UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
DEPARTAMENTO DE INFORMÁTICA
MESTRADO EM INFORMÁTICA

RÔMULO HENRIQUE ARPINI

# A FRAMEWORK TO SUPPORT THE ASSIGNMENT OF ACTIVE STRUCTURE AND BEHAVIOR IN ENTERPRISE MODELING APPROACHES

VITÓRIA, AUGUST 2012

RÔMULO HENRIQUE ARPINI

# A FRAMEWORK TO SUPPORT THE ASSIGNMENT OF ACTIVE STRUCTURE AND BEHAVIOR IN ENTERPRISE MODELING APPROACHES

**Dissertação submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Mestre em Informática.**

VITÓRIA, AUGUST 2012

RÔMULO HENRIQUE ARPINI

# A FRAMEWORK TO SUPPORT THE ASSIGNMENT OF ACTIVE STRUCTURE AND BEHAVIOR IN ENTERPRISE MODELING APPROACHES

**Dissertação submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Mestre em Informática.**

**Aprovada em 31 de agosto de 2012.**

**VITÓRIA, AUGUST 2012**

## ABSTRACT

The need to relate the various architectural domains captured in partial descriptions of an enterprise is addressed in virtually all enterprise modeling approaches. One of these domains, namely that of *organizational behavior,* has received significant attention in recent years in the context of business process modeling and management. Another important domain, that of *organizational structure* is strongly inter-related with the process domain. 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, we argue that 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.*

Despite the importance of the relations between these architectural domains, 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 work aims to overcome these limitations by proposing a framework for active structure assignment that can be applied to 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.

**Keywords:** Enterprise Architecture, Enterprise Architecture Modeling, Business Process modeling, Organizational Modeling, Active Structure, Active Structure Assignment, BPMN, ArchiMate.

# LIST OF FIGURES

# LIST OF TABLES

# SUMMARY

# 1. Introduction

## 1.1 Motivation

Several approaches to enterprise modeling manage the complexity of an organization by describing the organization from different perspectives focusing on: (i) organizational structure (with actors, roles and organizational units); (ii) organizational behavior (structured into business processes, activities, and more recently, services); (iii) information systems that support organizational behavior, and (iv) technical infrastructure to support information systems.

The need to relate the various perspectives captured in partial descriptions of an enterprise is addressed in virtually all enterprise modeling approaches and has been recognized in Zachman (1987): “each of the different descriptions has been prepared for a different reason, each stands alone, and each is different from the others, even though all the descriptions may pertain to the same object and therefore are inextricably related to one another.”

This need has led to the development of *relations between architectural domains* in enterprise architecture and enterprise modeling approaches (LANKHORST, 2005). One of these domains, namely that of *organizational behavior*, has received significant attention in recent years in the context of business process modeling and management. Business process modeling addresses the way enterprises organize their work and resources showing how they contribute to fulfilling the enterprise’s strategies (SHARP; MCDERMOTT, 2001). Another important domain, namely that of *organizational structure*, is strongly inter-related with the process domain. 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.

The relationship between both domains and how an activity will be assigned to an organizational entity is commonly captured in process models by associating a role, an organizational unit or a human to each activity or task (AALST; HEE, 2002). In BPMN (OMG, 2011), for instance, entities that will perform the activities are often modeled in "swim lanes" and "pools", which may represent a role or an organizational human or other kinds of organizational concepts.

Organizational entities that perform work are often called resources in enterprise modeling and business process modeling approaches. According to Russel, Hofstede and Edmond (2005), a resource is an entity that is capable of performing activities, and may be classified as a human or non-human resource. We focus here on human resources. A human resource is typically a member of an organization, which is a formal grouping of resources that can undertake activities pertaining to a common set of business objectives. They usually have a specific position within that organization, and may also have specific privileges associated with it. They may also be members of an *organizational unit*, which is a permanent group of resources within an organization.

In many approaches, a human resource may have one or more associated *roles* (RUSSEL; HOFSTEDE; EDMOND, 2005), which in general serve as a classification mechanism for resources with similar job roles or responsibilities. Further, individual resources may possess *capabilities* that may be used to consider their suitability to undertake certain activities. Capabilities are often specialized into skills, qualifications or other attributes, such as previous working experience. Each resource may also have a schedule with future commitments (i.e., activities that the resource committed to undertake) and a history with the resource's past activities. Finally, a resource may have direct relationships with other resources, such as direct report and delegate relationships.

Given the strong connection between the organizational behavior and organizational resources, we argue that 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.*

The relations enable the basic expression of distribution of work between the various organizational resources, a relation that is historically discussed as *division of labor*, with its acknowledgment dating from a long time ago, as Plato (1992) mentions: "Well then, how will our state supply these needs? It will need a farmer, a builder, and a weaver, and also, I think, a shoemaker and one or two others to provide for our bodily needs." Adam Smith also recognized the importance of the division of labor in organizations, stating that "growth is rooted in the increasing *division of labor*." (SMITH, 2008).

Properly representing the assignment of active structure elements and behavioral elements at design time is important to allow the comprehensive analysis of an Enterprise Architecture, 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, as observed in (MUEHLEN, 2004).

Although most of the techniques offer some support for establishing these relations, the levels of support and expressiveness they offer vary significantly (ARPINI; ALMEIDA, 2012). Several of these, such as BPMN and UML activity diagrams are considered to offer simplistic support, as seen in the work by Awad et al. (2009), failing to provide required expressiveness with respect to active structure assignment (e.g., as evidenced by a low coverage of Workflow Resource Patterns (RUSSEL et al., 2006; WOHED et al., 2006). Since many of these approaches are based solely on business process models, they fail to identify relations with rich organizational structure models and are thus unable to express active structure assignment based on organizational relations.

## 1.1 Objectives

The main objective of this work is to address these limitations by defining a framework for active structure assignment for enterprise architecture and business process modeling approaches. This framework should enrich the expressiveness of existing techniques and 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 behavioral languages to the framework with minor changes.

A specific objective of our work is to apply the framework to enrich the expressiveness of BPMN, which represents a widely employed business process modeling technique. Different from most of the current proposals of extensions of BPMN active structure assignment capabilities, our work should not rely on modifying the BPMN metamodel to provide the required expressiveness, thus maintaining the interoperability between BPMN models. Finally, an additional objective of our work is to investigate the application of the framework to a second technique (namely ArchiMate) in order to provide some evidence in favor of the generality of the framework.

## 1.2 Approach

Initially, we conducted a study to review the active structure assignment support in some of the widely enterprise and business process modeling techniques and frameworks, including ArchiMate (THE OPEN GROUP, 2009a), ARIS (SCHEER, 2000), DoDAF (US DEPARTMENT OF DEFENSE, 2011), BPMN (OMG, 2011), XPDL (WFMC, 2008) and UML (OMG; 2010b).

Our review was done initially through a bottom-up approach: we analyzed and extracted the constructs that are used to define the assignments at design-time in each of the techniques. In addition, we have performed a top-down analysis considering the desirable requirements for the expression of assignments. This top-down analysis was performed employing Workflow Resource Patterns to evaluate the expressiveness of the various techniques, similarly to what has been done in (AWAD et al., 2009), (MEYER, 2009), (RUSSEL; HOFSTEDE; EDMOND, 2005), (WOHED et al., 2006).

The literature review showed us that the current support in the various techniques is extremely simplistic in the active structure assignment domain, justifying the need for a comprehensive framework for active structure assignment.

To address the identified limitations, a framework for active structure assigned has been proposed. A main requirement for this framework has been to avoid modification or heavyweight extension of currently employed modeling languages. Thus, the framework should be applicable to existing and widely used business process modeling and organizational modeling approaches, enriching their capabilities in the expression of active structure assignment.

The framework is designed using technologies that are driven by the MDA (Model-Driven Architecture) principles (KLEPPE; WARMER; BAST, 2003), in particular using the Eclipse Modeling Framework (EMF, 2012), as a *de facto* standard for Model-Driven Engineering. EMF is used to define all the proposed metamodels and relations between architectural domains. The assignment component of the framework is proposed as a metamodel that is dependent of behavioral and organizational metamodels, as complex assignments require information available across many enterprise aspects that may be represented in different models, like the organizational model.

Since assignments must consider some specific organizational aspects often not covered in business process modeling techniques (such as BPMN), we have defined our own abstract organizational language (using an organizational metamodel) that is embodied into the framework. This language captures the general organizational concepts to cover a range of assignments and model a complex organization. This allows us to represent expressive organization-based assignments.

Further, since assignments are often dependent on a history of past activities, we have defined a behavioral occurrence metamodel, which allows us to define assignments based on historical information.

In order to offer full support a number of the considered patterns we have proposed a mechanism to represent sophisticated expressions for complex assignments based on the Object Constraint Language (OCL) (OMG; 2010a). A prototype including support for OCL-based assignments was developed to demonstrate the feasibility of the approach.

To demonstrate the applicability of the framework to existing languages, we apply it to BPMN. This makes possible to evaluate the assignment framework and see how it helps to support the assignment requirements. The application prototype developed to BPMN specifically demonstrates the feasibility of the framework using current available tools.

The proposal of application of the assignment framework to ArchiMate serves to provide some evidence in favor of the generality of the framework.

## 1.3 Structure

This work is structured in six chapters. The content of each one is briefly described in the following:

- In Chapter 2 we review the support for the assignment of active structure and behavior in enterprise modeling approaches, presenting the Workflow Resource Patterns as a requirements framework and reviewing the current support of some modeling techniques, namely ArchiMate, ARIS, DoDAF, XPDL, UML 2.0 Activity Diagram and BPMN.
- In Chapter 3 we present the proposed assignment framework. We discuss the framework's general architecture and explain how each metamodel is integrated into this general architecture. Further, we show how the metamodel may be used in conjunction to define rich and expressive assignments. An evaluation of the framework with respect to the Workflow Resource Patterns is presented.
- In Chapter 4 we apply our framework to BPMN, showing how assignment is defined considering this specific technology. We also define a UML Class Diagram profile to enable the representation of the generic organizational metamodel. Finally, we illustrate the approach showing the range of assignments possible.
- In Chapter 5 we show how the framework could be applied to ArchiMate, defining a mapping of ArchiMate organizational constructs to the organizational metamodel of the assignment framework and showing how the behavioral elements from ArchiMate could be integrated into the assignment framework. Ultimately, we use the same example of Chapter 4 to illustrate the approach in ArchiMate and show how one could model assignments on it.
- In Chapter 6 we conclude our work, discussing its contributions, limitations and proposing future work that may be developed based on this work.

# 2. Support for the Assignment of Active Structure and Behavior in Enterprise Modeling Approaches

In this section, we analyze and review the support of the different kinds of active structure assignment in enterprise modeling techniques and frameworks, including ArchiMate, DODAF, and ARIS. Since we believe that these frameworks will be used in the description of an Enterprise Architecture in tandem with the detailed description of business processes, we also discuss the support for active structure allocation in processes modeling techniques. Instead of addressing an exhaustive list of business process and workflow modeling techniques, we have included here developments that we believe are representative of a large number of process techniques. First, we have included XPDL, since it was conceived as an interchange format for a number of process-related products, including "execution engines, simulators, BPA modeling tools, Business Activity Monitoring and reporting tools" (SHAPIRO, 2000). Second, we have addressed the support provided in UML 2.0 Activity Diagrams and BPMN 2.0 because of their wide acceptance to represent business processes.

We employ Workflow Resource Patterns to evaluate the expressiveness of the resource perspective in many process technologies and tools (similarly to what has been done for various techniques in (AWAD et al., 2009; MEYER, 2009; RUSSEL, HOFSTEDE, EDMOND, 2005; WOHED et al., 2006) and we will also use it as requirements to design our proposed assignment framework. We present each of the techniques in sequel, dedicating a sub-section in the review of each technique to discuss how they support the Workflow Resource Patterns.

## 2.1 Workflow Resource Patterns

The Workflow Resource Patterns form a comprehensive catalog of common types of resource allocation constraints. 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 use to compare a number of commercially available workflow management systems and business process modeling languages (RUSSEL et al., 2010).

We focus here on the core set of patterns that deals with task allocation to human resources and that are specified at design time, restricting the range of human resources that can undertake particular work items that correspond to the tasks. These are called the “creation patterns”. Figure 1 illustrates this. The creation patterns come into effect when a work item (instance of a task) is first created.

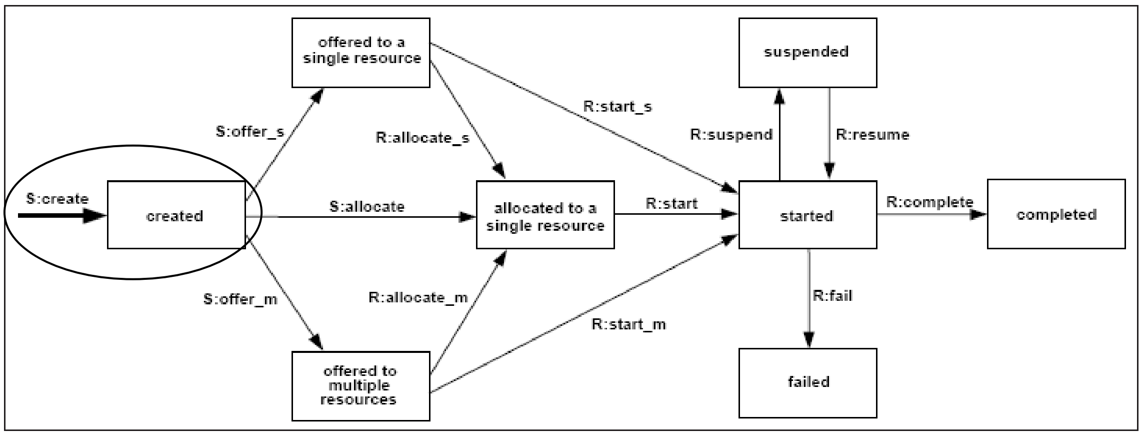


**Figure 1 - Creation Patterns (RUSSEL et al., 2010)**

The *Direct Distribution pattern (WRP-01)* captures the ability to specify the identity of the resources to which the work items will be distributed. It is particularly useful when we want a task to be performed by a specific resource.

The *Role-based Distribution pattern (WRP-02)* 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. So if this pattern is supported by a process technology, there must exist the role concept.

The *Deferred Distribution pattern (WRP-03)* 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. To support this pattern, the process technology must have some mechanism to specify that a task will not be assigned to a specific resource until runtime. For instance, we may want to specify that the identity of the resource who will perform the task ‘Review paper’ will be deferred until runtime.

The *Authorization pattern (WRP-04)* 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 (WRP-05)* captures the ability to specify that two work items must be performed by different resources. For instance, if we have a task 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. So the process technology that supports this pattern must have some mechanism to specify some kind of interdependence between work items.

The *Case Handling pattern (WRP-06)* is a specific approach based on the premise that all the tasks in a process or sub-process are related and must be performed by the same resource.

The *Retain Familiar pattern (WRP-07)* 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. For instance, we may want that the same resource who performed the task ‘Identify applicants’ to be the performer of the task ‘Select suitable applicants’. It is a more flexible version of the *WRP-06 Case Handlin*g pattern.

The *Capability-Based Distribution pattern (WRP-08)* 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. For instance, we may want that the task “Audit Critical Project” to be performed by an auditor with previous work experience on this job greater than ten years.

The *History-Based Distribution pattern (WRP-09)* captures the ability to distribute tasks to the resources based on the history of execution they have in the tasks. For instance, we may want that the task “Conduct Heart Surgery” to be performed by a surgeon that already carried out similar procedures before. The operationalization of this pattern requires information about previous executions.

The *Organizational Distribution pattern (WRP-10)* 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. For instance, the task “Approve Project Budget” must be performed by the manager directly superior to the Project Manager that performed the task “Elaborate Project Budget”.

The *Automatic Execution pattern (WRP-11)* 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. For instance, the task ‘Calculate balance’ may be performed automatically based on functionality implemented in an information system.

## 2.2 ArchiMate

ArchiMate is a modeling language that offers an integrated architectural approach that describes and visualizes the different architecture domains and their underlying relations and dependencies, aiming to offer an unambiguous specification and descriptions of enterprise architecture’s components and specially their relationships with a consistent alignment (THE OPEN GROUP, 2009a). The language is currently standardized by The Open Group in its version 1.0 and used to support architectural descriptions produced using TOGAF (THE OPEN GROUP, 2009b).

The language distinguishes three layers with different abstraction levels: (i) the business layer, which offers products and services to external customers, realized in the organization by business processes performed by business actors; (ii) the application layer, which supports the business layer with application services which are realized by software applications; and (iii) the technology layer, which offers infrastructural services for software applications. Each one of these layers includes modeling constructs to represent active structure elements, behavioral elements and passive structure elements, as shown in Figure 2.

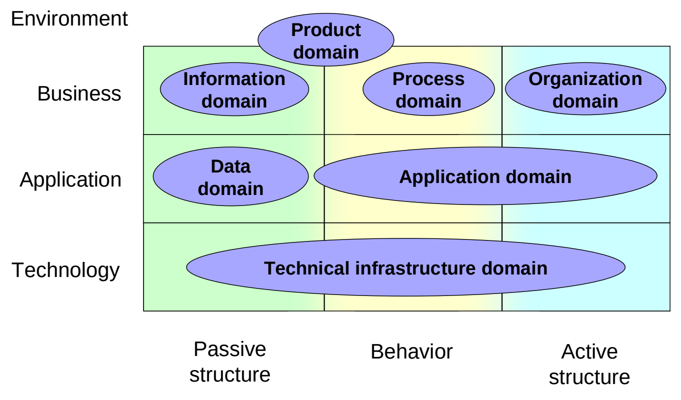


**Figure 2 - ArchiMate Framework (THE OPEN GROUP, 2009a)**

We focus on the concepts of the business layer, whose metamodel is presented in Figure 3. The abstract concept *Business Behavior Element* groups all concepts related to the behavioral structure. The link with the active structure is done through the assignment relationship, which allows a modeler to relate a *Business Behavior Element* to a *Business Role*. A *Business Role* may, in turn, be related to a *Business Actor* through an assignment relationship.

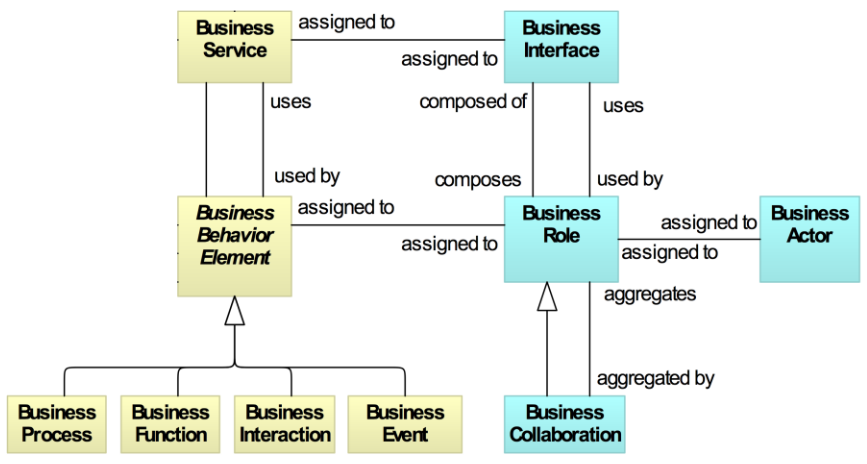


**Figure 3 - Fragment of Business Layer Metamodel (THE OPEN GROUP, 2009a)**

A *Business Actor* is an organizational entity capable of performing behavior, and performs the behavior assigned to one or more *Business Roles*. *Business Roles* are defined as a named specific behavior of a business actor in a particular context. A *Business Role* may be assigned to one or more business processes or *Business Functions*. *Business Processes* are defined as units of internal behavior or collections of causally-related units of internal behavior intended to produce products or services, while *Business Functions* are defined as units of internal behavior that group behavior according to some criteria, such as knowledge, resources and skills. A *Business Service* is an externally observable behavior that is realized internally by *Business Behaviour Elements*. A *Business Service* may be assigned to a role's *Business Interface*.

Figure 4 shows a small example of active structure assignment in ArchiMate, relating process, role and actor. The "ArchiSurance" actor is composed of two departments, namely, "Luggage Insurance Department" and "Travel Insurance Department". The "Travel Insurance Department" is assigned to the "Travel Insurance Seller" role, which is associated with the "Take out Insurance" process. Whichever actor is assigned to the "Travel insurance seller role" will perform the "Take out insurance process". In this specific example, the process should be performed by the "Travel Insurance Department". The example also reveals the assignment of the "Offering travel insurance" service (a behavioral element), by the means of a *Business Interface* provided by the "Travel insurance seller" role and realized by the "Take out insurance" process.

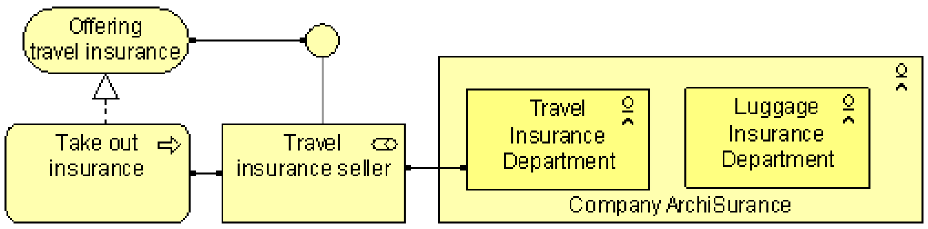

Offering travel insurance
Take out insurance
Travel insurance seller
Travel Insurance Department
Luggage Insurance Department
Company ArchiSurance

**Figure 4 - Process, Actor and Role (THE OPEN GROUP, 2009a)**

A *Business Collaboration* is defined as a temporary configuration of two or more roles, resulting in specific collective behavior in a particular context. Unlike the case in which a *Business Process* or *Function* is assigned to a single *Role*, *Business Collaborations* aggregates two or more *Roles*, meaning it represents a collective effort which may be more than the sum of the behavior of the separate roles. Collaborations are assigned to *Business Interactions*, which are used to describe the behavior that takes places within these collaborations. Figure 5 shows how both *Business Collaboration* and *Business Interaction* may be used. "Combined Insurance Seller" is the collaboration that aggregates the "Travel insurance seller" and "Luggage insurance seller" roles. The "Take out combined insurance" interaction involves the execution of the "Prepare travel policy" process, performed by the "Travel insurance seller" role, and the "Prepare luggage policy" process, performed by the "Luggage insurance seller" role.

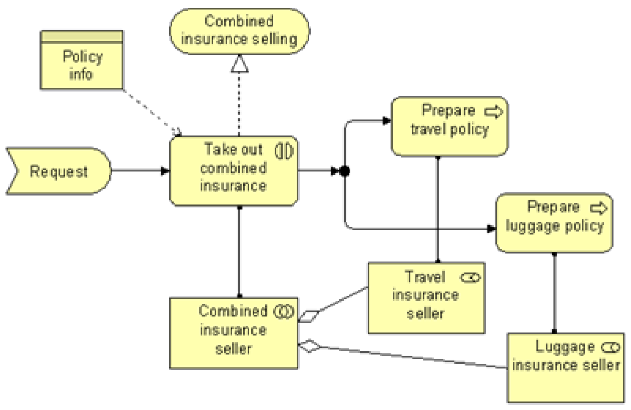


**Figure 5 - Business Colaboration and Interactions (THE OPEN GROUP, 2009a)**

### 2.2.1 Workflow Resource Patterns Support in ArchiMate

ArchiMate provides direct support to the Direct Distribution, Role-based Distribution and Automatic Execution patterns. Direct Distribution is fully supported because a *Business Actor* may be explicitly "assigned to" a *Business Role* which in its turn may be "assigned to" a *Business Behavioral Element*. Role-based Distribution is also fully supported by omitting the assignment of a *Business Actor* to a *Business Role*. Automatic Execution is supported by specifying an assignment relationship between an application component (part of a system) and a business process.

Since ArchiMate is aimed predominantly to model enterprise architectures at design-time, it cannot be said to fully support Deferred Distribution, as run-time support (from an execution environment) is not in the scope of the language and framework. Nevertheless, it is possible to state that is supports Deferred Distribution partially since it enables one to omit assignment for a particular behavioral element. We assume that this leaves certain flexibility for runtime assignment, hence characterizing the partial support for the pattern.

Organizational-based Distribution is also partially supported as one may define an organizational model in ArchiMate, and use the roles and actors in this model in the assignment. *Business roles* may be composed of other business roles; and the same can be said of *business actors*. Nevertheless, relations between roles and between actors cannot be used in the assignment.

The remaining "creation patterns" are not supported. ArchiMate lacks an integrated authorization framework (not supporting the Authorization pattern), lacks the possibility of relating the performers of processes (not supporting the Separation of Duties, Case Handling and Retain Familiar patterns), does not consider that business actors or business roles may have specific attributes to characterize them (not supporting the Capability-based Distribution pattern). Further, it does not aim at considering execution history (not supporting the History-based Distribution pattern).

## 2.3 ARIS

ARIS (Architecture for integrated Information Systems) method (SCHEER, 2000) is structured in four inter-related views (organizational, data, control and function) that support the description of an organization and its information system. The framework includes three abstraction layers (requirements, design and implementation), dealing with different levels of details, separating specific concerns. The organizational view describes all the hierarchy of an organization, i.e., the communication and relationship between organizational units and reveals the roles of the individuals in an organization, whereas the functional view is used to describe the tasks performed by the organization (SCHEER, 2000). The control view shows the relationship between the business processes of an organization and the remaining entities of the organization (organizational structure, resources, information) of the business environment (DAVIS, 2001). We focus on the control view at the requirements level.

Business processes in ARIS are modeled in Event-driven Process Chains (EPCs), whose main elements include *Functions, Events* and *Rules*. *Functions* are the main behavioral concept, representing organizational activities. *Functions* of an EPC can be placed within swim-lanes, as shown in the example of Figure 6. In this example, a "Client" performs the "Request Purchase" *Function*, while a "Seller" performs the "Analyze purchase request" and the "Finish purchase" or the "Inform Client" *Functions*, depending on whether the purchase is approved.

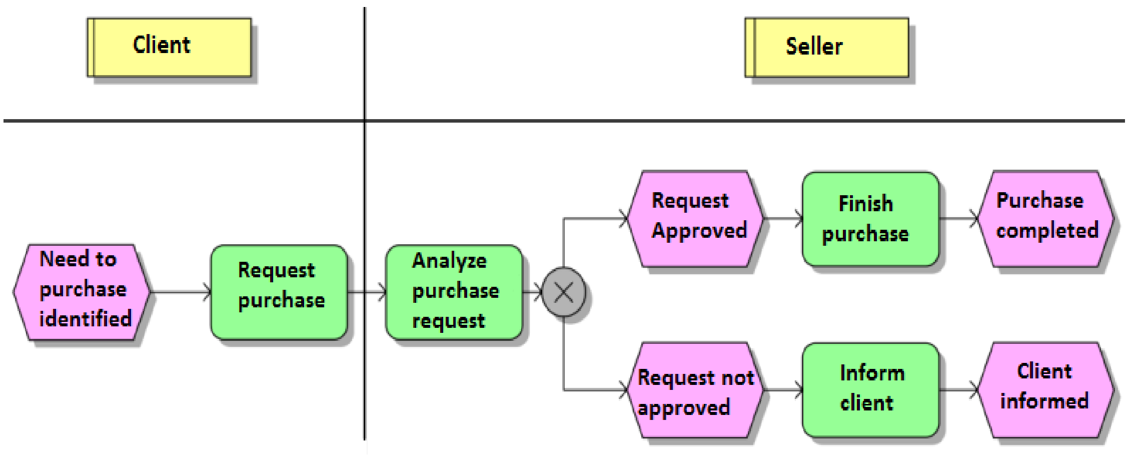


**Figure 6 - Example of Business Process Model in ARIS (SANTOS JR.; ALMEIDA; GUIZZARDI, 2010b)**

In our analysis, we consider the metamodel excavated in (SANTOS JR.; ALMEIDA; PIANISSOLLA, 2011), where the authors have identified that the ARIS toolset recognizes the following relations between the active structure (represented by the abstract metaclass *Participant*) and behavioral (represented by *Function*): *is technically responsible for*, *carries out*, *is IT responsible for*, *decides on*, *must be informed about*, *contributes to*, *accepts*, *has consulting role in*, *must be informed on cancellation* and *must inform about result of*. *Carries out* is the main relationship, indicating who will be responsible for performing the *function*, and is one of the relationships represented in Figure 7.

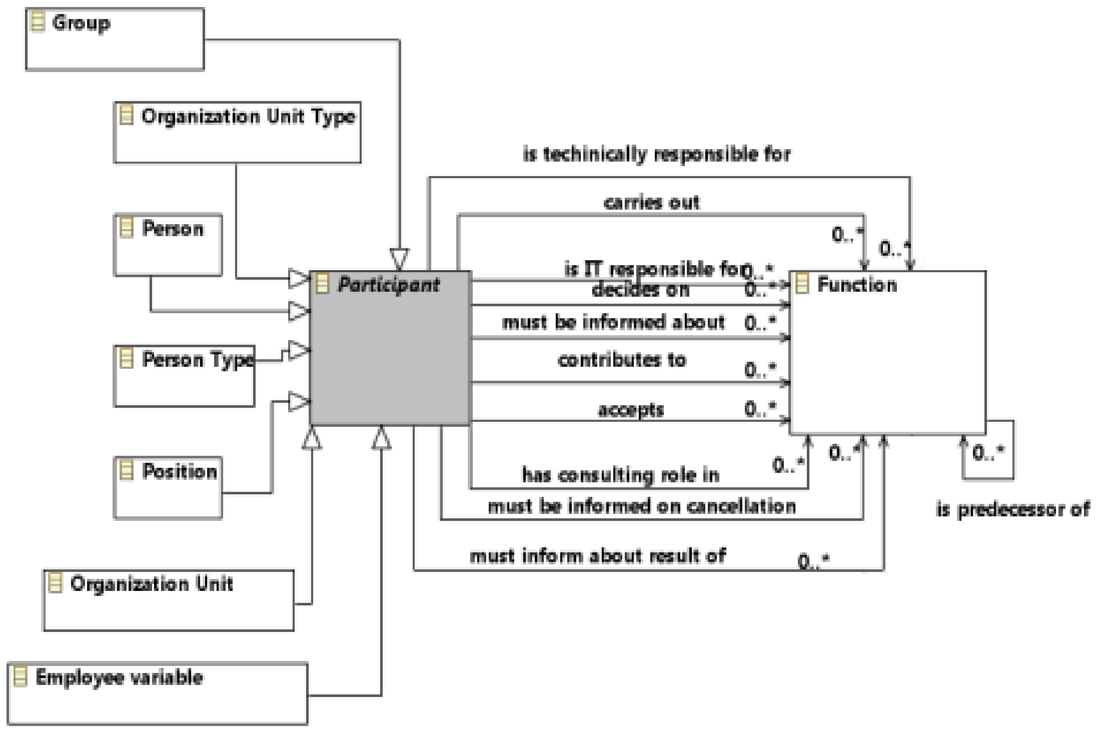


**Figure 7 - Fragment of the Metamodel Adapted from (SANTOS JR.; ALMEIDA; PIANISSOLLA, 2011)**

Figure 7 also shows the organizational concepts used to describe the potential *participants* in the organizational activities, they are: *Organization Unit Type*, *Organization Unit*, *Position*, *Person Type*, *Person*, *Group* and *Employee Variable*. All of these concepts can be related with the *Function* concept through all the aforementioned relations. These elements are used in the so-called Organization Charts, which allows one to capture hierarchical and others active structure specific relations.

ARIS has a rich set of elements to describe organizational structure at instance-level and type-level. An *Organizational Unit* represents "an entity that is responsible for achieving organizational goals", being a real-world entity. An *Organizational Unit Type* is described as "a type of organization unit, i.e., an element that represents the common features (duties, responsibilities, etc.) of a set of organization units". A *Position* represents "the smallest organizational unit possible. The responsibilities and duties of a position are defined in the Position Description". A *Position Type* represents a "type of position, i.e. an element that represents the common features (duties, responsibilities, etc.) of a set of positions". A *Person* "is used to represent a person who is assigned to an organization". A *Person Type* represents a "generalization of person, i.e., an element that represents the common features (duties, responsibilities, feature, etc.) of a set of people". A Group represents "a group of employees (person) or a group of organizational units (Organizational Unit) that cooperate to achieve a goal". Finally, the semantics of the *EmployeeVariable* metaclass is not discussed in the ARIS documentation. ARIS also has a rich set of relations between those organizational structure elements, which include hierarchical relations (of technical and managerial nature), delegation relations, etc. We refrain from discussing them here due to space constraints. 3

### 2.3.1 Workflow Resource Patterns Support in ARIS

The EPC implemented by ARIS toolset provides direct support to the Direct Distribution, Role-based Distribution and Automatic Execution patterns. Direct Distribution is fully supported because a Person, an Organization or a Group may be defined as the performer of a function (through the "carries out" relation). Role-based Distribution is also fully supported because *Person Types* and *Positions* may carry out a function, and both may assume the notion of role. Automatic Execution is also fully supported because one may allocate the concepts of Application System and Application System Type that represent computer and software applications to functions (DAVIS, 2001).

As a function may be defined without the need to specify which kind of Participant will be the performer, we consider that Deferred Distribution is partially supported. Organizational-based Distribution is also partially supported as there exists in ARIS an organizational metamodel (excavated and presented in (SANTOS JR.; ALMEIDA; GUIZZARDI, 2010a)), and it does consider concepts of Position, Position Type, and others. But as it does not consider the more complex relationships when defining the assignment, it only provides a partial support.

The remaining "creation patterns" are not supported.

## 2.4 DoDAF

The Department of Defense Architecture Framework (DoDAF) is a comprehensive framework and conceptual meta-model that has been designed specifically to meet the business and operational needs of the US Department of Defense (US DEPARTMENT OF DEFENSE, 2011). Although the focus of the framework is clearly oriented to military systems, it can be extended to architectures that are more general (LANKHORST, 2005), and provides concepts to model behavioral and active structure concepts.

In DoDAF, a *Performer* represents who may execute an *Activity,* and an activity represents specific operational actions. DoDAF introduces a few concepts to address the relation between performers and activities. A fragment of the metamodel with *Performer* and its related concepts is shown on Figure 8.

*Performer* is a type of *Resource*. A *Performer* may be further specified into one of the following types: (i) *System*, which is defined as “a functionally, physically, and/or behaviorally related group of regularly interacting or interdependent elements”, (ii) *Service*, described as a “*Performer* to enable access to a set of one or more resources, such as Information, Data, Material and Performers”; (iii) *OrganizationType*, which is the type of an individual *Organization*. For example, we may have a “ForProfitOrganization” and “NonProfitOrganization” types; or (iv) *PersonType,* which defines a category of *IndividualPersons* that share common skills. A *PersonType* may also be used to represent a role that may be played with a more general *PersonType*, through the *personTypePartofPerformer* relation.

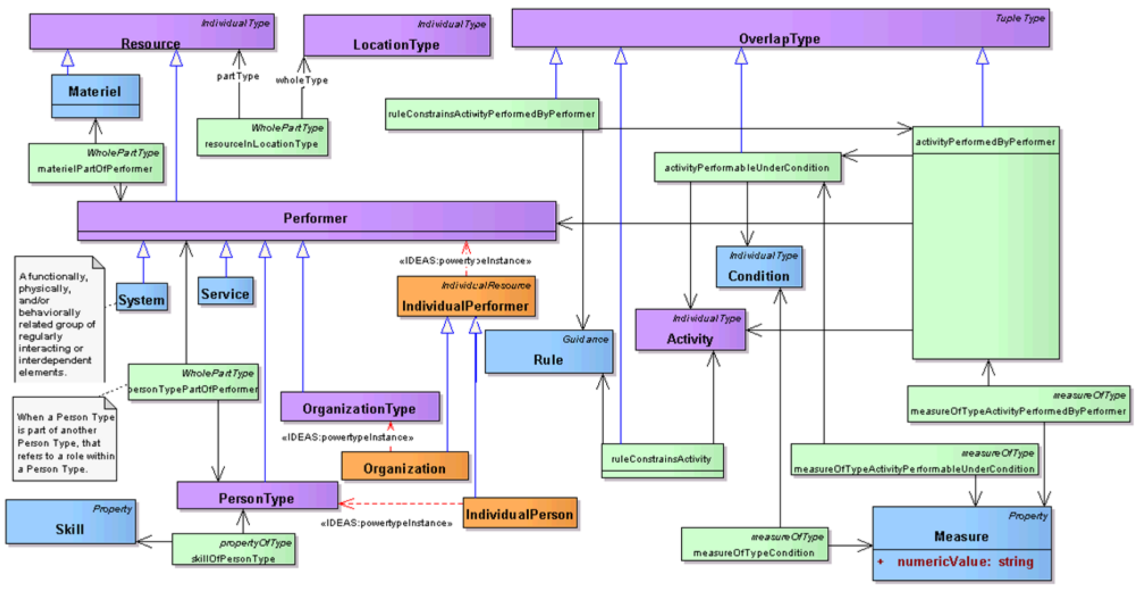


**Figure 8 - Excerpt of DoDAF Performer Metamodel (LANKHORST, 2005)**

### 2.4.1 Workflow Resource Patterns Support in DoDAF

DoDAF provides direct support to the Direct Distribution, Role-based Distribution and Automatic Execution. Direct Distribution is fully supported because an *IndividualPerformer* may be defined as the performer of an *Activity*, and that includes *Organizations* and *IndividualPersons*. Role-based Distribution is supported because there does exist the concept of *PersonType*, which may assume the notion of role. Automatic Execution is also fully supported because the performer may be a *Service*, which may be any software to a business service.

As an activity may be defined without the need to specify what type of *Performer* will perform the activity, then Deferred Distribution is partially supported. Although it does have an organizational structure built-in, it is very basic and in regards to the Capability-based Distribution, we consider that it is partially supported because there may exist rules that constrain the performers of the activities, and these rules may refer to the skills that a *PersonType* has within an organization.

The remaining “creation patterns” are not supported.

## 2.5 XPDL

XPDL (XML process definition language) (WFMC, 2008) was developed by the Workflow Management Coalition (WfMC) to support the interchange of workflow process definitions (AALST, 2004).

The topmost entity of an XPDL 2.1a model is a *Package*, which includes one or more process definitions (HAVEY, 2005) and one or more *Participant* definitions. A *Participant* represents the "description of resources that can act as the performer of the various activities in the process definition" (WFMC, 2008). Process definitions in a *Package* automatically "inherit" the *Participants* defined on that *Package*.

Figure 9 depicts the basic set of entities and relations for the exchange of process definitions. The entity *Participant* is further classified into one of the following basic types (WFMC, 2008): (i) *Resource*, when the participant represents a specific resource agent; (ii) *ResourceSet*, when the participant represents an aggregation of resources; (iii) *Organizational Unit*, when the participant represents a department or any other unit within an organization model; (iv) *Human*, when the participant represents is a single person; (v) *System*, when the participant represents an automatic agent; (vi) *Role*, when the participant is a placeholder for a human which can perform a specific function. Note that XPDL does not provide a clear semantics for each one of the basic types.

Figure 9 shows an association between the *Participant* entity and a *Resource Repository or Organizational Model*, meaning that the *Participant* declaration may refer organizational structure definitions outside the scope of the specification, but which may be used with the extensibility mechanisms provided by XPDL.

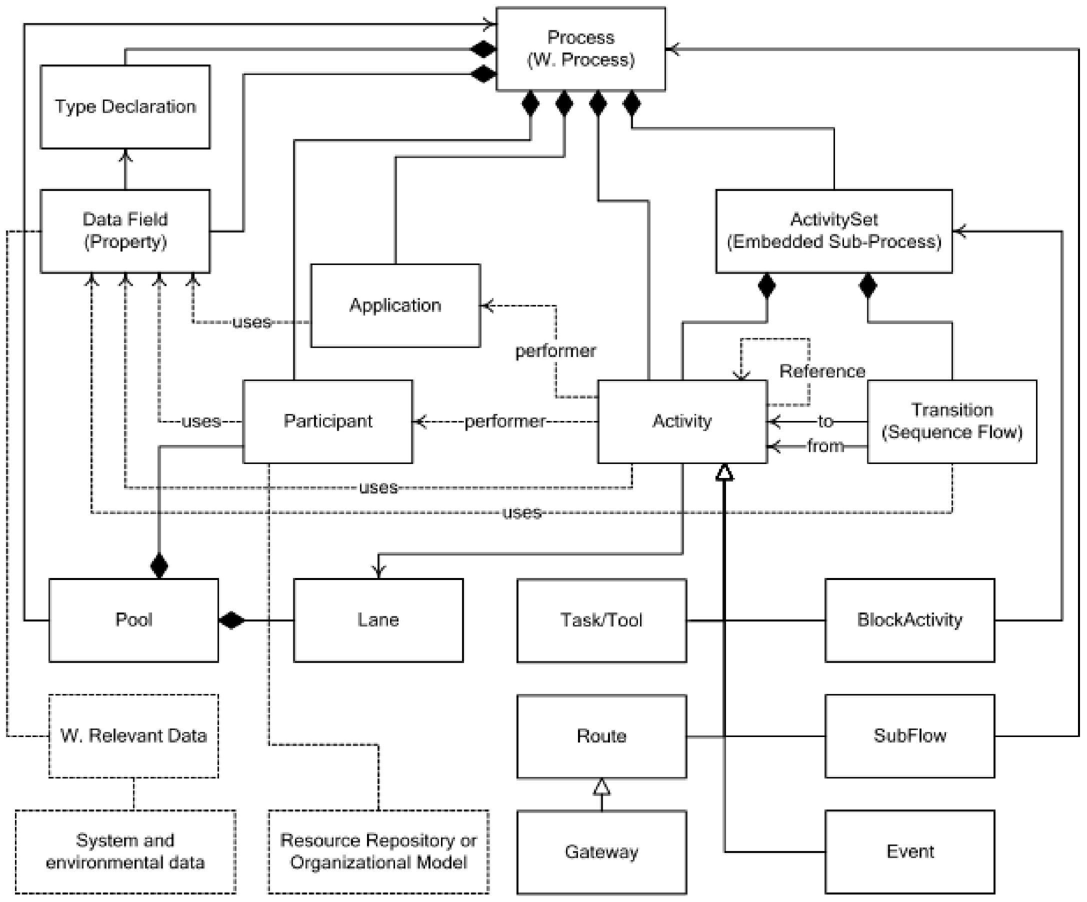


**Figure 9 – Excerpt of the Process Definition Metamodel (WFMC, 2008)**

In XPDL, a *Process* is structured into *Activities*. The link between the active structure and *Activity*, is given by the *performer* relationship. The *Participant* identifiers that are used in this relationship must be declared either in the surrounding Process Definition or inherited from the surrounding *Package* declaration or coming from external packages, like an *Organizational Model*. The specification mentions the use of expressions to define the *Participants* of an *Activity*, without specifying exactly the syntax and semantics of these expressions. The specification also mentions that when the expression evaluation returns an empty set of performers or when it returns a non-unique performer, then this must be handled by the execution engine of the Workflow System and is outside the scope of the specification.

### 2.5.1 Workflow Resource Patterns Support in XPDL

XPDL provides direct support to the Direct Distribution and Role-based Distribution. Direct Distribution is fully supported because a human or a specific resource may be defined as the performer of an activity. Role-based Distribution is also fully supported because there exists the notion of role and it may be used when defining the performer of an activity. Automatic Execution is also fully supported because the performer may be a *System*, considered to be an automatic agent.

As an activity may be defined without the need to specify which kind of *Participant* will be the performer, we consider that Deferred Distribution is partially supported.

The remaining "creation patterns" are not supported.

## 2.6 UML Activity Diagrams

UML is a standardized general-purpose language that aims "to provide system architects, software engineers, and software developers with tools for analysis, design, and implementation of software-based systems as well as for modeling business and similar processes" (OMG; 2010b).

The modeling concepts of UML are grouped into language units represented by different diagrams, which consists of tightly-coupled modeling concepts that provide users the ability to represents aspects of a system under study according to a particular formalism. For instance, the Activity Diagram groups concepts related to behavior modeling.

UML 2.0 does not provide a specific language unit to model an organization; however, as shown on the work by Dumas, Aalst and Hofstede (2005), general organizational structures can be modeled by UML class diagrams, and concrete organizations can be treated as instances of these general organizational structures.

Activity diagrams can also be used for process modeling in UML. An *Action* is one of the main constructs of an activity diagram, and a fundamental unit of behavior specification, taking a set of inputs and transforming them on a set of outputs (though either or both sets may be empty). An action represents a single step within an *Activity*, that is, one that is no further decomposed (OMG; 2010b).

The connection of the active structure to the process models is done within an activity diagram using the notational element *ActivityPartition*, which divide the nodes and edges to constrain and show a view of the contained nodes. Constraints vary according to the type of element that the partition represents, which may be one of the following (OMG; 2010b): (i) *Classifier*, meaning that the behaviors of invocations contained by the partition are the responsibility of instances of the classifier representing the partition. Thus different instances of the same classifier may execute the contained actions; (ii) *Instance*, imposing the same constraints as a classifier-based partition, but requiring a particular instance of the classifier. (iii) *Part*, meaning that the behaviors contained in the partition will be executed by parts of the same instance of a structured classifier. (iv) *Attribute a*nd *Value*, meaning that certain attributes are restricted to certain values. The specification includes an example of a partition representing a location attribute and sub-partitions representing specific values of that attribute, such as “Rio de Janeiro” (OMG; 2010b). Nevertheless, this latter kind of partitioning is not well documented in the specification, as it does not specify whether the attributes apply to actions inside the sub-partition or to objects (instances) executing the actions. Figure 10 exemplifies multidimensional partitioning.

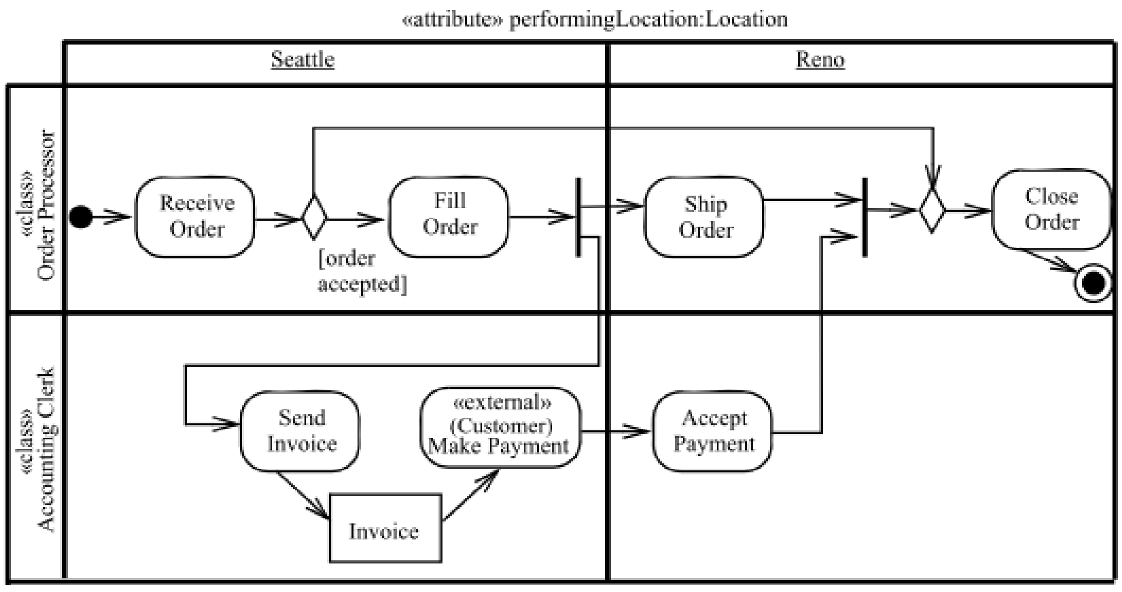


**Figure 10 - ActivityPartition Usage (OMG, 2010b)**

The actions “Receive Order” and “Fill Order” in Figure 10 are performed by an instance of the “Order Processor” class, situated in “Seattle”, but not necessarily the same instance for both. Although the “Make Payment” action is contained within the “Seattle/Accounting” partitions, its performer and location are not specified by these partitions since this action is stereotyped as «external».

### 2.6.1 Workflow Resource Patterns Support in UML Activity Diagrams

In this section we basically summarize the results of the work done by Russel et al. (2006), which evaluated the suitability of UML in regards to a number of workflow patterns, including the Workflow Resource Patterns (RUSSEL et al., 2010).

Activity Diagrams provide direct support to the Direct Distribution and Role-based Distribution, as it is possible to define as a partition a specific instance of a classifier, thus allowing the definition of specific single resources. Although there is not the concept of role in Activity Diagrams, it is possible to define the performer of an *ActivityPartition* as a *Classifier*, which may be used to be a role. As there may exist *CallActions* that invoke behavior from some target object, which may be a non-human object, we consider that the WRP-11 is directly supported.

Russel et al. (2006) considers that the remaining patterns are not supported in UML Activity Diagrams, however, we consider that UML offers partial support to Deferred Distribution, Capability-based Distribution and Organizational-based Distribution. We consider that Deferred Distribution is partially supported because it is possible to define actions that do not belong to any partition, thus deferring the identity of the resource that may perform the action. We consider that Capability-based Distribution is also partially supported because although there is no organizational model, it is possible to use the attributes and values as the performers of the *ActivityPartition*, so the support may be inferred. Lastly, we consider it to offer partial support to Organizational-based Distribution because although it lacks any kind of reference to an organizational metamodel, it is possible to infer some kind of organizational hierarchy through the use of sub-partitions.

The remaining “creation patterns” are not supported.

## 2.7 BPMN

Business Process Modeling Notation (BPMN) is a standard graphical notation for business process modeling adopted by the OMG. Its main goal is to provide a notation that is easy to understand by all business users (WHITE, 2004). The BPMN 2.0 specification clearly states that the language is constrained to support only concepts of modeling applicable to business processes, meaning that other domains of an organization are out of scope, with one of them being the domain of organization modeling (OMG, 2011). Although BPMN does not include elements for organizational modeling the specification clearly assumes the existence of these elements when defining who will be responsible for a process or for the execution of an activity.

BPMN defines a number of diagrams to model business processes under a certain perspective. We focus here on the Process and Collaboration Diagrams, not explicitly discussing Choreography (a specialization of Collaboration) and Conversation (a specialized use of Collaborations) (OMG, 2011).

The Process Diagram is used to model a business process internal to an organization. It essentially describes a sequence or flow of Activities in an organization with the objective of carrying out work. This type of diagram does not include a textual nor graphical way to explicitly specify the responsible for the *Process* or the activities contained within it. Nevertheless, *Lanes* can be used *informally* for that purpose. As discussed in the specification, “the meaning of the Lanes is up to the modeler” (OMG, 2011). In practice, “Lanes are often used for such things as internal roles (e.g., Manager, Associate), systems (e.g., an enterprise application), an internal department (e.g., shipping, finance), etc.” (OMG, 2011). Figure 11 shows a small example of a *Process* defined in BPMN. *Activities* are represented by rectangles with rounded corners, and represent points in a *Process* where work is performed, being the main behavioral concept in the language. An *Activity* is an abstract metaclass specialized into either a *Sub-Process* or a *Task* (which in turn is further specialized into specific kinds of tasks).

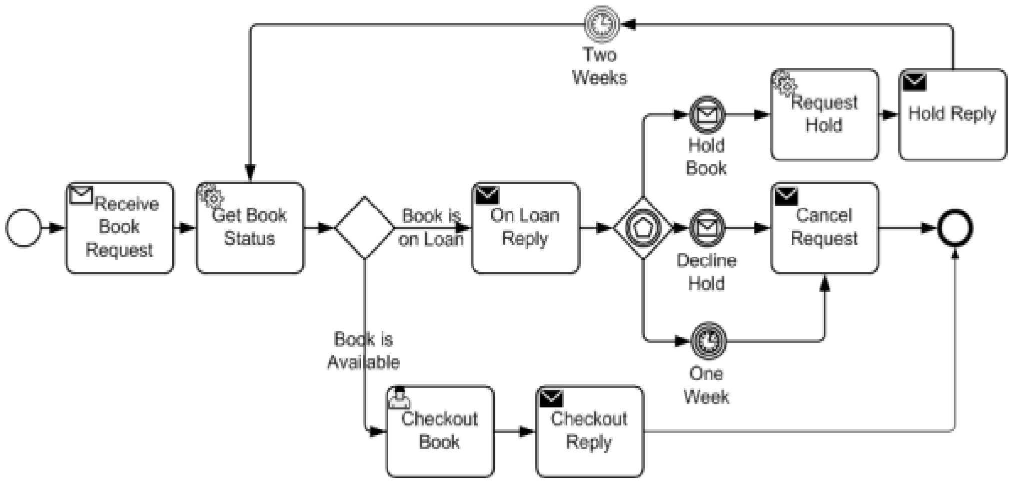


**Figure 11 - A Process Example (OMG, 2011)**

Although BPMN does not provide graphical or textual elements to represent the performers of activities in process diagrams, the metamodel includes elements to define them. Figure 12 shows the main concepts and associations related to this aspect of the language. The *Resource* metaclass is used to specify resources that may be referenced by activities. These resources may be human resources or any other resource assigned to an activity during process runtime. Resources are defined at type-level, e.g., “Professor” and “Student”. Specific resources (instances such as, e.g., “João Paulo” and “Rômulo”) would be described in a deployment phase, which is outside the scope of the specification (OMG, 2011), and may be addressed in a BPMN-conformant infrastructure. A modeler may characterize *resources* by defining its properties using *ResourceParameters* (OMG, 2011). The assignment of active structure to behavior may be defined by the modeler using the *ResourceRole* element shown in Figure 12. The assignment may be done by defining either: (i) an association between the *ResourceRole* and a *ResourceAssignmentExpression* or (ii) between the *ResourceRole* and a *Resource*.

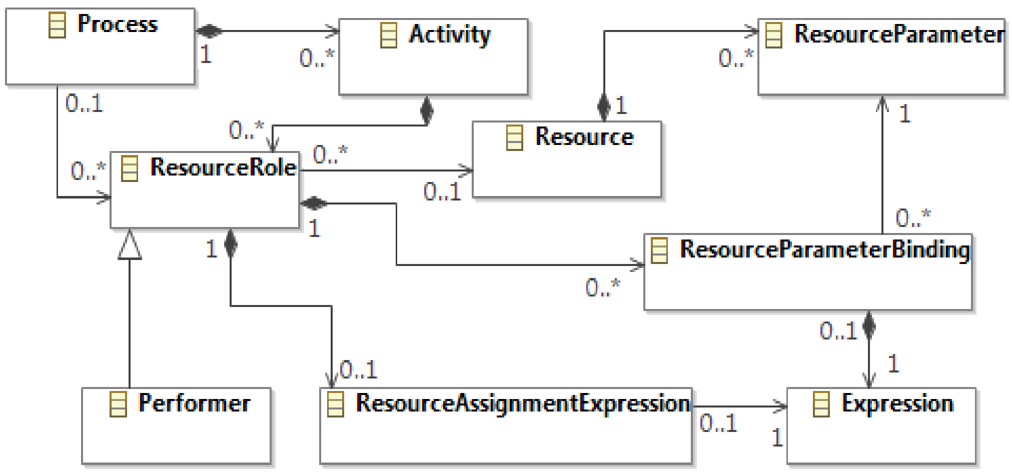


**Figure 12 - Fragment of the BPMN metamodel centered in ResourceRole, adapted from (OMG, 2011)**

In the former case (i), the modeler provides an *Expression* written in natural language or in a formal expression of choice (by default formal expressions are defined in XPath (CLARK; DEROSE, 1999). This expression is used at runtime to assign resource(s) to a *ResourceRole* element.

In the latter case (ii), a specific resource (type) is selected at modeling time. Optionally, the modeler may define which parameters of the resource specified may be used or overridden through the definition of an Expression, that may also use data of the instance task in which the resource is being referred.

Figure 12 shows that a *ResourceRole* may be further specialized in a *Performer*, meaning that the resources selected must be the ones responsible for the execution of the activity ("A performer can be specified in the form of a specific individual, a group, a position or role in an organization, or an organization" (OMG, 2011)).

In addition to the Process Diagram, BPMN defines a Collaboration to describe the interactions (messages exchange) between two or more business entities. These business entities are called *Participants* in the scope of a *Collaboration* and are represented graphically as pools. A *Participant* can be a specific entity (*PartnerEntity*, e.g. a company) or a more generic one (*PartnerRole*, e.g. a buyer). However, there are no graphical elements o distinguish these concepts, with all being done in natural language. A *Participant* may be associated with a *Process* in a *Collaboration*, meaning that it is responsible for the execution of the process. Figure 13 shows the metaclass *Participant* and its main associations.

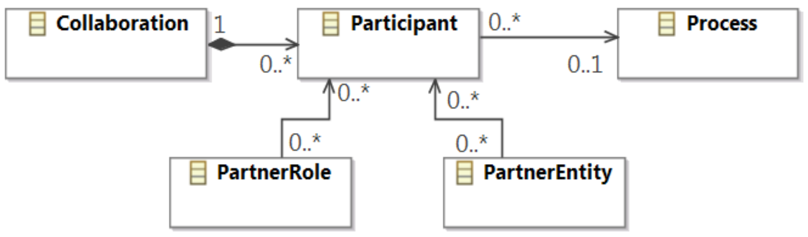


**Figure 13 - Fragment of the metamodel centered in Participant, adapted from (OMG, 2011)**

Figure 14 shows an example of a Collaboration Diagram. “Financial Institution” and “Supplier” are the *Participants*. Each one of them is assigned to a process.

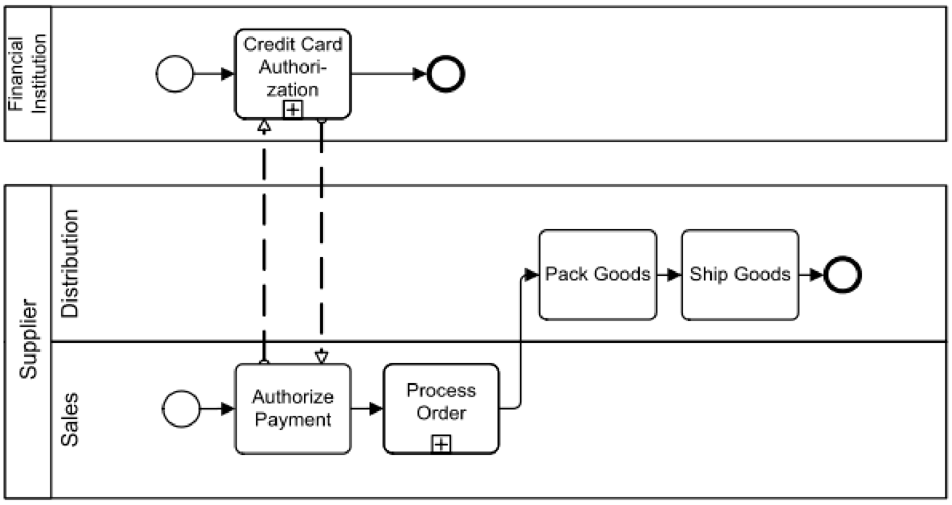


**Figure 14 - A Collaboration Diagram (OMG, 2011)**

If activities are represented in a collaboration, they may also be allocated to perfomers using the mechanisms discussed for the process diagram. We believe this may cause certain semantic problems in the language, because it allows modelers to mix activity-level assignment and process-level assignment with no consistency rules. (*Performer* is defined at activity level, i.e., a performer is assigned to an activity defined in a process, “being the resource that will perform or be responsible for an activity”, while *Participants* are defined at process-level. The metamodel does not define relations or constraints involving the metaclasses *Performer* and *Participant*.)

### 2.7.1 Workflow Resource Patterns Support in BPMN

Although the support of BPMN for the Workflow Resource Pattern has been considered in the past, for instance the one performed by members of the Workflow Resource Patterns initiative in (WOHED et al., 2006), we perform here our own analysis, consolidating several evaluations such as those in (MEYER, 2009; GROSSKOPF, 2007; CABANILLAS, RESINAS, RUIZ-CORTÉS, 2011; STROPPI, CHIOTTI, VILLARREAL, 2011) and considering new concepts introduced in the BPMN 2.0 specification (OMG, 2011).

BPMN 2.0 offers direct support to Direct Distribution and Role-Based Distribution, both through the *Pool* construct which represents a *Participant* in the process and may be a business entity (a company) or a more general business role (WOHED et al., 2006). At the activity level, BPMN 2.0 also supports Role-Based Distribution through the *Performer* metaclass, which may be related to an Activity and associated with a *Resource*.

Automatic Execution is also directly supported through the *Service Task*, which is a *Task* that "uses some sort of service, which could be a Web service or an automated application." (OMG, 2011).

BPMN provides partial support to three others patterns. As it is possible to have no performers defined (either at activity or process level), we consider that it offers partial support to Deferred Distribution. It also offers partial support to the Capability-based Distribution (at activity level), as one may use the metaclass *ResourceParameters* to characterize *Resource*, and that may be used when defining the *Performer*. We consider this support partial since capabilities (represented here as resource parameters) cannot be captured in a general organizational model, and cannot be used at process level. Lastly, BPMN provides partial support to Organizational-based Distribution, because *Pools* have participants and may also have sub-partitions called *Lanes*. Nevertheless, Lanes may be used to organize Pools using arbitrary criteria (for example, it may be used to represent geographical locations, or the level of importance of the tasks within a Lane). Thus, while the modelers may employ *Lanes* to determine organizational roles and positions and define a basic organizational structure to categorize activities, this is informal and lacks semantics. Because of this, we disagree with the work by Meyer (2009) which considers BPMN to fully support Organizational-based Distribution. The remaining “creation patterns” are not supported (WOHED et al., 2006; GROSSKOPF, 2007).

## 2.8 Support Summary

Table 1 summarizes the constructs of the approaches reviewed, presenting the constructs they adopt to model the active structure domain; the constructs they adopt to model the behavioral domain; and the constructs to express the relations between active structure and behavior.

**Table 1 - Summary of current support**

| | **Active Structure Domain** | | **Relations between Active Structure and Behavioural Domain** | **Behavioral Domain** |
|---|---|---|---|---|
| | **Main Concepts** | **Relations Between Concepts** | | **Main Concepts** |
| ArchiMate 1.0 | *Business Actor*, *Business Role*, and *Business Collaboration* | A *Business Actor* may be assigned to a *Business Role* and a *Business Collaboration* aggregates *Business Roles.* | A *Business Role* is *assigned to* a *Business Behavior Element*, with different semantics when used to relate different kinds of elements. | *Business Behavior Element*, specialized into *Business Process*, *Business Function*, *Business Interaction* and a *Business Event* |
| DODAF 2.02 | *Performer*, which may be a *System*, *Service*, *PersonType* or *OrganizationType*; and *IndividualPerformer*, which may be a specific *Organization* or *IndividualPerson* | *IndividualPerson* is instance of *PersonType*; *Organization* is instance of *OrganizationType*; part-whole relations | *activityPerformedByPerformer*, *Rule*, *ruleConstrainsActivityPerformedByPerformer*, *Condition*, *activityPerformableUnderCondition*, *Measure*, *measureOfTypeActivityPerformedByPerformed*, *measureOfTypeActivityPerformableUnderCondition* | *Activity* |
| ARIS | *Organization Unit Type*, *Organization* | Numerous relations, which | An element of the active structure domain may be related to a | *Function* |

| | | | | |
|---|---|---|---|---|
| | *Unit*, *Position*, *Person Type*, *Employee Variable*, *Person*, *Group* | are omitted here due to space constraints - see (SANTOS JR.; ALMEIDA; PIANISSOLLA, 2011). | *Function* through these relations: *is technically responsible for*, *carries out*, *is IT responsible for*, *decides on*, *must be informed about*, *contributes to*, *accepts*, *has consulting role in*, *must be informed on cancellation* and *must inform about result of*. | |
| XPDL 2.1a | *Participant* may be a *Resource*, *Resource Set*, *Organizational Unit*, *Role*, *Human* or *System*. | None. | A *Process* includes the definition of *Participants*. The link between an *Activity* and a *Participant* defines the *performer* attribute, which may be defined using expressions. *Participant* identifiers are used in the *performer* attribute and must be declared in the surrounding process definition or coming from external packages, like an organizational model. | *Activity*, *Process* |
| UML 2.0 Activity Diagram | No specific elements, although active structure can be represented through *class diagrams* and *object diagrams* (DUMAS; AALST; HOFSTEDE, 2005). | *Associations* in *class diagrams* and *links* in *object diagrams (*when these diagrams are used to represent active structure). | *Partitions* are contained within *Activities*, constraining and providing a view on the *Actions* performed in *Partitions*. May be used to indicate who/what will perform the *actions* contained within it, referring to an element such as *Classifier*, *Instance*, *Part*, *Attribute* and *Value*. | *Activity* and *Action*, which represents a single step within an *Activity*, that is, one that is no further decomposed. |
| BPMN 2.0 | *Resource*, *PartnerRole* and *PartnerEntity*. | None. | A *Resource* may be associated to an *Activity* through the *Performer* metaclass. A *Performer* may explicitly specify a *Resource* who will perform the activity or an *Expression* that returns *Resources* that will perform the activity. A *Participant* of a *Collaboration* may refer to a *PartnerRole* or a *PartnerEntity* who will participate in the *Collaboration*. A *Participant* may also explicitly refer to a *Process*, which in turn contains *Activities.* | *Collaboration*, *Process*, *Activity* |

Table 2 summarizes the support for the workflow resource "creation patterns" in the reviewed approaches; '+' stands for full support; '+/-' stands for partial support; '-' stands for no support.

**Table 2 - Support for the "creation" Workflow Resource Patterns in the reviewed approaches**

| | ArchiMate | ARIS | DoDAF | XPDL | UML Activity Diagram | BPMN |
|---|---|---|---|---|---|---|
| WRP-01: Direct Distribution | + | + | + | + | + | + |
| WRP-02: Role-Based Distribution | + | + | + | + | + | + |
| WRP-03: Deferred Distribution | +/- | +/- | +/- | +/- | +/- | +/- |
| WRP-04: Authorization | - | - | - | - | - | - |
| WRP-05: Separation of Duties | - | - | - | - | - | - |
| WRP-06: Case Handling | - | - | - | - | - | - |
| WRP-07: Retain Familiar | - | - | - | - | - | - |
| WRP-08: Capability-Based Distribution | - | - | +/- | - | +/- | +/- |
| WRP-09: History-Based Distribution | - | - | - | - | - | - |
| WRP-10: Organizational Distribution | +/- | +/- | - | - | +/- | +/- |
| WRP-11: Automatic Execution | + | + | + | + | + | + |

## 2.9 Conclusions

A mature approach to enterprise modeling should clearly establish relations between the various architectural domains addressed. In this section, we have reviewed the mechanisms employed in ArchiMate, DODAF, ARIS, XPDL, UML and BPMN to support the assignment of active structure including the review of their support to the Workflow Resource Patterns. We can observe in our analysis that most of the approaches offer simplistic support for the active structure assignment, including few modeling constructs to relate each of the architectural domains.

With respect to the coverage of the workflow resource patterns by the various surveyed techniques, 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 (RUSSEL et al., 2010). 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. Given the need to refer to past executions of tasks in history-based distribution, the lack of support for this pattern is not surprising 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 allow one to specify some properties that resources may have and to use that when defining the assignment. However, because they do not offer a full-fledged mechanism to allow the specification of resource properties and their types and to use that when describing the assignment, we consider the support for this pattern “partial”. Ultimately, 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 because they do not offer the possibility to both define complex organizational structure and use organizational relationships when defining the assignment, we consider the support for this pattern “partial”.

With respect to ArchiMate, *Business Actors* are assigned to *Business Behavioral Elements* indirectly, through the *Business Role* element. The language also includes a notion of *Business Collaboration* which may be used to assign a behavioral element to several *Business Actors* (through an aggregation of *Business Roles*). The objective of the language is to establish a high level abstract view on an enterprise architecture, and thus the language cannot be used to model details of the assignment.

DoDAF, in its turn, offers more expressiveness when considering the constraints on its assignment relation, defining *Conditions* under which the *Activity* should be performed and *Rules* on the *Performer*, possibly including quantitative constraints using a notion of *Measure*.

Regarding ARIS, we observed that it is the only one of the studied languages to define relationships beyond assignment or responsibility for behavior execution. The relations between active structure elements and behavioral elements include technical responsibility, participation in decision making, general contribution, general interest in, need to consult and inform, etc.  Nevertheless, the semantics of each of the different relations is not discussed explicitly, and can only be superficially inferred from the names of the meta-associations.

Regarding XPDL, which is designed with the main goal to provide interoperability between workflow systems, the support for active structure assignment is rather primitive: it only identifies a direct relationship between a *Participant* and an *Activity*. XPDL makes no assumptions on the organization model (beyond defining a list of participant types, whose semantics is poorly defined.) The specification also mentions that expressions may be used to define the performers of activities, but a language for these expressions is not defined.

UML provides the generic mechanism of *ActivityPartitions* which can be used to define the classes or instances which execute actions in an activity diagram. The same mechanism can be used to capture any other criteria which modelers may define for grouping actions. The construct is similar to that of Lanes in BPMN, although specific stereotypes facilitate the identification of the types of partitions in a model, defining more precise semantics for each of them.

With respect to BPMN, the assignment of the performers may be done directly or through *expressions*. Differently from XPDL it provides a default language for such expressions. Nevertheless, it only assumes the existence of attributes in a (external) resource model. No kinds of relations between resources (performers) are assumed, and thus the *expressions* cannot take advantage of using relationships between active structure elements. Further, we have identified some issues in the combination of process-level and activity-level assignment relations. Some of the limitations in BPMN to address the assignment of active structure and behavior have been addressed in (AWAD et al., 2009) and (MEYER, 2009), which propose an extension to BPMN in order to support various kinds of active structure allocation proposed by the Workflow Resource Patterns.

# 3. Assignment Framework

In this chapter we discuss the Assignment framework, which is centered on an Assignment metamodel and includes a number of related metamodels to enable the expression of a wide range of assignments. We first discuss the requirements and assumptions for the framework. We then present the framework's overall architecture, presenting its dependencies and relationships with other metamodels, and discuss how they are integrated with each other. We also discuss in details the elements of each metamodel involved with the various kinds of assignments, showing how the framework satisfies the various requirements. The framework is presented independently of process or enterprise modeling techniques.

## 3.1 Requirements/Assumptions

In this section we consider the requirements and assumptions for the framework proposed in this work and which should be applicable to technologies for active structure assignment. We consider the set of Workflow Resource Patterns discussed in Chapter 2 as a requirements framework and thus assume in this work that they all must be somehow supported in an expressive assignment technique. Further, the analysis of the constructs of the various enterprise architecture approaches and business process languages surveyed in Chapter 2 provides us with some mechanisms that must be incorporated in the framework and that are often tied to resource patterns too. These sort of "bottom-up" requirements are incorporated here.

First of all, there must be some support to manually specify at design-time the resource to which the instances of a behavioral element will be assigned. This is directly related to the Direct Distribution pattern (WRP-01). Further, it must be possible to specify the assignment of instances of a behavioral element to agents that fulfill a specific role in an organization. This is directly related to the Role-Based Distribution pattern (WRP-02).

The premise that one must not force the designer to prescribe the assignment at design-time must also be supported, which is related to the Deferred Distribution pattern (WRP-03). When there is no assignment specified to a behavior, we assume the assignment is the least restrictive, because, in principle, any entity may perform it at run-time.

In order to support the Authorization pattern (WRP-04), there should be support to an authorization framework dealing with the privileges that resources may have in regards to the execution of instances of a task. The supported authorization framework should focus on the privileges that may be specified at design-time concerning the performers of behavioral elements. We envision that beyond specifying the resources that are *authorized* to perform behavioral elements such a framework may also allow the specification of prohibitions and obligations, incorporating features from access control (e.g. (BOTHA; ELOFF, 2001), (BOTHA, 2001), (AHN; SANDHU, 2000) and (ZHOU, 2008)), policy-based approaches (e.g. (ISO, 2010), (BRUCKER et al., 2012)) and compliance techniques (KHARBILI et al., 2011) currently missing in all of the approaches surveyed in Chapter 2.

Various techniques for behavioral specification adopt a hierarchical approach in which behavioral elements may be further decomposed into finer-grained behavioral elements (e.g., from high-level "processes" to fine-grained "tasks"). In these cases, the framework should enable the assignment of behavioral elements at any level of aggregation or abstraction. If there is an assignment to a behavioral element that is further refined into others, the composing behavioral elements must be performed by the active structure element assigned to the "container" behavioral element. This is more general than, but hinted by the Case Handling pattern (WRP-06).

The framework should be able to support the specification of the assignment of a behavioral element based on interdependence with other behavioral elements. This is hinted by the Separation of Duties pattern (WRP-05) and Retain Familiar pattern (WRP-07).

Given the prominence of organizational roles, capabilities and relations in the distribution of work, the framework should allow the specification of the assignment by means of the properties that resources have in regards to the organization. This is directly related to the Capability-Based Distribution pattern (WRP-08) when attributes of resources are used when defining the assignment and to Organizational Distribution

(WRP-10) pattern because the organizational relationships are used when defining the assignment. This is also indirectly related to History-Based Distribution pattern (WRP-09) because the past executions of resources may also be seen as characterizing resources, and also indicative of a resources capabilities in performing similar or related tasks.

Some of the techniques surveyed, such as BPMN and XPDL, consider that the details of assignments may have to be specified in *Expressions*. This is necessary, for example, to specify Capability-Based Distribution, representing what is called a “capability function” (RUSSEL et al., 2010). In this case, an expression represents this function. Expressions may also be used to support the definition of more complex assignments, possibly combining various patterns. The assignment framework should support the precise specification of complex assignments, possibly through a formal expression language.

## 3.2 Framework Architecture Overview

Figure 15 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 external Behavioral metamodel is shown in dashed lines, as it is in fact a placeholder for a specific metamodel of the technique being extended by the framework. The metamodels in this middle layer provide the metaclasses and meta-associations which will define the elements that may be part of the various kinds of assignments.

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.. For a discussion of how OCL is used and a brief explanation of its main concepts, one may refer to Appendix A.

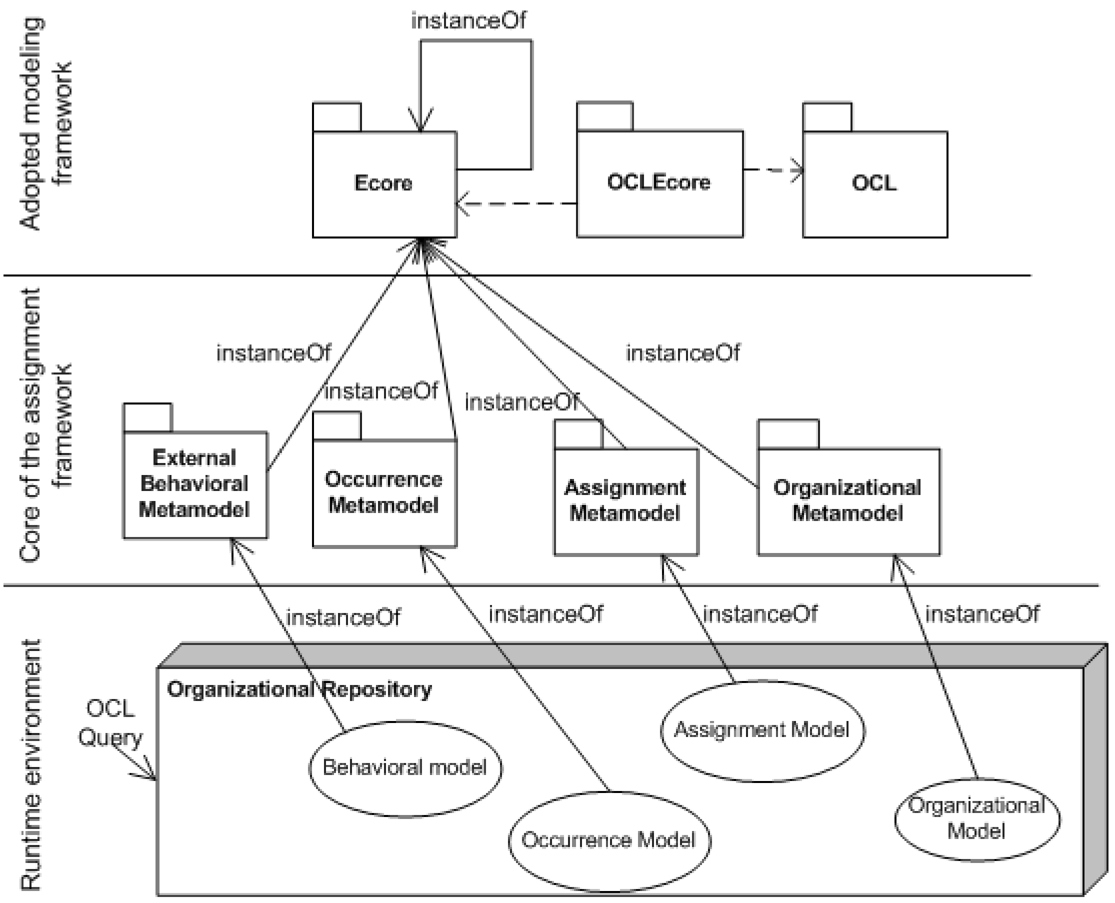


**Figure 15 - Assignment Framework Architecture**

Figure 16 shows the distinct phases the models are defined in the environment. 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).

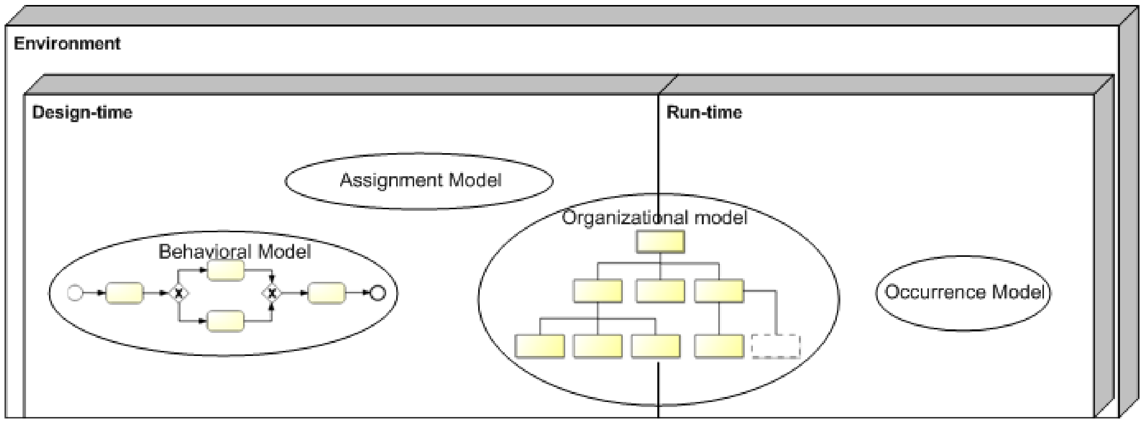


**Figure 16 - Design-time and Run-time models**

The framework is designed such that it can be applied as a lightweight extension to existing technologies. As a consequence, the assignment metamodel is built to be as loosely-coupled as possible. Its concepts are used to assign the behavioral to the organizational elements, therefore we cannot avoid specifying this dependency relationship in the metamodel level at some degree.  Hence, the assignment metamodel is not entirely independent of the behavioral and organizational metamodel that may be chosen because we still need to have an insight of the behavioral and active structure elements present in the metamodels to be able to define which one of them may be covered by the assignment metamodel.

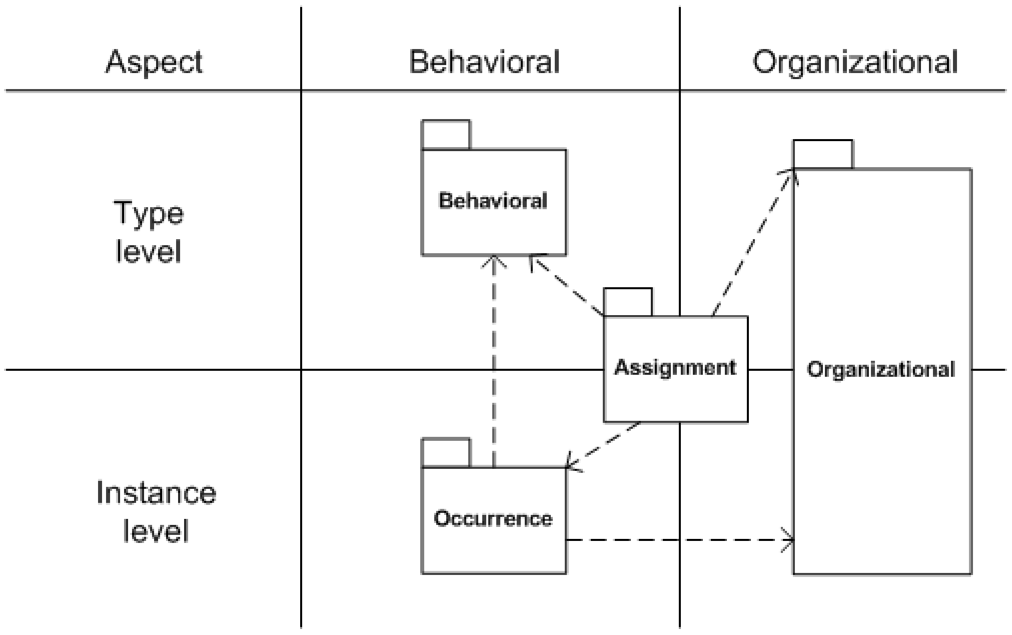


**Figure 17 - The different metamodeling levels and their dependencies**

Figure 17 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 type 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 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 (i.e., are instances of metaclasses in the occurrence metamodel).

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 3.4 and 3.5 employing an abstraction of the various behavioral metamodels and the generic capabilities of EMF.)

## 3.3 Organizational Metamodel

Many of the surveyed modeling techniques 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 (OMG, 2009)). Thus, we have consolidated many of these elements into an abstract organizational metamodel, which provides us with basic elements required for organizational-based assignments.

The metamodel was designed to provide more general organizational concepts while also covering all the desired requirements/assumptions. As discussed in Chapter 1, we focus on human resources, thus leaving out non-human resources from the model. The organizational metamodel is shown in Figure 18.

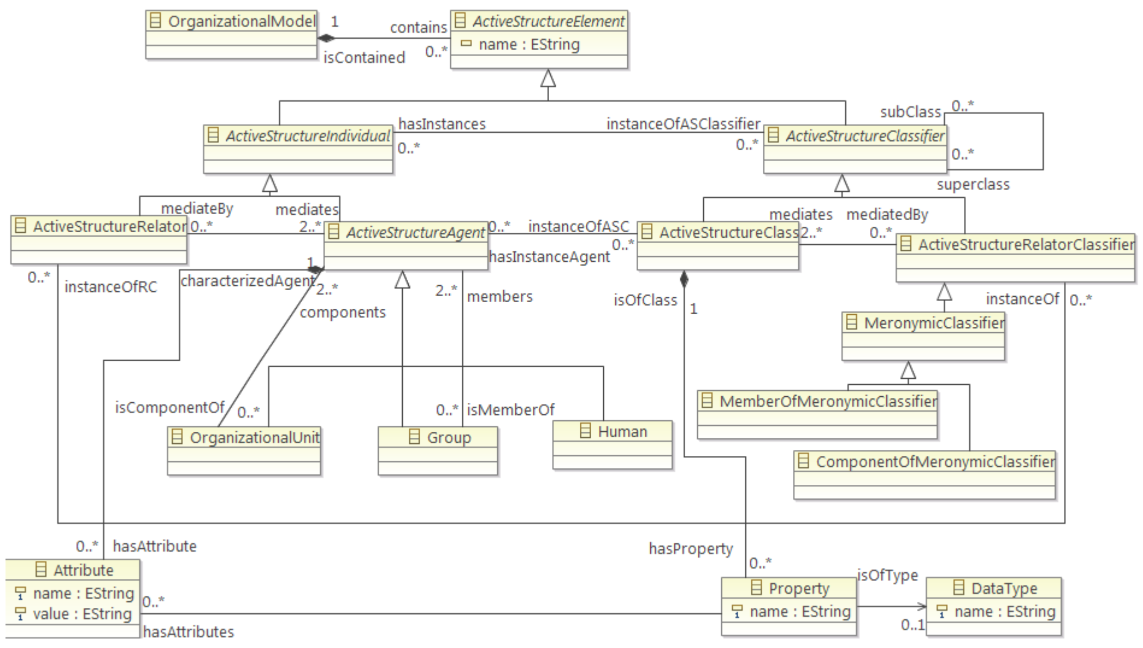


**Figure 18 - Organizational metamodel**

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. It also has an attribute called *name* of type String, defining that all ActiveStructureElements will be named. In a model, the value of this attribute must also be unique, i.e., there must not exist two elements with the same name, assuring that there won't be name clashes in an instantiating model. 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. A Human represents the persons that work in an organization. An OrganizationalUnit is composed of other ActiveStructureAgents, meaning it is a functional complex entity with parts (other ActiveStructureAgents) playing various roles. For instance, we may have the OrganizationalUnit 'Petrobras' that is composed of the OrganizationalUnits 'Engineering Department' and 'Human Resources Department'. When the agents that are components of an OrganizationalUnit change or even cease to exist, the OrganizationalUnit will remain the same. An

OrganizationalUnit also has the notion of persisting through time. A Group is also a whole to ActiveStructureAgents, but the difference to an OrganizationalUnit is that a group is considered temporary, being constituted with mandates for specific tasks. In addition, a group may be a functional complex or a collective (GUIZZARDI, 2005), i.e., the entities that are members of a Group may play the same role in the scope of the group. For instance, we may have the Group 'Project X Committee', which has as members Humans that perform the 'Functional Manager' or 'Project Manager' role that will analyze the feasibility of the specified project; or we may have a Group of Humans that perform the 'Programmer' role, in this case all the agents composing the Group play the same role. In case the Group is a functional complex, if some ActiveStructureAgent leaves the group or cease to exist, the identity of the Group also changes. A Group also has the notion of being temporary, meaning that it will not persist through time.

An ActiveStructureAgent may also have Attributes that characterizes then. 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'. 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 represents the specific data type of Property. For instance, we have the Property 'Experience', which is of the type 'Integer'. This Property must be of one ActiveStructureClass, for instance an ActiveStructureClass called 'Position'. An ActiveStructureRelatorClassifier represents a relation between two or more ActiveStructureClasses. For instance, consider a ‘Supervision’ ActiveStructureRelatorClassifier, which mediates the ActiveStructureClasses ‘Professor’ and ‘Master Student’. This mediates relationship is nonUnique, such that we can model a relationship with elements which are instances of the same ActiveStructureClass. For instance, we may have a ‘reports To’ relationship between persons (instances of the same ActiveStructureClass ‘Person’). An ActiveStructureRelatorClassifier may be further specialized into a MeronymicClassifier, which in its turn may be further classified into an MemberOfMeronymicClassifier and ComponentOfMeronymicClassifier relator classifiers. The MemberOfMeronymicClassifier metaclass must be such that its relator instances mediate only a Group and its members. Similarly, the ComponentOfMeronymicClassifier metaclass must be such that its relator instances mediate only an OrganizationalUnit and its components. Thus, to ensure that, we defined two OCL invariant constraints, which are shown below:

```
context MemberOfMeronymicClassifier
   inv  validMember:
         instanceOfRC->collect(
         mediates->at(1))->forAll(
        oclIsKindOf(Group));
```

```
context ComponentOfMeronymicClassifier
   inv  validComponent:
         instanceOfRC->collect(
         mediates->at(1))->forAll(
        oclIsKindOf(OrganizationalUnit));
```

An ActiveStructureClassifier may also be superclass of another class. For instance, the ActiveStructureClass ‘Engineer’ is superclass of the ActiveStructureClass ‘Civil Engineer’. The ActiveStructureRelatorClassifier ‘Supervision’ is a superclass of the ActiveStructureRelatorClassifier ‘SupervisionMasterDegree’.

As the Assignment framework is designed to be as general and generic as possible, the organizational metamodel has been defined to enable flexibility concerning the typing of ActiveStructureAgents, Attributes and ActiveStructureRelators. In other words, one may choose at model level whether to type those elements or not. Concerning ActiveStructureAgents, we may have, for instance, Humans that are instance of the ActiveStructureClass ‘Professor’, while others being instances of the ActiveStructureClasses ‘Manager’, or others that are instance of both, or we may also have Humans that don’t instantiate any ActiveStructureClass at all. This flexibility allows us to cope with approaches that are typed and those that do not include the concept of a type for organizational resources. Attributes represent particularized properties such as Joao Paulo's experience as a professor (measured in years), (let us call this 'JoaoPauloExperienceAsProfessor') which is an Attribute of the Human 'Joao Paulo', and this Attribute may also be instance of the Property 'Experience', which is of the Datatype 'Integer'. So the value of the Attribute that is instance of this must be parsed to an integer type. Concerning ActiveStructureRelators, we may have, for instance, an ActiveStructureRelator 'SupervisionJoaoPauloCarlos' that mediates the Humans 'Joao Paulo' and 'Carlos'. The mediates eReference owned by an ActiveStructureRelator is ordered, such that the various roles in the relationship can be distinguished even if the relator is not typed. If the relator is typed (i.e., when it is related to an ActiveStructureRelatorClassifier) the mediated elements must respect the types of the mediates eReference owned by an ActiveStructureRelatorClassifier. For example, if there is a 'SupervisionJoaoPauloCarlos' ActiveStructureRelator which mediates the Humans 'Joao Paulo' and 'Carlos', with this ActiveStructureRelator being instanceOf the 'Supervision' ActiveStructureRelatorClassifier, then 'Joao Paulo' and 'Carlos' must conform to the ActiveStructureClasses that the ActiveStructureRelatorClassifier is mediating.

## 3.4 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* that 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).

## 3.5 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 of these past executions and its main elements are shown in Figure 19.

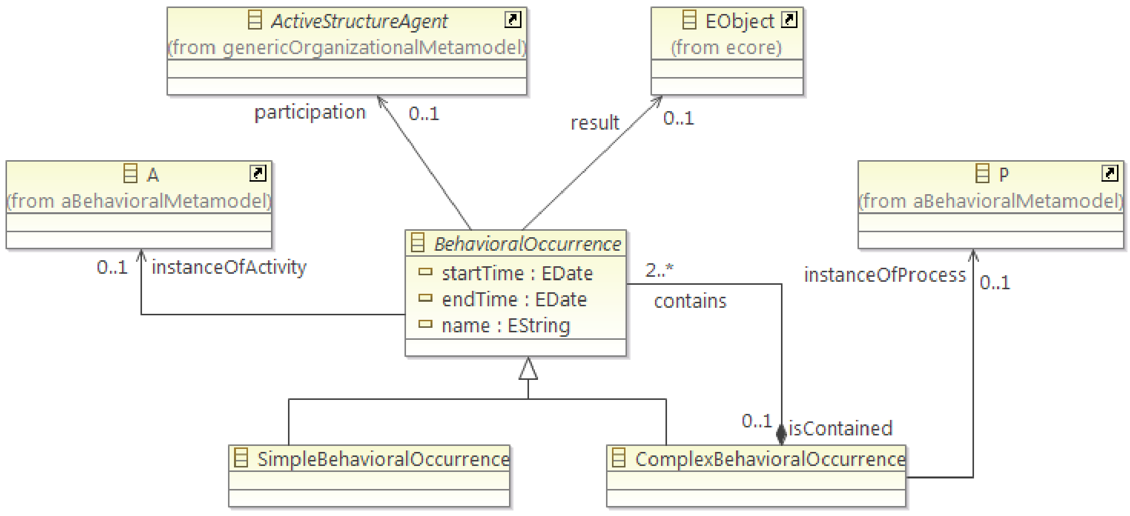


**Figure 19 - The Behavioral Occurrence Metamodel**

The main element of the metamodel is the BehavioralOccurrence abstract metaclass, which represents the actual occurrence of some behavior. It has a start date and time and an end date and time, with the former being the point in time in which the behavior begins to occur and the latter being the point in time in which the behavior ceases to occur.

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 of the organizational metamodel previously presented, meaning that the ActiveStructureAgent is responsible for the performance of that BehavioralOccurrence.

Lastly, a result relationship has been included to represent the result of some piece of behavior. Given the generic nature of "results", this is typed with the generic metaclass EObject. In general, a behavior occurrence may create, change, select or destroy an EObject. The cases in which it creates, changes or selects an EObject may be relevant for the assignment of a future behavior which refers to the resulting EObject. We may have, for instance, an activity called 'Define the person to head the expedition', in which the BehavioralOccurrences of this activity will select an already existing Human that will be heading a future expedition.

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 ComplexBehavioralOcurrence may refer to either an activity through the instanceOfActivity relationship or refer to a process through the relationship instanceOfProcess. The fragment below shows an OCL invariant constraint named ‘eitherProcessOrActivityDefined’ to guarantee that:

```
context ComplexBehavioralOccurrence
  inv  eitherProcessOrActivityDefined:
       instanceOfActivity.oclIsUndefined() xor instanceOfProcess.oclIsUndefined()
```

In case it refers to an activity, it means that the activity being instantiated by the ComplexBehavioralOccurrence must be one that is not atomic, i.e., must be an activity that may be further decomposed in finer-grained behavior elements, like a sub-process in BPMN, which is an activity that is composed of other activities. In case it refers to a process, it means that the ComplexBehavioralOccurrence represents a single execution of the process being instantiated. The composing BehavioralOccurrences of a ComplexBehavioralOccurrence must have a start date and time that precedes those of each of the composing BehavioralOccurrences and an end date and time that follows those of each of the composing BehavioralOccurrences.

## 3.6 Assignment Metamodel

Figure 20 shows the Assignment metamodel.

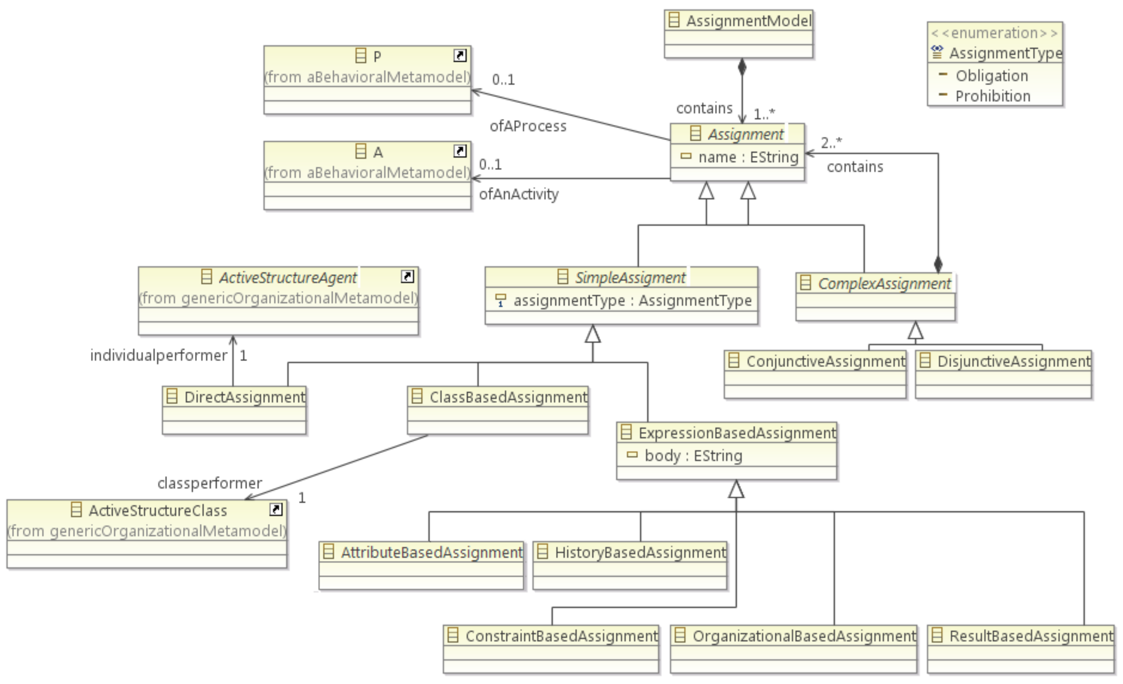


**Figure 20 - Assignment Metamodel**

An AssignmentModel represents the specification of assignments, including thus at least one Assignment, which captures the relation between the behavioral and organizational models.

Assignment is the top-level abstract metaclass and represents either a SimpleAssignment or a ComplexAssignment. There must be at most one Assignment for each behavior present in the behavioral model and it is the metaclass that establishes the relationship to the behavioral model through either the ofAnActivity or the ofAProcess relationships, one of which must be set for an Assignment. The fragment below shows an OCL invariant constraint named ‘eitherProcessOrActivityDefined’ to guarantee that.

```
context Assignment
  inv  eitherProcessOrActivityDefined:
      ofAnActivity.oclIsUndefined() xor ofAProcess.oclIsUndefined()
```

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).

In the sequel, we discuss the metaclass SimpleAssignment and its specializations. Subsequently, we discuss how SimpleAssignments may be used to compose ComplexAssignments.

### 3.6.1 SimpleAssignment

SimpleAssignment is an abstract metaclass that is further specialized into the various different types of SimpleAssignments, which we discuss in the following sections. 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 (e.g. must be an instance of that class, must be that specific agent, all depending on the specialization of SimpleAssignment).
- Prohibition, stating that the active structure element(s) referred to in the assignment **may not** perform the referred behavioral element (e.g. cannot be the one that performed a previous activity).

Often found in deontic logic theories, these two types are usually accompanied by a third type, namely permission, which we chose not to include as an explicit third type of Assignment because we assume in our model that everything that is not explicitly prohibited is permitted. In other words, permission is the default assignment type in the absence of assignments for a behavior element. As a consequence, the absence of an assignment model allows any active structure element to perform any behavior element, not constraining the performance of activities and processes in any way. 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.

### 3.6.2 DirectAssignment

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 to specify that the Human ‘Romário’ should be set as the performer of the activity ‘Analyze World Cup 2014 expense’, we should use a DirectAssignment of type Obligation. As so, ‘Romário’ will be responsible for the execution of every instance of the aforementioned activity. In the case of DirectAssignment involving an OrganizationalUnit, when we assign for instance the activity ‘Sign Contract’ to the OrganizationalUnit ‘Petrobras’, we mean that literally the OrganizationalUnit is responsible for the execution of the activity, even if in the end a Human will be the one that will perform the activity of signing the contract (acting in the name of the organization). Similarly, when there is a DirectAssignment to a Group, when we assign for instance the activity ‘Debug the source code’ to the Group ‘Debugging Programmers’, we mean that the entire Group is collectively responsible for the execution of the activity.

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. This type of Assignment would be used, for instance, if we would like to specify that ‘Roberto Jefferson’ is prohibited to perform the activity ‘Verify reports of corruption in government’. With this assignment, any other agent that is not ‘Roberto Jefferson’ could be chosen to be the performer of each occurrence of the activity. 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.

### 3.6.3 ClassBasedAssignment

A ClassBasedAssignment determines at design-time an ActiveStructureClass for the assignment. A ClassBasedAssignment 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. We would use a ClassBasedAssignment of type Obligation, for instance, if we would like to specify that an instance of the ActiveStructureClass ‘Professor’ must perform the activity ‘Analyze Students’ Exams’.

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 ActiveStructureClass 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. Figure 21 shows the possible outcomes of a ClassBasedAssignment of type Obligation at the moment of the evaluation of the assignment. When there is only one agent instantiating the referred ActiveStructureClass, the agent to be assigned will be fully determined and no further actions from the run-time infrastructure are required. This means that the assignment is then considered *independent of run-time policies*. When there are many agents that instantiate the referred ActiveStructureClass, the run-time infrastructure must choose one instance to be the performer based on its own policies. This means that the assignment is then considered *dependent of run-time policies*. Ultimately, when there is no agent that instantiates the referred ActiveStructureClass, the run-time infrastructure must invoke its exception handling mechanisms to deal with this case.

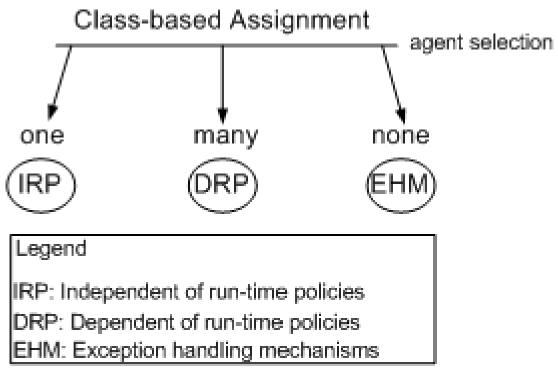

Class-based Assignment
agent selection
one
many
none
IRP
DRP
EHM
Legend
IRP: Independent of run-time policies
DRP: Dependent of run-time policies
EHM: Exception handling mechanisms

**Figure 21 - The possible outcomes of ClassBasedAssignment of type Obligation**

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. Specifically in the scenario where every agent is instance of the referred ActiveStructureClass, exception handling mechanisms from the run-time infrastructure would be required, since every single agent is prohibited. Finally, if there is only one agent that is not instance of the referred class, the assignment exactly determines which will be the performer.

In the case of a ClassBasedAssignment that refers to a non-atomic (higher-level) behavior, we should consider the consequences for the assignment of the contained (finer-grained) behaviors. In fact, we consider that assignments applied to non-atomic behaviors are propagated from the container behavior to the containing behaviors. This means that each assignment is evaluated separately (when each of the containing behaviors is enabled) and thus different agents may be chosen to perform each one of the containing behaviors.

### 3.6.4 ExpressionBasedAssignment

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. It is a metaclass that can be further specialized into various specific types, satisfying a variety of expressiveness requirements. But before proceeding to the types of ExpressionBasedAssignments defined, we firstly discuss the structure of the OCL expression that must be built.

The first thing that needs to be determined in an OCL expression is its context. In ExpressionBasedAssignments, the context 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 many of the requirements demand information that is only available during the performance of the behavior. We may want to reference information of the result of the execution of previous activities in the on-going instance of a process, or we may want to reference performers of previous activities in the on-going instance of a process. 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.

Another thing to note concerns the return type that the OCL expression may have. The OCL expression may return a single ActiveStructureAgent, a set of ActiveStructureAgents (which may also be an empty set) or a single ActiveStructureClass. Any other return types are considered invalid in the framework and would indicate an error in the specification of the assignment.

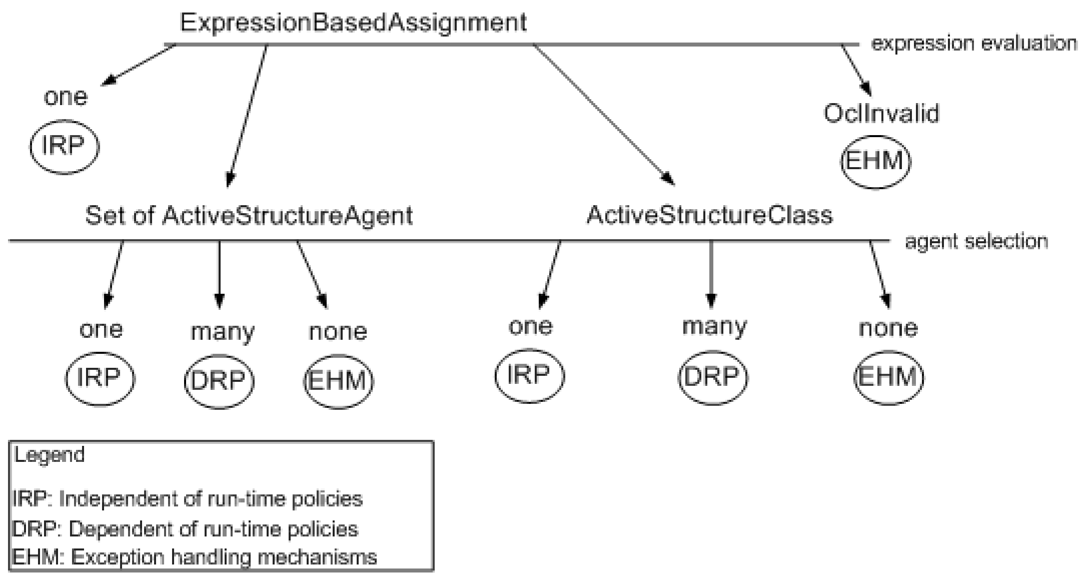

ExpressionBasedAssignment
expression evaluation
one
IRP
OclInvalid
EHM
Set of ActiveStructureAgent
ActiveStructureClass
agent selection
one
many
none
IRP
DRP
EHM
one
many
none
IRP
DRP
EHM
Legend
IRP: Independent of run-time policies
DRP: Dependent of run-time policies
EHM: Exception handling mechanisms

**Figure 22 - The possible outcomes of an ExpressionBasedAssignment of type Obligation**

Figure 22 shows all the possible outcomes an ExpressionBasedAssignment of type Obligation. If the evaluation of the expression returns a single ActiveStructureAgent, then the assignment is straightforward and is independent of run-time policies. If the evaluation of the expression returns a set of ActiveStructureAgents, and there is only agent in the set, then it is independent of run-time policies; if there are many agents in the set, then the run-time infrastructure must choose an agent in this set based on its own policies (i.e., the assignment is dependent of run-time policies); if it is an empty set, then the run-time infrastructure must invoke its exception handling mechanism to deal with this case. If the evaluation of the expression returns an ActiveStructureClass, then the same outcome discussed in the ClassBasedAssignment section applies here. Ultimately, if the evaluation of the expression returns the *OclInvalid* type (resulting from an invalid expression, e.g., when trying to select an element from an empty collection), the run-time infrastructure must invoke its exception handling mechanism to deal with this case.

In case of type Prohibition, if the assignment prohibits all agents, exception handling mechanisms are required; if it prohibits all but one agent, then there is an exact determination of the performer, and if the evaluation of the assignment prohibits just some agents, then the run-time infrastructure must choose an agent that is not prohibited based on its own policies.

Similarly to what is discussed in ClassBasedAssignment, an ExpressionBasedAssignment that refers to a non-atomic behavior will also propagate the assignment to the containing behaviors.

Having the context and the return types that the OCL expression must have, we now proceed to start how the expression itself may be written. That will entirely depend on what kind of expression one may want to write. We may want to write an expression that will select agents based on some criteria; or we may want to write an expression that will select agents based on the repository of execution of the processes (the previously finished BehavioralOccurrences); and, of course, we may want to write an expression that will select agents based on the information of the actual on-going BehavioralOccurrence of a process, which may be referred through the aforementioned context. These various kinds of ExpressionBasedAssignment are going to be addressed in the following sections.

In case we want to select the performer based on an organizational model, we first need to determine from which organizational metaclass we want to start navigating. If we want to select an ActiveStructureAgent, we should probably start the expression with ActiveStructureAgent.allInstances(). In fact, we may write an expression to assign a specific ActiveStructureAgent to the referred behavior of the assignment, like we already may do using a DirectAssignment. For example, if we would like to specify that the Human 'Romário' should be set as the performer of the activity 'Analyze World Cup 2014 expense', we could have written an ExpressionBasedAssignment of type Obligation that is composed of the following expression:

```
genericOrganizationalMetamodel::ActiveStructureAgent.allInstances()->select (name = 'Romário')
```

The expression firstly starts with the name of the organizational metamodel package, 'genericOrganizationalMetamodel', given that the context of the expression is actually in another package (*behavioraloccurrence*). We may also write an expression to define the assignment by means of an ActiveStructureClass, like with the ExpressionBasedAssignment of type Obligation that is composed of the following expression:

```
genericOrganizationalMetamodel::ActiveStructureClass.allInstances()->select (name = 'Professor')
```

Thus the evaluation of this expression returns a single ActiveStructureClass and thus the actual performer will be an ActiveStructureAgent that is instance of the resulting ActiveStructureClass, like in the ClassBasedAssignment.

Note that several expressions may have the same effect in terms of the implied assignment. For example, the assignment above (for "Romário"), could have been written as a prohibition with the following expression:

```
genericOrganizationalMetamodel::ActiveStructureAgent.allInstances()->select (
            name <> 'Romário')
```

This expression prohibits agents that are not the ActiveStructureAgent 'Romário' to perform the referred behavior. In other words, the performer of the referred behavior would have to be the ActiveStructureAgent 'Romário'.

### 3.6.5 AttributeBasedAssignment

AttributeBasedAssignment is a specific ExpressionBasedAssignment that defines an OCL expression that consists of selecting the ActiveStructureAgent to be assigned based on the Attributes it may have within an organization. For example, we may want to express that only Humans with at least one year of employment may be assigned to the activity 'Go on holidays'. To do so, we may have an AttributeBasedAssignment of type Obligation that consists of the following expression:

```
genericOrganizationalMetamodel::Property.allInstances()->select (
  name = 'Employment')->collect(hasAttributes)->select (
         value.toInteger()>1).characterizedAgent
```

This expression firstly selects the Property that is named 'Employment' and then collects every Attribute that is instance of the Property, filtering the ActiveStructureAgents that has the value of the attribute greater than the value of one.

Similarly, one could express the same constraint using an AttributeBasedAssignment of type Prohibition. The difference in the expression would be that instead of selecting the agents that have one year or more of employment, we would select the ones that have less than one year.

### 3.6.6 ConstraintBasedAssignment

ConstraintBasedAssignment is a specific ExpressionBasedAssignment that defines an OCL Expression that consists of selecting the ActiveStructureAgent to be assigned based on the execution of a previous behavior (a previous BehavioralOccurrence) in the same "case" (within the same higher-level BehavioralOccurrence). A ConstraintBasedAssignment of type Prohibition determines that the ActiveStructureAgent that performed a previous instance of an activity is not allowed to perform the activity, thus directly supporting the Separation of Duties pattern (RUSSEL et al., 2010). For example, we may want to determine that the performer of the activity 'Analyze report' must not be the same agent that performed the previous instance of the activity 'Write report'. To do so, we may write a ConstraintBasedAssignment of type Prohibition that consists of the following expression:

```
self.isContained.contains->select(
        instanceOfActivity.name = 'Write report').participation
```

This expression firstly navigate through the BehavioralOccurrence of the process that contains the BehavioralOccurrence of the activity 'Analyze report', then it selects the BehavioralOccurrence of the activity 'Write report' and navigate through the participation relationship, returning the ActiveStructureAgent which has performed the 'Write report' activity in the given BehavioralOccurrence of the process, he is the one that is prohibited of performing 'Analyze report'. This ExpressionBasedAssignment starts with 'self', indicating that the context of interpretation is the behavior occurrence which is the subject of the assignment.

A ConstraintBasedAssignment of type Obligation determines that the ActiveStructureAgent that performed a previous instance of an activity must be the one to perform the activity in which we are defining the assignment, thus directly supporting the Retain Familiar pattern (RUSSEL et al., 2010). We may write different ConstraintBasedAssignments with the same effect by varying the AssignmentType. We could have written an expression of type Obligation and selected the agents that are not the one that performed the previous activity.

### 3.6.7 HistoryBasedAssignment

HistoryBasedAssignment is a specific type of ExpressionBasedAssignment that defines an OCL expression that consists of selecting the ActiveStructureAgent to be assigned based on the history of ActiveStructureAgents in relation to the previous performances of behaviors (e.g. how many times the agent participated in BehavioralOccurrences of an activity). For example, we may want to assign the ActiveStructureAgent who has performed the activity 'Manage Project' more than ten times (the one who has more than ten participations in BehavioralOccurrences of this activity) to be the one that will perform the activity 'Explain the basics of project management'. To do so, we may have a HistoryBasedAssignment of type Obligation that consists of the following expression:

```
let manageProject:
Set (SimpleBehavioralOccurrence) =
SimpleBehavioralOccurrence.allInstances()->select (
        instanceOfActivity.name = 'Manager Project')
in
manageProject.participation->asSet()->select (
        agent | manageProject->select (
                participation = agent)->size() >= 10)
```

This expression firstly select all the SimpleBehavioralOccurrences and select the ones that are instances of the activity 'Manage Project' and then get the set of all the ActiveStructureAgents that participated in BehavioralOccurrences of the activity. Then we count the times that each ActiveStructureAgent participated in the BehavioralOccurrences of the referred activity and then select the ones that participated more than times in the BehavioralOccurrences of the referred activity.

An assignment with the same effect could have written with a HistoryBasedAssignment of type Prohibition specifying an expression that selects every ActiveStructureAgent that performed the activity ten times or less.

### 3.6.8 OrganizationalBasedAssignment

OrganizationalBasedAssignment is a specific type of ExpressionBasedAssignment that defines an OCL expression that consists of selecting the ActiveStructureAgent to be assigned based on the organizational relationships that it has with other organizational concepts. For example, we may want to assign the ActiveStructureAgent who is the manager of the Human ‘Carlos’ to be the one that will perform the activity ‘Analyze student performance’. To do so, we may have an OrganizationalBasedAssignment of type Obligation that consists of the following expression:

```
genericOrganizationalMetamodel::ActiveStructureRelatorClassifier.allInstances()->select(
        name='Management')->any(true).instanceOfRC->select(
                mediates->at(2).name='Carlos').mediates->at(1)
```

This expression firstly select the ActiveStructureRelatorClassifier named ‘Management’ and then navigates through all the ActiveStructureRelators of it and then select the ones in which ‘Carlos’ is the second element of the mediation relationship (we assume the convention that the second element of a ‘Management’ relationship is the element being managed.) Finally, we select his/her managers by picking up the first agent in the ActiveStructureRelator mediation relationship.

### 3.6.9 ResultBasedAssignment

ResultBasedAssignment is a specific type of ExpressionBasedAssignment that defines an OCL expression that consists of selecting the ActiveStructureAgent to be assigned based on the result of the instance of a previous behavior (the result of a previous BehavioralOccurrence). For example, we may want to assign the Group ‘Management Committee’ that was the result of the previous instance of the activity ‘Define Committee’ to be one that will perform the activity ‘Analyze Project Feasibility’. To do so, we may write a ResultBasedAssignment of type Obligation that consists of the following expression:

```
self.isContained.contains->select(
        instanceOfActivity.name = 'Define Committee').result
```

This expression firstly navigate through the BehavioralOccurrence of the process that contains the BehavioralOccurrence of the activity ‘Analyze Project Feasibility’, then it select the BehavioralOccurrence of the activity ‘Define Committee’ and selects the Group through the result of the BehavioralOccurrence.

### 3.6.10 SimpleAssignments and determinism levels

As we briefly discussed in some of the types of SimpleAssignment before, there is a varying level of determinism concerning the range of agents that may perform the assigned behaviors in the various types of assignment. Figure 23 summarizes the types in regards to this.

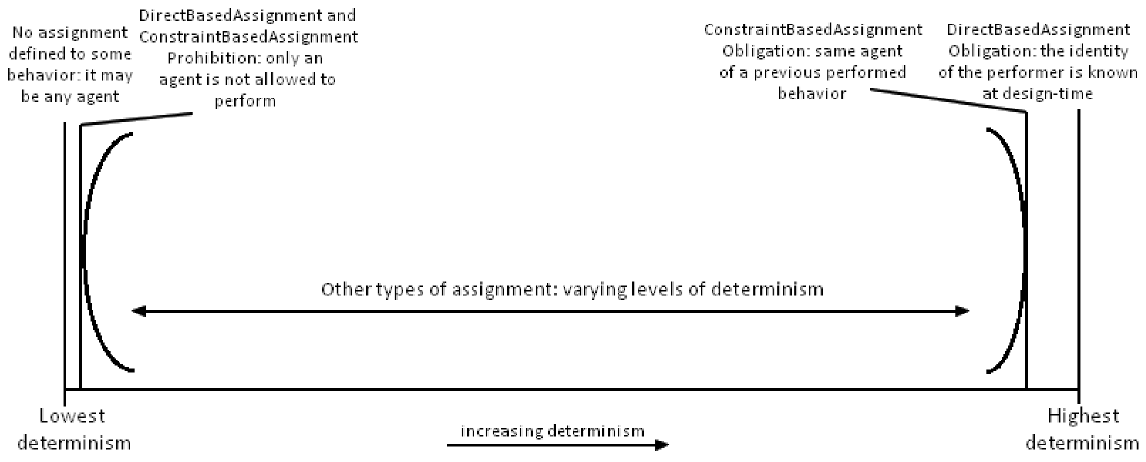

No assignment defined to some behavior: it may be any agent
DirectBasedAssignment and ConstraintBasedAssignment Prohibition: only an agent is not allowed to perform
ConstraintBasedAssignment Obligation: same agent of a previous performed behavior
DirectBasedAssignment Obligation: the identity of the performer is known at design-time
Other types of assignment: varying levels of determinism
Lowest determinism
increasing determinism
Highest determinism

**Figure 23 - Determinism levels**

Note that we are talking about determinism in real-world scenarios of organizations, so we consider them to be populated by many agents, and these agents playing various roles, and so on. Extreme and unlikely scenarios like an organization composed of one or few agents are excluded from this assessment.

The highest level of determinism comes from the DirectAssignment of type Obligation: the identity of the performer will be defined at design-time, and the chosen agent will be responsible for the execution of every instance of that behavior. On the other hand, the lowest level of determinism comes when there is no assignment defined to a behavior: any agent may be chosen to be the performer. A DirectAssignment of type Prohibition has the next lowest level of determinism: doing an assignment of this type, we are only forbidding at design-time a specific agent of an organization to be able to perform some behavior.

A ConstraintBasedAssignment is slightly less deterministic than a DirectAssignment in both assignment types: in type Obligation, the agent that will be the performer of some behavior will be the one that performed a previous behavior. But unlike DirectAssignment, the identity of the agent is unknown at design time because it depends on who performed the previous instance of the previous behavior, and this agent may also change because of this nature in each occurrence of the behavior. In type Prohibition, the agent that will be prohibited to perform some behavior will be only known at run-time.

In the case of ClassBasedAssignments and the other ExpressionBasedAssignments (AttributeBasedAssignment, HistoryBasedAssignment, OrganizationalBasedAssignment and ResultBasedAssignment), determinism varies greatly. The highest level of determinism from these would come with an expression which chooses only one agent in the end. In the case of a ClassBasedAssignment, the highest level of determinism would come when the ActiveStructureClass chosen is one that is instantiated by only a few agents in an organization and it is of type Obligation. For instance, we may want the president of a company to perform the activity 'Sign Marketing Contracts'. Usually, there is only one Human at an organization that is instance of the ActiveStructureClass 'President', so there is a high level of determinism in this case.

For the ExpressionBasedAssignments, the AssignmentType does not directly interferes the level of determinism. This is because an ExpressionBasedAssignment with different type (Obligation or Prohibition) may have the same effect (as we discussed in the ExpressionBasedAssignment and its types sections).

### 3.6.11 ComplexAssignment

A ComplexAssignment is an abstract metaclass that specifies types of Assignment which contains two or more assignments to a specific behavior. It may be a ConjunctiveAssignment or a DisjunctiveAssignment. We will see how each one of these may be used in the following.

### 3.6.12 ConjunctiveAssignment

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. For example, we may have a ConjunctiveAssignment composed of a AttributeBasedAssignment 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, as we may want to combine Assignments with different AssignmentTypes. For instance, we may write a ConjunctiveAssignment to the activity 'Analyzing World Cup 2014 expenses' with a DirectAssignment of type Prohibition, selecting the agent 'Joao Paulo Cunha' and a ClassBasedAssignment of type Obligation selecting the ActiveStructureClass 'Deputy'.

Nevertheless, some assignments types are conflicting with others, so they must not be allowed to be in a ConjunctiveAssignment. For example, it makes no sense to define a conjunction between two DirectAssignments of type Obligation that refer to different agents. Table 3 shows the possible outcomes of the conjunction of a pair of assignments based on their evaluations (which may select one, many or no agents) and based on their types (Obligation or Prohibition). Since conjunction is commutative, the table is symmetric along the diagonal.

**Table 3 - The possible outcomes of two of assignments in a ConjunctiveAssignment**

| | Obligation One agent B | Obligation Many agents (set $S_B$) | Obligation No agent | Prohibition (One, Many, No agent) |
|---|---|---|---|---|
| Obligation One agent A | IRP if A and B are the same agent. EHM otherwise. | IRP if A is contained in $S_B$. EHM otherwise. | EHM | IRP if A is not prohibited. EHM if so. |
| Obligation Many agents (set $S_A$) | IRP if B is contained in $S_A$. | IRP if only one agent intersects. | EHM | IRP if prohibition applied to all agents but one in the |

| | | | | |
|---|---|---|---|---|
| | EHM otherwise. | DRP if a set of agents intersects.<br><br>EHM if there is no intersection. | | obligation set.<br><br>DRP if prohibition does not apply to two or more agents in the obligation set.<br><br>EHM if prohibition applies to all agents in the obligation set. |
| Obligation No agent | EHM | EHM | EHM | EHM |
| Prohibition (One, Many, No agent) | IRP if B is not prohibited.<br><br>EHM if so. | IRP if prohibition applied to all agents but one in the obligation set.<br><br>DRP if prohibition does not apply to two or more agents in the obligation set.<br><br>EHM if prohibition applies to all agents in the obligation set. | EHM | IRP if the union of both prohibitions include all agents but one.<br><br>DRP if the union of both prohibitions excludes at least two agents in the set.<br><br>EHM if the union of both prohibitions includes all agents. |
| Legend:<br>IRP: Independent of run-time infrastructure<br>DRP: Dependent of run-time infrastructure<br>EHM: Exception handling mechanism | | | | |

The first line of the table shows the possible outcomes when the first composing assignment is of type Obligation and returns only one agent. If the second composing assignment is also of type Obligation and returns only one agent, if it is the same agent then no further actions from the run-time infrastructure are required. Otherwise, exception handling mechanisms of the runtime infrastructure are required. If the second composing assignment is also of type Obligation and returns a set of agents, if the agent of the first composing assignment is contained in the set then no further actions from the run-time infrastructure are required. Otherwise, exception handling mechanisms are required. If the second composing assignment is also of type Obligation and returns no agent, then exception handling mechanisms are required. If the second composing assignment is of type Prohibition, then if it does not prohibit the agent obliged of the first assignment, then no further actions from the run-time infrastructure are required. Otherwise, exception handling mechanisms are required.

New combination scenarios include when the first and the second assignments are of type Obligation and both return many agents. If the intersection of the both sets is only one agent, then no further actions from the run-time infrastructure are required. If the intersection is a set of agents, then the run-time infrastructure must choose one agent of the set based on its own policies. If the intersection is empty, exception handling mechanisms are required. If the first composing assignment is of type Obligation and return many agents and the second is of type Prohibition, prohibiting every single but one agent, then no further actions from the run-time infrastructure are required. If it does not prohibit two or more agents of the first composing assignment, then the run-time infrastructure must choose one of them based on its own policies. If it prohibits every single agent of the first composing assignment, then exception handling mechanisms are required. In case one of the composing assignments is of type Obligation and when evaluated selects no agents, then exception handling mechanisms are always required. If the two composing assignments are of type Prohibition, the conjunctive assignment is independent of run-time policies if the union of both prohibition assignments includes all agents but one. If the union of both composing assignments of type Prohibition includes agents but leaves at least two agents out of it then the run-time infrastructure must choose one that is not on the set of the union based on its own policies. If the union includes all agents, then exception handlings mechanisms must be invoked.

We generalize this analysis for conjunctive assignments with more than two composing assignments, by first considering every assignment of type prohibition $AP_i$ to have the same effect of a corresponding assignment of type obligation ($AO_i$) that selects all agents that were not selected in $AP_i$. In this view, the resulting conjunctive assignment includes only assignments of type obligation, including original assignments of type obligation and those that represent corresponding assignments of type prohibition ($AO_i$). The set of possible performers for the conjunctive assignment is given by the intersection of all the agents selected by the composing assignments. If there is at least one composing assignment that selects no agents, then exception handling mechanisms are required (as the intersection is empty). If the intersection of the composing assignments results in only one agent, then no further actions from the run-time infrastructure are required, as the assignment is fully determinate. If the intersection of the composing assignments has two or more agents, then the run-time infrastructure must choose one of these based on its own policies.

### 3.6.13 DisjunctiveAssignment

DisjunctiveAssignment is a specific type of ComplexAssignment indicating that at least one of the composing assignments must be satisfied during the run-time evaluation of the composing Assignments. For example, we may have a DisjunctiveAssignment composed of a ClassBasedExpression which indicates that the performer must have a civil engineering role or a AttributeBasedAssignment indicating that the performer must be a human with at least 5 years of experience in managing construction site projects. The assignments in a DisjunctiveAssignment result in a complex assignment with a lower level of determinism (when compared to the composing assignments).

Similary to the case of ConjunctiveAssignment, we consider the effects of DisjunctiveAssignment by first considering every assignment of type prohibition $AP_i$ to have the same effect of a corresponding assignment of type obligation ($AO_i$) that selects all agents that were not selected in $AP_i$. In this view, the resulting disjunctive assignment includes only assignments of type obligation, including original assignments of type obligation and those that represent corresponding assignments of type prohibition ($AO_i$). The set of possible performers for the disjunctive assignment is given by the *union* of all the agents selected by the composing assignments. If the union of the composing assignments results in only one agent, then no further actions from the run-time infrastructure are required, as the assignment is fully determined. If the union of all composing assignments is empty, then exception handling mechanisms are required. Lastly, if the union of the composing assignments results in a set with at least two agents, then the run-time infrastructure must choose an agent based on its own policies.

### 3.6.14 Assignments specified at different levels of abstraction

When the assignment refers to a high-level behavioral element (such as a process), that means the assignment is applicable to all finer-grained elements (ultimately atomic behavior elements) that are contained within the high-level behavioral element. For instance, if one defines a ClassBasedAssignment for a process containing several activities, the semantics of this assignment is equivalent to the semantics of several ClassBasedAssignments for each of the activities within that process. In other words, the assignment applies transitively with respect to the behavior refinement hierarchy.

If more than one assignment is applied (directly or transitively) to a behavioral element, the semantics is equivalent to that of a ConjunctiveAssignment. Thus, the assignments defined for higher-level behavioral elements and the assignment defined for the finer-grained behavioral element must be considered in conjunction to determine the semantics of the assignment. This implies that, there must be consistency through the "chain" of assignments at the various levels of behavior refinement in order to avoid the specification of assignments with undefined semantics.

## 3.7 Evaluation

Table 4 describes how each of the considered Workflow Resource Patterns is satisfied in the assignment framework.

**Table 4 - Support of workflow resource patterns in the assignment framework**

| Patterns | Covering Metamodel Concepts | Support description |
|---|---|---|
| WRP-01: Direct Distribution | DirectAssignment | The pattern is supported by the DirectAssignment metaclass (of type Obligation). This assignment specifies the human agent which will be responsible for every single execution of the referred behavior. We could also specify a direct assignment to a group of human agents defined in the organizational model, so that will mean that the group as a whole will be responsible for the execution of the referred behavior. |
| WRP-02: Role-Based Distribution | ClassBasedAssignment | The pattern is supported by the ClassBasedAssignment metaclass (of type Obligation), in which the assignment specifies a class defined within the organization and thus only agents that are instances of that class perform the activity. |
| WRP-03: Deferred Distribution | ResultBasedAssignment (with expressions) | The pattern is supported partially at design time by the absence of assignment for the behavioral element. ResultBasedAssignment may also be considered as a kind of Deferred Distribution because the resource to which a work item will be allocated will only be known at run-time, and it may change based on the result of previous work items.<br>Full support for this pattern is outside the scope of this work, as it concerns runtime support. |

| | | |
|---|---|---|
| WRP-04: Authorization | SimpleAssignment, AssignmentType | SimpleAssignments have an AssignmentType, which may assume the values ‘obligation’ or ‘prohibition’. In the absence of an assignment defined to a behavior, any agent defined in the organizational model is permitted to perform the behavior. |
| WRP-05: Separation of Duties | ConstraintBased-AssignmentConstraint-BasedAssignment (with expressions) | The pattern is supported by the ConstraintBasedAssignment metaclass, in which the the assignment will be specified by the means of an OCL expression that will refer to the actual performer of a previous activity in a given instance of a process and will have the ‘prohibition’ AssignmentType. |
| WRP-06: Case Handling | ConstraintBased-Assignment (with expressions), (DirectAssignment at process level) | The pattern is supported by the ConstraintBasedAssignment of type Obligation, specifying that each activity of the process must be performed by a same agent. |
| WRP-07: Retain Familiar | ConstraintBased-Assignment | A more flexible pattern of the Case Handling pattern discussed before, this pattern is supported by the ConstraintBasedAssignment metaclass, in which the assignment will be specified by the means of an OCL expression that will refer to the actual performer of a previous activity in a given instance of a process and will have the Obligation AssignmentType. |
| WRP-08: Capability-based Distribution | AttributeBased-Assignment | The pattern is supported by the AttributeBasedAssignment metaclass, in which the assignment will be specified by the means of an OCL expression and will directly refer to the attributes that an agent has within an organization. |
| WRP-09: History-based Distribution | HistoryBased-Assignment | The pattern is supported by the HistoryBasedAssignment metaclass, in which the assignment will be specified by the means of an OCL expression and will directly refer to a repository of previous executions of the activities and process. |
| WRP-10: Organizationa l Distribution | Organizationalbased-Assignment | The pattern is supported by the metaclass OrganizationalBasedAssignment metaclass, in which the assignment will be specified by the means of an OCL expression and will directly refer to the relationships that agents have within an organization. |
| WRP-11 – Automatic Execution | - | Not directly supported by any of the assignments. |

Additional support includes the definition of complex assignments by the means of a ConjunctiveAssignment and a DisjunctiveAssignment.

# 4. Application to BPMN

In order to show the applicability of the general approach presented in Chapter 3 to a widely employed business process modeling technique, we present in this chapter 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.

Of all the process technologies analyzed in Chapter 2, we have opted for BPMN due to its wide adoption for communicating business processes (maximizing potential impact) and its simplistic mechanisms for the assignment of active structure to behavioral concepts (emphasizing that the extended technique will profit from significantly increase support).

In the sequel, we show the fragments of the BPMN metamodel that interest us, and show how they are integrated into the assignment framework. Since BPMN does not provide constructs for organizational modeling, we define a UML class diagram profile for the organizational metamodel concepts defined and presented in the previous section. This provides us with all the behavioral and organizational elements required. Finally, we present an example with a specific organizational model (concerning a university), a BPMN model (concerning the writing and defense of a master's work), and define the assignment model that will enrich the relationships between the behavioral aspects of the BPMN model to the organizational model.

## 4.1 BPMN Metamodel Integration

In BPMN, all the work that is performed in the scope of a particular business process is represented through the Activity concept (OMG, 2011), which is the abstract class for all the concrete Activity types, like a SubProcess and a Task. Thus, the Activity metaclass will be the direct target of the relationship instanceOfActivity of the BehavioralOccurrence metaclass, presented in section 3.5. It will also be the direct target of the ofAnActivity relationship of the SimpleAssignment metaclass presented in section 3.6.1.

In BPMN, Process is described 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 not considered to be a higher level Activity, it only is comprised of Activities, thus Process (graphically represented as Pools in collaboration diagrams) will also be a direct target of the instanceOfProcess relationship of the BehavioralOccurrence metaclass, presented in section 3.5. It will also be the direct target of the ofAProcess relationship of the SimpleAssignment metaclass presented in section 3.6.1.

Process is used in process diagrams and in collaboration Diagrams. In process diagrams, there is only one Process to each diagram defined. In collaboration diagrams, Processes represent the internal behavior of the Participants, which are graphically represented by Pools. There may exist many processes in collaboration diagrams, and assignments may be defined at their level.

We consider BPMN Choreography and Conversation diagrams outside the scope of this integration and focus here on Process Diagrams and the more general Collaboration Diagrams.

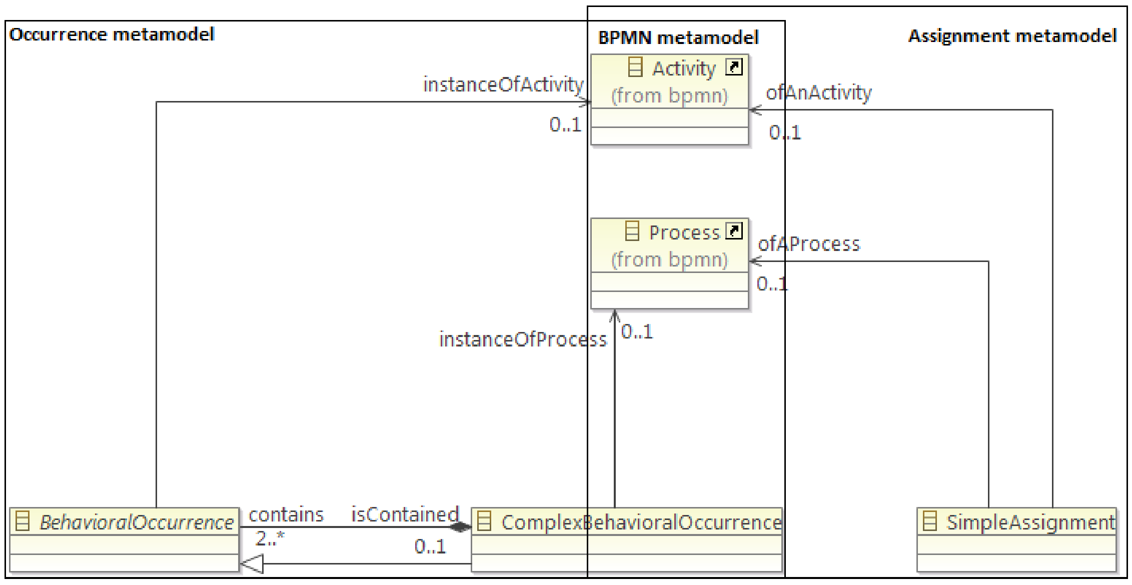


**Figure 24 - BPMN integration into the Assignment framework**

Figure 24 summarizes how the BPMN metamodel is integrated into the assignment framework. The assignment may be done through an Activity or a Process, and because of that, a BehavioralOccurrence will be or an instance of an Activity or an instance of a Process.

## 4.2 UML Class Diagram Profile for Organizational Modeling

We now define a UML Class Diagram Profile for the Organizational metamodel, allowing us to define Organizational models in a well-known concrete syntax. Figure 25 summarizes the profile using a UML 2.0 Profile Diagram.

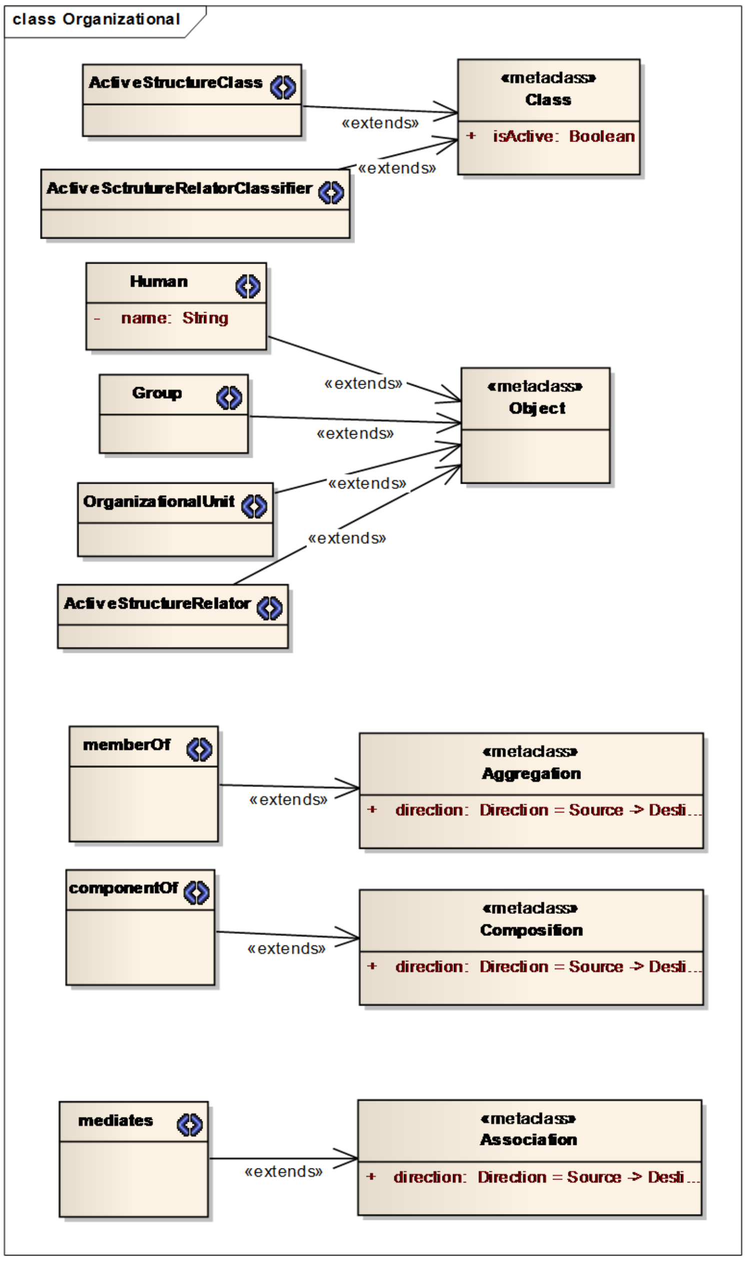


**Figure 25 - UML Class Diagram Profile to the Organizational Metamodel**

ActiveStructureClass and ActiveStructureRelatorClassifier are the only metaclasses at type level and both stereotypes extend the UML concept of Class. Human, Group, OrganizationalUnit and ActiveStructureRelator are entities at instance level, therefore, all of them extend the concept of an Object, i.e., an entity that is instance of a Class.

Regarding the meta-associations we adopt the following stereotypes:

(i) *componentOf*, which stereotype extends the Composition metaclass . When applied to relate UML InstanceSpecifications it concerns ActiveStructureAgents which are components of an OrganizationalUnit. When applied to relate UML Classes, it concerns a ComponentOfMeronymicClassifier relating ActiveStructureClasses;

(ii) *memberOf*, which stereotype extends the Aggregation metaclass. When applied to relate UML InstanceSpecifications it concerns ActiveStructureAgents which are members of a Group. When applied to relate UML Classes, it concerns a MemberOfMeronymicClassifier relating ActiveStructureClasses.

(iii) *mediates*, which stereotype extends the Association metaclass. It is applied to represent the relationship that exists between an ActiveStructureRelator and ActiveStructureAgents and between an ActiveStructureRelatorClassifier and ActiveStructureClasses.

There is a superclass relationship that is applicable to either an ActiveStructureClass or an ActiveStructureRelatorClassifier. As this relationship shares the same semantics of the standard generalization metaclass in UML class diagram, we use that instead of defining an additional stereotype in the profile. . The *instanceOf* relationship that agents have to classifiers is also natively supported by UML, as it is possible to specify the classifier of which an object is an instance.

Attributes and Properties will be represented directly by the Property metaclass from UML.

## 4.3 Example

We now present an example to show how the assignment framework may be used. We will first present a BPMN model, then an Organizational Model, and ultimately, the Assignment model, showing how we can build an expressive and complete set of assignments using the framework.

### 4.3.1 BPMN model

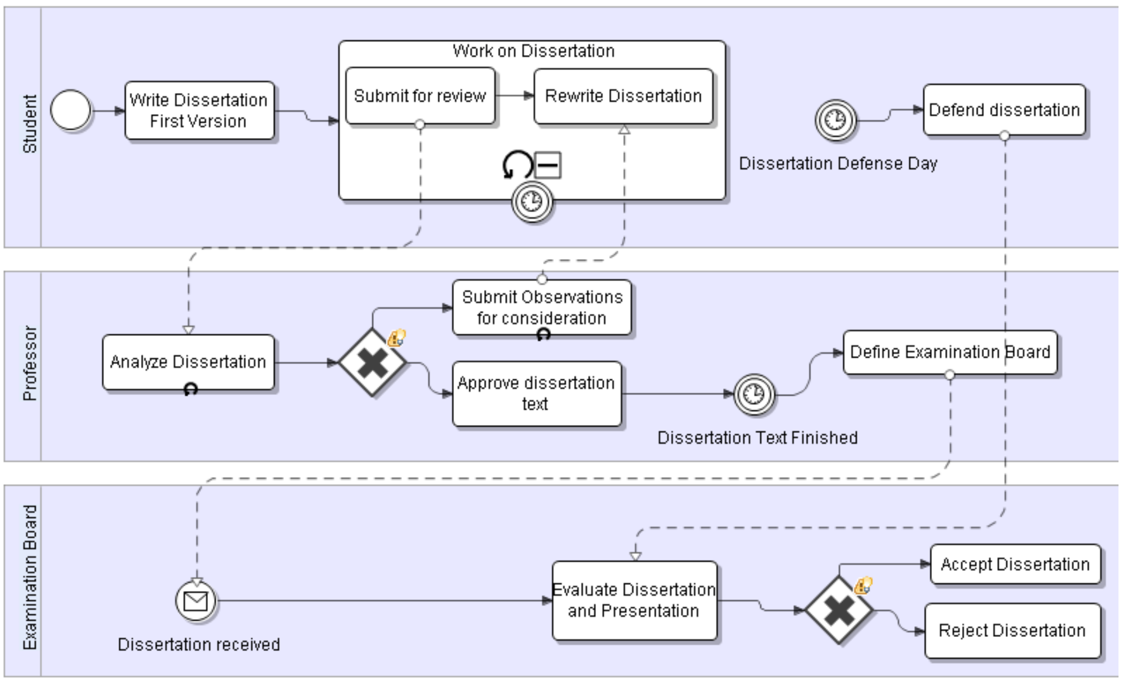


**Figure 26 - Example of a BPMN model**

Figure 26 shows the business process model that will be the subject of this illustration. 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 keep getting 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 to that dissertation's defense 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, ending the process.

### 4.3.2 Organizational Model

Figure 27 shows a small example showing an organizational model.

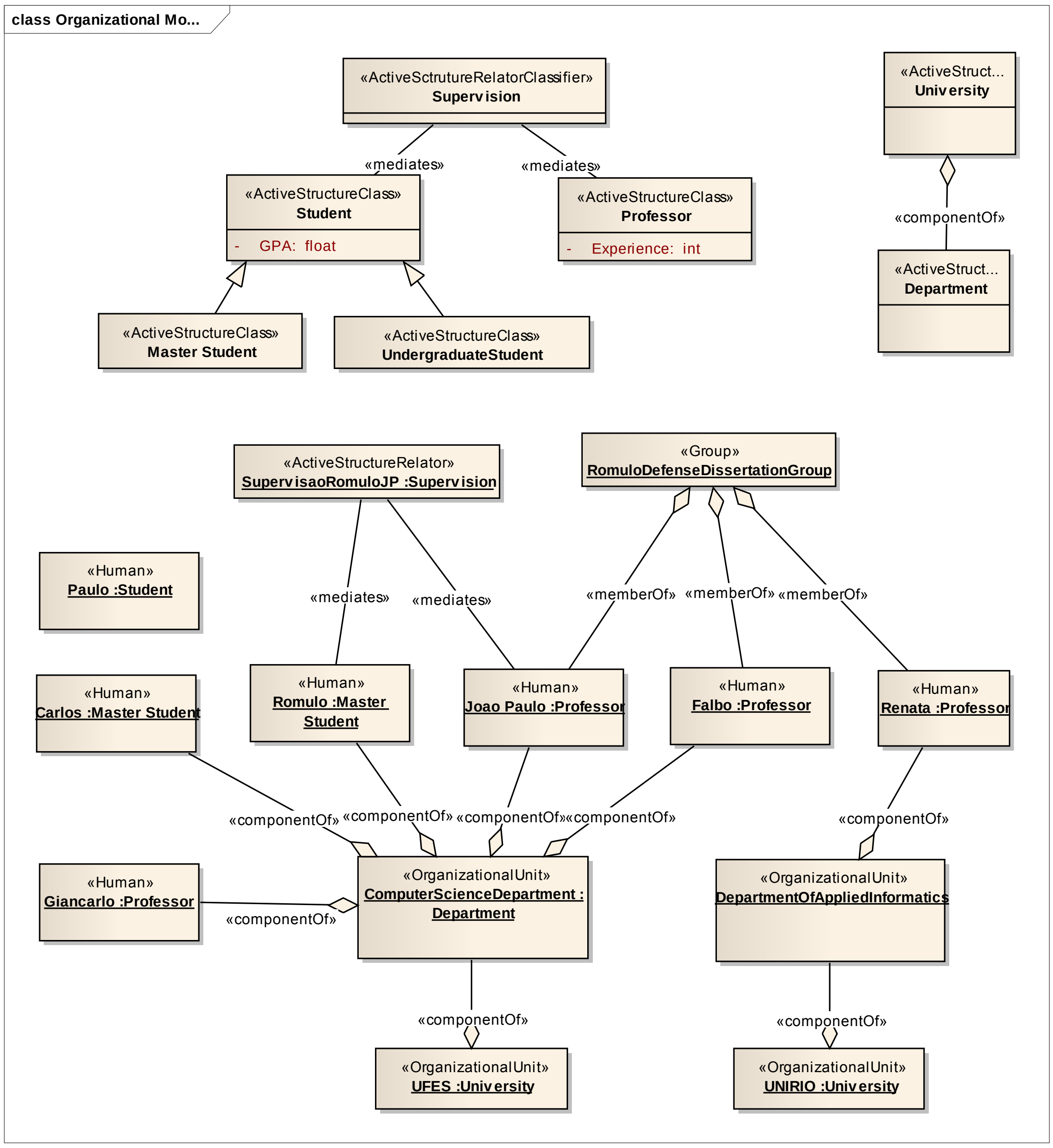

class Organizational Mo...
«ActiveSctrutureRelatorClassifier» Supervision
«ActiveStruct... University
«mediates»
«mediates»
«ActiveStructureClass» Student
- GPA: float
«ActiveStructureClass» Professor
- Experience: int
«componentOf»
«ActiveStruct... Department
«ActiveStructureClass» Master Student
«ActiveStructureClass» UndergraduateStudent
«ActiveStructureRelator» SupervisaoRomuloJP :Supervision
«Group» RomuloDefenseDissertationGroup
«Human» Paulo :Student
«mediates»
«mediates»
«memberOf»
«memberOf»
«memberOf»
«Human» Carlos :Master Student
«Human» Romulo :Master Student
«Human» Joao Paulo :Professor
«Human» Falbo :Professor
«Human» Renata :Professor
«componentOf»
«componentOf»
«componentOf»
«componentOf»
«componentOf»
«Human» Giancarlo :Professor
«componentOf»
«OrganizationalUnit» ComputerScienceDepartment : Department
«OrganizationalUnit» DepartmentOfAppliedInformatics
«componentOf»
«componentOf»
«OrganizationalUnit» UFES :University
«OrganizationalUnit» UNIRIO :University

**Figure 27 - Example of an organizational model**

The classifiers defined in this organizational model include the classes ‘Professor’, with the attribute ‘Experience’ typed as ‘Integer’, which will represent the experience in years of the ‘Professor’. ‘Student’ has the attribute ‘GPA’ typed as ‘Real’, representing the grade point average of a student. ‘Student’ is also further specialized in ‘Master Student’ and ‘Undergraduate Student’. ‘University’ has ‘Department’ as components. The only relator classifier defined was ‘Supervision’, which mediates the classes ‘Student’ and ‘Professor’.

The agents defined include the humans ‘Paulo’, which is instance of ‘Student’, ‘Romulo’ and ‘Carlos’, both being instances of ‘Master Student’ and finally, ‘Joao Paulo’, ‘Falbo’, ‘Giancarlo’ and ‘Renata’ being instances of ‘Professor’. ‘Joao Paulo’, ‘Falbo’ and ‘Renata’ are also members of the group ‘RomuloDefenseDissertationGroup’, which will have a brief existence in the organizational model. The relator ‘SupervisionRomuloJP’ mediates ‘Romulo’ and ‘Joao Paulo’.

‘Renata’ is a component of the ‘DepartmentOfAppliedInformatics’ department which in its turn is a component of the ‘UNIRIO’ University. The remaining humans (except ‘Paulo’) and the only group are components of the ‘ComputerScienceDepartment’ department, which in its turn is also a member of the UFES University.

### 4.3.3 Assignment Model

The following assignment constraints were identified when we designed the business process model:

- Activity ‘Write Dissertation First Version’
- The activity must be performed by a Master Student. Sub-Process ‘Work on Dissertation’, which encompasses the activities ‘Submit for Review’ and ‘Rewrite Dissertation’
    - The sub-process, i.e., all the activities contained within the sub-process, must be performed by the same agent that performed the previous activity ‘Write Dissertation First Version’.
- Activity ‘Defend Dissertation’

  - Must be performed by the same agent that performed the previous activities ‘Work on Dissertation’ and ‘Write dissertation first version’
  - Students that have a grade point average below 7.0 are not allowed to defend a dissertation, thus they cannot perform these activities.
- Activity ‘Analyze Dissertation’
  - The agent to be assigned should have performed this activity at least three times before.
  - The agent must have at least 5 years of experience as a professor.
  - The agent should be a supervisor of the specific student that wrote the dissertation (performed the previous activity ‘Write Dissertation’).
- Activity ‘Submit observations for consideration’
  - The activity must be performed by the same agent that performed the previous activity ‘Analyze Dissertation’.
- Activity ‘Approve Dissertation text’
  - The activity must be performed by the same agent that performed the previous activity ‘Analyze Dissertation’.
- Activity ‘Define examination board’
  - The activity must be performed by the same agent that performed the previous activity ‘Analyze Dissertation’.
- Pool ‘Examination Board’, which encompasses the activities ‘Evaluate Dissertation and Presentation’, ‘Accept Dissertation’, ‘Reject Dissertation’
  - The activity must be performed by the group that was defined in the previous activity ‘Define Examination Board’.

With these requirements in mind, we designed the assignment model. The result can be seen in Figure 28.

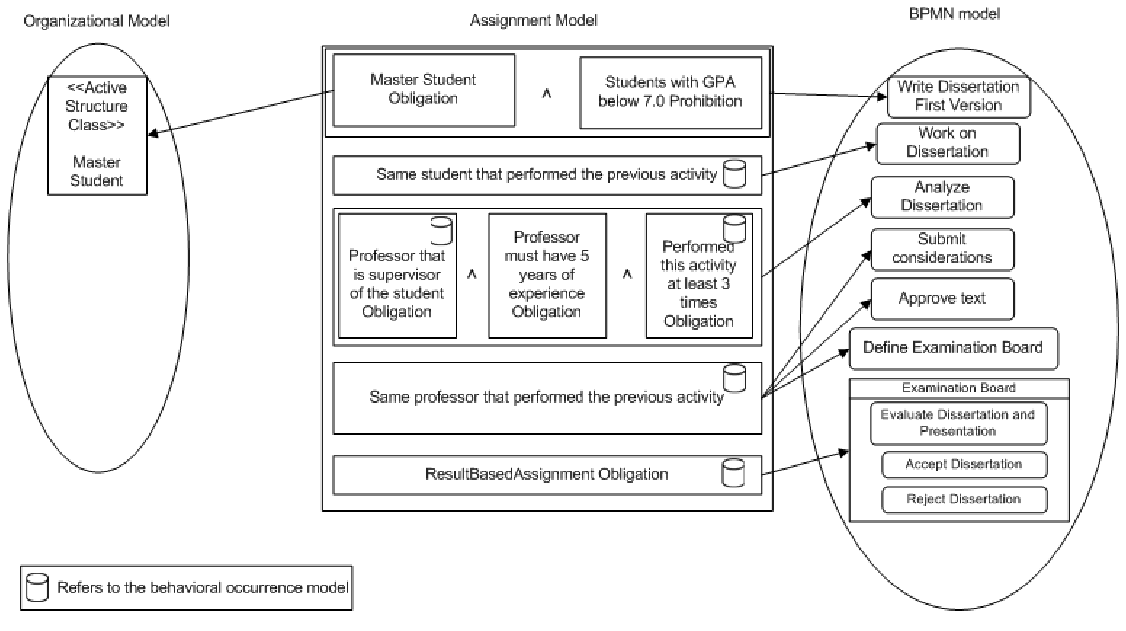

Organizational Model
Assignment Model
BPMN model
<<Active Structure Class>> Master Student
Master Student Obligation
∧
Students with GPA below 7.0 Prohibition
Same student that performed the previous activity
Professor that is supervisor of the student Obligation
Professor must have 5 years of experience Obligation
Performed this activity at least 3 times Obligation
Same professor that performed the previous activity
ResultBasedAssignment Obligation
Write Dissertation First Version
Work on Dissertation
Analyze Dissertation
Submit considerations
Approve text
Define Examination Board
Examination Board
Evaluate Dissertation and Presentation
Accept Dissertation
Reject Dissertation
Refers to the behavioral occurrence model

**Figure 28 - Example of an assignment model**

As seen in Figure 28, the mapping of the assignment constraints to the assignment model concepts was very straightforward: each assignment constraint resulted in an instance of the corresponding metaclass of the assignment metamodel. We now describe the model in details.

The first defined assignment is a ClassBasedAssignment of type Obligation. In this case, an ActiveStructureClass of the organizational model namely the 'Master Student' class is referred to in the assignment.

The next defined assignment is a ConstraintBasedAssignment of type Obligation to the sub-process 'Work on Dissertation', to ensure that the one who will perform this sub-process will be the same agent that performed the instance of the previous activity 'Write Dissertation First Version'. This assignment contains the following OCL expression:

```
self.isContained.contains->select(
  instanceOfActivity.name = 'Write Dissertation First Version')->any(true).participation
```

The context of the expression is self, which refers to the newly created behavioral occurrence of the sub-process 'Work on Dissertation'. It starts navigating through the behavioral occurrence of the process that contains the newly created behavioral occurrence, and then selects the agent which was the performer of the previous activity

'Write Dissertation First Version' ("any(true)" is required in the expression because "select" returns an OCL collection, even in the case in which only one element is contained in the collection).

We apply a ConjunctiveAssignment to the activity 'Defend Dissertation'. The first composing assignment in the conjunction is a ConstraintBasedAssignment of type Obligation, to ensure that the one who will perform this activity will be the same agent that performed the instance of the previous sub-process 'Work on Dissertation' and the activity 'Write Dissertation First Version'. As the agent that performed these two behaviors are the same, we can use the same expression defined at the sub-process 'Work on Dissertation' to this assignment:

```
self.isContained.contains->select(
  instanceOfActivity.name = 'Write Dissertation First Version')->any(true).participation
```

The second composing assignment is an AttributeBasedAssignment of type Prohibition, to ensure that a 'Master Student' with a GPA of a value less than '7.0' is prohibited to perform the activity. This assignment contains the following OCL expression:

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

The expression firstly starts querying the 'Master Student' class and then collects all its properties. Then, it specifically selects the 'GPA' property and collects all the attributes that instantiate this property, and then selects the ones with a value lower than '7.0'. Finally, it returns the agents that carry these attributes (and hence satisfy the required constraints).

Next, we have a ConjunctiveAssignment to the activity 'Analyze Dissertation'. The first composing assignment is an OrganizationalBasedAssignment of type Obligation, to ensure that the agent analyzing the dissertation should be a supervisor of the specific student that performed the instance of the activity 'Write Dissertation First Version' according to the organizational model. This assignment contains the following OCL expression:

```
genericOrganizationalMetamodel::ActiveStructureRelatorClassifier.allInstances()->select(
name='Supervision')->any( true ).instanceOfRC->select(
  mediates->at(2).name=self.isContained.isContained.contains.oclAsType(ComplexBehavioralOccurrence)->select (
    instanceOfProcess.name = 'Student').contains->select(
      instanceOfActivity.name = 'Write Dissertation First Version')->any(true).participation.name).mediates->at(1)
```

The expression firstly selects the 'Supervision' relator classifier and then selects the relators that are instances of it and have as the 'Supervised' agent the same one that performed the activity 'Write Dissertation First Version' and then returns the agents that are 'Supervisors' of the aforementioned agent. The second composing assignment is an AttributeBasedAssignment of type Obligation, to ensure that the supervising professor have at least five years of experience. This assignment contains the following OCL expression:

```
genericOrganizationalMetamodel::ActiveStructureClass.allInstances()->select (
    name = 'Professor')->collect (hasProperty)->select(
        name = 'Experience')->collect (hasAttributes)->select (
            value.toInteger()>5).characterizedAgent
```

The expression firstly starts querying the 'Professor' class and then collects the property named 'Experience', and then collects all the attributes that are instances of the 'Experience' property, and then returns the agents that has a value greater than '5' in the aforementioned attribute.

The last composing assignment is a HistoryBasedAssignment of type Obligation, to ensure that the supervising professor has at least analyzed some dissertation three times. This assignment contains the following OCL expression:

```
let analyseDissertation:
Set (SimpleBehavioralOccurrence) =
SimpleBehavioralOccurrence.allInstances()->select (
            instanceOfActivity.name = 'Analyze Dissertation')
in
analyseDissertation.participation->asSet()->select (
            agent | analyseDissertation->select (
                        participation = agent)->size() >= 3)
```

The context of the expression is self, which refers to the newly created behavioral occurrence of the activity 'Analyze Dissertation' the assignment is referring to. It starts navigating through the behavioral occurrence of the process that contains the newly created behavioral occurrence, and then select the agent which has performed this same activity at least three times. Only an agent that satisfies all these three composing assignments will be able to perform the activity.

Next we have the same ConstraintBasedAssignment of type Obligation, which will be the assignment to three different activities: ‘Submit Considerations’, ‘Approve Text’ and ‘Define Examination Board’. It has the following OCL expression:

```
self.isContained.contains->select(
    instanceOfActivity.name = 'Analyze Dissertation')->any(true).participation
```

The expression starts by navigating through the behavioral occurrence of the process that contains the newly created behavioral occurrence of each referred activity, and then returns the agent that performed an instance of the ‘Analyze Dissertation’ activity. In a given instance of the process, there may be multiple behavioral occurrences of the ‘Analize Dissertation’ activity, but all of them will be performed by the same agent, so it is not necessary to see the end time of all of them to see which one was the last.

Lastly, there is a ResultBasedAssignment of type Obligation to the process ‘Examination Board’, to ensure that the group that will be the performer will be the one that was defined in the ‘Define Examination Board’ activity. The process encompasses the activities ‘Evaluate Dissertation and Presentation’, ‘Accept Dissertation’ and ‘Reject Dissertation’. This assignment contains the following OCL expression:

```
self.isContained.contains.oclAsType(ComplexBehavioralOccurrence)->select(
    instanceOfProcess.name = 'Professor')->collect(contains)->select(
        x | x.instanceOfActivity.name = 'Define Examination Board')->any(true).result
```

The context of the expression is self, which refers to the newly created behavioral occurrence of the process graphically represented by the ‘Examination Board’ pool, which the assignment is referring to. It starts navigating through the collaboration diagram that contains the referred pool and then selects the pool ‘Professor’; then it navigates through the activity whose result is the group for which we want to assign the referred process.

### 4.3.4 Considerations and Limitations

We discussed the integration of BPMN into our assignment framework, but regarding specifically the integration of Process, there are some noteworthy considerations. In collaboration diagrams, there are multiple participants involved, thus there are multiple pools. For the sake of not having our framework dependent of specific BPMN concepts, in our integration we do not consider that a complex behavioral occurrence may also be a collaboration diagram, although it is clearly the case and our own example needed to navigate through a collaboration diagram to specify some assignments. Thus, there will exist behavioral occurrences in behavioral occurrence models representing those that won't be instances of a process and neither of an activity, an exception of what we stated in section 3.5.

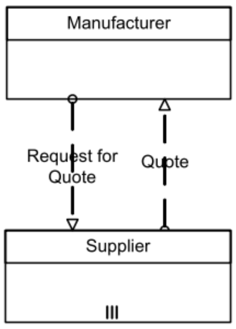


**Figure 29 - Multiple Participants in black-boxes Pools**

Figure 29 shows an example that has some peculiarities in regards to collaboration diagrams. Firstly, both pools represent participants without referring a process, therefore the details (e.g. activities) of each of them is not shown. Nevertheless, the message flows exchanged between then clearly shows that they are send and receive tasks (or events) that have been omitted for modeling reasons. This suggests that some collaboration models may require assignments in regards to the content of the message flows between the pools. We consider these special cases of ResultBasedAssignments, but do not currently address them in the scope of our integration.

Figure 29 also shows a pool with a multi-instance marker, i.e., in an instance of that collaboration diagram there are multiple instances of the 'Supplier' process and only one of the 'Manufacturer' process. In such cases, we did not define of how we may differentiate each instance of the collaboration, thus we chose to not support this multiple instances marker in our assignments.

## 4.4 Prototype

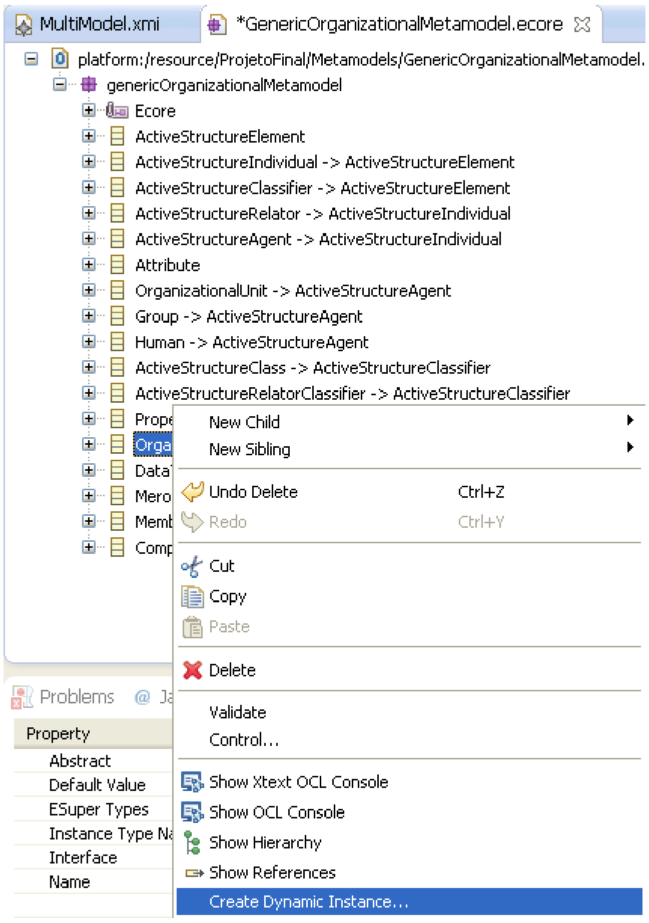
To test the integration of our framework to BPMN, we have developed a prototype to simulate a working environment. It was implemented using the native EMF capabilities to manipulate models that are built in Ecore.

Our example BPMN model was designed and edited in the STP BPMN Modeler[1]. The modeler is implemented on top of EMF and generates two files, one with layout information and the other holding the XMI content of the BPMN model, thus allowing us to serialize and load the model with ease.

An organizational repository containing the organizational model based on the organizational metamodel and the occurrences of behaviors information based on the occurrence metamodel are also required.

We simulate the required organizational repository by creating dynamic instances of the corresponding metamodels as shown in Figure 30. Through this mechanism we created a behavioral occurrence model and an organizational model and populated them.

[1] http://www.eclipse.org/stp/bpmn



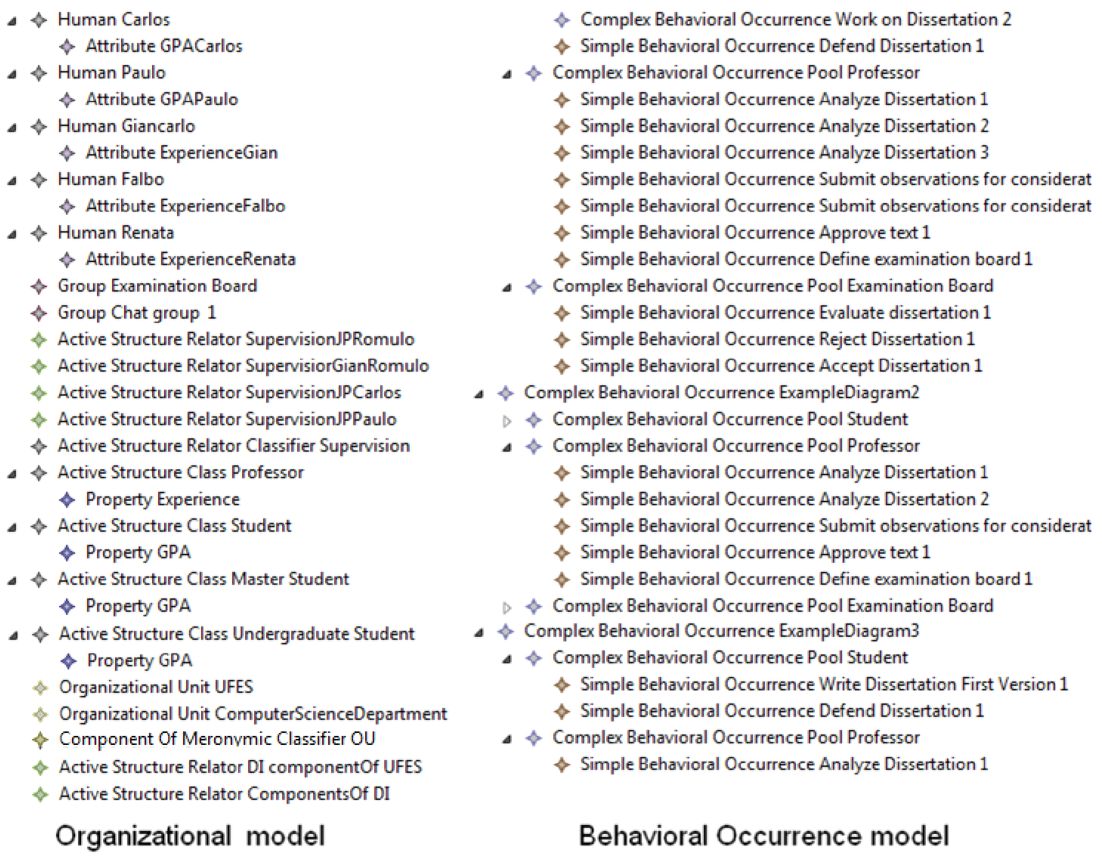
```
MultiModel.xmi    *GenericOrganizationalMetamodel.ecore
platform:/resource/ProjetoFinal/Metamodels/GenericOrganizationalMetamodel.
  genericOrganizationalMetamodel
    Ecore
    ActiveStructureElement
    ActiveStructureIndividual -> ActiveStructureElement
    ActiveStructureClassifier -> ActiveStructureElement
    ActiveStructureRelator -> ActiveStructureIndividual
    ActiveStructureAgent -> ActiveStructureIndividual
    Attribute
    OrganizationalUnit -> ActiveStructureAgent
    Group -> ActiveStructureAgent
    Human -> ActiveStructureAgent
    ActiveStructureClass -> ActiveStructureClassifier
    ActiveStructureRelatorClassifier -> ActiveStructureClassifier
    Prope
    Orga
    Data
    Mero
    Memb
    Comp

Problems  @ Ja
Property
  Abstract
  Default Value
  ESuper Types
  Instance Type Na
  Interface
  Name
```

**Figure 30 - Creating metamodel instances**

Figure 31 shows a snapshot of the organizational repository. It is presented in a tree-view using the Sample Ecore Model Editor. We purposely decide to collapse some information to not bring overwhelmingly large information. Also omitted from the figure are the properties (attributes and not containment associations) of each node.

In the Behavioral Occurrence model, there are three complex behavioral occurrences of the collaboration diagram defined in section 4.3.3: ‘ExampleDiagram1’, ‘ExampleDiagram2’ and ‘ExampleDiagram3’. The first two executions of the collaboration diagram are already finished, and the third one is still unfinished, so there will be assignments to be still evaluated and performed on it. The complex behavioral occurrences of the collaboration diagram contain three other complex behavioral occurrences each, representing the occurrence of the processes defined in the BPMN model. The occurrence of each process contains all the other occurrences of the activities contained within the process, including the occurrences of activities that were performed more than once (e.g. the activity ‘Analize Dissertation’).



```
Human Carlos
    Attribute GPACarlos
Human Paulo
    Attribute GPAPaulo
Human Giancarlo
    Attribute ExperienceGian
Human Falbo
    Attribute ExperienceFalbo
Human Renata
    Attribute ExperienceRenata
Group Examination Board
Group Chat group 1
Active Structure Relator SupervisionJPRomulo
Active Structure Relator SupervisiorGianRomulo
Active Structure Relator SupervisionJPCarlos
Active Structure Relator SupervisionJPPaulo
Active Structure Relator Classifier Supervision
Active Structure Class Professor
    Property Experience
Active Structure Class Student
    Property GPA
Active Structure Class Master Student
    Property GPA
Active Structure Class Undergraduate Student
    Property GPA
Organizational Unit UFES
Organizational Unit ComputerScienceDepartment
Component Of Meronymic Classifier OU
Active Structure Relator DI componentOf UFES
Active Structure Relator ComponentsOf DI
```

Organizational model

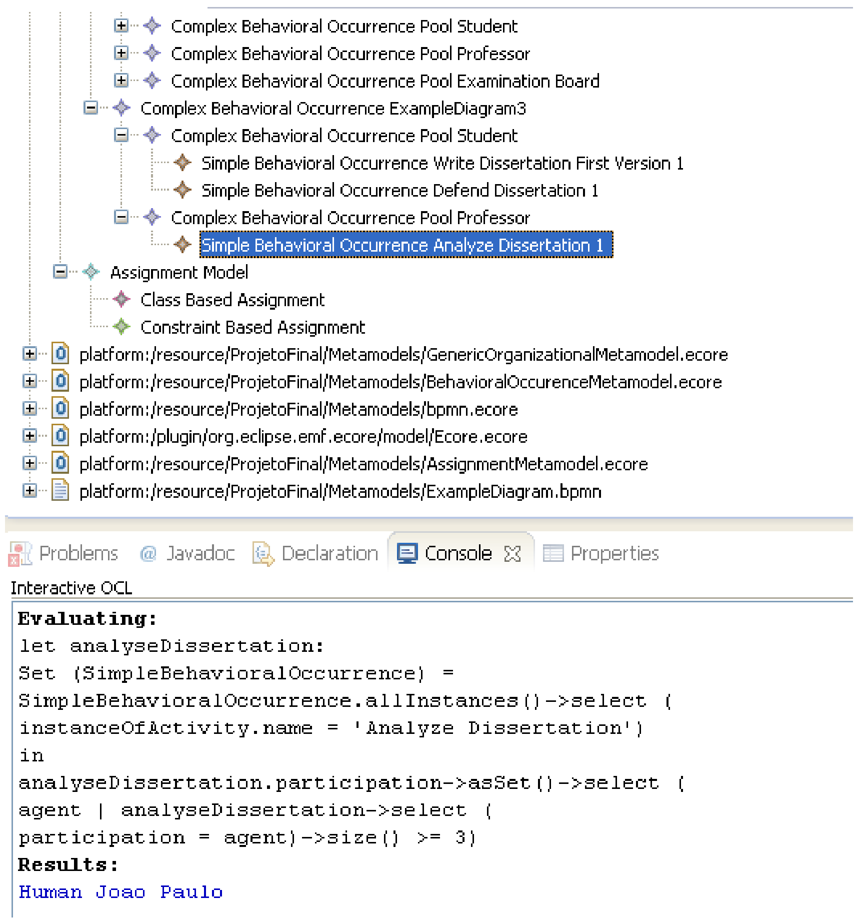
```
        Complex Behavioral Occurrence Work on Dissertation 2
        Simple Behavioral Occurrence Defend Dissertation 1
    Complex Behavioral Occurrence Pool Professor
        Simple Behavioral Occurrence Analyze Dissertation 1
        Simple Behavioral Occurrence Analyze Dissertation 2
        Simple Behavioral Occurrence Analyze Dissertation 3
        Simple Behavioral Occurrence Submit observations for considerat
        Simple Behavioral Occurrence Submit observations for considerat
        Simple Behavioral Occurrence Approve text 1
        Simple Behavioral Occurrence Define examination board 1
    Complex Behavioral Occurrence Pool Examination Board
        Simple Behavioral Occurrence Evaluate dissertation 1
        Simple Behavioral Occurrence Reject Dissertation 1
        Simple Behavioral Occurrence Accept Dissertation 1
Complex Behavioral Occurrence ExampleDiagram2
    Complex Behavioral Occurrence Pool Student
    Complex Behavioral Occurrence Pool Professor
        Simple Behavioral Occurrence Analyze Dissertation 1
        Simple Behavioral Occurrence Analyze Dissertation 2
        Simple Behavioral Occurrence Submit observations for considerat
        Simple Behavioral Occurrence Approve text 1
        Simple Behavioral Occurrence Define examination board 1
    Complex Behavioral Occurrence Pool Examination Board
Complex Behavioral Occurrence ExampleDiagram3
    Complex Behavioral Occurrence Pool Student
        Simple Behavioral Occurrence Write Dissertation First Version 1
        Simple Behavioral Occurrence Defend Dissertation 1
    Complex Behavioral Occurrence Pool Professor
        Simple Behavioral Occurrence Analyze Dissertation 1
```

Behavioral Occurrence model

**Figure 31 - Snapshot of an excerpt of the organizational repository**

Having the organizational repository snapshot we just mentioned, we tested and ran the OCL expressions that we previously defined in the assignment model. To initially assess and test the expressions, we used the OCLInEcore[2] console. In Figure 32 we show an OCL expression and its result.

[2] http://help.eclipse.org/helios/topic/org.eclipse.ocl.doc/tutorials/oclinecore/oclInEcoreTutorial.html



```
Complex Behavioral Occurrence Pool Student
Complex Behavioral Occurrence Pool Professor
Complex Behavioral Occurrence Pool Examination Board
Complex Behavioral Occurrence ExampleDiagram3
  Complex Behavioral Occurrence Pool Student
    Simple Behavioral Occurrence Write Dissertation First Version 1
    Simple Behavioral Occurrence Defend Dissertation 1
  Complex Behavioral Occurrence Pool Professor
    Simple Behavioral Occurrence Analyze Dissertation 1
Assignment Model
  Class Based Assignment
  Constraint Based Assignment
platform:/resource/ProjetoFinal/Metamodels/GenericOrganizationalMetamodel.ecore
platform:/resource/ProjetoFinal/Metamodels/BehavioralOccurenceMetamodel.ecore
platform:/resource/ProjetoFinal/Metamodels/bpmn.ecore
platform:/plugin/org.eclipse.emf.ecore/model/Ecore.ecore
platform:/resource/ProjetoFinal/Metamodels/AssignmentMetamodel.ecore
platform:/resource/ProjetoFinal/Metamodels/ExampleDiagram.bpmn
```

```
Evaluating:
let analyseDissertation:
Set (SimpleBehavioralOccurrence) =
SimpleBehavioralOccurrence.allInstances()->select (
instanceOfActivity.name = 'Analyze Dissertation')
in
analyseDissertation.participation->asSet()->select (
agent | analyseDissertation->select (
participation = agent)->size() >= 3)
Results:
Human Joao Paulo
```

**Figure 32 - OCL expression evaluation**

Note that this approach uses the native EMF environment to prototype our framework, however, no integration to a runtime environment (such as a business process engine) has been implemented. Nevertheless, using the EMF capabilities and the OCL API we have been able to run the expressions programmatically. This may be used in future work to integrate the framework in an existing run-time infrastructure.

## 4.5 Related Work

Recently, numerous works have been proposed to extend BPMN to support the workflow resource creation patterns. In (AWAD et al., 2009), 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*. Thereby, the extended BPMN metamodel mixes design-time and run-time elements, which is undesirable from the process model management perspective and also characterize a heavyweight extension of the language. It is possible to define the assignments to activities using OCL constraints.

The work proposed by Meyer (2009) has a similar premise, extending 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. The perspective is formally represented through three models: an organizational metamodel, a metamodel, and a task lifecycle model. Similarly to (AWAD et al., 2009), the metamodel which extends BPMN also includes design-time and run-time elements, like *Case* and *Work Item*. Being a bit more restrictive, it only considers the allocation of resources to tasks, not considering the allocation to *Processes*.

Grosskopf (2007) firstly does an assessment of BPMN and BPDM (OMG, 2007) in regards to a considered set of relevant workflow resource patterns. He then proposes a metamodel extension based on BPDM, introducing new associations and attributes to capture the not yet supported patterns. He considers the existence of an expression language to define allocation constraints, although he only considers abstractly, not defining any semantics for the expressions that can be built because he considers that to be a technical choice.

Cabanillas, Resinas and Ruiz-Cortés (2011) 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.

Similarly to the previously related works, Stroppi, Chiotti and Villarreal (2011) propose a heavyweight 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. 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 own BPMN mechanism for extending, it keeps the models interchangeable because the standard elements are not modified. The extension is divided in three aspects: resource structure, work distribution and authorization. The resource structure is concerned with the characterization of resources, and regarding to this, this work extends the *Resource* and *ResourceParameters* concepts of BPMN. The work distribution aspect is concerned with how the work is advertised and assigned to specific resources for execution, extending the *UserTask* concept of BPMN. The authorization aspect is concerned with the privileges that a resource has with regard to check and progress the work distributed to them.

Finally, differently from our approach all the works cited here (but (AWAD et al., 2009)) 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.

# 5. Application to ArchiMate

In this chapter, we discuss how our framework may also be applied in ArchiMate. While BPMN aims to provide detailed business process models, ArchiMate aims to provide a high-level description of enterprise architecture, not focusing on the details of business process behavior. In addition, while BPMN explicitly states that the organizational domain is out of the scope of the language, ArchiMate offers constructs to model organizations and, as seen in Chapter 2, it also provides some relationships to assign organizational elements and behavioral elements.

The first subsection of this chapter explains how the organizational constructs of ArchiMate should be mapped to the constructs of the organizational language defined in our framework.

Further, we discuss how the ArchiMate behavioral concepts may be integrated into the framework and we model the same example business process model presented in the previous chapter using ArchiMate constructs. Finally, we discuss the differences of the assignment model when ArchiMate is applied to the framework, comparing to the approach using BPMN.

## 5.1 Organizational Structure Constructs Mapping

To determine and define our mapping from the proposed organizational metamodel to the ArchiMate organizational structure constructs, we mostly refer to the definitions and examples from the official ArchiMate documentation (THE OPEN GROUP, 2009a).

Business Actor is one of the main elements to model organizational actors of ArchiMate and based on the definition and the proposed interpretation of the concept by the work of Almeida and Guizzardi (2008), we consider it to be mapped to the ActiveStructureAgent metaclass.

Business Role is defined as “named specific behavior of a business actor participating in a particular context”. Business Roles may be assigned to many Business Actors. We consider this concept is directly mapped to an ActiveStructureClass.

Business Collaboration is another active structure concept in ArchiMate and is defined as "a (temporary) configuration of two or more business roles resulting in specific collective behavior in a particular context". The word temporary used in the specification seems to imply that it is related to the concept of Group, specifically, to types of Group, because it aggregates two or more Business Roles.

Table 5 summarizes the mapping also including ArchiMate's relations of aggregation, composition and specification. Aggregation and composition are mapped into the MemberOfMeronymicClassifier and ComponentOfMeronymicClassifier metaclasses, respectively. The Specialization relationship is mapped into the superclass/subclass meta-association from the metamodel (and we consider it is only used between Roles and between Collaborations). As we can also observe from the table, a number of metaclasses from the organizational metamodel of the proposed framework have no mapping to ArchiMate constructs. This is due to the high-level nature of ArchiMate. (The consequence is that organizational relations and actor's attributes cannot be visualized in an ArchiMate model, and only represented in our organizational model.)

**Table 5 - Summary of the organizational structure concepts from ArchiMate to the organizational metamodel of our framework**

| **ArchiMate Construct** | **Framework Organizational Metamodel** |
|---|---|
| Business Actor | ActiveStructureAgent<br>(Human, OrganizationalUnit, Group) |
| Business Role | ActiveStructureClass |
| Business Collaboration | ActiveStructureClass (referring to types of Groups) |
| Aggregation | MemberOfMeronymicClassifier |
| Composition | ComponentOfMeronymicClassifier |
| Specialization | Superclass |

| | |
|---|---|
| Not Available | ActiveStructureRelatorClassifier, ActiveStructureRelator, Attribute, Property |

## 5.2 Behavioral constructs integration and limitations

As we previously discussed, the behavioral occurrence metamodel and the assignment metamodel depend on some behavioral concepts. Both metamodels have generic classes (namely *Activity* and *Process*) that must be replaced by the corresponding concepts from the concrete language, in this case, ArchiMate.

ArchiMate has a number of concepts that covers the behavioral aspects. In an excerpt of its metamodel, shown in Figure 3, we see that *Business Event*, *Business Function*, *Business Process* and *Business Interaction* are considered to be *Business Behavior Elements*. We must now identify which one of them must replace the *Process* and *Activity* concepts through the definitions given in the official documentation (THE OPEN GROUP, 2009a).

*Business Event* is discarded as a potential candidate, because it is described as "something that happens (internally or externally) and influences behavior (business process, business function, business interaction)". They trigger or are triggered by behavior. Thus, it is not a concept that may have an assignment to an active structure element to perform the event.

*Business Process* is described as "a unit of internal behavior or collection of causally-related units of internal behavior intended to produce a defined set of products and services". It describes an internal behavior that must also be assigned to a *Business Role*. It may be used to describe atomic processes, the ones that may not be further subdivided, or to describe complex processes that are composed of other processes. Therefore, it seems to be related to the concept of *Activity*.

*Business Function* is described as "a unit of internal behavior that groups behavior according to, for example, required skills, knowledge, resources, etc., and is performed by a single role within the organization". Although this seems to imply that Business

Functions could be subject of assignment in our framework, business functions do not have a definite temporal extension and thus, unlike ArchiMate's Business Processes, Business Functions are incompatible with our notion of behavior elements (and would be incompatible with history-based assignments, since we cannot talk about past occurrences of a function).

The last behavioral concept, *Business Interaction* is described as "a unit of behavior performed as a collaboration of two or more business roles". It is like a process that is performed by multiples roles in a *Business Collaboration*. It may also be decomposed in smaller units. Because of its nature, the roles that compose the collaboration generally have to perform some behavior themselves to accomplish the interaction as a whole. Like a *Business Process*, a *Business Interaction* is a unit of behavior that may be composed into smaller interactions or may also be an atomic collaborative behavior. Therefore, it also seems to be related to the concept of *Activity*.

In order to integrate the assignment framework with ArchiMate, we should bind the *Activity* generic concept to ArchiMate's *Business Process* and *Business Interaction* concepts. The expressive assignments described in the framework would be superimposed in ArchiMate's (simple) assignments involving these concepts.

To implement the binding without affecting the ArchiMate metamodel, we would bind *Activity* to a *Business Behavior Element*, and assure by the means of an OCL invariant that only *Business Process and Business Interactions* may be used, ruling out the possibility of referring to *Business Events* and *Business Functions* in the assignment.

## 5.3 Example

### 5.3.1 Behavioral and Organizational models

In this section we revisit the example presented in Chapter 4. Figure 33 shows ArchiMate model capturing the behavioral elements that will be subject to assignment, and further representing assignments using the simplistic ArchiMate assignment relation. Our aim was to translate the BPMN model and *Processes* were mapped to either *Business Processes* or *Business Interactions* (in the case of the behavior performed by the ‘Examination Board’). Accordingly, the *Participants* of the *Pools* representing the processes were mapped to either *Business Roles* or *Business Collaborations* (in the case of ‘Examination Board’). *Business Roles* are assigned to *Business Processes*, which are further decomposed into others *Business Processes* and the only *Business Collaborations* are assigned to *Business Interactions*, which are further decomposed into others *Business Interactions*. The roles and the collaboration are also part of an organizational model whose view is shown in Figure 34.

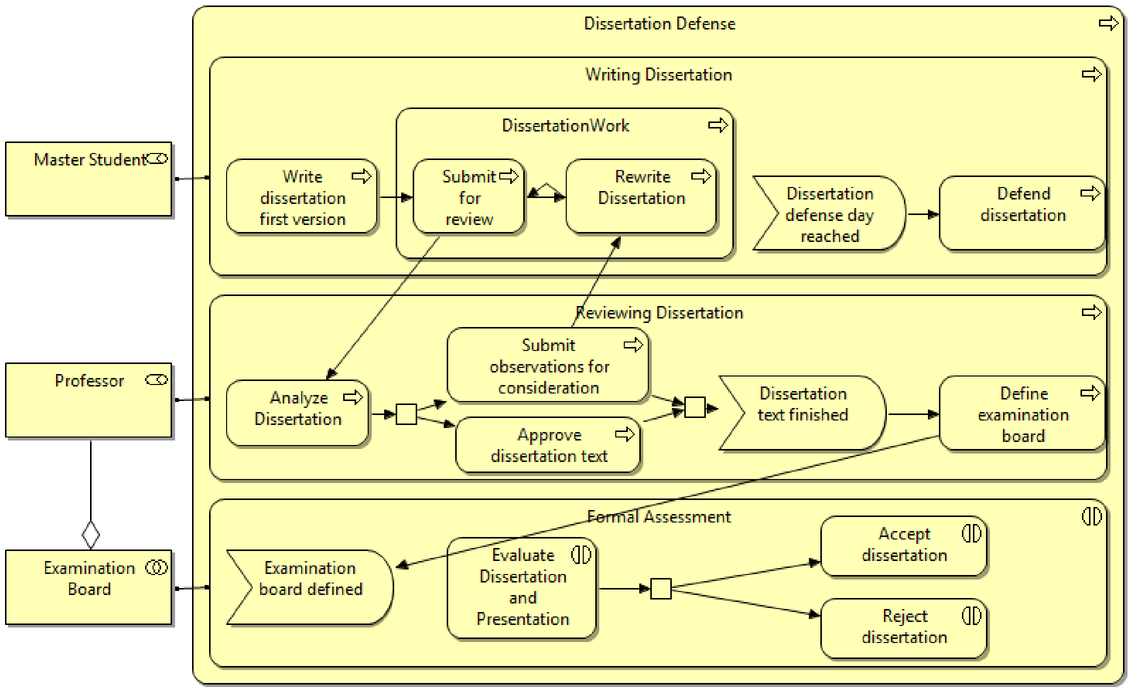


**Figure 33 - Example of an ArchiMate model with the Process view**

The organizational model shown in Figure 34 represents the organizational model in ArchiMate (in conformance with the model presented in section 4.3.2). As discussed previously, the organizational metamodel defined in the assignment framework is more expressive than ArchiMate and *Attribute*, *Property*, *ActiveStructureRelatorClassifier* and *ActiveStructureRelator* have no correspondence in ArchiMate. In order to perform the integration (and enable organizational-based assignments, as well as attribute-based assignments), one must map the organizational model in ArchiMate to an organizational model defined in the assignment framework and enrich it with the other constructs the last has.

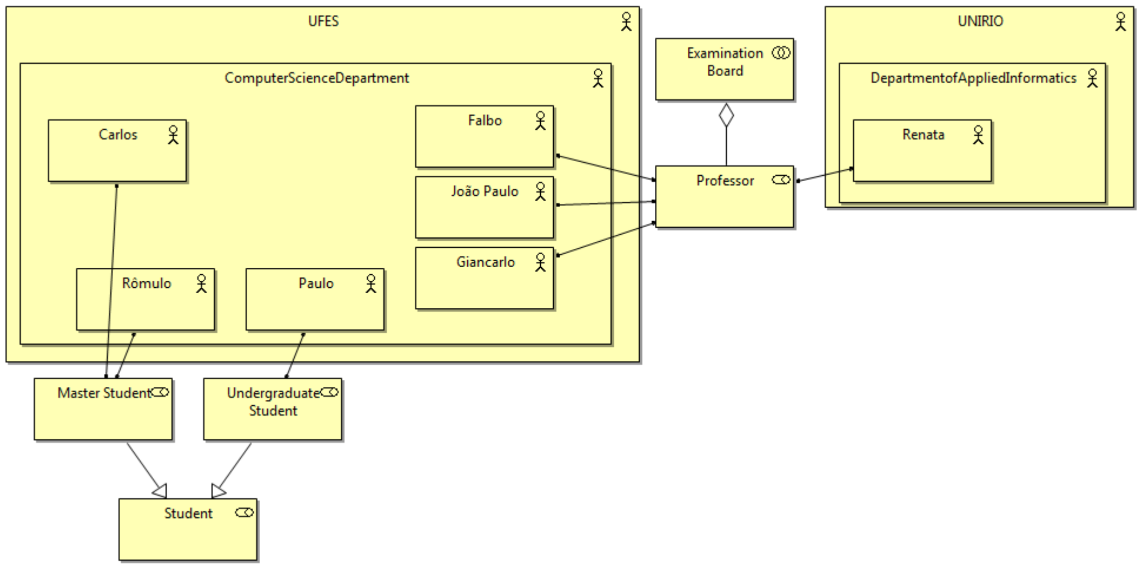


**Figure 34 - Organizational Structure view**

### 5.3.2 Assignment Model

The assignment constraints defined in section 4.3.1 are also applied here: they are constraints defined in natural language that should be applied to any technology. The constraints were represented in an assignment model with BPMN integrated into the framework. This assignment model does not changes when applied to ArchiMate. What changes would be how it would be graphically represented (its concrete syntax). For instance, Direct Assignment could be directly represented in ArchiMate when only one business actor is assigned to a business role and the business role is assigned to a business process (or only a business actor is assigned to business collaboration). Role-based assignment is directly represented with the assignment relationship between a business role and a business process (or a business collaboration and business interaction). Expression-based assignments changes would be that there is the need to add the expression-based assignments in the assignment relationship of ArchiMate. This could be done as shown in Figure 35, when a result-based assignment enriches the assignment relationship, precisely specifying what ‘Examination Board’ will perform the assigned business interactions.

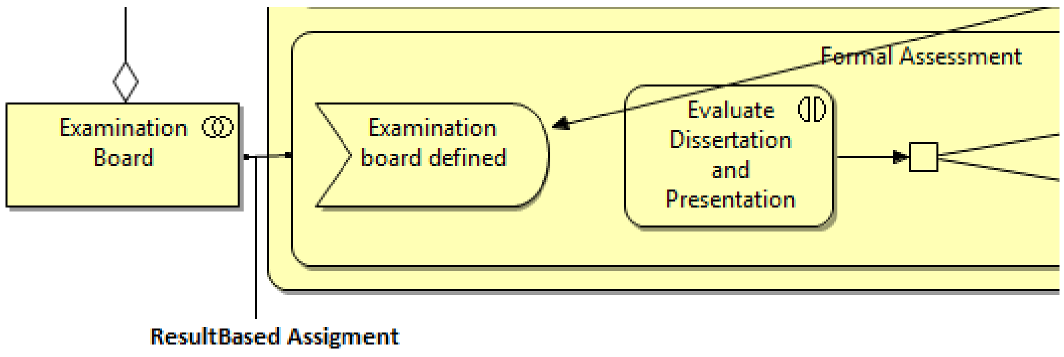

Examination Board
Examination board defined
Formal Assessment
Evaluate Dissertation and Presentation
ResultBased Assigment

**Figure 35 - An assignment constraint in ArchiMate applied to the framework**

### 5.3.3 Conclusions

The proposal of integration to ArchiMate discussed in this chapter provides some evidence in favor of the generality of the assignment framework. Nevertheless, differently from the application to BPMN discussed in Chapter 4, we have not implemented a prototype to test and simulate assignment constraints in ArchiMate models. In the future, we intend to address that by selecting an ArchiMate metamodel (preferably one built in EMF) and performing a full-fledged application of the framework. That would enable us to further refine the proposal of this chapter and could provide us with some feedback to improve the generality of the assignment framework.

# 6. Concluding Remarks

## 6.1 General Conclusions

This work has 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, the approach 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 must be bound to specific concepts from the metamodels of the adopted languages. We have applied the framework to BPMN, using the concepts of activity and process, and to ArchiMate, using the concept of business process. We believe that the framework could be applied to some of the other reviewed modeling techniques, such as UML activity diagrams (binding to action and activity), XPDL (binding to activity and pool) and ARIS (binding to the concept of function). As some of these offer support to model organizational structures, transformations to/from our organizational metamodel would be required for full integration of our approach.

In contrast to the other works that we previously discussed, our approach is more general because it is not dependent of a specific business process technology. As such, the referred behavioral metamodel will not need to be heavily modified (as we can observe when applying to BPMN). With the behavioral occurrence metamodel, we have also covered the aspect of execution history that is required in some of the patterns, without the need of modifying the behavioral metamodel (differently from e.g. (AWAD et al., 2009) and (MEYER, 2009), which requires such modification). We have also proposed an organizational metamodel which is general enough to model human resources and organizational structures for the perspective of assignment.

Regarding the assignment metamodel, we defined some assignment metaclasses that directly refer to organizational concepts to specify the assignment, in particular to refer to specific agents or to classes of agents. Other assignments are based on (OCL) expressions, which are used to query organizational resources and behavioral occurrences models. These assignments are categorized into attribute-based assignment, constraint-based assignment, history-based assignment, organizational-based assignment, and result-based assignment. The result-based assignment is not directly addressed in any of the considered workflow resource patterns, but was identified when studying the assignment domain and when trying to apply the assignment model in particular scenarios (like the one we provided). More complex assignments may also be defined using conjunctive and disjunctive assignments.

Ultimately, we can briefly list the contributions of our work in the following:

1. A bottom-up review of the active structure assignment mechanisms in widely used enterprise architecture frameworks and business process languages, presenting the constructs that each one adopts to cover the assignment. A review of these approaches against the considered set of workflow resource patterns. Through these analyses we identified that the reviewed approaches offer simplistic constructs and mechanisms in regards to the specification of the assignment of the active structure elements to the behavioral elements.

2. The definition of an assignment framework consisting of an organizational metamodel, a behavioral occurrence metamodel and an assignment metamodel, employing generic mechanisms that ease the integration of existing languages into the proposed framework. One can design an organizational model, refer to the execution history of behavior and finally define expressive assignments using all this information through the assignment framework.

3. The application of the framework to a standard in business process modeling and a widely used language to design business process model, namely BPMN. We show which BPMN concepts will be integrated into the framework, replacing the generic assumed concepts, and we show some considerations and limitations in regards to BPMN models and how the assignment framework

covers them. We also present a UML class diagram profile to model the organizations in a well-known syntax and finally we illustrate assignments in an example, consisting of an organizational model, a BPMN model and an assignment model that is related to both of them.

4. A prototype implementation of the framework. This implementation has been applied to a concrete and available BPMN tool, which has allowed us to simulate a runtime environment and test the expression assignment constraints. We believe this prototype could be used as part of other tools to extend them with capabilities to define and manage expressive assignments.

5. A preliminary study of how the framework could be applied to a general language to model enterprise architecture, namely ArchiMate. We show how the organizational structure language of ArchiMate may be mapped to our organizational metamodel and how the behavioral concepts required by the framework may be replaced by the ArchiMate behavioral concepts. We then design the same business process example presented in the BPMN chapter using the ArchiMate constructs and show how the assignment model would be defined.

The application of the results of this work in practice should aid organizations in managing how the work is distributed. This should help them in analyzing the relation between organizational activities and organizational actors, considering the perspectives of responsibility, authorization and accountability.

## 6.2 Future Work

Future work will firstly focus on use cases to validate the usability of the proposed framework. These use cases may reveal lack of expressiveness that may require extension of the assignment framework proposed here. Furthermore, we should define an integration of the approach in a process aware system considering the runtime environment, giving support to the actual execution of the assigned behaviors, like in jBPM. This may support us in addressing other workflow resource patterns that mostly focus on the dynamics of allocation at run-time (beyond the “creation patterns”).

A consistency analysis of assignment models should also be considered in future research, because as of now, the modeler is responsible to design the assignment constraints and also verify if they are indeed evaluating as expected. They could return unforeseen active structure elements, or they may never return anything at all. In a conjunctive assignment, the composing assignments may also be conflicting with each other and there should be a way to highlight that to the modeler.

We also consider that ontology-based semantics should be defined to all the meta-concepts defined in the framework using the Foundational Ontology (UFO) (GUIZZARDI, 2005) as basis, to ensure that the organizational metamodel, behavioral occurrence metamodel and the assignment metamodel indeed have well-founded semantics. Further, we should define a formal semantics for the notions of obligation and prohibition using deontic logics (MCNAMARA, 2010).

When testing the applicability of the framework in some very simple examples we have already identified that some expression-based assignments may occur several times in models. This is an indication that the reuse of previously-defined expressions or parts of an expression should be considered. In (THOM; REICHERT; IOCHPE, 2009) and (THOM et al., 2008), for example, activity patterns that are considered to be recurrent in business process models are defined and empirically evidenced in real-world process models. Thus, having a way to define expressions patterns to be reused based on gathered empirical evidence may also have promising results.

Defining a simpler concrete syntax for the assignment expressions should also be target in future research, because expressions are often poorly readable depending on the type of the expression assignment. An end user environment could transform expressions in a simpler concrete syntax into OCL in our framework, thus profiting from the well-defined syntax, semantics and interpretation tooling for OCL.

We also envision the applicability of the framework in others domains of study, for example the domain of project management, which, similarly to business process management, is concerned with distribution of work. Further research should be conducted to investigate how the framework could be applied in such domains and to consider how they would benefit from rich assignment capabilities.

Finally, the organizational metamodel proposed should also be extended to include the management of non-human resources as well as the combined assignment of human and non-human resources to behavioral elements.

# 7. References

AHN G.; SANDHU, R. *Role-based authorization constraint specification*. ACM Trans. Inf. Syst. Sec. 3, 4 (Nov.), 2000.

AKEHURST, D. H.; BORDBAR, B. *On Querying UML Data Models with OCL*. Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools, p.91-103, October 01-05, 2001.

ALMEIDA, J. P. A.; GUIZZARDI, G. *A Semantic Foundation for Role-Related Concepts in Enterprise Modelling*. In: Proc. 12th Int’l IEEE EDOC Conference, IEEE Computer Society Press, pp 31–40, 2008.

ARPINI, R. H.; ALMEIDA, J. P. A. *On the support for the assignment of active structure and behavior in enterprise modeling approaches*. In Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC '12), pp 1686 – 1693, 2012.

AWAD, A.; GROSSKOPF, A.; MEYER, A.; WESKE, M. *Enabling resource assignment constraints in bpmn*. Technical report, Business Process Technology – Hasso Plattner Institute, 2009.

BOTHA, R. A. *CoSAWoE – A Model for Context-sensitive Access Control in Workflow Environments,* South Africa, 2001.

BOTHA, R. A.; ELOFF, J. H. P. *Separation of duties for access control enforcement in workflow environments*. IBM Syst. J. 40, 666–682, 2001.

BRUCKER, A. D.; HANG, I.; LUCKEMEYER, G.; RUPAREL, G. *SecureBPMN: Modeling and enforcing access control requirements in business processes*. In SACMAT. ACM Press, 2012.

CABANILLAS, C.; RESINAS, M.; RUIZ-CORTÉS, A. *Towards the definition and analysis of resource assignments in BPMN 2.0,* tech. rep., Universidad de Sevilla, 2011.

CLARK, J.; DEROSE, S. *XML path language (XPath) version 1.0*. Technical report, World Wide Web Consortium (W3C) Recommendation, 1999.

DAVIS, R. *Business Process Modelling with ARIS - A Practical Guide*, Springer, 2001.

DUMAS, M.; van der AALST, W. M. P.; ter HOFSTEDE, A. H. M. *Process Aware Information Systems: Bridging People and Software Through Process Technology*. Wiley-Interscience, 2005.

GOGOLLA, M.; RICHTERS, M. *On Constraints and Queries in UML*; Proc. UML'97 Workshop The Unified Modeling Language - Technical Aspects and Applications', 1997.

GROSSKOPF, A. *An extended resource information layer for BPMN*. tech. rep., BPT, 2007.

GUIZZARDI, G. *Ontological Foundations for Structural Conceptual Models*. Ph.D. Thesis. University of Twente, The Netherlands, 2005.

HAVEY, M. *Essential Business Process Modeling*. O'Reilly Media, Inc., 2005.

ISO - International Organization for Standard. *Information Technology – Open Distributed Processing – Reference Model – Foundations*. Final Draft International Standard, ISO/IEC10746-2, 2010.

KHARBILI, M. E.; MA, Q.; KELSEN, P.; PULVERMUELLER, E. *Corel: Policy-based and model-driven regulatory compliance management*. In Proceedings of the 15th IEEE International EDOC Conference, 2011.

KLEPPE, A.; WARMER, J.; BAST, W. *MDA Explained: The Model Driven Architecture: Practice and Promise*, 2003.

LANKHORST M., et al. *Enterprise Architecture at Work - Modelling, Communication, and Analysis*. Springer, 2005.

MANDEL, L.; CENGARLE, M. V. *On the Expressive Power of OCL*. FM'99 - Formal Methods, World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, Springer LNCS 1708, pp 854 – 874, 1999.

MEYER, A. Resource Perspective in BPMN: Extending BPMN to Support Resource Management and Planning. Master's Thesis, Hasso Plattner Institute, 2009.

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

RUSSEL, N.; ter HOFSTEDE, A.H.M; EDMOND, D. *Workflow resource patterns: Identification, representation and tool support*. In Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE05), volume 3520 of Lecture Notes in Computer Science, pp 216–232, 2005.

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

Object Management Group (OMG). *Business Process Definition Metamodel*. (BPDM) - Final Submission, 2007.

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

Object Management Group (OMG). *Object Constraint Language specification version 2.2*. OMG document, formal/2010-02-01, 2010a.

Object Management Group (OMG). *The Unified Modeling Language: Superstructure. Version 2.3*, OMG document, formal/2010/05/03, 2010b.

PLATO (Author); GRUBE, G. M. A. (Translator); Reeve, C. D. C (Editor). *The Republic*. 2nd edition. Hackett Publishing Company, 1992.

RUSSEL, N.; ter HOFSTEDE, A.H.M.; EDMOND, D; 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.

RUSSEL, N.; van der AALST, W. M. P.; ter HOFSTEDE, A. H. M.; WOHED, P. *On the suitability of UML 2.0 activity diagrams for business process modelling*. Pages 95–

104 of: APCCM '06: Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling. Darlinghurst, Australia, Australia: Australian Computer Society, 2006.

SANTOS JR., P. S.; ALMEIDA, J. P. A.; PIANISSOLLA, T. L. *Uncovering the Organizational Modelling and Business Process Modelling Languages in the ARIS Method*. International Journal of Business Process Integration and Management (IJBPIM), Vol. 5, No. 2, pp 130-143, 2011.

SANTOS JR., P. S.; ALMEIDA, J. P. A.; GUIZZARDI, G. *An Ontology-Based Semantic Foundation for Organizational Structure Modeling in the ARIS Method*. In Joint 5th International Workshop on Vocabularies, Ontologies and Rules for The Enterprise (VORTE) - International Workshop on Metamodels, Ontologies and Semantic Technologies (MOST), Vitória, Espírito Santo. Proceedings of the 2010 14th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW 2010). Los Alamitos, CA : IEEE Computer Society Press, 2010a.

SANTOS JR, P.S.; ALMEIDA, J. P. A.; GUIZZARDI, G. *An Ontology-Based Semantic Foundation for ARIS EPCs*, In: 25th ACM Symposium on Applied Computing (Enterprise Engineering Track), 2010b.

SCHEER, A. W. *ARIS – Business Process Modeling (3rd edition)*, Springer, 2000.

SHAPIRO, R. M. *XPDL 2.2: Incorporating BPMN 2.0 Process Modeling Extensions*, in Fischer, L. (ed.), 2010 BPM and Workflow Handbook, Future Strategies, Inc., 2010.

SHARP, A.; MCDERMOTT, P. *Workflow Modelling Tools for Process Improvement and Application Development*. Artech House, 2001.

SMITH, A. (Author); SUTHERLAND, K. (Editor). *An Inquiry into the Nature and Causes of the Wealth of Nations: A Selected Edition.* Oxford Paperbacks, 2008.

STROPPI, L.; CHIOTTI, O.; VILLARREAL, P. *A BPMN 2.0 Extension to Define the Resource Perspective of Business Process Models*, XIV Congresso Iberoamericano in Software Engineering (CIBSE), Rio de Janeiro, Brasil, 2011.

The Eclipse Modeling Framework. http://www.eclipse.org/emf/, last access at 02/02/2012.

The Open Group. *ArchiMate Technical Standard*. 2009a. http://www.opengroup.org/archimate/doc/ts_archimate/, last accessed at 21/05/2011.

The Open Group. *TOGAF Version 9*. 2009b. http://pubs.opengroup.org/architecture/togaf9-doc/arch/, last accessed at 06/09/2011.

THOM, L. H.; REICHERT, M.; IOCHPE, C. *Activity Patterns in Process-aware Information systems: Basic Concepts and Empirical Evidence*. In: IJBPIM - International Journal of Business Process and Information Management, 2009.

THOM, L. H.; REICHERT, M.; IOCHPE, C.; CHIAO, C.M.; HESS, G. N. *Inventing Less, Reusing More, and Adding Intelligence to Business Process Modeling*. In: 19th International Conference on Database and Expert Systems Applications (DEXA '08), Turin, Italy. Springer, LNCS 5181, pp. 837-850, 2008.

US Department of Defense. *DoD Architecture Framework Version 2.02,*. http://cio-nii.defense.gov/sites/dodaf20/, last accessed at 25/05/2011.

van der AALST, W. M.P.; van HEE, K. M. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.

van der AALST, W.M.P. *Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management*, LNCS, Vol. 3098, pp 1-64, 2004.

WARMER, J.; KLEPPE, A. *The Object Constraint Language: Getting Your Models Ready for MDA*. Addison Wesley, second edition, 2003.

Workflow Management Coalition (WfMC), *Process Definition Interface – XML Process Definition Language (XPDL)*. WfMC Standards, WFMC-TC-1025, The Workflow Management Coalition, 2008.

WHITE S. A. *Introduction to BPMN*. IBM White Paper, 2004.

WOHED, P.; van der AALST, W.M.P.; DUMAS, M.; ter HOFSTEDE, A.H.M.; RUSSEL, 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, 2006.

ZACHMAN, J. *A Framework for Information Systems Architecture*. IBM Systems Journal, Vol. 26, No. 3, pp 276-292, 1987.

ZHOU, W. *Authorization Constraints Specification and Enforcement.* Journal of Information Assurance and Security, 3 (1), pp 38-50, 2008.

MCNAMARA, P. *Deontic Logic*. Stanford Encyclopedia of Philosophy, 2010, http://plato.stanford.edu/entries/logic-deontic/, last accessed at 25/06/2012.

# APPENDIX A: OCL USAGE

Structural diagrams, such as UML class diagrams, are typically not refined enough to provide all the relevant aspects of a specification. There is the need to describe additional constraints about the models. Often these constraints are described in natural language. However the practice has shown that this inevitably leads to ambiguities. Thus, formal languages have been developed to write unambiguous constraints. Most of them usually had a traditional disadvantage of requiring persons with a strong mathematical background to be used, bringing difficulties to the average modelers to use (OMG; 2010a).

OCL has been developed to fill this gap. It is a formal language that is easy to read and write and it has been developed as a business modeling language. It is a pure specification language thus an OCL expression won’t have any side effects, not changing anything in the model. That means that the state of a system will never change because of an evaluation of an OCL expression. Although OCL was originally designed for describing constraints about models, its ability to navigate models and form collection of objects has led to its usage as a query language, as seen, for instance, in (MANDEL; CERGARLE, 1999), (GOGOLLA; RICHTERS, 1997) and (AKEHURST; BORDBAR, 2001).

In the remainder of this section we introduce the main concepts of OCL, and give some examples, all based on Figure 36. For a complete documentation of the language, we suggest the read of the OCL specification (OMG; 2010a), where the remainder of this section is also based on. For those who are unfamiliar with OCL and want a reference manual and explanations in a relatively informal way, we recommend the reading of the work by Warmer and Kleppe (2003).

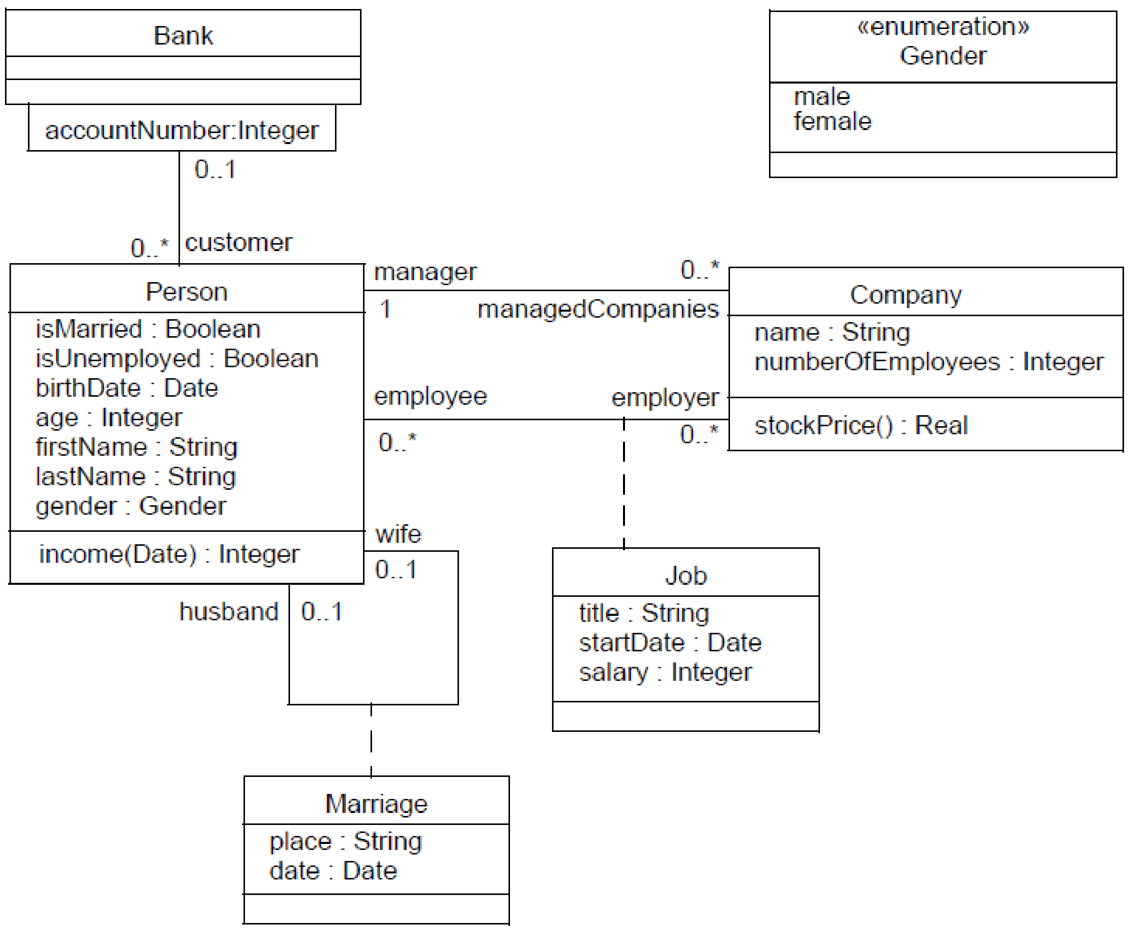


**Figure 36 - Example Class Diagram (OMG; 2010a)**

## Basic Elements

Every OCL expression is written in the context of an instance of a specific type. Firstly, to specify the type that will be the context, we use the reserved word context. To refer to the instance of the specified type, the reserved word *self* is used. Suppose the following example:

***context*** *Company* ***inv:***
*self.numberOfEmployees > 50*

In the example, the context will be the class 'Company' and self will refer to every single instance of this class. The reserved word *inv* in the above example indicates that the expression will be an invariant constraint, which means that every instance of the specified context must evaluate to true for the model, or else it would be in an invalid state. Thus, the model would only be valid if every instance of ‘Company’ has more than fifty employees.

## Basic Types

In OCL, there is a number of basic types that are predefined and available to the modeler at all times. These basic types, with the corresponding examples of their values, are shown inTable 6.

**Table 6 - Basic OCL Types and their values**

| Type | Values |
|---|---|
| Boolean | true, false |
| Integer | 1, 10, -2, 2134, … |
| Real | 0.8, 300.74, … |
| String | ‘To be or not to be...’ |

OCL also defines a number of operations that are used on the predefined types. Table 7 gives examples of that.

**Table 7 - Some operations in OCL primitive types**

| Type | Operations |
|---|---|
| Boolean | and, or, xor, not, implies, if-then-else |
| Integer | *, +, -, /, abs() |
| Real | *, +, -, /, floor() |

| | |
|---|---|
| String | concat(), size(), substring() |

Collection, Set, Bag, Sequence, and Tuple are basic types as well that have major roles in OCL expressions. We talk about them in the following.

## Retyping

In some cases, it is desirable to use a property of an object which is defined in a subtype of the current type of the object. When you are certain that the actual type of an object is it subtype, the object may have his re-type it by using the operation *oclAsType (OclType)*. This operation does not change the object, only its type in the context of the operation.

Suppose we have an *object* and types *Type1* and *Type*, which are different types. We could write:

```
objeto.oclAsType(Type2)
```

This will only be valid if, at evaluation time, Type2 is a subtype of the type of the object and the object is also an instance of the subtype.

## Let expressions

Often there are occasions in which a sub-expression is used more than once in an expression. In such cases, we may use let expressions which allows one to define a variable that can be used in the OCL expressions. The example below illustrates how it may  be used:

```
context Person inv:
  let income : Integer = self.job.salary->sum() in
  if isUnemployed then
     income < 100
  else
    income >= 100
  endif
```

In this we defined the variable income, which has as declared type Integer and its initial value set to *self.job.salary->sum()*.

## allInstances operation

An important operation that is predefined in OCL and is applicable to all classifiers in a given model, is the *allInstances* operation. The example bellow illustrates its use.

*Person.allInstances()*

The above excerpt of an expression would return the collection of all instances of Person.

## Type checking

The operation *oclIsTypeOf (Type)* returns true if the type of *self* and *Type* are exact the same, otherwise it returns false. For instance:

***context*** *Person*
***inv****: self.oclIsTypeOf( Person )*
***inv****: self.oclIsTypeOf( Company )*

The first invariant of the above example returns true. The second one returns false. If one would want to check if Type is the exact same or any of the supertypes, it may use the operation *oclIsKindOf (Type)*.

## Collections

The *Collection* type is the type that we most used when defining queries to define the assignments. It is a predefined type which defines a large number of operations to allow the modeler to manipulate the collections. OCL distinguishes three different collection types: (i) *Set*, which does not contain duplicate elements; (ii) *Bag*, which may contain duplicate elements; (iii) *Sequence*, which is like a *Bag* but the elements are ordered.

When we navigate in the model through relationships with multiplicity greater than one, we would have as the return type a Collection. In the following we show some important predefined operations that permit us to manipulate collection in a flexible and powerful way.

## select operation

When we have a collection and we are interested in only a subset of it that conforms to a certain criteria, we may use the *select* operation. It has as a parameter a special sintax which allows one to specify the elements from the collections that we want to appear in the new subset. This is done through a boolean expression. In the below example, we obtain a subset of persons that has age greater than fifty and states that this collection is not empty:

```
context Company inv:
self.employee->select(age > 50)->notEmpty()
```

As shown, the context of the boolean expression in the argument of the operation is actual element of the collection on which the select operation was invoked. Thus the property *age* is considered in the context of a *Person*.

In the above example, it is impossible to refer to the persons themselves, it may only refer to properties of them (e.g. *age*). To allow that, there is another syntax to refer to each person explicitly:

```
context Company inv:
self.employee->select(p | p.age > 50)->notEmpty()
```

The above example is identical to the previous one. With the sole difference being the presence of the *p* element, which iterates over each member of the collection and evaluates it by the boolean expression specified after the ‘|’.

## collect operation

As shows in the previous sub-section, the *select* operation always results in a sub-collection of the original collection. When it is desired to specify a collection that is derived from other collection, containing different objects from the original collection (i.e. it is not a sub-collection), we may use the *collect* operation. The below example show its usage:

```
self.employee->collect( person | person.birthDate )
```

In the above example, we specify the collection of *birthDates* for all employees in the context of a company.