A Framework to Support the Assignment of the Active Structure and Behavior in Business Process Modeling

Rômulo H. Arpini and João Paulo A. Almeida

Ontology and Conceptual Modeling Research Group (NEMO),
Federal University of Espírito Santo (UFES)
Vitória, ES, Brazil
romuloarpini@gmail.com, jpalmeida@ieee.org

Abstract—Despite the importance of the relations between the organizational domain and the business process domain, many of the current enterprise architecture and business process modeling approaches lack support for the expressiveness of a number of important active structure allocation scenarios. This paper aims to overcome these limitations by proposing a framework for active structure assignment that can be applied to existing enterprise architecture and business process modeling approaches. This framework enriches the expressiveness of existing techniques and supports the definition of precise active structure assignments. It is designed such that it should be applicable to a number of enterprise architecture and business process modeling languages, i.e., one should be able to use and apply different (enterprise and business process) modeling languages to the framework with minor changes. We show the application of this general framework to BPMN.

Business Process Modeling, Organizational Modeling, Behavior, Active Structure, Assignment Framework, BPMN.

I. INTRODUCTION

Business process modeling addresses the way enterprises organize their work and resources showing how they contribute to fulfilling the enterprise's strategies [11]. While the process domain focuses on "how" the business process activities are structured and performed, the organizational structure domain focuses on "who" performs these activities, i.e., which kinds of entities in an organization are capable of performing work.

Given the strong connection between the organizational behavior and organizational resources, any comprehensive enterprise modeling technique should explicitly establish the relations between the modeling elements that represent organizational behavior, called here *behavioral elements,* and those used to represent the organizational resources (organizational actors) involved in these activities, called here *active structure elements.*

Properly representing the assignment of active structure elements and behavioral elements at design time is important to allow the comprehensive analysis of business process and enterprise architectures, e.g., from the perspectives of accountability, authorization, and responsibility of organizational actors with respect to the activities they execute. The assignment of active structure and behavioral elements also supports business process enactment and later phases of process management, such as monitoring and evaluation [6].

Although several techniques (such as ArchiMate, ARIS, DoDAF, XPDL, UML activity diagram and BPMN) offer some support for establishing these relations, the levels of support and expressiveness they offer vary significantly [1]. Several of these, such as BPMN and UML activity diagrams are considered to offer simplistic support (as seen in [2]), failing to provide required expressiveness with respect to active structure assignment (e.g., as evidenced by a low coverage of Workflow Resource Patterns ([10], [13]). Further, approaches based solely on business process models (such as BPMN and XPDL), fail to identify relations with rich organizational structure models and are thus unable to express active structure assignment based on organizational relations. Further, the semantics of active structure assignment is poorly defined in many of these techniques, leading to ambiguous or imprecise models. For example, in BPMN, while "Lanes" have often been used to specify the assignment of active structure elements to process fragments, such interpretation is informal and not defined in the language semantics.

In this paper we intend to address these limitations by proposing a framework for active structure assignment for enterprise architecture and business process modeling approaches. This framework should enrich the expressiveness of existing techniques and support the definition of precise active structure assignments.

We offer two main contributions: a generic assignment framework applicable to a number of enterprise architecture and business process modeling languages; and, an application of this framework to BPMN, enriching its capabilities to express active structure assignment.

This paper is further structured as follows: section II presents the expressiveness requirements for the framework, which are based on the Workflow Resource Patterns [9] and discusses the level of support for these patterns in the existing enterprise and business process modeling approaches; section III presents the assignment framework, including the proposed generic assignment metamodel; section IV discusses the application of the framework to BPMN, binding the generic metamodels to the BPMN metamodel and showing a usage example. Section V discusses related work and, finally, section VI presents concluding remarks and outlines topics for future work.

II. SUPPORT FOR WORKFLOW RESOURCE PATTERNS

The Workflow Resource Patterns form a comprehensive catalog of common types of human resource allocation constraints [9]. They were developed by the Workflow Patterns Initiative, with the goal of providing a conceptual basis for process technology. The Workflow Resource Patterns capture the various ways in which resources are represented and utilized in process technologies and have been used to compare a number of commercially available workflow management systems and business process modeling languages. As discussed in [9],

workflow patterns can be used as requirements of expressiveness for process-aware technologies, and this is role they will serve with respect to our framework which is presented in section III.

We focus here on the core set of patterns that deals with task allocation to human resources, and in particular those that may be used at process definition time to restrict the range of human resources that can undertake particular work items (task instances). They are called the 'creation patterns'.

The following 'creation patterns' have been defined in [9].

The *Direct Distribution pattern* captures the ability to determine at design-time the specific resources to which the work items will be distributed. The *Role-based Distribution pattern* captures the ability to specify that a work item is to be performed by resources that fulfill a specific role. For instance, we may want to specify that the task 'Review technical report' is to be performed by a manager (any manager, not a specific one). The *Deferred Distribution pattern* captures the ability to specify that the identification of the resource(s) that will be distributed to instances of a task will be deferred until runtime (and thus not specified at design-time). The *Authorization pattern* captures the ability to specify privileges that a resource have regarding the execution of a work item, for example, defining whether a resource is authorized to execute or delegate a work item. The *Separation of Duties pattern* captures the ability to specify that two work items must be performed by different resources. For instance, if we have a task that whose result is a report that will be audited by a following task, we may want to guarantee that the two tasks will be performed by different resources. The *Case Handling pattern* is a specific approach based on the premise that all the tasks on a process or sub-process are related and must be performed by the same resource. The *Retain Familiar pattern* captures the ability to specify that the resource who will undertake a work item is the same that undertook the previous one. It is particularly useful when there are sequential tasks and also may help minimizing the switch time. It is a more flexible version of the *Case Handlin*g pattern. The *Capability-Based Distribution pattern* captures the ability to allocate resources to work items based on specific capabilities they must have, so there must exists some mechanism that allows to specify resource's capabilities and to use these when deciding the performer of a task. The *History-Based Distribution pattern* captures the ability to distribute tasks to the resources based on the history of execution they have on the tasks. The operationalization of this pattern requires information about previous executions. The *Organizational Distribution pattern* captures the ability to distribute tasks to the resources based on their positions within an organization and their relations with other resources. Therefore, the process technology that supports this pattern must assume an organizational model with positions and some relationships between them. The *Automatic Execution pattern* captures the ability to perform a task without needing to be allocated to a specific human resource. Therefore, there must exist some way to declare a task to be automatic and it will be performed without any human interference.

We have reviewed ArchiMate, ARIS, DoDAF, XPDL, UML activity diagram and BPMN for their support for these patterns. We can observe that Direct Distribution, Role-Based Distribution and Automatic Execution are directly supported by all of them. Deferred Distribution is considered to be partially supported by all of them, because they allow the modeler to refrain from specifying the performer of the behaviors. We consider this kind of support partial, since full support would require not only to defer identification of a resource but also would require some run-time mechanism for resource identification [9]. Authorization is not supported by any of them, because they consider the assigned performer to be the one that will execute a behavior, not discussing other range of privileges that resources may have in regards to behavioral elements. Separation of Duties, Case Handling and Retain Familiar are not supported by any of them, because they ignore the interdependences between performers of behavioral elements. History-Based Distribution is also not supported by any of them as the approaches cover mainly aspects of design-time. Capability-Based Distribution is partially supported in DoDAF, UML 2.0 Activity Diagram and BPMN, because they offer some kind of mechanism to specify properties that resources should have. However, because they do not offer a full-fledged mechanism to allow the specification of resource properties and their types and to use that in the assignment, we consider the support for this pattern "partial". Finally, Organizational Distribution is partially supported in ArchiMate, ARIS, UML and BPMN because they allow one to define a basic organizational structure and use its hierarchy to define the assignment, but they do not offer the possibility to use organizational relationships when defining the assignment.

III. ASSIGNMENT FRAMEWORK

In this section, we present the assignment framework, which is intended to address the limitations of the various techniques discussed in section II. The framework is composed of a number of metamodels, which together enable the expression of the assignment of active structure and behavior.

A. Architecture

Figure 1 provides a general overview of the Assignment Framework architecture. The middle layer shows the core of the assignment framework and aims at covering the range of assignments to be expressed. It includes an Assignment metamodel which is integrated with an external Behavioural metamodel, an Occurrence metamodel and Organizational metamodel The metamodels in this middle layer provide the metaclasses and meta-associations which will define assignments as well as the elements that may be referred to in the various kinds of assignments. The external Behavioral metamodel is a placeholder for a specific metamodel of the technique being extended by the framework (e.g., BPMN).

The top layer shows the Ecore metametamodel, which is instantiated by all the metamodels in the middle layer, represented by the *instanceOf* relationships. The OCLEcore package is built-in feature of the Eclipse Modeling Framework (EMF) that allows a designer to use OCL for queries and constraints on the instantiating metamodels. These queries will be used in the run-time environment to be able to satisfy the expression-based requirements stated in the previous section. The bottom layer shows how the model-based runtime environment works when the framework is applied. Assignment, Behavioral, Occurrence and Organizational models populate an organizational repository. OCL queries referencing the models will be evaluated as required to satisfy particular assignments in the Assignment model.

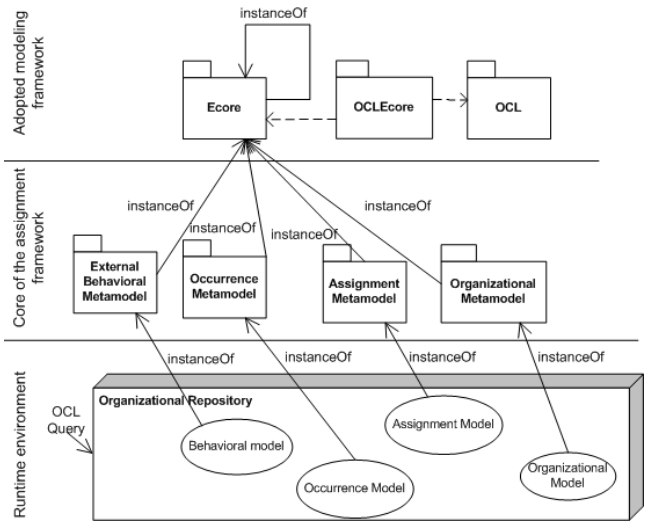

Figure 1 - Framework Architecture Overview

We assume the Behavioral model is defined at design-time, and focus also on the design-time specification of active structure assignment (although active structure assignment may refer to runtime information as we will see in the following). An Organizational model is defined and modified at design-time and run-time in order to accommodate a changing organizational structure. An Occurrence model deals only with run-time information, getting populated automatically by a process-aware application or a process enactment environment (such as a workflow system or business process management engine).

The framework is designed such that it can be applied as a lightweight extension to existing technologies (thus not involving the modification of existing metamodels). As a consequence, the assignment metamodel is built to be as loosely coupled as possible.

Figure 2 shows the basic relationships between the metamodels as well as the levels of modeling that they deal with. As we can see, the behavioral metamodel covers the behavioral aspects at type level, defining the types of processes and activities that will be instantiated at process run-time. The occurrence metamodel is considered to be at instance level, as it represents actual occurrences (instances) of types of processes and activities defined in a behavioral model. Suppose we have an activity called "Send report" defined in a behavioral model (at type level). The records of execution(s) of this activity are represented at instance level and are covered in the occurrence metamodel.

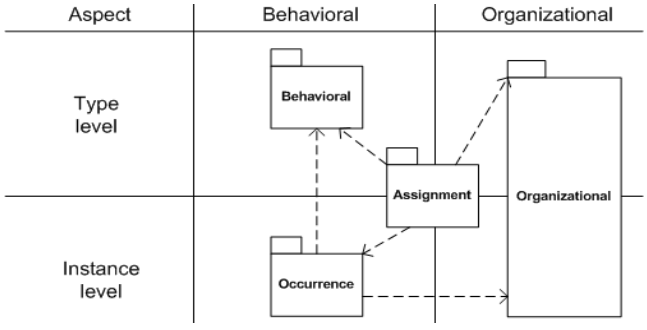

Figure 2 - The different metamodeling levels and their dependencies

The organizational metamodel is considered to cover both levels, as seen in many modeling techniques, such as ARIS. For instance, in an organizational model there will be type level elements, for instance positions like 'Engineer', 'Manager' and instance level elements, like the humans that work at the organization being modeled, i.e., 'John', 'Paul', etc.

The occurrence metamodel depends on the behavioral metamodel to determine the processes or activities in the behavioral model that are instantiated in particular occurrences. It also depends on the organizational metamodel because it refers to the particular individuals that performed the behaviors. The Assignment metamodel depends on all the other metamodels in the framework because it needs to be able to refer to specific activities in the behavioral model, possible past occurrences of activities in the occurrence model and resources in the organizational model. We will see how these dependencies are used in assignments in the subsequent sections.

The behavioral model is independent of the other metamodels, and is only referred to by other metamodels. This is an important characteristic of the approach as it enables us to employ previously existing behavioral metamodels (such as, e.g., the BPMN metamodel) without alteration. In order to cope with different behavioral metamodels, the relation between the Assignment metamodel and the behavioral metamodel is parameterized (this is discussed further in sections III.C and III.D employing an abstraction of the various behavioral metamodels and the generic capabilities of EMF.)

B. Organizational Metamodel

Many of the modeling techniques we have considered in section 2 include elements to model organizational elements. Nevertheless, there is a wide range of differences in the coverage of concepts, ranging from very simplistic (e.g., BPMN, with no organizational relations) to sophisticated (e.g., ARIS, with various kinds of relations). Unfortunately, there is no standard or reference model developed for this domain yet (although there were some efforts, such as, e.g., an Organizational Structure Metamodel effort of the Object Management Group [7]). Thus, we have consolidated many of these elements into an abstract organizational metamodel (Figure 3), which provides us with basic elements required for organizational-based assignments.

The organizational metamodel has the OrganizationalModel metaclass, which will serve as the container for all the elements that comprise a specific organizational model. These elements are what we call the ActiveStructureElements, the topmost abstract class that subsumes almost all the concepts defined in the metamodel. An ActiveStructureElement is further specialized into two classes: ActiveStructureIndividual, which is the topmost class covering active structure elements at the instance level and ActiveStructureClassifier, which is the topmost class covering active structure elements at the type level.

An ActiveStructureIndividual may be an ActiveStructureAgent, which in its turn may be an OrganizationalUnit, a Group or a Human. An ActiveStructureAgent may have Attributes that characterizes them. For instance, a Human named 'João Paulo' may have an Attribute 'experience as professor', with its value set to 10 (years) in a given time. An ActiveStructureRelator represents a relation between two or more ActiveStructureAgents. For instance, we may have an ActiveStructureRelator 'SupervisionJoaoPauloRomulo' that relates a specific human named 'Joao Paulo' to another specific human named 'Romulo'.

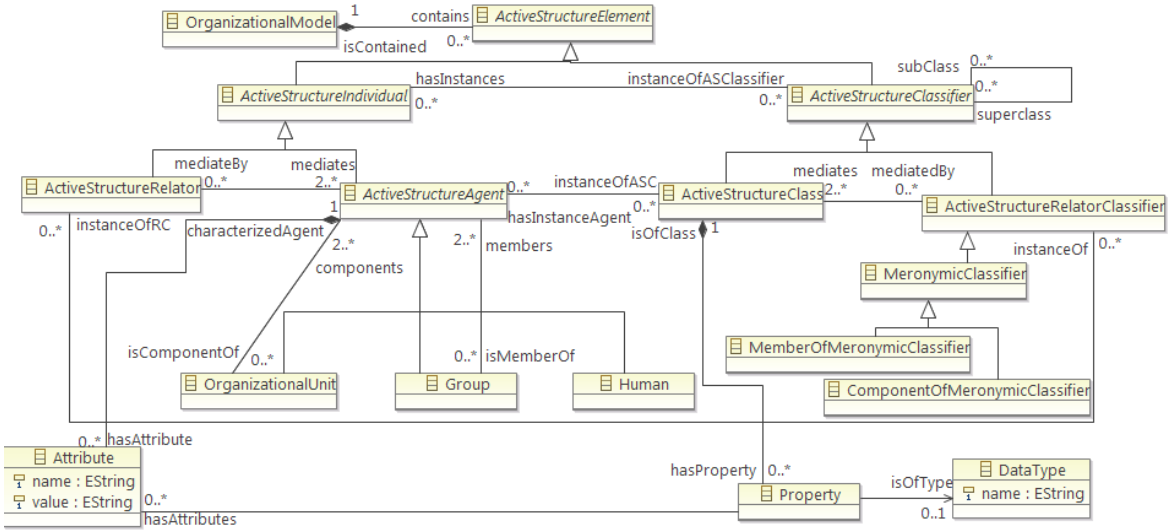

Figure 3 - Organizational Metamodel

This relationship between two or more ActiveStructureAgents, which we call mediates, is ordered, because each part being mediated has a different role in the relationship. In the previous example, for instance, 'Joao Paulo' is the 'Supervisor' and Romulo is the 'student being supervised'.

An ActiveStructureClassifier may be an ActiveStructureClass or an ActiveStructureRelatorClassifier. An ActiveStructureClass is the main element for being the one that will represent the various types that are defined within an organization and they may have Properties, which are the types of attributes that agents may have. The isOfType relationship to DataType will represent the specific data type of Property. An ActiveStructureRelatorClassifier represents a relation between two or more ActiveStructureClasses. For instance, consider a 'Supervision' ActiveStructureRelatorClassifier, which mediates the ActiveStructureClasses 'Professor' and 'Master Student'. An ActiveStructureRelatorClassifier may be further specialized into a MeronymicClassifier, which in its turn may be further classified into a MemberOfMeronymicClassifier and ComponentOfMeronymicClassifier relator classifiers to represent the different categories of whole-part relations.

C. Assumptions on a Behavioral Metamodel

Our framework assumes that a behavioral metamodel includes elements that represent the units of behavior that will be assigned to perform some work. In the reviewed techniques, these elements are often called *Activities*, *Tasks* or *Processes*. In some of those techniques, *Activity* is a more general concept while *Task* is a specialized *Activity* which represents the most refined unit of work, as is the case in XPDL and BPMN. Further, in some of the reviewed techniques, *Process* is considered a special unit of behavior that may include other units of behavior, as is the case in XPDL and BPMN. A behavioral metamodel may or may not consider *Activities*, *Tasks* and *Processes* as specializations of a more abstract metaclass. For example, XPDL and BPMN do not have such a more abstract metaclass, while ArchiMate includes only the more abstract *Business Processes*.

Given the possible variations in behavioral metamodels, in order to cope with most of the modeling techniques, the assignment metamodel must be able to assign active structure elements to any of the elements that represent units of behavior. We assume thus that the behavioral metamodel may have two separate types of behavior elements (which we call conveniently activity and process) or a single type of behavior element (either an activity or a process).

D. Behavioral Occurrence Metamodel

Since we need to be able to specify assignments based on the history of execution of activities, we are required to refer to past executions. The behavioral occurrence metamodel was created to define the structure of information on these past executions and its main elements are shown in Figure 4.

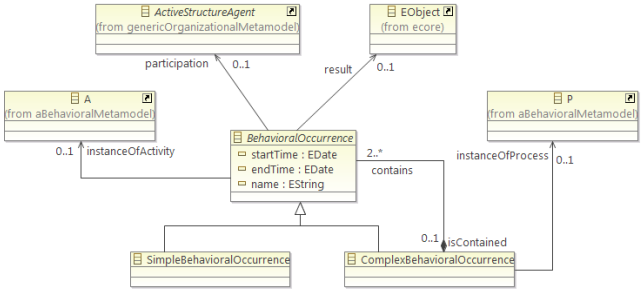

Figure 4 - The Behavioral Occurrence Metamodel

The main element of the metamodel is the BehavioralOccurrence abstract metaclass, which represents the actual occurrence of some behavior. A BehavioralOccurrence has a number of relationships to metaclasses of other metamodels. The instanceOfActivity relationship shows that a BehavioralOccurrence may instantiate an "activity" concept from some behavioral metamodel, meaning that the BehavioralOccurrence is an actual performance (instance level) of the referred "activity" (type level). In order to avoid the direct integration of an existing metamodel of a process technology, we use EMF generic capabilities to parameterize the occurrence metamodel. Thus, the "A" metaclass that is being referred to is a parameter of the metamodel and will be replaced when this metamodel is instantiated by a metaclass of an existing behavioral metamodel of a specific process technology (e.g. BPMN) with the similar behavioral concept of an activity (e.g. Activity in BPMN). The participation relationship shows that a BehavioralOccurrence may have the participation of an

ActiveStructureAgent. Lastly, a result relationship has been included, to represent the result of some piece of behavior (using the generic metaclass EObject.) This will be used in a special kind of assignment which refers to the results of previous occurrences.

BehaviouralOcurrences are further specialized into SimpleBehaviourOcurrence and ComplexBehaviouralOcurrences. A SimpleBehavioralOccurrence represents the execution of a behavior that may not be further divided in finer grained behaviors (often called 'tasks' or 'atomic activities' in process modeling techniques). The instanceOfActivity relationship of a SimpleBehavioralOccurrence must refer to an activity of the behavioral metamodel that is atomic, i.e., that is not further subdivided. A ComplexBehavioralOccurrence is composed of two or more BehavioralOccurrences and represents a single execution of a behavior that may be further decomposed into finer grained behaviors (often represented by processes and subprocesses in process modeling techniques). A ComplexBehavioralOccurrence may also have a relationship to a process concept of a behavioral metamodel, which is reflected in the "P" parameter of the instanceOfProcess meta-association. Thus, a ComplexBehavior-alOcurrence may refer to either a (non-atomic) activity through the instanceOfActivity relationship or refer to a process through the relationship instanceOfProcess.

E. Assignment Metamodel

Figure 5 shows the metaclasses and the main attributes of the Assignment metamodel. An AssignmentModel represents the specification of assignments, including thus at least one Assignment, which captures the relation between the behavioral and organizational models.

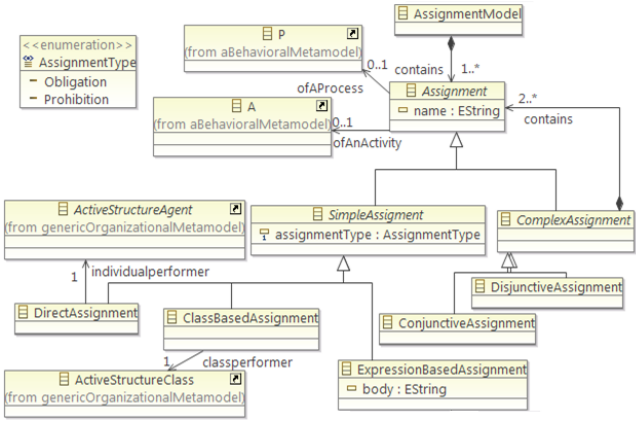

Figure 5 - Assignment Metamodel

Assignment is the top-level abstract metaclass is further specialized into SimpleAssignment and ComplexAssignment. There must be at most one Assignment for each behavior present in the behavioral model, which may be an instance of an "A" or "P" metaclass. Similarly to the behavioral occurrence metamodel, the "A" and "P" metaclasses are parameters of this metamodel and will be replaced when this metamodel is instantiated by metaclasses that represent the different types of behavior elements in the behavioral metamodel ("A" stands for activity and "P" stands for process).

SimpleAssignment is an abstract metaclass that is further specialized into the various different types of Assignments, which we discuss in the following sections. In order to address the pattern concerning Authorization, all SimpleAssignments must have an AssignmentType, which may be one of the following: *Obligation*, stating that the active structure element(s) referred to in the assignment must perform the referred behavioral element; or, *Prohibition*, stating that the active structure element(s) referred to in the assignment may not perform the referred behavioral element (*Permission* is the default assignment type in the absence of assignments for a behavior element, following the motto "everything that is not explicitly prohibited is permitted". We chose this approach to avoid forcing the modeler to explicitly state the entities that would be permitted to perform the behaviors, which would often lead to models that are unnecessarily verbose.)

A DirectAssignment determines at design-time the specific agent (OrganizationalUnit, Group or Human) involved in the assignment. A DirectAssignment of type Obligation determines at design-time the agent who must execute all instances of the referred behavior element, either a process or an activity. This is the only type of assignment for which we know at design-time what real-world entity will perform all instances of the referred behavioral element, and thus is an assignment with the highest level of determinism. For example, if we would like a specific Human to be the performer of every occurrence of a certain activity, we should use a DirectAssignment of type Obligation. The same applies if we would like to specify that the responsibility for the execution of every instance of a behavior is to be set to an OrganizationalUnit or Group.

A DirectAssignment of type Prohibition specifies that one real-world entity (i.e., one Human, Group or OrganizationalUnit) is not allowed to perform any instance of that referred behavioral element. Considering an organizational model with many active structure elements, there is still a high level of indetermination in the execution of the instances of the referred behavioral element. The identity of the agent that will perform the referred behavior will only be known at run-time and the selection of the performer is dependent of run-time infrastructure policies, which is outside the scope of this work.

A ClassBasedAssignment determines at design-time an ActiveStructureClass for the assignment. A ClassBasedAssign-ment of type Obligation determines at design-time that the performer who must execute all instances of the referred behavior element must be an instance of the referred ActiveStructureClass.

As an ActiveStrutureClass is a type level entity, this means that at run-time, an ActiveStructureAgent must be chosen to perform an instance of the selected behavior in case it is of type Obligation. From the perspective of the assignment framework, the exact instant in which the assignment is evaluated and the ActiveStructureAgent is chosen will be defined non-deterministically at run-time and may happen at any moment after the behavioral element of the referred assignment is enabled (i.e., when its preconditions and dependencies are satisfied) and *before* its execution has started. There may exist zero, one, or many ActiveStructureAgents that instantiate the selected ActiveStruc-tureClass when the assignment is evaluated. For the type Obligation, run-time mechanisms, which are outside the scope of this work, are required to deal with the cases in which no agent instantiate the selected class and in which several agents instantiate the selected class. For instance, the run-time infrastructure may randomly choose one particular agent to perform an activity in the case several agents instantiate the selected class. In any case, the identity of the real-world entity that will perform each instance of the behavior will only be

known at run-time, as the extension of the class may change arbitrarily at run-time.

A ClassBasedAssignment of type Prohibition specifies that any real-world entity that is an instance of that ActiveStructureClass is not allowed to perform any instance of the referred behavior. This applies to all possible cases, irrespective of whether there is one, many or none agents that are instances of the ActiveStructureClass. The run-time infrastructure will have to choose one agent that is not an instance of the referred ActiveStructureClass to perform the assigned behavior

An ExpressionBasedAssignment defines at design-time an OCL expression that will be evaluated at run-time constraining the possible ActiveStructureAgents that will perform the referred behavior. In ExpressionBasedAssignments, the context of the OCL expression will always be the newly created BehavioralOccurrence of the referred behavior of the ExpressionBasedAssignment. This behavioral occurrence is created non-deterministically at runtime after the occurrence is enabled (i.e., when its preconditions and dependencies are satisfied). That is mandatory because expression may refer to information that is only available during process run-time. For instance, we may want that the specific agent that performed the previous activity *A* in a specific instance of a process to be the performer of the next activity *B*. In this case, the identity of the agent will only be known when the performer for *A* is known at process run-time. Similarly to what we have discussed for class-based assignment, the exact instant in which the expression is evaluated will be defined non-deterministically at run-time and may happen at any moment after the behavioral occurrence of the referred behavior is enabled and *before* its execution has started.

The OCL expression may return either a single ActiveStructureAgent, a set of ActiveStructureAgents (one of which will be selected in case of type obligation) or a single ActiveStructureClass (which is treated similarly to a ClassBasedAssignment).

An ExpressionBasedAssignment is used in our framework to address a variety of patterns, namely: (i) Capability-Based Distribution, in which case an expression will navigate through attributes and properties of agents and classes to define the possible performers; (ii) Case Handling, Retain Familiar and Separation of Duties, in which case an expression will navigate through the occurrences of the same complex behavior ocurrence to define the performer (or to prohibit specific performers); (iii) History-Based Distribution, in which case an expression will navigate through the historical occurrences to define possible performers; and (iv) Organizational Distribution, whose expression will navigate through the organizational relationships that agents have to define the possible performers. By employing expression-based assignments we are also able to cover an additional kind of assignment which we call Result-Based Assignment. In a result-based assignment, an expression will navigate through the result of past occurrences to determine the performer of the behavior referred to in the assignment

ConjunctiveAssignment is a specific type of ComplexAssignment indicating that all the composing Assignments must be satisfied at the same time during the run-time evaluation of the composing Assignments. Each instance of this type of ComplexAssignment refers to a specific behavior, so it may be used when a SimpleAssignment is not expressive enough to define the assignment of a behavior. It may be composed of SimpleAssignments and/or DisjunctiveAssignments. For example, we may have a ConjunctiveAssignment composed of a CapabilityBasedAssignment of type Obligation as an expression that queries the Professors with at least 5 years of experience in an organizational model, and a HistoryBasedAssignment of type Obligation indicating that the professor must have performed that task at least five times. This type of ComplexAssignment does not have an AssignmentType: this will come from the composing SimpleAssignments. When all composing assignments are of type Obligation, the set of possible performers for a conjunctive assignment is given by the intersection of all the agents selected by the composing assignments. When all composing assignments are of type Prohibition, the set formed by the union of all the agents selected by the composing assignments is prohibited from performing the behavior referred to in the assignment. Due to space constraints, we do not elaborate here on the semantics of complex assignments which mix obligation and prohibition. Our framework also supports DisjunctiveAssignments for which at least one of the composing assignments (either SimpleAssignments or ConjunctiveAssignments) must be satisfied during the run-time evaluation of the composing Assignments. When all composing assignments are of type Obligation, the set of possible performers for the disjunctive assignment is given by the *union* of all the agents selected by the composing assignments. When all composing assignments are of type Prohibition, the set formed by the intersection of all the agents selected by the composing assignments is prohibited from performing the behavior referred to in the assignment.

IV. APPLICATION TO BPMN

In this section, we discuss the application of the Assignment framework to BPMN. This enables us to: (i) instantiate the framework with respect to a concrete behavioral metamodel (that of BPMN) and (ii) illustrate the application of the approach in a concrete usage scenario which exercises the expressiveness of the assignment framework. Since BPMN does not provide constructs for organizational modeling we adopt the organizational metamodel embodied into the framework to provide us with all the organizational elements required.

In BPMN, all the work that is performed in the scope of a particular business process is represented through the Activity concept [8], which is the abstract class for all the concrete Activity types, like a SubProcess and Task. Thus, the Activity metaclass will be the direct target of the relationship instanceOfActivity of the BehavioralOccurrence metaclass, presented in section III.C. It will also be the direct target of the ofAnActivity relationship of the SimpleAssignment metaclass presented in section III.D. Process is described in BPMN as "a sequence or flows of activities in an organization with the objective of carrying out work" and they "can be defined at any level from enterprise-wide processes to processes performed by a single person". Process is the target of the instanceOfProcess relationship of the BehavioralOccurrence metaclass and the ofAProcess relationship of the SimpleAssignment metaclass. Our work is focused on Process and Collaboration Diagrams; Choreography and Conversation diagrams are considered outside the scope.

Figure 6 shows the business process model that will be the subject of the illustration of how the Assignment framework enriches the expressiveness of the active structure assignment in BPMN. It represents a process of writing and defending a master's dissertation. The process begins with a master's student writing his dissertation's first version, which is the first activity of the

process. After concluding this activity, the student submits the manuscript for review. Then a professor, which supervises his master's degree, analyzes the dissertation. The outcome of this activity defines which activity will follow. If the professor considers that there are issues on the text that must be addressed, he/she submits his considerations to the student, and the student then considers that to rewrite the dissertation. These activities will be performed until the professor approves the dissertation text. Then, the next step of the process will be the activity in which the professor defines the examination board and schedules the defense. Afterwards, when the scheduled time arrives, the master's student defends his dissertation and the next activity will be the evaluation of the dissertation and presentation, performed by the examination board. There may be two outcomes for this activity: the acceptance or the rejection of the dissertation, completing the process. The process relies on an organizational model that defines the classes "Student" (and its property "GPA"), "Professor", and, at run-time, a group to represent the examination board (we have defined a profiled UML class diagram to represent the organizational model; the model is omitted here due to space constraints.)

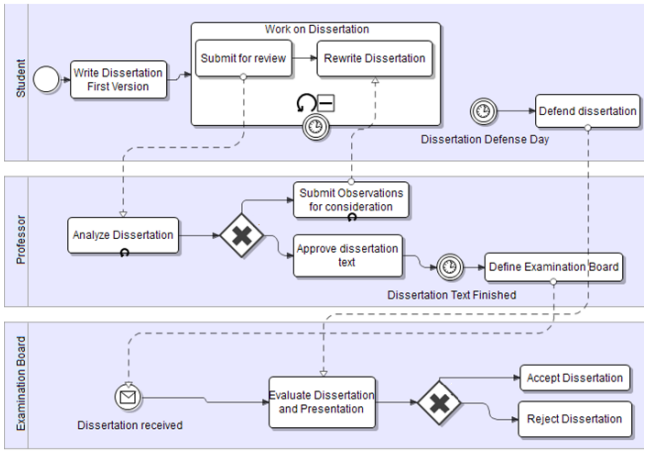

Figure 6 - BPMN model example

The following assignment constraints apply to the process:

The activity 'Write Dissertation First Version' must be performed by a Master Student. The sub-Process 'Work on Dissertation' must be performed by the same agent that performed the previous activity 'Write Dissertation First Version'. The activity 'Defend Dissertation' must be performed by the same agent that performed 'Write Dissertation First Version' and according to the rules, students that have a grade point average ('GPA') below '7.0' are not allowed to perform this activity. The activity 'Analyze Dissertation' must be assigned to an agent that: (i) has performed this activity at least three times before; (ii) has at least 5 years of experience as a professor; and (iii) is a supervisor of the specific student that wrote the dissertation (i.e., that performed the previous activity 'Write Dissertation'). The activity 'Submit observations for consideration' must be performed by the same agent that performed the previous activity 'Analyze Dissertation'. The activity 'Approve Dissertation text' must be performed by the same agent that performed the previous activity 'Analyze Dissertation'. The activity 'Define examination board' must be performed by the same agent that performed the previous activity 'Analyze Dissertation'. Finally, the activities 'Evaluate Dissertation and Presentation', 'Accept Dissertation', 'Reject Dissertation' must be performed by the group that was defined in the previous activity 'Define Examination Board'.

In order to specify the assignments described informally above, we instantiate the assignment metamodel. Due to space constraints, we focus here on the assignment involving the activity 'Defend Dissertation'. This is a conjunctive assignment (an instance of ConjunctiveAssignment) with two composing ExpressionBasedAssignments (one with assignment type Obligation and one with assignment type Prohibition). The first assignment determines that the agent that performs 'Defend Disseration' should be the same one that performed 'Work on Dissertation', with the following expression:

```
self.isContained.contains->select(
  instanceOfActivity.name = 'Work on Dissertation').participation
```

The second composing assignment is an ExpressionBasedAssignment of type Prohibition, to ensure that a 'Master Student' with a 'GPA' below '7.0' is prohibited to perform the activity. This assignment contains the following OCL expression, which retrieves all agents with a GPA below '7.0':

```
genericOrganizationalMetamodel::ActiveStructureClass.allInstances()->select(
  name = 'Master Student').hasProperty->select(
    name = 'GPA').hasAttributes->select(
      value.toReal()<7.0).characterizedAgent
```

The expression retrieves the 'Master Student' class and then collects all its properties. Then, it specifically selects the 'GPA' property, collects all the attributes that instantiate this property, selecting the ones with a value lower than '7.0'. Finally, it returns each agents that carry these attributes.

A. Prototype

To test the integration of our framework to BPMN, we have developed a prototype. It was implemented using the native EMF capabilities to manipulate models that are built in Ecore. Our example BPMN model was designed and edited in the STP BPMN Modeler. We have simulated the required organizational repository by creating dynamic instances of the organizational and occurrence metamodels through the EMF mechanisms. This repository allows us to evaluate the OCL expressions in the assignments above.

An integration to a runtime environment (such as a particular business process engine) is yet to be implemented. Since the approach we have employed so far enables the programmatic evaluation of assignment expressions, we believe this could be the basis for the integration the framework into an existing run-time infrastructure (such as that of jBPM, for example).

V. Related Work

Recently, a number of works have been proposed to extend BPMN to support the workflow resource creation patterns. In [2], the authors extend the BPMN metamodel to include concepts related to human resources to accomplish the work presented in a process. Roughly, the extended metamodel includes run-time concepts, like *Case* and *WorkItem*, respectively instances of *Process* and *Task*. Thus differently from our approach, the extended BPMN metamodel mixes design-time and run-time elements, which is undesirable from the process model management perspective and also characterizes a heavyweight extension of the language. Similarly to our approach, they rely on OCL constraints to specify assignments constraints.

The work proposed in [5] also extends the BPMN metamodel to support the resource perspective, taking into account not only the

creation patterns, but all of the workflow resource patterns. Furthermore, it also specifies a set of advanced resource patterns which the author considers to be new patterns identified in newly presented scenarios. Similarly to [2], the metamodel which extends BPMN also mixes design-time and run-time elements.

In [4], Grosskopf firstly performs an assessment of BPMN and BPDM in regards to a considered set of relevant workflow resource patterns. It then proposes a metamodel extension based on BPDM, introducing new associations and attributes to capture the not yet supported patterns. It assumes the existence of an expression language to define allocation constraints, although it does not adopt a particular language, considering this to be a technical choice.

The authors in [3] define a Resource Assignment Language (*RAL*), which is a "textual language to express resource assignments in the activities of a business process in BPMN". *RAL* is used considering an extension of the BPMN metamodel that include organizational features. As a limitation, the history of past executions is not considered in RAL. The approach supports all creation patterns except the history-based distribution pattern.

The authors in [12] propose an extension to the BPMN 2.0 metamodel to support the modeling and visualization of the resource perspective. The proposed BPMN extension is also validated against a large set of the workflow resource patterns, going beyond the creation patterns, although it does not directly support history-based distribution. Differently from the previously mentioned efforts, the authors extend BPMN with its built-in extension mechanisms, which allow attaching additional attributes and elements to BPMN elements. As it uses the BPMN mechanism for extensions, it keeps the models interchangeable because the standard elements are not modified.

Finally, differently from our approach all the works cited here (but [2]) consider the allocation of resources to activities, not considering the allocation to *Processes*. Further, none of the approaches explicitly include deontic notions such as *prohibition* as a primitive element.

VI. Conclusions and Future Work

We have introduced an assignment framework to enrich the expressiveness of existing enterprise and business process modeling techniques and support the definition of precise active structure assignments. We have proposed a model-driven framework that employs an organizational metamodel, a behavioral occurrence metamodel, and an assignment metamodel. The resulting assignment metamodel is able to express all of the creation workflow resource patterns involving allocation of organizational agents. Further, it supports an expressive constraint language to define sophisticated assignments.

To apply our framework to existing business process modeling or enterprise architecture modeling languages, the generic behavioral concepts referred to by the behavioral occurrence metamodel and the assignment model are bound to specific concepts from the metamodels of the adopted languages. In this paper, we have shown the application of the framework to BPMN, using the concepts of activity and process.

We have defined our framework making as little assumptions as possible concerning the behavioral metamodel. Thus we believe that the approach has the potential to be applied to different metamodels. In our future work, we intend to report on our efforts to apply the assignment framework to other process and enterprise modeling languages (such as ArchiMate). Further, we should define an integration of the approach in a process-aware system considering the runtime environment, in order to support to the actual execution of the assigned behaviors. We should also focus on use cases to validate the usability of the proposed framework. These may reveal lack of expressiveness that may require extending the framework proposed here. Finally, defining a simpler concrete syntax for the assignment expressions should also be target of future investigation. An end user environment could transform expressions defined in a simpler concrete syntax into OCL, thus enhancing the framework's usability while still profiting from OCL's well-defined syntax, semantics and tooling.

Acknowledgments

This research is funded by the Brazilian Research Funding Agencies CAPES, FAPES (grant number 59971509/2012) and CNPq (grants number 310634/2011-3 and 485368/2013-7).

References

[1] Arpini, R. H., Almeida, J. P. A., *On the support for the assignment of active structure and behavior in enterprise modeling approaches*, Proc. of the 27th Annual ACM Sym-posium on Applied Computing (SAC '12), 1686–1693, 2012.

[2] Awad, A., Grosskopf, A., Meyer, A., and Weske, M. *Enabling resource assignment constraints in BPMN*. Technical Report, Hasso Plattner Institute, 2009.

[3] Cabanillas, C., Resinas, M., and Ruiz-Cortés, A. *Towards the definition and analysis of resource assignments in BPMN 2.0,* tech. rep., Universidad de Sevilla, 2011.

[4] Grosskopf, A. *An extended resource information layer for BPMN*. Technical Report, Hasso Plattner Institute, 2007.

[5] Meyer, A. *Resource Perspective in BPMN: Extending BPMN to Support Resource Management and Planning*. Master's Thesis, Hasso Plattner Institute, 2009.

[6] Muehlen, M. Z. Organizational Management in Workflow Applications – Issues and Perspectives. *Information Technology and Management*, 5 (3-4), 2004, 271-294.

[7] Object Management Group (OMG*). Organization Structure Metamodel (OSM) 3rd initial submission*. OMG document, bmi/09-08-02, 2009.

[8] Object Management Group (OMG). *Business Process Modeling Notation (BPMN) 2.0 Specification*. OMG document, formal/2011-01-03, 2011.

[9] Russell, N., ter Hofstede, A.H.M., D. Edmond, and van der Aalst, W.M.P. *Workflow Resource Patterns*. Technical report, Queensland University of Technology, Australia, 2010. http://www.workflowpatterns.com/patterns/resource, last accessed at 17/05/2011.

[10] Russell, N., van der Aalst, W. M. P., ter Hofstede, A. H. M., and Wohed, P. On the suitability of UML 2.0 activity diagrams for business process modelling. *APCCM '06: Proceedings of the 3rd Asia-Pacific conference on Conceptual modeling*, Darlinghurst, Australia, 2006, 95-104.

[11] Sharp, A., and McDermott, P. *Workflow Modelling Tools for Process Improvement and Application Development*. Artech House, 2001.

[12] Stroppi, L, Chiotti, O, Villarreal, P. *A BPMN 2.0 Extension to Define the Resource Perspective of Business Process Models*. XIV Ibero-American Conference on Software Engineering (CIbSE), Rio de Janeiro, Brazil, 2011.

[13] Wohed, P., van der Aalst, W.M.P., van der Dumas, M., ter Hofstede, A.H.M., Russell, N. *Pattern-based Analysis of BPMN - An extensive evaluation of the Control-flow, the Data and the Resource Perspectives*. BPM Center Report BPM-06-17, BPMcenter.org, 2006.