DEALING WITH BUSINESS PROCESS EVOLUTION USING VERSIONS

Mohamed Amine Chaâbane, Eric Andonoff
Laboratoire IRIT, Université Toulouse 1, 2 rue du Doyen Gabriel Marty, 31042 Toulouse Cédex, France

Lotfi Bouzgenda, Rafik Bouaziz
Laboratoire MIRACL, ISIMS, Route de Tunis, km 10, BP 242, 3021 Sakeit Ezzit, Sfax, Tunisia

Keywords: Version, Business Process Evolution, Meta-model, Petri net with Objects.

Abstract: Competition in which enterprises and organizations are involved nowadays imposes them to often make evolve their business processes in order to meet, as quickly as possible, new business or production requirements. This paper proposes to adopt a version-based approach to support these dynamic changes of business processes. This approach permits to keep chronological business process changes: it is then possible to allow several instances of a same business process to own different schemas, each one representing a possible schema for the considered business process. Consequently, this approach is very suitable to deal with long-term business process evolution: it does not necessarily impose the adaptation and migration of running instances of business processes to a new business process schema. The paper contribution is threefold. First, it defines a meta-model for designing versions of business processes considering the three dimensions of business processes: the informational, organizational and process dimensions. Then, it introduces a taxonomy of operations for business process version management. Finally, it proposes to formalize and visualize modeled versions of business processes using a Petri net-based formalism, namely Petri net with Objects.

1 INTRODUCTION

Nowadays, the importance of business processes in enterprises' and organizations' Information Systems (IS) is widely recognized. As a consequence, these last few years, there has been a shift from data-aware IS to process-aware IS (Aalst and al, 2007). However, even if important advances have been done in business process management, several problems are still to be dealt. Among them, the business process evolution problem that can be posed as follows: *how to support dynamic change of business processes* (Smith and Fingar, 2003), (Aalst and al, 2003-a)?

The competitive and dynamic worldwide economic context, in which enterprises and organizations are involved, lead them to often change and adapt their business processes in order to meet new business or production requirements. Consequently, the business process evolution problem is really a relevant problem. This problem has mainly been addressed in the Workflow context using two main approaches: an adaptive-based approach and a version-based approach. The *adaptive-based* approach consists in defining a set of operations supporting both workflow process schema changes, and adaptation and migration of their corresponding instances (Casatia and al, 1996), (Reichert and Dadam, 1998), (Kammer and al, 2000), (Rinderle and al, 2004). In this approach, only one schema is kept for all modelled workflow processes. This approach has been investigated intensively and its implementations, ADEPT (Reichert and Dadam, 1998) and JOpera (Heinis and al, 2005), are probably the most successful Workflow Management Systems (WfMS) regarding workflow process' schema evolution.

In the *version-based* approach, different instances of a same workflow process can have different schemas. Thus, it is possible to distinguish between temporary and permanent updates for workflow processes since it is possible to keep track of chronological workflow process changes, each

Amine Chaâbane M., Andonoff E., Bouzgenda L. and Bouaziz R. (2008).
DEALING WITH BUSINESS PROCESS EVOLUTION USING VERSIONS.
In *Proceedings of the International Conference on e-Business*, pages 267-278
DOI: 10.5220/0001903702670278

one representing a possible schema for the considered workflow process.

In the workflow context, where long-term processes are involved, adaptation and migration of workflow process instances according to a new schema are not always easy and are sometimes impossible (Casati and al, 1996). So, it is important to be able to manage different schemas for a workflow process in order to allow several instances of this workflow process to own different schemas (Kradolfer and Geppert, 1999). Thus, the version-based approach is a promising solution to deal with business process evolution.

Versions are used in several fields of computer science in which was highlighted the need to describe evolution of real world entities over time. Thus, versions are used in the database field mainly in object-oriented databases (Cellary and Jomier, 1990), (Sciore, 1994), or scientific databases (Chen and al, 1996) but also for specific database application fields such as computer aided design or computer aided manufacturing (Chou and Kim, 1986), (Katz, 1990). Versions are also used in software engineering to handle software configurations (Kimball and Larson, 1991). Versions are also considered in conceptual models such as the Entity Relationship model (Roddick and al, 1993) or the OMT model (Andonoff and al, 1996).

Although versions are used in several areas of computer science, to the best of our knowledge, only few efforts have been put on version management in the business process (workflow) context (in the remainder of the paper, the terms workflow and business process will be used equally).

We distinguish two main contributions about versions of business processes in literature. (Kradolfer and Geppert, 1999) have proposed to deal with dynamic workflow evolution, i.e. modification of workflow process schemas in the presence of active workflow process instances, introducing versions of workflow process schemas. This work has defined a set of operations for workflow process schema modification and, if possible, a strategy for migration of workflow process instances. Recently, (Zhao and Liu, 2007) have also defended the advantages of a version-based approach to face business process evolution. More precisely, this work proposes to model versions of workflow process schemas using graphs. It also presents a set of operations enabling updates of graphs and defines two strategies to extract versions of workflow process schemas from these graphs.

We believe that these two propositions need to be revisited. Indeed, both (Kradolfer and Geppert, 1999) and (Zhao and Liu, 2007) addressed the issue of business process versioning only considering, what is called in the workflow literature, “process model”. Such a model describes tasks involved in the process and their coordination. But, using only this model is not enough to have a comprehensive description of business processes (Aalst, 1999). Two others models have to be considered: the organizational and the informational models. The organizational model structures the business process actors and authorizes them, through the notion of role, to perform tasks making up the process. The informational model defines the structure of the documents and data required and produced by the process. These two models are glued together with the process model since, in addition to the tasks and their coordination, the process model also defines the required resources (information, actors) to perform the tasks.

Consequently, this paper proposes to revisit the business process evolution problem using a version-based approach and considering both organizational, informational and process models of business processes. More precisely, this paper introduces:

- A meta-model for designing versions of business processes;
- A taxonomy of operations for business process version management;
- A formalization and a visualization of versions of business processes designed with the previous meta-model.

The remainder of this paper is organized as follows. Section 2 introduces the Business Process (BP) meta-model we use for designing business process, while section 3 introduces the Versioned Business Process (VBP) meta-model we propose for business process versioning. More precisely, section 3 first recalls the notion of version, then presents the versioning kit we propose for handling versions of business processes, and finally explains how the kit is merged with the BP meta-model to define the VBP meta-model. This section also gives an example of business process versioning. Section 4 is dedicated to the dynamic aspects of the meta-model: it presents a taxonomy of operations for business process version management. Section 5 presents our proposition for both formalization and visualization of workflow process versions using a formal model, namely Petri Net with Objects (Sibertin, 1985). Finally, section 6 stands our contribution according to related works and then concludes the paper.

2 MODELING BUSINESS PROCESSES

As mentioned before, a business process meta-model must allow the expression of three complementary aspects, usually described through three different interacting models: the organizational, informational and process models. The main important model is the process model which defines component tasks and their coordination, but this model also refers to the organizational model and to the informational model defining required and produced resources before and after tasks execution (Aalst, 1999).

Another important requirement for such a meta-model is its simplicity and efficiency: it must be comprehensive and must define the core (basic) concepts of the three complementary aspects of business processes: it must play the role of a Business Process Virtual Meta-model, i.e. a minimal meta-model for the design of business processes. This idea of Business Process Virtual Meta-model is the same as the one of Workflow Virtual Machine introduced in (Fernandes and al, 2004) to deal with the development of a Workflow Management System (WfMS) that supports changes in its workflow definition language(s).

But does such a meta-model for business process modelling (i.e. meeting the previous requirements) already exist, or do we have to define a new one by ourselves?

Despite the standization efforts of the Workflow Management Coalition (WfMC), different workflow or business meta-models exist in literature. The used vocabulary differs from one model to another, and yet, so far, the workflow and business process community seem to not have reached an agreement on which model to adopt, even if XPDL, BPMN and BPEL are standards recommended by the WfMC.

Some business process and workflow meta-models proposed in literature mainly focus on the process model -i.e. tasks description and their coordination- (e.g. (Zhao and Liu, 2007), BPEL, XPDL). Others also consider the informational model in addition to the process model (e.g. (Casati and al, 1995), (Kradolfler and Geppert, 1999), (Vossen and Weske, 1999), (Aalst and al, 2004)). Finally, some meta-models have a comprehensive approach for business process modeling considering the three complementary aspects (e.g. FlowMark and its successors MQSeries Workflow and WebSphere MQ Workflow (Leymann and Roller, 1999), Exotica (Mohan and al, 1995), OpenFlow (Halliday and al, 2001)). For instance, as illustrated in (Rosemann and zur Muehlen, 1998), the FlowMark meta-model proposes a very detailed description of workflow processes along with involved data flows and actors. However, these meta-models are very complex, specially with respect to the organizational dimension.

Consequently, we have defined our own meta-model which fulfils the previous requirements: (i) a comprehensive meta-model considering three complementary aspects of business processes (organizational, informational and process models), and (ii) a business process virtual meta-model as it defines the core (basic) concepts of the three complementary aspects of business processes. This meta-model is shown in the UML diagram of figure 1.

In this UML meta-model, a business process is either a composite process or an atomic process. A composite process is itself recursively composed of atomic or composite processes. It also uses a control pattern, which participates to the definition of business process coordination. In our meta-model, and as in, for instance (Manolescu, 2001), the main control patterns described in the literature are provided. Some of them are conditional (e.g. if, while…), while others are not (e.g. sequence, fork…). Their semantics is the following:

- *Sequence* pattern: it permits the execution of processes in a sequential order;
- *If* pattern: it allows processes execution according to a condition;
- *Fork* pattern: it spawns the parallel execution of processes and waits for the first to finish;
- *Join* pattern: it spawns the parallel execution of processes but waits for all of them before completing;
- *While* and *Repeat* patterns: they cyclically execute a process while or until a condition is achieved.

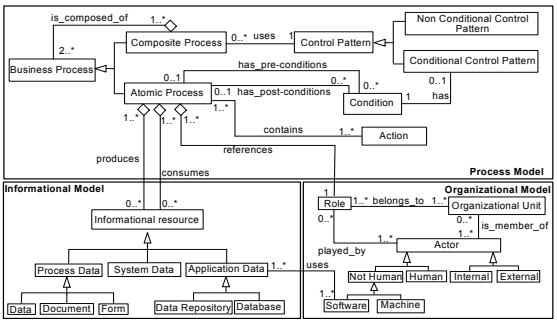

Figure 1: The Business Process meta-model.

An atomic process corresponds to a task to perform. It can have pre-conditions and post-conditions, and executes one or several actions. An atomic process is performed by a role (belonging to

the organizational model) and consumes and/or produces informational resources (belonging to the informational model). Informational resources correspond to system data, process data (i.e. data, document or form), and application data (i.e. database and data repository). A role is played by an actor belonging to some organizational units. An actor is a human resource or not (machine or software). Finally, an actor may be internal or external.

Going back to control patterns, our meta-model only includes low level (basic) control patterns; all the high level workflow patterns of (Aalst and al, 2003-b) are not considerer here (they are much more complex than what we need). In this way, the meta-model we propose could be seen as a Business Process Virtual Meta-model gathering the core (basic) concepts of business process models.

3 MODELING VERSIONS OF BUSINESS PROCESSES

First, this section briefly recalls the version notion as it is introduced in object-oriented databases and software engineering. Then, this section presents the Versioned Business Process (VBP) meta-model: it consists of a versioning kit to handle versions of business processes which is merged with the BP meta-model introduced before. Finally, this section illustrates the VBP meta-model instantiation to design versions of business processes.

3.1 Concept of Version

A real world entity has characteristics that may evolve during its life cycle: it has different successive states. In object-oriented database systems that provide version management, this entity is described by a set of objects called versions. A version corresponds to one of the significant entity states. Then, it is possible to manage several entity states (neither only the last one as in classical databases nor all the states as in temporal databases).

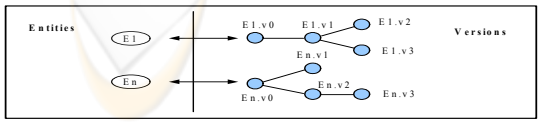

Figure 2: Representing entities with versions.

As illustrated in figure 2, the entity versions are linked by a derivation link; they form a version derivation hierarchy. When created, an entity is described by only one version. The definition of every new entity version is done by derivation from a previous one. Such versions are called derived versions (e.g. E1.v1 is a derived version from E1.v0). Several versions may be derived from the same previous one. They are called alternatives (e.g. E1.v2 and E1.v3 are alternatives derived from E1.v1).

A version is either frozen or working. A frozen version describes a significant and final state of an entity. A frozen version may be deleted but not updated. To describe a new state of this entity, we have to derive a new version (from the frozen one). A working version is a version that temporarily describes one of the entity states. It may be deleted and updated to describe a next entity state. The previous state is lost to the benefit of the next one.

3.2 The Versioned Business Process Meta-model

This meta-model consists of a versioning kit to handle versions of business processes, which is merged to the BP meta-model previously introduced.

3.2.1 Versioning Kit

This kit is very simple: it is composed of a class and a set of properties and relationships that make classes of the previous meta-model "versionable". A "versionable" class is a class whose instances are versions (Katz, 1990).

Thus, for each of these "versionable" classes, we define a new class which contains versions, called "Version of…". We also specify two new relationships: (i) the is_version_of relationship which links a class to its corresponding "Version of…" class, and (ii) the derived_from relationship which describes version derivation hierarchies. This latter relationship is reflexive. The underlying idea of our proposition is to describe both entities and their corresponding versions as indicated in figure 2. Consequently, (i) versions are therefore involved in the process definition, and (ii) a couple (version, entity) is obviously created when the first version of an entity is created. Regarding properties of these "Version of…" classes, we introduce the classical version number, creator name, creation date and status properties (Sciore, 1994).

3.2.2 Merging the Versioning Kit with the Business Process Meta-model

Regarding the process model, we propose to keep versions for only two classes: the Atomic Process

and the Business Process classes. It is indeed interesting to keep changes history for both atomic processes (i.e. tasks) and workflow processes since these changes correspond to changes in the way that business is carried out. At the atomic (task) level, versions describe evolutions in activity realization while at the business process level, versions describe evolutions in work organization (i.e. coordination of activities). We defend the idea that atomic process (task) and business processes versioning is enough to help organizations to face the fast changing environment in which they are involved nowadays.

Regarding the other models, it is necessary to handle versions for the Informational resource class from the informational model, and versions for the Role class from the organizational model. Regarding this latter model, it is also possible to handle versions for the Actor and Organizational Unit classes. However, keeping changes history for these two classes is, in our opinion, quite useless to handle versions of business processes.

Figure 3 below presents the new obtained meta-model in terms of classes and relationships.

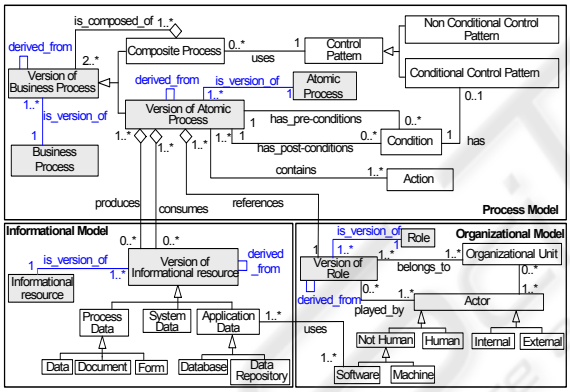

Figure 3: The Versioned Business Process meta-model.

3.3 Example

In order to illustrate the VBP meta-model instantiation, we propose to use the example introduced by (Zhao and Liu, 2007). Because of space limitation, we only focus on the instantiation of the process model of this example.

This example describes a production business process and involves a factory, which owns one production pipeline following the business process shown in figure 4(a). It includes several activities: production scheduling, production using a work centre, quality checking and packaging. In order to increase its productivity, the factory decides to add a new work centre. The business process is then updated as shown in figure 4(b). If one of the two work centres, for instance work centre#1 (Pc#1), has a technical problem and consequently is removed from the process, two solutions are proposed to attempt keeping the production output: fixing unqualified products or using employees for manual production. The business process is then updated as shown in figure 4(c) and 4(d).

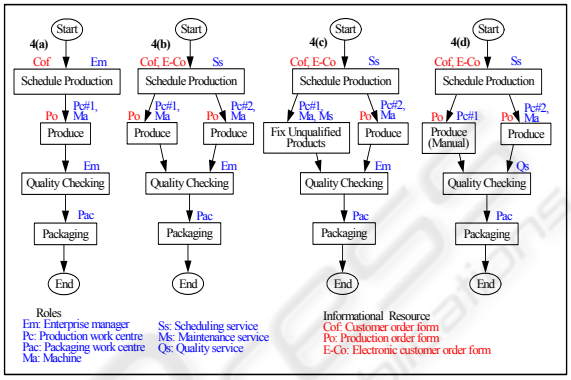

Figure 4: Change in the Production BP.

The solution we provide to model theses derivation hierarchies consists in instantiating the VBP meta-model. The Business Process, Atomic Process, Role and Informational resource "versionable" classes and their "Version of…" corresponding classes are involved in this instantiation, along with the Composite Process and Control Pattern non "versionable" classes. This instantiation is visualized in figure 5.

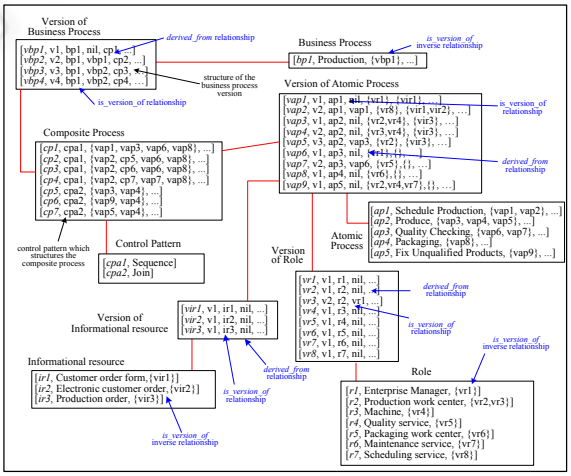

Figure 5: Instantiation of the VBP Meta-Model.

4 OPERATIONS FOR BUSINESS PROCESS VERSIONING

In this section, we introduce a taxonomy of operations for business process versioning. These operations are defined as methods in the "Version of …" classes ("versionable" classes). They correspond to classical operations for versions (Katz, 1990): create, derive, delete, update and froze, but this taxonomy also includes operations for version selection. Of course, create, delete and update are also available for the other classes of the meta-model (non "versionable" classes), but their presentation is out of the scope of the paper.

This section introduces the create, derive, delete, update and froze operations first giving a state chart which indicates when these operations are available, and second detailing the actions they perform according to the classes in which they are defined. Moreover, this section also discusses about version selection, more precisely business process version selection.

4.1 State Chart for Versions

The UML state chart of figure 6 indicates when these operations are available. Some of them are available whatever the state of versions on which they are applied, while others are only available in some cases. In this state chart, each operation is described using the notation Operation:Event/Action whose meaning is "for Operation when Event is triggered then Action is performed".

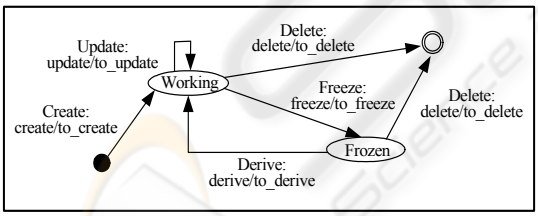

Figure 6: State Chart for Versions of Business Processes.

When the create event is triggered by the a version designer, the to_create action is performed to both create the entity and its corresponding first version. The state of the created version is Working. In this state, the version can be updated (update event and to-update action).

It also can be deleted (delete event and to-delete action): its state is then the final state of the chart. It also can be frozen (freeze event and to-freeze action): its state is then Frozen. Triggering the freeze event, the designer means that the considered version is definitive and does not need additional updates. A frozen version (i.e. a version in a Frozen state) can be deleted or can serve as a basis for the creation of a new version using the derive event and to-derive operation. This new created version has the same value as the version from which it is derived from: its state is Working.

In addition to the previous state chart, these operations require further details. For instance, the Create and Update operations permit to add and delete references to the components of versions. These components change according to the considered type of versions: versions of business processes, versions of atomic processes (tasks), versions of informational resources or versions of roles. Regarding the Derive operation, it can trigger the derivation of versions of its components. The sections below gives additional details for these operations.

4.2 Creating and Updating Versions

Table 1 and 2 below give the semantics of these two operations (Create and Update) according to the classes in which they are defined. The four "Version of…" classes are considered.

Table 1: Creating and Updating Versions of Business Processes and Atomic Processes (task).

Business Process	Atomic Process
1. Change structure	1. Change conditions
1.1. add/delete composite process in the workflow process structure	1.1. add/delete pre-conditions (has-pre-conditions relationship)
1.2. add/delete atomic process in the work-flow process structure	1.2. add/delete post-conditions (has-post-conditions relationship)
2. Change pattern	2. Change action
2.1. choose a pattern for a composite process (use relationship)	2.1. add/delete actions (contains relationship)
	3. Change information
	3.1. add/delete input information (consumes relationship)
	3.2. add/delete output information (produces relationship)
	4. Change role
	4.1. add/delete roles (references relationship)

These two tables indicate that Create and Update operations 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. Moreover, relationships referencing versions may only reference frozen versions (i.e. versions in the Frozen state).

Table 2: Creating and Updating Versions of Informational Resource and Role.

Informational Resource	Role
1. Change software	1. Change actors
1.1. add/delete software (uses relationship)	1.1. add/delete actors (played_by relationship)
2. Change the structure of information resource	2. Change organization
	2.1. add/delete organizational units (belongs_to relationship)

4.3 Derivation of Versions

The Derive operation allows the creation of a new version from an existing frozen one. The new created version is a working version (its state is working). Before being updated, the value of this new created version is the same than the derived one. Moreover, derivation of a version may trigger the derivation of other versions, which are linked to the derived one. Figure 7 below illustrates this derivation propagation.

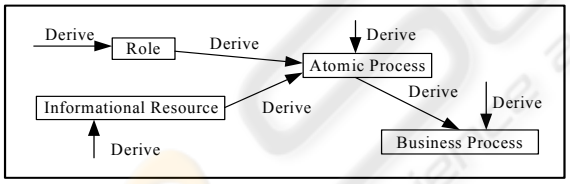

Figure 7: Derivation Propagation.

This propagation is due to the composition relationships existing between Business Process, Atomic Process, Informational Resources and Role classes. Thus, derivation of an Informational Resource version or a Role version triggers the derivation of its corresponding Atomic Process version. In the same way, derivation of an Atomic Process version triggers the derivation of its corresponding Business Process version.

4.4 Selection of Versions

In addition to the previous presented operations, we also propose specific operations for version selection and version hierarchy selection: Select, Slice, Display, among others... Because of space limitation, the paper only details the version selection operation and illustrates its use for business process version selection.

4.4.1 Select Operation

This operation allows the selection of versions. Its syntax is: *Select(Class,Predicate)* where Class is a name of a VBP class containing versions (i.e. a "versionable" class) and Predicate a condition permitting the filtering of versions.

The result of this operation is a set of versions verifying the predicate along with versions and/or objects that are (directly or not) linked to it by a relationship. In other words, the result of the Select operation is a set of instances of the VBP meta-model linked (directly or not) to a version belonging to the "versionable" class on which the Select operation is performed. We call such a group of instances VBP-instances. This notion of VBP-instances corresponds to the notion of Configuration introduced for handling versions in Software Engineering (Kimball and Larson, 1991). It is also close to the notion of Database Version introduced in (Cellary and Jomier, 1990) in order to reduce the complexity of version management in object-oriented databases.

4.4.2 VBP-Trees for representing VBP- Instances

Regarding business process version selection, the result of a Select operation performed to the Version of Business Process class is a set of business process versions verifying the predicate along with instances (versions and/or objects) of the Composite Process, Control Pattern, Version of Atomic Process, Version of Informational Resource and Version of Role classes which are (directly or not) linked to them.

In this case, a VBP-instance corresponds to a business process version along with versions and/or objects linked to it. It can be represented as what we call an VBP-Tree from which we distinguish two kinds of nodes: terminal nodes (leaves) and non terminal nodes. *Terminal nodes* correspond to VBP atomic processes while *non terminal nodes* correspond to VBP composite processes. A non terminal node is described by the following data structure:

- *NodeName*: name of the node (corresponds to the name of the corresponding composite process);

- *CPName*: name of the control pattern used for the composite process;
- *Condition*: optional property associated to conditional control patterns;
- *SetOfNodes*: set of nodes (terminal or non terminal) composing it.

A terminal node is described by the following data structure:

- *NodeName*: name of the node (corresponds to the name of the corresponding atomic process);
- *SetOfActions*: set of actions to perform;
- *PreCondition*: condition associated to the execution of the atomic process; it must be evaluated to true to perform actions of the atomic process that the node represents;
- *PostCondition*: condition associated to the atomic process after execution of actions of the node;
- *ConsumesInformation*: set of informational resources required to perform actions of the node;
- *ProduceInformation*: set of informational resources produced by the performing the actions of the node;
- *PlayedBy*: role defining a set of actors able to perform the actions of the node.

For instance, the VBP-Tree corresponding to the third version of the Production business process (vbp3 i.e. Production.v3) introduced in section 3.3 is visualized in figure 8.

This VBP-Tree illustrates the structure of the considered business process distinguishing terminal nodes (visualized as ellipses) from non terminal nodes (visaulized as rectangles).

In fact, figure 8 only gives a simplified view of the VBP-Tree since nodes are not described in details (according their corresponding structures defined before).

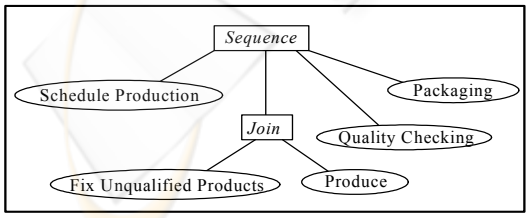

Figure 8: VBP-Tree for Production v3.

The function implementing the mapping from a VBP-instance to a VBP-Tree uses the mapping rules given in Table 3 below.

Table 3: Mapping Rules from VBP-instance to VBP-Tree.

VBP meta-model concepts	VBP-Tree concepts
Instance of Version of Business Process class	VBP-Tree
Instance of Composite Process class	Non Terminal node
Instance of Version of Atomic Process class	Terminal node

Moreover, this function uses a set of functions permitting the handling of processes and nodes:

- IsAtomicProcess(i) indicates if i is an instance of the Version of Atomic Process class;
- BuildTerminalNode(i) returns the corresponding terminal node of an atomic process i taking into account the relationships flowing from i (has_pre-conditions, …);
- BuildNonTerminalNode(i) returns the corresponding non terminal node of the composite process i using the relationship flowing from i (uses);
- AddNode(n, tr) adds the node n to a VBP-Tree tr.

This function is the following.

```
Function BuildVBP-Tree (i:VBP-
                    Instance):VBP-Tree
Local n:Node
Begin
   If IsAtomicProcess(i) Then
      n = BuildTerminalNode(i)
      BuildVBP-Tree = AddNode(n, tr)
   Else
      -- i is a composite process
      n = BuildNonTerminalNode(i)
      BuildVBP-Tree = AddNode(n, tr)
      For Each c ∈ IsComposedOf(i)
         BuildVBP-Tree =
                  BuildVBP-Tree(c)
      Next c
   End If
End
```

5 FORMALIZING BUSINESS PROCESS VERSIONS: FROM VBP-TREE TO PNO

Representing versions of business processes as VBP-Tree is not sufficient to visualize and formalize the semantics of the modeled versions of business processes. To compensate this drawback, we propose to use a Petri net-based formalism, namely Petri Nets with Objects (PNO) (Sibertin, 1985) for

workflow process version visualization and formalization.

This section first presents the PNO formalism and gives the reasons of the choice of this language for workflow process versions. Then, this section explains the mapping from a VBP-Tree onto a PNO.

5.1 Petri-net with Objects

5.1.1 What are PNO?

Petri Nets with Objects (PNOs) (Sibertin, 1985) are a formalism combining coherently Petri nets (PN) technology and the Object-Oriented (OO) approach. While PN are very suitable to express the dynamic behavior of a system, the OO approach permits the modeling and the structuring of its active (actor) and passive (information) entities. In a conventional PN, tokens are atomic, whereas they are objects in a PNO. As any PN, a PNO is made up of places, arcs and transitions, but in a PNO, they are labeled with inscriptions referring to the handled objects. More precisely, a PNO features the following additional characteristics:

- Places are typed. The type of a place is a (list of) type of an (list of) object(s). A token is a value matching the type of a place such as a (list of) constant (e.g. 2 or 'hello'), an instance of an object class, or a reference towards such an instance. The value of a place is a set of tokens it contains;
- Arcs are labeled with parameters. Each arc is labeled with a (list of) variable(s) of the same type, as the place the arc is connected to. The variables on the arcs surrounding a transition serve as formal parameters of that transition and define the flow of tokens from input to output places. Arcs from places to a transition determine the possible condition of the transition: a transition may occur (or is possible) if there exists a binding of its input variables with tokens lying in its input places;
- Each transition is a complex structure made up of three components: a precondition, one (or several) action(s) and emission rules. A transition may be guarded by a precondition, i.e. a side-effect free Boolean expression involving input variables. In this case, the transition is only permitted by a binding if this binding evaluates the precondition to be true. Passing through a transition depends on the precondition, on the location of tokens and also on their value. A transition also includes one or several actions, which consists of a piece of code in which transitions' variables may appear and object methods may be invoked. These actions are executed at each occurrence of the transition and they process the values of tokens. Finally, a transition may include a set of emission rules i.e. side-effect free Boolean expressions that determine the output arcs that are actually activated after the execution of the action.

5.1.2 Motivations for using PNO

Petri nets are widely used for workflow specification (Aalst, 1998). Several good reasons justify their use:

- An appropriate expressive power that permits the description of the different tasks involved in a workflow process and their coordination;
- A graphical representation that eases the workflow process specification;
- An operational semantics making an easy mapping from specification to implementation possible;
- Theoretical foundations enabling analysis and validation of behavioral properties and simulation facilities.

Unfortunately, conventional Petri nets focus on the process definition and do not perfectly capture the organizational and the informational dimensions of business processes. As mentioned before, PNO extend Petri nets by integrating high-level data structures represented as objects, and, therefore provide the possibility to integrate in a coherent way the two missing dimensions. Thus, using PNO, actors/roles of the organizational model are directly represented as objects and they may be invoked through methods in the action part of a transition. In the same way, data and documents (from the informational model) are also represented by objects flowing in the PNOs and transformed by transitions.

Consequently, we use PNO as a graphical tool to visualize versions of business processes, and as a formal tool to define executable specifications in order to analyze, simulate, check and validate workflow process versions.

5.2 From VBP-Trees to PNOs

Table 4 and figure 9 give the mapping rules in order to obtain, from a VBP-Tree, i.e. a VBP-instance, the corresponding Petri net with objects. We distinguish mapping rules for concepts from mapping rules for control patterns. Table 4 introduces mapping rules for concepts while figure 9 presents mapping rules for control patterns.

Table 4: Mapping Rules for Concepts.

VBP-Tree concepts	PNO concepts
Name of a Terminal node	Name of a transition
SetofActions of a Terminal node	Actions of a transition
PreCondition of a Terminal node	Pre-condition of a transition
PostCondition of a Terminal node	Emission rule of a transition
ConsumesInformation of a Terminal node	Begin place of a transition
ProduceInformation of a Terminal node	End place of a transition
PlayedBy of a Terminal node	Begin place of a transition representing a role

Figure 9 below details how modeled control patterns are represented using PNOs. In this figure, P, P1 and P2 correspond to business processes while condition corresponds to a condition used in conditional control patterns. Finally, Empty is a transition for which no actions are executed.

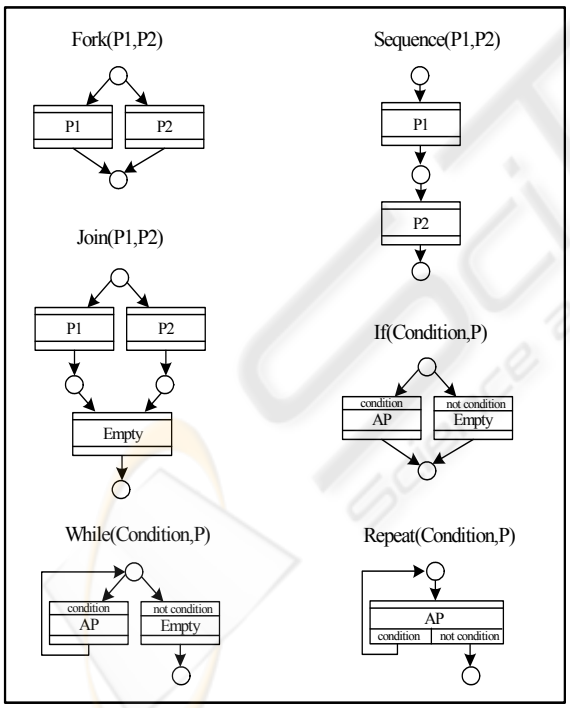

Figure 9: Mapping Rules for Control Patterns.

We also provide a function for building a PNO from a VBP-Tree. This function uses mapping rules presented in table 4 and figure 9 for defining transitions of the PNO and their coordination. Moreover, this function uses a set of functions permitting the handling of a tree:

- ListOfChildren(n) returns the children of a node n (non-terminal or terminal nodes);
- ListOfLeaves(t) returns the terminal nodes (leaves) of a tree t;

and also a set of functions for building transitions and their coordination:

- BuildTransition returns the corresponding transition to a node using mapping rules defined in table 4;
- BuildPattern uses mapping rules defined in figure 9 to return the corresponding PNO according to the control pattern specified in a node;
- AddTransition (tr,PNO): add a transition tr to a PNO;
- AddPattern (pa,PNO): adds a pattern pa to a PNO.

This function is the following.

```
Function BuildPNO (n:Node):PNO
Local c:Node; tr:Transition
Global t:VBP-Tree
Begin
   If n ∈ ListOfLeaves(t) Then
      tr = BuildTransition(n)
      BuildPNO =
         AddTransition(tr,BuildPNO)
   Else
      -- n is a non terminal node
      pa = BuildPattern(n)
      BuildPNO =
         AddPattern(pa, BuildPNO)
      For Each c ∈ ListOfChildren(n)
         BuildPNO = BuildPNO(c)
      Next c
   End If
End
```

6 CONCLUSIONS

As mentioned in the introduction, the problem stated as "how to support dynamic change of business process" has already been addressed in the workflow context. We distinguish two main approaches to deal with this problem.

Concerning the adaptive-based approach, relevant works in this area propose solutions to deal with workflow schemas changes, adaptation and migration of their corresponding instances. (Casatia and al, 1996) presents a workflow modification language that supports updates of workflow

schemas. It also defines a set of evolution policies that a workflow administrator can adopt to manage instances of updated workflow schemas in order to migrate (or not) them as instances of the new schema. Three mains policies are defined: abort, flush and progressive. (Kammer and al, 2000) investigates exception handling as a way to support dynamic change to workflow process schemas. Consequently, it introduces a taxonomy for exceptions and defines functionalities that Workflow Management Systems must have in order to be able to deal with these exceptions. The ADEPTflex project (Reichert and Dadam, 1998), (Rinderle and al, 2004) extensively studies process schema evolution. This work formally defines change operations for both process schemas and workflow instances as well as related migration policies in handling potential conflicts. We can also mention van der Aalst's work to address dynamic change of workflow (Aalst, 2001). This work uses a generic process model to describe a family of variants of a same workflow process and the notion of inheritance is used to link these different variants. In the same vein, (Adams and al, 2006) proposes, for dealing with dynamic evolution in workflows, to use accepted ideas of how people actually work to define sets of worklets (i.e. processes) and a stategy for runtime selection of a specific worklet.

However, none of these works mention the notion of workflow versions. Consequently, none of them enables several different schemas of a same workflow process to conjointly exist.

Relevant works from the version-based approach allow to different instances of a same workflow process to own different schemas. Two main contributions are relevant from this approach. First, (Kradolfer and Geppert, 1999) have proposed to deal with dynamic workflow evolution, i.e. modification of workflow process schemas in the presence of active workflow process instances, introducing versions of workflow process schemas. This work has defined a set of operations for workflow process schema modification and a strategy for migration of workflow process instances. Second, and more recently, (Zhao and Liu, 2007) have also defended the advantages of a version-based approach to face business process evolution. More precisely, this work proposes to model versions of workflow process schemas using a graph. It also presents a set of operations enabling to update this graph and defines two strategies to extract versions of workflow process schemas from this graph.

However, these two works only consider the process model of workflow. They do not integrate the two other dimensions of workflow, that are the informational and the organizational dimensions.

Consequently, this paper revisits the dynamic change of business process issue following a version-based approach and considering the organizational, informational and process models of business processes. More precisely, it introduces:

- A meta-model for designing versions of business processes;
- A taxonomy of operations for business process version management;
- A formalization and a visualization, using Petri net with Objects, of versions of business processes, designed with the previous meta-model.

Our solution has the following advantages:

- It permits a comprehensive modeling of business processes considering the three dimensions of business processes;
- The VBP meta-model is simple: it only integrates core concepts for both business process modeling and business process versioning (our versioning kit is very simple),
- Dynamics aspects of business process version management are investigated in depth according to the state of the art for versions in databases;
- It provides rules and algorithms to derive modeled versions of business processes onto Petri net with objects specifications.

As future work, we have planed to implement the VBP meta-model in order to model version of business processes and to derive versions of business processes specified using BPEL.

REFERENCES

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

Andonoff, E., Hubert, G., Le Parc, A., Zurfluh, G., 1996. Integrating Versions in the OMT Models. *Int. Conference on the Entity Relationship Approach*, Cottbus, Germany, pp. 472–487.

Casati, F., Ceri, S., Pernici, B., Pozzi, G., 1995. Conceptual Modelling of Workflows. *Int. Conference on the Entity Relationship Approach*, Goald Cost, Autralia, pp. 341–354.

Casati, F., Ceri, S., Pernici, B., Pozzi, G., 1996. Workflow Evolution. *Int. Conference on the Entity Relationship Approach*, Cottbus, Germany, pp. 438–455.

Cellary, W., Jomier, G., 1990. Consistency of Versions in Object-Oriented Databases. *Int. Conference on Very Large DataBases*, Brisbane, Australia, pp. 432–441.

Chen, I., Markowitz, V., Letovsky, , Li, P., Fasman, K., 1996. Version Management for Scientific Databases. *Int. Conference On Extended Database Technology*, Avignon, France, pp. 289–303.

Chou, H.T., Kim, W., 1986. A Unifying Framework for Version Control in a CAD Environment. Int. Conference on Very Large DataBases, Kyoto, Japan, pp. 336–344.

Fernandes, S., Cachopo, J., Silva, R., 2004. Version Supporting Evolution in Workflow Definition Language. *Int. Conference on Current Trends in Theory and Practice of Computer Science*, Merin, Czech Republic, pp. 208–217.

Halliday, J., Shrivastava, SK., Wheater, SM., 2001. Flexible Workflow Management in the OPEN-flow System. *Int. Conference on Enterprise Distributed Object Computing*, Seattle, Washington, USA, pp. 82–92.

Heinis. T., Pautasso. C., Alonso G., 2005. The JOpera Autonomic Workflow Engine, *the 2nd International Conference on Autonomic Computing (ICAC-05),* Seattle, Washington, June 2005.

Kammer, P., Bolcer, G., Taylor, R., Bergman, M., 2000. Techniques for supporting Dynamic and Adaptive Workflow. *Int. Journal on Computer Supported Cooperative Work*, 9(3/4), pp. 269–292.

Katz. R., 1990. Towards a Unified Framework for Version Modelling in Engineering Databases. *Int. Journal on Computing Surveys*, 22(4), pp. 375–408.

Kimball, J., Larson, A., 1991. Epochs: Configuration Schema, and Version Cursors in the KBSA Framework CCM Model. *Int. Workshop on Software Configuration Management*, Trondheim, Norway, pp. 33–42.

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

Leymann, F., Roller, D., 1999. *Production Workflow: Concepts and Techniques*. Prentice-Hall Press.

Manolescu, DA., 2001. *Micro-Workflow: A Workflow Architecture Supporting Compositional Object-Oriented Development*. PhD Thesis, University of Illinois.

Mohan, C., Alonso, G., Gunthor R., Kamath, M., 1995. Exotica: A Research Perspective on Workflow Management Systems. *IEEE Data Engineering Bulletin*, 18(1), pp. 19–26.

Reichert, M., Dadam, P., 1998. ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control. *Int. Journal on Intelligent Information Systems*, 10(2), pp. 93–129.

Rinderle, S., Reichert, M., Dadam, P., 2004. Disjoint and Overlapping Process Changes: Challenges, Solutions and Applications. *Int. Conference on Cooperative Information Systems*, Agia Napa, Cyprus, pp.101–120.

Roddick, J., Craske, N., Richards, T., 1993. A Taxonomy for Schema Versioning based on the Relational and Entity Relationship Models. *Int. Conf. on the Entity Relationship Approach*, Arlington, Texas, USA, pp. 137–148.

Rosemann, M., zur Muehlen, M., 1998. Evaluation of Workflow Management Systems: a Meta-model Approach. *Australian Journal of Information Systems*, 6(1), pp. 103–116.

Sciore, E., 1994. Versioning and Configuration Management in Object-Oriented Databases. *Int. Journal on Very Large Databases*, 3(1), pp. 77–106.

Sibertin-Blanc, C., 1985. High Level Petri Nets with Data Structure. *Int. Workshop on Petri Nets and Applications*, Espoo, Finland.

Smith, H., Fingar, P., 2003. Business Process Management - The Third Wave. Megan-Kiffer Press.

van der Aalst, W., 1998. The application of Petri Nets to Workflow Management. *Int. Journal on Circuits, Systems and Computers*, 8(1), pp. 21–66.

van der Aalst, W., 1999. Inter-Organizational Workflows: An Approach Based on Message Sequence Charts and Petri Nets. *Int. Journal on Systems Analysis, Modeling and Simulation*, 34(3), pp. 335–367.

van der Aalst, W., 2001. How to Handle Dynamic Change and Capture Management Information: an Approach based on Generic Workflow Models. *Int. Journal of Computer Systems, Science, and Engineering*, 16(5), pp. 295–318.

van der Aalst, W., Aldred, L., Dumas, M., ter Hofstede, A., 2004. Design and Implementation of the YAWL System. *Int. Conference on Advanced Information Systems Engineering*, Riga, Latvia, pp. 142–159.

van der Aalst, W., Benatallah, B., Casati, F., Curbera, F., Verberk, E., 2007. Business Process Management: Where Business Processes and Web Services Meet. *Int. Journal on Data and Knowledge Engineering*, 61(1), pp. 1–5.

van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A., 2003-b. Workflow Patterns. *Int Journal on Distributed and Parallel Databases*, 14(1), pp. 5–51.

van der Aalst, W., Ter Hofstede, A., Weske, M., 2003-a. Business Process Management: A Survey. *Int. Conference on Business Process Management*, Eindhoven, The Nederlands, pp. 1–12.

Vossen, G., Weske, M., 1999. The WASA2 Object-Oriented Workflow Management System. *Int. Conference on Management of Data*, Philadelphia, Pennsylvania, USA, pp. 587–589.

Zhao, X., Liu, C., 2007. Version Management in the Business Change Context. *Int. Conf. Business Process Management*, Brisbane, Australia, pp. 198–213.