fragment) and AP5 (Swap Process Fragment).

AP1: Insert Process Fragment	**AP8**: Embed Process Fragment in Loop
AP2: Delete Process Fragment	**AP9:** Parallelize Activities
AP3: Move Process Fragment	**AP10:** Embed Process Fragment in Conditional Branch
AP4: Replace Process Fragment	**AP11**: Add Control Dependency
AP5: Swap Process Fragments	**AP12**: Remove Control Dependency
AP6: Extract Sub Process	**AP13**: Update Condition
AP7: Inline Sub Process	**AP14**: Copy Process Fragment

Fig. 1. Adaptation Patterns for Intra-Organisational Processes [30]

In addition to [30], we can also mention [23], which has defined variability patterns for variants. More precisely, [23] proposed to extend BPMN to address variability of processes using process variants, and defined a set of patterns to easily derive a specific process variant from a base process schema.

However, these contributions do not address inter-organizational process flexibility, which is still an open issue. More precisely, in the context of inter-organizational processes, *i.e.* Collaborative Processes (CPs), flexibility may be related to the availability of involved processes or to the change of collaboration schemas [31]. Research in CP flexibility mainly addresses process availability in the context of dynamic inter-organizational processes. Dynamic inter-organizational processes refer to CPs where the different partners involved are not necessarily known at design-time, or can evolve at run-time (*e.g.*, they become unavailable or their quality of service decreases significantly, etc.) [14–15, 32–33]. The provided solutions support finding new partners offering requested services [14, 32–33], along with negotiation [34], contracting and service execution, in separate or comprehensive frameworks [35].

Collaboration schema change has been rather neglected even if we found some contributions in the SOA context [36–40]. Contributions [36] and [37] mainly consider chained execution and subcontracting collaborative processes, and they provide high level patterns for service adaptation (adding, removing, substituting services). These contributions address CPs evolution but they do not address CPs variability and adaption. Contribution [38] recommends an extension of the WS-BPEL language to deal with exceptions in collaborative processes. However, the proposed solution is specific since it depends on a particular execution language and it only deals with CPs adaptation. Finally, contributions [39] and [40] focus on the propagation of private process changes towards processes of the other partners involved in a collaborative process. More precisely, they provide a set of algorithms to deal with changes of process schema by adding, deleting, replacing or updating process fragments, and they do not consider changes that can affect messages (*i.e.*, information) exchanged between collaborative process partners. Note also that change propagation has been investigated in the context of choreographies [41].

In this work, we address the collaborative process (*i.e.*, collaboration) schema change issue introducing both versions of collaboration and high level adaptation patterns. The reasons justifying such an approach are the following.

On the one hand, versions are known to be a powerful mechanism to address process flexibility. First, handling versions of processes facilitate the migration of instances from an initial schema to a final one, allowing, if the migration is not possible, two different instances of the same process to run according to two different schemas [8–10]. Secondly, the notion version has been recognized as a key notion to deal with process flexibility and more precisely, with *process variability*, *process evolution* and *process adaptation* (when adaptation can be defined a priori, at design-time) [27]. Therefore the collaborative process flexibility should benefit from these advantages. On the other hand, providing adaptation patterns, which are high level primitives to make collaboration schema update easier, is an interesting contribution to address the flexibility of collaborative processes. They should be the support for a comparison of PaIS with respect to collaborative process flexibility support.

Note that this work extends our previous contributions on BPMN and process versioning ([28–29]). It addresses collaborative processes schema flexibility issue using versions and introducing adaptation patterns to make BPMN 2.0 collaboration schema changes easier to perform.

III. BPMN4VC for Modelling and Handling Versions of Collaborative Processes

A collaborative process defines a collaboration between companies (each one represented by a process) in order to carry out a common business target. BPMN2.0 introduces collaboration diagrams for modelling such process schemas, crossing companies' boundaries [42]. These diagrams include an explicit representation of permanent interactions between the partners involved. These interactions are defined as message flows, *i.e.* messages exchanged between partners.

To address the flexibility of collaborative process schemas, we recommend introducing the notion of version in collaboration diagrams. Consequently, this section first introduces the notion of version and then presents BPMN4VC, an extension of BPMN for modelling and handling versions of BPMN2.0 collaboration diagrams, considering both static and dynamic aspects of collaborative process versions.

A. Notion of Version

As illustrated in Fig. 2 below, a version corresponds to one of the significant states (*i.e.*, values) an entity may have during its life cycle [43]. In our context, the considered entity is a collaborative process schema.

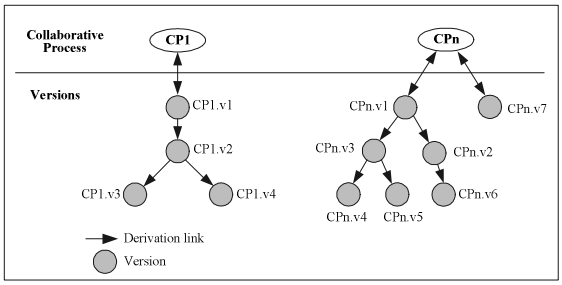

Fig. 2. Versions to Model Collaborative Process Flexibility

So, it is possible to describe the changes of a collaborative process schema through its different versions. These versions are linked by a derivation link; they form a derivation hierarchy. When created, a collaborative process is described by only one version. The definition of every new version is done by derivation from a previous one (except the first level versions, which are created directly from the process –*e.g.*, CPn.v1, CPn.v7): such versions are called derived versions and they capture the evolution of the corresponding process. Of course, several versions may be derived from the same previous one: these are called alternatives; they capture the variability of the corresponding process and they correspond

to their various variants. The derivation hierarchy looks like a tree if only one version is created directly from a collaborative process (*e.g.*, CP1), and it looks like a forest if several versions are created directly from a collaborative process (*e.g.*, CPn).

Thus, using the notion of version, it becomes possible to model collaborative process flexibility and more precisely, collaborative process schema variability (through the notion of alternative or variant), collaborative process schema adaptation which can be modelled a priori in the schema, and collaborative process schema evolution [27].

B. Statics Aspects of BPMN4VC: the BPMN4VC Meta-Model

We extended BPMN2.0 to model versions of collaborative processes providing extensions to BPMN2.0 collaboration meta-model and as well as defining operations for collaborative process version management. This section (III.B) focuses on the extensions provided to BPMN2.0 meta-model while the next one (III.C) deals with defined operations.

More precisely, BPMN4VC meta-model for collaboration results from the merging of BPMN2.0 meta-model for collaboration and a versioning kit introduced to make classes of BPMN2.0 meta-model versionable, *i.e.* able to handle versions. We briefly present these two layers.

1) Versioning kit. The versioning kit we propose is very simple: it includes only two classes and two relationships as illustrated in Fig. 3.

A versionable class is a class for which we would like to handle versions: so, versions are stored in a *Version_of_Versionable* class while corresponding entities are stored in a *Versionable* class. The *is_version_of* relationship links a versionable class to its corresponding versions. The *derived_from* relationship allows for building version derivation hierarchies (cf. Fig. 2). This latter relationship is reflexive and the definition of both relationship sides is the following: (i) a version (SV) succeeds another one in the derivation hierarchy, and (ii) a version (PV) precedes another one in the derivation hierarchy. Regarding properties of a *Version_of_Versionable* class, we introduce the classical version properties, *i.e.* version number, creator name, creation date and status [43].

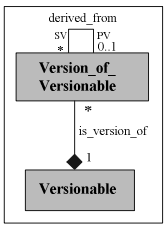

Fig. 3. Versioning Kit

2) Main Concepts of BPMN 2.0 Collaboration. In BPMN2.0, a *collaboration* depicts interactions between two or more processes belonging to business entities [42]. Each one is seen as a *participant* that represents a *partner entity* (*e.g.*, a company) or a *partner role* (*e.g.*, a buyer, a seller, or a manufacturer). A participant is often responsible for the execution of a p*rocess*. In a collaboration, only public processes are provided within *tasks*, *events* and the way these tasks and events are synchronized using *sequence flow* and *gateways*. Within a collaboration, participants are prepared to send and receive *messages* within *message flows*. A *message flow* illustrates the flow of messages between two interaction nodes. An *interaction node* is used to provide a single element as the source (send relationship) or the target (receive relationship) of a message flow, and therefore of a message. An interaction node can be a *participant*, a *task* or an *event*. Fig. 4 below gives the main BPMN2.0 concepts for modelling collaborations.

Note that within a collaboration, tasks and events are considered as the "touch point" between participants. Only those tasks or events that are used to communicate with the other participants are included. They define the public part of the process. As a consequence, all other internal (*i.e.*, private) tasks or events of the process are not shown in a collaboration diagram [42].

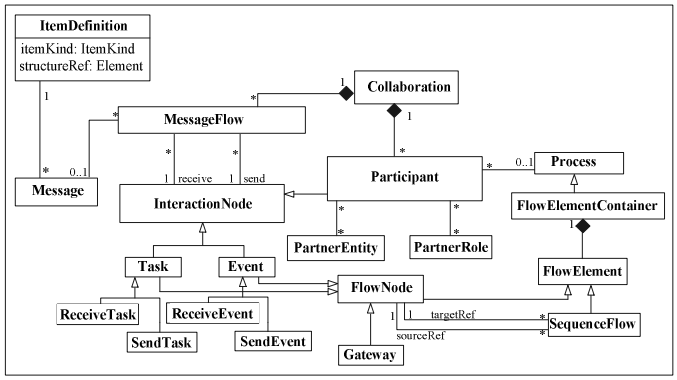

Fig. 4. Extract of the BPMN2.0 Collaboration Meta-model

3) Extending BPMN for Versioning Collaborations. We use the versioning kit introduced before to make some classes of BPMN2.0 meta-model versionable. Fig. 5 below presents the BPMN4VC resulting meta-model.

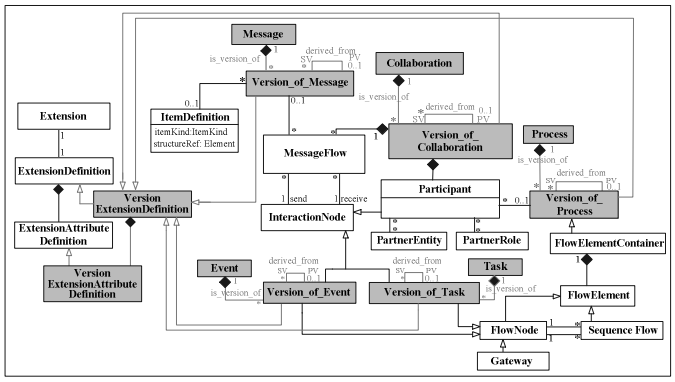

Fig. 5. BPMN4VC Meta-model for Modelling Versions of Collaborations

On the one hand, to take collaborative process flexibility into account, we recommend handling versions for the following five BPMN 2.0 classes: *Collaboration, Message*, *Process*, *Task* and *Event*. In fact, each of these classes represents key concepts for collaborations and plays a strong role in the definition of a collaborative process. The idea is to keep track of changes occurring to components which play a

part in the description of how the collaboration is carried out.

Generally speaking, a new version of an element (*e.g.*, collaboration) is defined according to changes occurring to it: these changes may correspond to the addition of information (property or relationship) or to the modification or the deletion of existing ones. More precisely, regarding messages, we consider that a modification of their property *ItemDefinition* results in the creation of a new version of message. For instance, if *Report* is a message referring to a paper document (*Itemkind* value is *physical*), and as a result of technical changes, if it becomes an electronic document (*Itemkind* value is *information*) then a new version of *Report* has to be created. However we do not necessarily create a new version of message if there is change in the interaction in which the message is involved. Indeed an interaction (*i.e.*, a message flow) being defined as the triplet (message, send node, receive node), where send and receive nodes are interaction nodes involved in the message exchange that either correspond to versions of task or versions of event, changing the interaction does not necessarily leads to create a new message. For instance, if a message M is sent from task A to task B, and if a new task C is defined after an organizational change and the message is no longer sent from A to B but rather from C to B, then we do not create a new version of the message M if it carries out the same information. Thus we manage M.v1 as a message exchanged between A and B, and M.v1 and we also manage M.v1 as a message exchanged between C and B.

Regarding processes, we create new versions when there are changes to the involved tasks and/or events or in the way they are linked together using sequence flows and gateways. In the same way, changes to tasks and events may result in the creation of new task and event versions. In addition, we create new versions of tasks or events involved in message exchange, when there are changes to the exchanged messages.

Finally, regarding collaborations, new versions may result from changes to participants involved. Thus when we add or delete a participant, it is necessary to adapt the current collaborative process to this change: we have to incorporate the added participant or to possibly replace the deleted one. New versions of collaborations may also result from changes to involved processes or exchanged messages. Exchanged messages have an important impact in collaboration flow. Thus any change in a sent or a received message affects the involved tasks or events, and consequently the involved process. So, when we add (or delete) a message, we have to add (or to delete) a received and a send activity, which leads to changing the process schema. In this case, the other processes involved in the collaboration have in turn to be adapted to this change to ensure continued collaboration.

On the other hand, BPMN 2.0 meta-model provides extension mechanisms through classes *Extension*, *ExtensionDefinition* and *ExtensionAttributeDefinition*, and, as suggested in [42], each recommended extension has to be assigned to these classes. Therefore, we recommend adding the classes *VersionExtensionDefinition* and *VersionExtension AttributeDefinition* to model the specific attributes which versionable classes include (version number, creator name, creation date and status). Thus each *Version_of_Versionable* class of BPMN4VC is a sub-class of the abstract class *VersionExtensionDefinition*.

C. Dynamic Aspects of BPMN4VC: Collaboration Version Management

In this section, we first give an overview of the operations available for collaboration version management within a state chart, before detailing them.

1) State Chart. In order to handle versions of collaborations modelled as instances of BPMN4VC meta-model, we propose a taxonomy of operations which allows to create, derive, update, validate and delete collaboration versions.

The UML state chart given in Fig. 6 indicates when these operations are available with respect to the version state. Some of them are available whatever the state of the version on which they are performed, while others are available only in some cases. In the state chart, each operation is described using the notation Event/Operation whose meaning is "if Event appears then Operation is triggered".

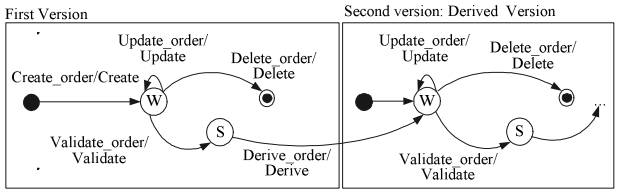

Fig. 6. UML State Chart for Versions

When the create order event appears, the create operation is carried out to create both a collaboration and its corresponding first version. The state of the created version is *working* (W state). In this state, a version is not yet a final one and it can be updated using the Update operation. Create and update operations can be specified using a set of primitives. These primitives change according to the classes in which they are defined. However, they share the same general idea that is to give values to properties and relationships of the considered classes (cf. III.C.2).

A working version can be deleted (Delete operation) or validated (Validate operation). When the Validate operation is performed, the corresponding version becomes *stable* (S state). This state indicates that a version is a final one, on which no additional updates can be performed. Moreover, validation of a version may trigger the validation of other versions as illustrated in III.C.3.

Finally, a stable version can be deleted or can serve as a basis for the creation of a new version using the Derive operation. The created version is a working version. Before being updated, it has the same value as the derived one. Moreover, derivation of a version may trigger the derivation of other versions, which are linked to the derived one (cf. III.C.4).

2) Update Operation. Table 1 below gives the definition of Update operation, indicating the low-level primitives this operation includes.

For instance, the update of a collaboration includes low-level primitives supporting the addition (+) or deletion (-) of participants (*i.e.*, their process) involved in the collaboration along with their interactions (*i.e.*, message flows between their processes). In the same way, the update of an interaction includes low-level primitives supporting the addition or deletion of messages, and the addition and deletion of message flows which are triplet gathering a message sent from a send node to a receive node. Finally, the update of a collaboration also includes low-level primitives supporting the addition or deletion of tasks, events, sequence flows and gateways.

Table 1. Primitives of the Update Operation

BPMN4VC Concepts	Primitives
Collaboration	+/- Participant/Process +/- Interaction
Interaction	+/- Message +/- Message flow
Participant/Process	+/- Task +/- Event +/- Sequence flow +/- Gateway

In order to make update operation easier to perform, we recommend a set of 6 adaptation patterns allowing process designers to modify collaboration schemas more easily by using high-level primitives (*e.g.*, AddInteraction to add an interaction into a collaboration) instead of low-level primitives (*e.g.*, AddNode, AddMessage, AddMessageFlow). These adaptation patterns will be detailed in section 4.

3) Validate Operation. This operation is performed to make a working version stable, *i.e.* when the considered version does not need additional updates. Validation of a version may trigger the validation of other versions, which are linked to it. Fig. 7 illustrates the validation propagation. More precisely, the black arrows correspond to initial validations while the grey arrows correspond to propagated validations.

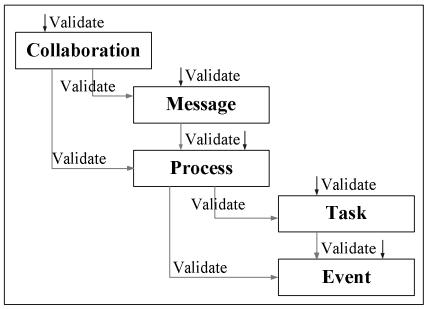

Fig. 7. Validation Propagation

According to Fig. 7, the validation of a collaboration version triggers the validation of processes and messages within this collaboration. In the same way, the validation of a process version triggers the validation of its versioned components, *i.e.* versions of its tasks and versions of its events.

4) Derive Operation. This operation allows the creation of a new version from an existing stable one. The created version is a working version. Before being updated, it has the same value as the derived one. Moreover, derivation of a version may trigger the derivation of other versions, which are linked to the derived one. Fig. 8 illustrates this derivation propagation. Again, the black arrows correspond to initial derivations, while the grey ones correspond to propagated derivations.

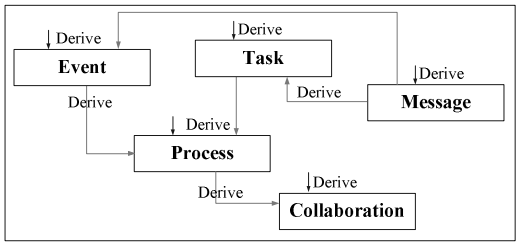

Fig. 8. Derivation Propagation

This propagation is due to the composition relationships existing between collaboration, message, process, task and event. Therefore, the derivation of a task or event triggers the derivation of its corresponding process. In the same way, the derivation of a process triggers the derivation of the corresponding collaboration. In addition the derivation of a message triggers the derivation of its corresponding events or tasks (*i.e.*, the events or tasks involved with the message in a message flow).

IV. Adaptation Patterns for Collaboration Update

We introduce 6 adaptation patterns to implement the update operation previously presented. These patterns focus on interaction within collaborations. As indicated before, an interaction is a message flowing from a send node to a receive node. Therefore, an interaction can be considered as a triplet (message, send node, receive node).

As illustrated in Fig. 9, the recommended patterns are the following: *AddInteraction*, *DeleteInteraction, Move Interaction, Replace Interaction*, *SwapInteraction* and *ModifyInteraction*. They are used to update a version of collaboration, created directly from scratch (first collaboration version) or derived from an existing one (derived collaboration version). Of course, the state of the collaboration version is working (see Fig. 6). In both cases, the patterns recommended move the considered collaboration from an initial schema IS to a final schema FS, adding, deleting, moving, replacing, swapping or modifying interaction between the processes involved. This schema update generally leads to the definition of derived process versions, derived task versions, derived messages and/or derived event versions. Thus, version management operations and adaptation patterns are both used to update a collaboration.

CPAP1: Add Interaction	**CPAP4**: Replace Interaction
CPAP2: Delete Interaction	**CPAP5**: Swap Interaction
CPAP3: Move Interaction	**CPAP6**: Modify Interaction

Fig. 9. Adaptation Patterns for Collaborative Process

In the following sub-sections, we present the recommended adaptation patterns. For each one, we consider the case when a new collaboration derived from an existing one moves from an initial schema IS to a final one FS.

A. AddInteraction Pattern

AddInteraction is a high-level pattern used to add an interaction, *i.e.* a message and its send and receive nodes to an existing collaboration. This pattern includes the following low-level primitives, briefly introduced in Table 1:

- AddNode (N, P, G, pN, fN, Type). This primitive uses the AP1 Insert Process Fragment defined in [30] to add in the process P of the current collaboration the interaction node *N* between the flow nodes *pN* (the previous node of N) and *fN* (the following node of N). The way in which the added node N is coordinated with the other nodes is indicated by *G*: it may be sequence, parallel or conditional. In addition, the type of added node N is indicated by *Type*, which can be a (send or receive) task or event. Note that this primitive implements the following operations of Table 1: +task and +event.
- AddMessage(M). This primitive adds the message *M* in the current collaboration.
- AddMessageFlow(M, sN, rN). This primitive adds the message flow (M, sN, rN) into the current collaboration, where *M* is a message between *sN* (the send node of M) and *rN* (the receive node of *M)*. Note that both sN and rN are interaction nodes.

The *AddInteraction* pattern may be achieved in four possible cases: (i) Add a message and the corresponding send and receive nodes, (ii) Add a message between existing send and receive nodes, (iii) Add a message and the corresponding send node, and (iv), Add a message and the corresponding receive node.

1) Case #1: Add a message and the corresponding send and receive nodes. In this case, the send and receive nodes (*i.e.*, tasks or events) of the added message do not exist in the collaboration. Therefore, the nodes have to be added to the corresponding processes, and the new message has to be added into the collaboration. Finally a new message flow linking the message and the nodes has to be defined.

Depending on the way the nodes are inserted within the collaborative processes (sequence, parallel or conditional insert), two options have to be considered. Due to a lack of space, Fig. 10 illustrates only two of these options, which allows (i) the insertion of the first version of the send task Ai and receive task Bi respectively in the second version of processes A and B and (ii) the definition of the first version of message M sent from the first version of Ai to the first version of Bi. The first option corresponds to a sequential insert of tasks Ai and Bi respectively in processes A and B, while the second one corresponds to a parallel insert. To address a conditional insert, we only have to have the gateway and use the conditional one instead of the parallel one.

For each option of case#1, we provide the corresponding interaction pattern order along with the corresponding low-level primitives performed when applying it. For instance, in possibility #1, we use *AddMessage* (M_{v1}) primitive to add the first version of message M, *AddNode* (Ai_{v1}, A_{v2}, *'Sequence'*, $A1_{v1}$, $A2_{v1}$, *'SendTask'*) primitive for a sequential insert of the first version of the send task Ai in the second version of process A, *AddNode* (Bi_{v1}, B_{v2}, *'Sequence'*, $B1_{v1}$, $B2_{v1}$, *'ReceiveTask'*) primitive for a sequential insert of the first version of the receive task Bi into the second version of process B, and finally *AddMessageFlow*(M_{v1}, Ai_{v1}, Bi_{v1}) primitive to define that the first version of message M is sent from the first version of Ai to the first version of Bi. These modifications, along with sequence flow modifications are shown in grey. The modifications which are shown in thick lines correspond to version management operation modifications.

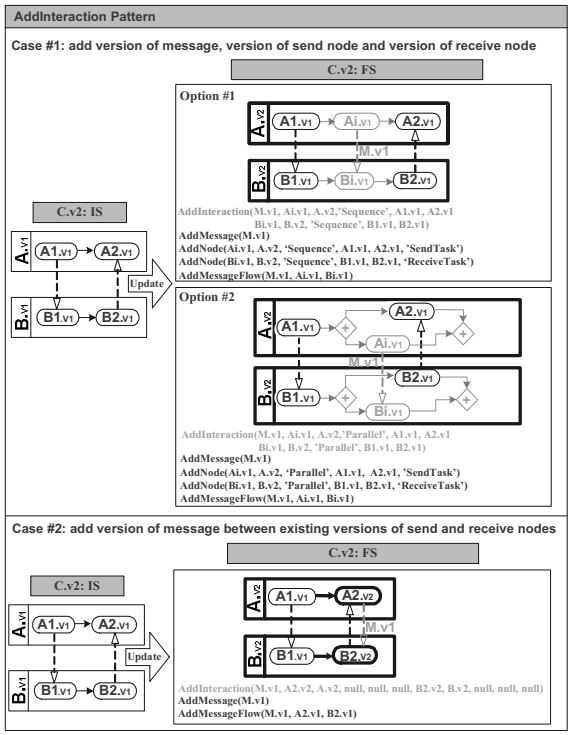

Fig. 10. Case #1 and Case #2 for AddInteraction Pattern

2) Case #2: Add a message between existing send and receive nodes. In this case, the send and receive nodes of the added message already exist within the collaboration. Therefore, a new message has to be added first into the collaboration and a new message flow linking the added message and the existing nodes has to be defined subsequently (cf. Fig. 10).

More precisely, the primitive *AddMessage*(M_{v1}) is used to add the first version of message M whereas the primitive *AddMessageFlow*(M_{v1}, $A2_{v2}$, $B2_{v2}$) allows for the adding of the message flow involving the first version of the message M, the second version of task B2, and the second version of task A2.

3) Case #3: Add a message and the corresponding send node. In this case, only the send node of the added message does not exist in the collaboration. Therefore, this node has to be added first into the corresponding process, and the new

message has to be added into the collaboration subsequently. Finally a new message flow linking the added message and nodes has to be defined.

Depending on the way the send node is inserted within a process (sequence, parallel or conditional insert), three options have to be considered. Fig. 11 illustrates two of these options, allowing (i) the insertion of the first version of the send task Bi in the first version of process B and (ii) the definition of the first version of the message M sent from the first version of the task Bi to the second version of task A2. The first option corresponds to a sequential insert of Bi in process B, while the second and third options correspond to a parallel insert and a conditional insert respectively.

For each option of case#3 and case#4, we provide the corresponding interaction pattern order along with the corresponding low-level primitives performed when applying it. For instance, in option #1 of case #3, we use *AddMessage*(M_{v1}) primitive to add the first version of message M, *AddNode (*$Bi_{.v1}$*,* $B_{.v2}$*, 'Sequence', B1*$_{.v1}$*,* $B2_{.v1}$*, 'SendTask')* primitive for a sequential insert of the first version of the send task Bi into the second version of process B and *AddMessageFlow(*M_{v1}*,* $Bi_{.v1}$*,* $A2_{.v2}$*)* primitive to define that the first version of message M is sent from the first

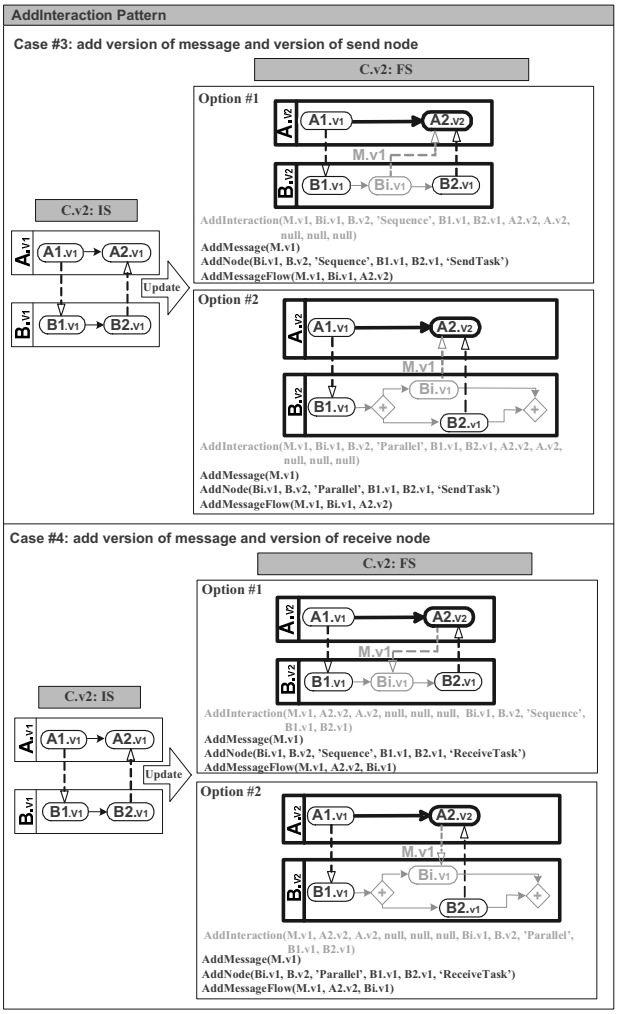

Fig. 11. Case #3 and Case #4 for *AddInteraction* Pattern

B. *DeleteInteraction Pattern*

DeleteInteraction is a high-level pattern used to delete a message and its interaction nodes in an existing collaboration. This pattern includes the following low-level primitives, briefly introduced in Table 1:

- DeleteNode(N, P). This primitive uses the AP2 Delete Process Fragment defined in [30] to remove the interaction node *N* from the process *P* in the current collaboration. Note that this primitive implements the following operations of Table 1: -task and -event.
- DeleteMessage(M). This primitive removes the message *M* from the current collaboration.
- DeleteMessageFlow(M, sN, rN). This primitive allows for the removal of the message flow involving the message *M*, and the send and receive interaction nodes *sN* and *rN* in the current collaboration.

The *DeleteInteraction* pattern may be achieved in four possible cases: (i) Delete a message and its corresponding send and receive nodes, (ii) Delete a message while keeping its corresponding send and receive nodes, (iii) Delete a message and its corresponding send node and (iv), Delete a message and its corresponding receive node.

1) Case #1: Delete a message and its corresponding send and receive nodes. In this case, the send and receive nodes of the deleted message are not part of other message flows of in the collaboration. So this message and its corresponding send and receive nodes can be deleted. More precisely, as illustrated in Fig.12, *DeleteMessageFlow(*M_{v1}*,* $A2_{.v1}$*,* $B2_{.v1}$*)* allows for the deletion of the message flow between A2.v1 and B2.v1, while *DeleteNode(*$A2_{.v1}$*,* $A_{.v2}$*), DeleteNode(*$B2_{.v1}$*,* $B_{.v2}$*)* and *DeleteMessage(*M_{v1}*)* allow for the deletion of the message and the nodes involved in the message flow.

2) Case #2: Delete a message and keep its corresponding send and receive nodes. In this case, the send and receive nodes of the deleted message are part of other message flows in the collaboration. Therefore, these nodes have to be kept in the collaboration. Consequently, only the message and its corresponding message flow have to be deleted (cf. Fig.12).

3) Case #3 (resp. case #4): Delete a message and its corresponding send (resp. receive) node. In this case, the send (resp.receive) node of the deleted message is not part of other message flows in the collaboration. So the message and its corresponding send (resp. receive) node have to be deleted, along with the corresponding message flow (cf. Fig. 12).

C. *MoveInteraction Pattern*

MoveInteraction is a high-level pattern used to shift an interaction from an initial position to a final one. It can be carried out by the combined use of the *AddInteraction* and *DeleteInteraction* patterns. It can also be defined using the following low-level primitives: *AddNode*, *DeleteNode*, *AddMessageFlow* and *DeleteMessageFlow*.

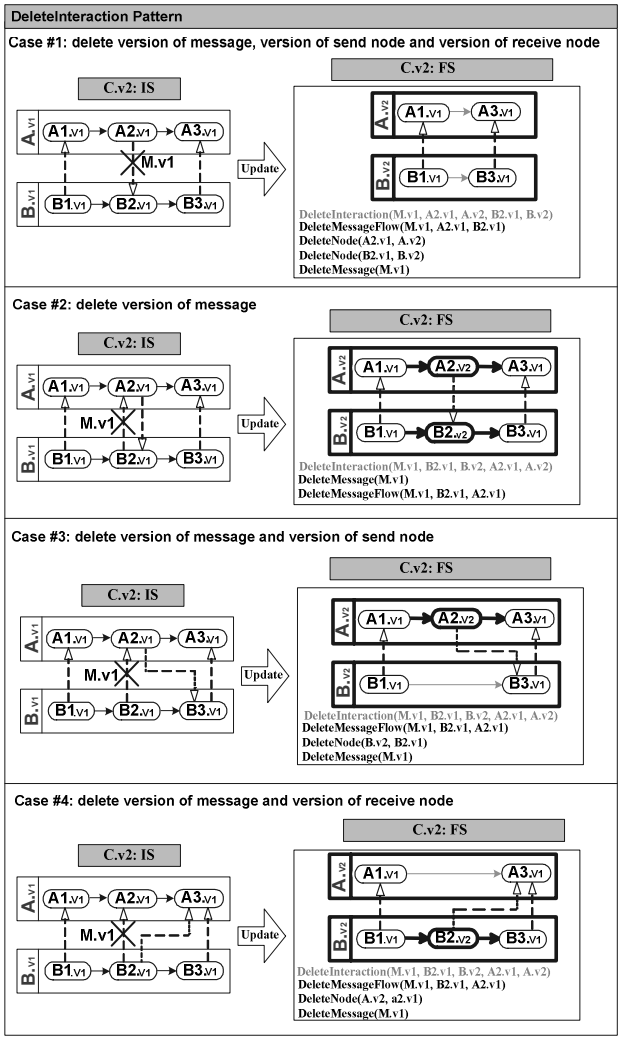

Fig. 12. *DeleteInteraction* Pattern

The possible options for moving an interaction from an initial position to a final one, according to the way the interaction is moved are: sequence move, parallel move and conditional move. These options are defined both in terms of adaptation patterns and low-level primitives. For instance, in the first option (sequence move), the following low-level primitives are performed: *DeleteMessage Flow(M.v1,B1.v1,A1.v1)* to delete the interaction between tasks B1.v1 and A1.v1, *DeleteNode(A1.v1,A.v2)* and *DeleteNode(B1.v1,B.v2)* to delete the first version of tasks A1 and B1, *AddNode(B1.v1, B.v2, 'Sequence', B3.v1, B4.v1, 'SendTask')* and *AddNode(A1.v1, A.v2, 'Sequence', A3.v1, A4.v1, 'ReceiveTask')* to re-insert B1.v1 and A1.v1 into their new positions, and *AddMessageFlow(M.v1, B1.v1, A1.v1)* to re-define the interaction involving M.v1 between B1.v1 and A1.v1. Due to a lack of space, Fig. 13 only illustrates a sequential move and a parallel move.

D. ReplaceInteraction Pattern

ReplaceInteraction is a high-level pattern which enables the replacement of an interaction with another one. Like *MoveInteraction*, this pattern can be carried out by the combined use of *InsertInteraction* and *DeleteInteraction* patterns. In addition, it can also be defined using the following

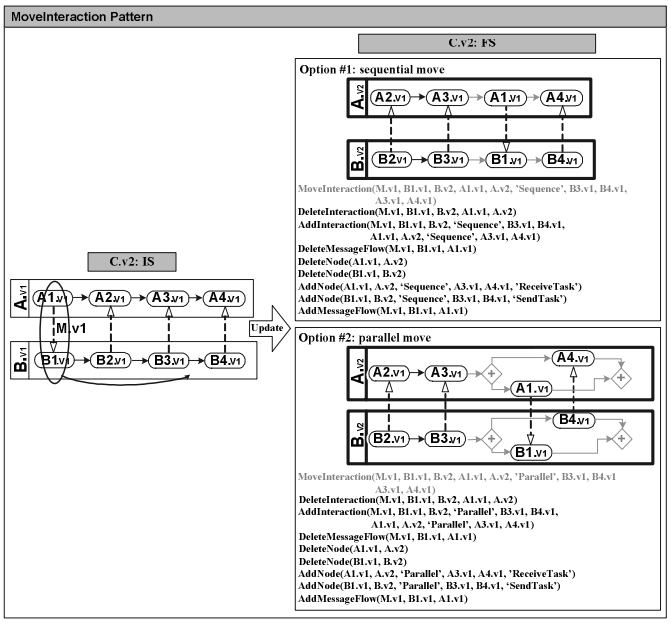

Fig. 13. *MoveInteraction* Pattern

low-level primitives: *AddNode*, *DeleteNode*, *AddMessage*, *DeleteMessage*, *AddMessageFlow* and *DeleteMessageFlow*.

Fig. 14 hereafter gives an example of the replacement of an interaction within a collaboration. In this example, the message flow involving M1.v1, A2.v1, and B2.v1 has to be replaced with the one involving M2.v1, Y.v1 and X. v1. The following low-level primitives are performed: (i) *DeleteMessageFlow(M1.v1, A2.v1, B2.v1)* and *Delete Message(M1.v1)* delete the interaction to be replaced along with the first version of the message M1, (ii) *DeleteNode(A2.v1, A.v2)* and *DeleteNode(B2.v1, B.v2)* delete the first versions of tasks A2 and B2 respectively from the second version of processes A and B, (iii) *AddNode(Y.v1, B.v2, 'Sequence', B1.v1, B3.v1, 'ReceiveTask')* and *AddNode(X.v1, A.v2, 'Sequence', A1.v1, A3.v1, 'SendTask')* insert the first version of tasks X and Y into the second version of processes A and B, (iv) *AddMessage(M2.v1)* add the first version of message M2, and finally (v) *AddMessageFlow(M2.v1, Y.v1, X.v1)* define the interaction involving M2.v1, Y.v1 and X.v1.

E. SwapInteraction Pattern

SwapInteraction is a high-level pattern which allows swapping an interaction for another one. This pattern can be defined using *MoveInteraction* pattern. It can also be defined using *AddInteraction* and *DeleteInteraction* patterns previously introduced, or using the following low-level primitives: *AddNode*, *DeleteNode*, *AddMessageFlow* and *DeleteMessageFlow*.

Fig. 14 gives an example of the *SwapInteraction* pattern allowing the exchange of the interaction involving the message M1.v1 sent from B1.v1 to A1.v1 with the interaction involving the message M2.v1 sent from B3.v1 to A3.v1. More precisely, the following low-level primitives are performed: (i) *DeleteMessageFlow(M1.v1, B1.v1, A1.v1)* and *DeleteMessage Flow(M2.v1, B3.v1, A3.v1)* to delete the message flows from the collaboration to swap, (ii) *DeleteNode(A1.v1, A.v2)*, *DeleteNode*

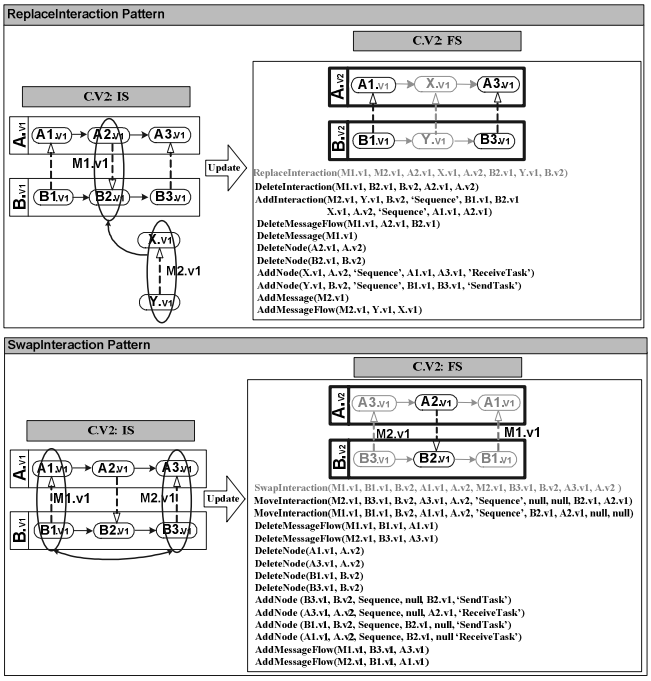

Fig. 14. *ReplaceInteraction* and *SwapInteraction* Patterns

(A3.v1, A.v2), *DeleteNode(B1.v1, B.v2)*, and *Delete Node(B3.v1, B.v2)* to delete the nodes involved in this swapping from the collaboration, (iii) *AddNode(A3.v1, A.v2, 'Sequence', null, A2.v1, 'ReceiveTask')*, *AddNode(A1.v1, A.v2, 'Sequence', A2.v1, null, 'ReceiveTask')*, *AddNode(B3.v1, B.v2, 'Sequence', null, B2.v1, 'SendTask')* and *AddNode(B1.v1, B.v2, 'Sequence', B2.v1, null, 'SendTask')* to re-insert the deleted nodes into their new positions, and finally (iv) *AddMessageFlow(M2.v1, B3.v1, A3.v1)* and *AddMessageFlow(M1.v1, B1.v1, A1.v1)* to re-define the swapped interactions involving M2.v1, B3.v1 and A3.v1 on the one hand and M1.v1, B1.v1 and A1.v1 on the other hand.

F. *ModifyInteraction Pattern*

ModifyInteraction is the last adaptation pattern aiming at updating an interaction by modifying its message and/or its interaction nodes. In addition to AddNode and DeleteNode primitives previously presented, this pattern includes the following low-level primitives:

- ModifySendNode(M, Ts). This primitive indicates that the message *M* is now sent from the interaction node *Ts*.
- ModifyReceiveNode(M, Tr).This primitive indicates that the message *M* in now received by the interaction node *Tr*.
- ModifyMessage (M, Id). This primitive indicates that the definition of *M* (property *ItemDefinition*) is replaced with *Id*. *Id* is the new value defining the structure and the type of information carried out by M.

Fig. 15 illustrates the different cases for *ModifyInteraction*, enumerating the possible combinations according to the new nodes of the interaction: do they already exist or do they have to be created? Due to lack of space, we detail only case #1 in the following.

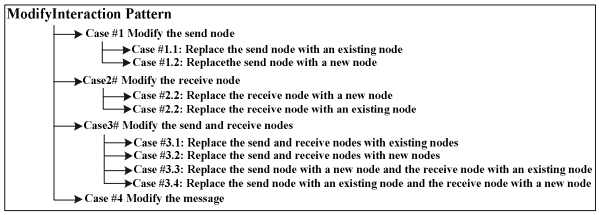

Fig. 15. Cases for *ModifyInteraction* Pattern

In case #1, the send node of the updated interaction has to be modified. This case may be carried out according to two possible sub-cases presented below.

1) Case #1.1: Replace the send node with an existing one. In this case, the send node of the updated interaction is replaced with an existing node. Consequently, it has to be deleted if it is not part of other message flows (option #1). Otherwise, it has to be kept (option #2). As illustrated in Fig. 16, the folowing low-level primitives are performed: *Modify SendNode(M.v1, A1.v2)* to specify that M.v1 is now sent from A1.v2 and *DeleteNode(A2.v1, A.v2)* to delete the first version of the task A2 into the collaboration.

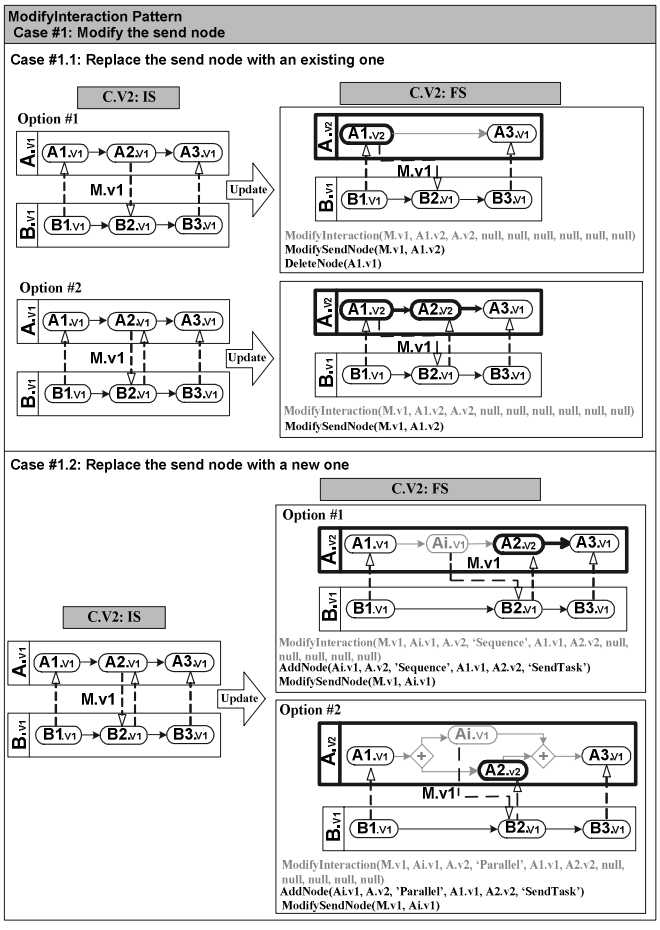

Fig. 16. Case #1.1 and Case #1.2 for ModifyInteraction Pattern

2) Case #1.2: Replace the send node with a new one. In this case, the send node of the updated interaction is replaced with a new interaction node. Therefore, the new node has first to be added and the interaction has to be updated subsequently.

Depending on the way the new send node is inserted (sequence, parallel or conditional insert), three options have to be considered. Due to lack of space, Fig. 16 illustrates two of these three options allowing (i) the insertion of the first version of the send task Ai in the first version of process A and (ii) the insertion of the corresponding message flow involving M.v1, Ai.v1 and B2.v1. The first option corresponds to a sequential insert of Ai in A, while the second one corresponds to a parallel insert. More precisely, if we consider a sequential insert, the following low-level primitives are performed: (i) *AddNode* ($Ai_{.V1}$, $A_{.V2}$, *'Sequence'*, $A1_{.V1}$, $A2_{.V1}$, *'SendTask'*) for a sequential insert of the first version of the task Ai between A1.v1 and A2.v1 and (ii) *ModifySendNode* ($M_{.V1}$, $Ai_{.V1}$) for modifying the corresponding message flow and defining Ai.v1 as the send task of the message M.v1.

V. Conclusion and Outlook

This paper has addressed collaborative process flexibility issue, which is an important challenge to address in Business Process Management area. It has presented BPMN4VC, an extension of BPMN using versions to address this issue along with a set of 6 adaptation patterns to make the update of collaborative process easier.

More precisely, the paper contributions are the following. Firstly, the paper has extended the BPMN2.0 meta-model to incorporate version modelling capability and to keep track of task, event, message, process and collaboration flexibility. The paper has also addressed the dynamic aspects of collaborative process version management, defining state charts for process versions and corresponding operations (create, update, delete, validate and derive operations). Secondly the paper has introduced the following collaborative process adaptation patterns: AddInteraction, DeleteInteraction, Move Interaction, Replace Interaction, SwapInteraction and ModifyInteraction. These patterns are high-level primitives used to update collaborative process schema considering the organizational, informational and behavioural dimensions of collaborative processes. They are well-suited for coping with the evolving nature of collaborative processes. Thirdly, the paper has illustrated the intertwining of the adaptation patterns and the versioning operations, taking the state of versions (W: working, S: stable) into account. We believe these contributions to be a step forward in addressing the flexibility of collaborative processes.

Our future work will take three directions. First, we have to evaluate our contributions. To do so, we will implement the BPMN4VC-Modeller, a specific tool intended for business process designers and which supports the modelling of BPMN collaborative process versions and the implementing of the dynamic aspects of version management including the recommended adaptation patterns. This tool will be used for a qualitative and a quantitative evaluation. More precisely, we intend to measure the usability (ease of use, understandability), and the efficiency (reduce modelling efforts) of the BPMN4VC-Modeller to address the collaborative process flexibility. A workshop at the University of Sfax in Tunisia will be organized for such an evaluation, involving graduate and PhD students having knowledge in the BPM area. We also intend to verify, from collaborative process practical cases, whether the recommended patterns cover any changes that may occur.

Secondly, regarding future works in relation to the notion of version, we will incorporate the notion of context in BPMN4VC in order to take the explicit definition of situations, in which collaborative process versions have to be used, into account, as we did for intra-organizational processes [29]. In addition, we also have to address the storage of versions (taking recommended adaptation patterns into account [44]) and the instance migration issues.

Thirdly, regarding future works in relation to adaptation patterns many challenges have to be addressed, in addition to their implementation in the BPMN4VC-Modeller. Among those identified in [45], we highlight the following ones:

- Formalization of collaborative process adaptation patterns, to obtain unambiguous pattern definitions that will allow their implementation in collaborative process support systems. This formalization will also help when using recommended adaptation patterns as a means for evaluating collaborative process change support in existing PaIS.
- Correctness of collaborative process schemas after using the recommended adaptation patterns: in particular, we have to check whether the compatibility and consistency properties of the collaborating business partners are to be kept [46].
- Use of the collaborative process adaptation patterns to answer the following question: how does the use of these adaptation patterns influence the modelling of collaborative process schemas?

VI. References

[1] M. Dumas, W. van der Aalst, and A. ter Hofstede, "Process-Aware Information Systems: Bridging People and Software through Process Technology". Wiley & Sons, 2005.

[2] M. Weske, "Business Process Management: Concepts, Languages, Architectures". Springer, 2007.

[3] M. Reichert, and B. Weber, "Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies". Springer, 2012.

[4] M. Rosemann, and W. van der Aalst, "A Configurable Reference Modeling Language". *Information Systems*, vol. 32, n°1, 2007, pp. 1–23.

[5] A. Hallerbach, T. Bauer, and M. Reichert, "Capturing Variability in Business Process Models: the Provop Approach". *Software Maintenance*, vol. 22, n°6-7, June 2010, pp. 519–546.

[6] S. Rinderle, and M. Reichert, "Data-Driven Process Control and Exception Handling in Process Management Systems". *Int. Conference on Advanced Information System Engineering*, Luxembourg, June 2006, pp. 273–287.

[7] M. Adams, A. ter Hofstede, D. Edmond, and W. van der Aalst, "Dynamic and Extensible Exception Handling for Worklows: A Service-Oriented Implementation". *Int. Conference on Cooperative Information Systems*, Vilamoura, Portugal, November 2007, pp. 95–112.

[8] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. "Workflow Evolution". *Data and Knowledge Engineering*, vol. 24, n°3, 1998, pp. 211–238.

[9] P. Kammer, G. Bolcer, R. Taylor, and R. Bergman, "Techniques for supporting Dynamic and Adaptive Workflow". *Int. Journal on Computer Supported Cooperative Work*, vol. 9, n°3/4, 1999, pp. 269–292.

[10] M. Kradolfer, and A. Geppert, "Dynamic Workflow Schema Evolution based on Workflow Type Versioning and Workflow Migration. *Int. Conference on Cooperative Information Systems*, Edinburgh, Scotland, September 1999, pp. 104–114.

[11] S. Rinderle, M. Reichert, and P. Dadam, "Flexible Support of Team Processes by Adaptive Workflow Systems". *Distributed and Parallel Databases*, vol. 16, n°1, 2004, pp. 91–116.

[12] X. Zhao, X., and C. Liu, "Version Management in the Business Change Context". *Int. Conference on Business Process Management*, Brisbane, Australia, September 2007, pp. 198–213.

[13] M. Adams, A. ter Hofstede, D. Edmond, and W. van der Aalst, "Worklets: a Service-Oriented Implementation of Dynamic Flexibility in Workflows". *Int. Conference on Cooperative Information Systems*, Montpellier, France, October 2006, pp. 291–308.

[14] E.Andonoff, L. Bouzguenda, and C. Hanachi, "Specifying Web Workflow Services for Finding Partners in the Context of Loose Inter-organizational Workflow". *Int. Conference on Business Process Management*, Nancy, France, September 2005, pp. 120–136.

[15] E. Andonoff, W. Bouaziz, C. Hanachi and L. Bouzguenda., "An Agent-based Model for Autonomic Coordination of Inter-Organizational Business Processes". *Informatica*, vol. 20, n°3, September 2009, pp. 323–342.

[16] R. Lu, S. Sadiq, and G. Governatori, "On Managing for Business Process Variant". *Data and Knowledge Engineering*, vol. 68, n°7, 2009, pp. 642–664.

[17] M. Pesic, H. Schonenberg, N. Sidorova, and W. van der Aalst, "DECLARE: full support for Loosely-Structured Processes". *Int. Conference on Enterprise Distributed Object Computing*, Annapolis, Maryland, USA, October 2007, pp. 287–300.

[18] M. Pesic, H. Schonenberg, and W. van der Aalst, "Constraint-based Workflow Models: Change made Easy". *Int. Conference on Cooperative Information Systems*, Vilamoura, Portugal, November 2007, pp. 77–94.

[19] D. Müller, M. Reichert, and J. Herbst., "Data-driven Modeling and Coordination of Large Process Structures". *Int. Conference on Cooperative Information Systems*, Vilamoura, Portugal, November 2007, pp. 131–149.

[20] Müller, D., Reichert, M., Herbst, J.: A New Paradigm for the Enactment and Dynamic Adaptation for Data-driven Process Structures. International Conference on Advanced Information Systems Engineering, Montpellier, France, June 2008, pp. 48–63.

[21] W. van der Aalst, M. Weske, and D. Grünbaur, "Case Handling: a New Paradigm for Business Process Support". *Int. Journal on Data Knowledge Engineering,* vol. 53, n°2, 2005, pp.129–162.

[22] G. Bruno, F. Dengler, B. Jennings, R. Khalaf, S. Nurcan, M. Prilla, M., Sarini, M., Schmidt, R., and R. Silva, "Key Challenges for Enabling Agile BPM with Social Software. *Software Maintenance and Evolution: Research and Practice*, vol. 23, n°4, June 2011, pp. 297–326.

[23] M. Döhring, B. Zimmermann, and L. Karg "Flexible Workflows at Design-time and Run-time using BPMN2 Adaptation Patterns". *Int. Conference on Business Information Systems*, Poznan, Poland, June 2011, pp. 25–36.

[24] R. Angles, P. Ramadour, C. Cauvet, and S. Rodier, "V-BPMI: A variability-oriented Framework for web-based Business Processes Modeling and Implementation". *Int Conference on Research Challenges in Information Science*, Paris, France, May 2013, pp. 313–323.

[25] V. Torres, S. Zugal, B. Weber, M. Reichert, C. Ayora, and V. Pelechano, "A Qualitive Comparison of Approaches Supporting Business Process Variability". *Business Process Management Workshops (rBPM)*, Tallin, Estonia, September 2008, pp. 560–572.

[26] P. Dadam, and M. Reichert, "The ADEPT Project: a decade of research and vevelopment for Robust and Flexible Process Support". *Computer Science – R&D*, vol. 23, 2009, pp. 81–97.

[27] MA. Chaâbane, E. Andonoff, L. Bouzguenda, and R. Bouaziz, "Versions to Address Business Process Flexibility Issue". *Int. Conference on Advances in Databases and Information Systems*, Riga, Latvia, September 2009, pp. 2–14.

[28] I. Ben Said, MA. Chaabane, E. Andonoff, and R. Bouaziz, "Extending BPMN 2.0 Meta-model for Process Version Modelling". *Int. Conference on Enterprise Information Systems*, Lisbon, Portugal, April 2014, pp. 384–393.

[29] I. Ben Said, MA. Chaabane, E. Andonoff, and R. Bouaziz, "Context-Aware Adaptive Process Information Systems: The Context-BPMN4V Meta-Model". *Int. Conference on Advances in Databases and Information Systems*, Ohrid, Macedonia, September 2014, pp. 366–382.

[30] B. Weber, M. Reichert, and S. Rinderle, "Change Patterns and Change Support Features – Enhancing Flexibility in Process-Aware Information Systems". *Data and Knowledge Engineering*, vol. 66, n°3, 2008, pp 438–466.

[31] W. van der Aalst, "Loosely Coupled Inter-Organizational Workflows: Modelling and Analysing Workflows Crossing Organizational Boundaries". *Information and Management*, vol. 37, n°2, 2000, pp. 67–75.

[32] I. Chebbi., S. Dustdar and S. Tata, "The View-based Approach to Dynamic Inter-Organizational Workflow Cooperation". *Data Knowledge Engineering*, vol. 56, no. 2, 2006, pp. 139–173.

[33] J. Meng, S. Su, H. Lam, A. Helal, J. Xian, X. Liu and S. Yang, "DynaFlow: a Dynamic Inter-Organisational Workflow Management System". *Business Process Integration and Management*, vol. 1, n°2, 2006, pp. 101–115.

[34] E. Andonoff and L. Bouzguenda, "Agent-Based Negotiation between Partners in Loose Inter-Organizational Workflow". *Int. Conference on Intelligent Agent Technology*, Compiègne, France, September 2005, pp. 619–625.

[35] W. Bouaziz, and E. Andonoff, "Autonomic Protocol-based Coordination in Dynamic Inter-Organizational Workflow". *Int Conference on Research Challenges in Information Science*, Paris, May 2013, pp. 555–566.

[36] S. Boukhedouma, M. Oussalah, Z. Alimazighi, and D. Tamzalit, "Adaptation patterns for Service-based Inter-Organizational Workflows". *Int. Conference on Research Challenges in Information Science*, Paris, France, May 2013, pp. 567–576.

[37] S. Boukhedouma, M. Oussalah, Z. Alimazighi, and D. Tamzalit, "Flexible Loosely Coupled Inter-Organizational Workflows using SOA". *Int. Conference on Computer Systems and Applications*, Ifrane, Morocco, May 2013, pp. 1–8.

[38] D. Domingosa , R. Martinhoa, and C. Cândidoa, "Flexibility in cross-organizational WS-BPEL Business Processes". *Procedia Technology*, vol. 9, 2013, pp. 584–595.

[39] W. Fdhila, A. Baouab, K. Dahman, C. Godart, O. Perrin, and F. Charoy, "Change Propagation in Decentralized Composite Web Services". *Int. Conference on Collaborative Computing: Networking, Applications and Worksharing*. Orlando, Florida, USA, October 2011, pp. 508–511.

[40] W. Fdhila, S. Rinderle-Ma, and M. Reichert, "Change Propagation in Collaborative Processes Scenarios". *Int. Conference on Collaborative Computing: Networking, Applications and Worksharing*. Pittsburg, Pennsylvania, USA, October 2012, pp. 452–461.

[41] W. Fdhila, C. Indiono, S. Rinderle-Ma, and M. Reichert, "Dealing with Change in Process Choreographies: Design and Implementation of Propagation Algorithms". *Information Systems*, vol. 49, 2015, pp. 1–24.

[42] OMG,. Business Process Model and Notation (BPMN) Version 2.0. OMG Document Number: formal/2011-01-03, available at: http://www.omg.org/spec/BPMN/2.0, 2011.

[43] E. Andonoff, G. Hubert, A. Le Parc, and G. Zurfluh, "Integrating Versions in the OMT Models". *Int. Conference on Conceptual Modeling*, Cottbus, Germany, October 1996, pp. 472–487.

[44] J. Kuster, C. Gerth, and G. Engels. "Dynamic Computation of Change Operations in Version Management of Business Process Models". *Int. Conference on Model Driven architecture – Foundations and Applications*, Paris, France, June 2010, pp. 201–216.

[45] M. Reichert, and B. Weber, "Process Change Patterns: Recent Research, Use Cases, Research Directions". *Seminal Contributions to Information Systems Engineering*, 2013, pp. 397–404.

[46] G. Decker, and M. Weske, "Behavioral Consistency for B2B Process Integration". *Int. Conference on Advanced Information Systems Engineering*, Trondheim, Norway, June 2007, pp.81–95.