Computing and Informatics, Vol. 33, 2014, 35–60

A SPEMONTOLOGY FOR SOFTWARE PROCESSES REUSING

Fadila Aoussat

LSI Laboratory, University of Sciences and Technology Houari Boumediene
BP 32, Bab Ezzouar, Algeria
e-mail: a_zahoua@yahoo.fr

Mourad Oussalah

LINA Laboratory, University of Nantes, CNRS UMR 6241
2, Rue de la Houssinière,BP 92208, 44322, Nantes, France
e-mail: Mourad.oussalah@univ-nantes.fr

Mohamed AhmedNacer

LSI Laboratory, University of Sciences and Technology Houari Boumediene
BP 32, Bab Ezzouar, Algeria
e-mail: anacer@mail.cerist.dz

Abstract. Reusing the best practices and know-how capitalized from existing software process models is a promising solution to model high quality software processes. This paper presents a part of AoSP (Architecture oriented Software Process) for software processes reuse based on software architectures. The solution is proposed after the study of existing works on software process reusing. AoSP approach deals with the engineering "for" and "by" reusing software processes, it exploits the progress of two research fields that promote reusing in order to improve the software process reusing: domain ontologies and software architectures. AoSP exploits a domain ontology to reuse software process know-how, it allows retrieving, describing and deploring software process architectures. This article details the engineering "for" reusing SPs step of AoSP, it explains how the software process architectures are described and discusses the software process ontology conceptualization and software process knowledge acquisition.

Keywords: Software process (SP) reuse, SP connectors, SP architecture, SP styles, domain ontology, System and Software Process Engineering Metamodel (SPEM), ATL transformations, SP knowledge acquisition

Mathematics Subject Classification 2010: 68U99, 68T30

1 INTRODUCTION

The quality of a software product depends on the quality of the software process models that are used for the development and maintenance of this software product [21].

Software Process (SP) models are complex structures used to define the steps performed during software development. Many kinds of information must be integrated to describe these steps (resources, roles, input and output products...). Therefore, an important number of concepts, paradigms and languages are identified to cover the different software development aspects. However, there are always difficulties to model SPs that deal with software development preoccupations such as flexibility, dynamicity and evolution [9].

Reuse SPs is one of the promised approaches used to improve SPs. Reusing approaches allows to exploit best practices and know-how capitalized from the precedent SP modeling and enactment experiments. However, diversity and wide range of SP models, in addition to SP rigidity, make the SP model reusing very difficult. A number of studies are being conducted nowadays in order to provide better support regarding SP reuse. Unfortunately, no reusing approach has emerged as reference in the SP reusing domain.

This article presents a part of an approach for reusing SPs: AoSP (Architecture oriented Software Process); this approach focuses on the insufficiencies of the existing solutions for reusing SPs and suggests pertinent ones.

In order to cover engineering "by" reusing SPs, we focus our researches on the SP reuse approaches based on software architectures. We believe that reusability, flexibility and abstraction of the software architectures combined with software architecture deployment techniques are relevant characteristics that can be used to provide a pertinent reusing approach to model high quality SPs. Thus, we describe and deploy SP architectures.

Moreover, in order to cover the engineering "for" reusing SPs, we focus our researches on the SP reuse approaches based on domain ontology. Our aim is to share common understanding among stakeholders by capitalizing the best practices and know-how extracted from different SP models. We think that using a domain ontology can offer a pertinent SP repository that can not only manage the heterogeneity of the SP models in terms of concepts and terminologies, but can also provide better support for logical inference that allows the emergence of new solutions for the SP improvement.

To suggest a standard solution the domain ontology must be coherent, not ambiguous, and commonly accepted by the SP community. It must not only capitalize knowledge extracted from heterogeneous SP models, but also retrieve the required comprehensible SP architecture knowledge in order to deploy the SP architectures.

To present the engineering "for" reusing step of AoSP approach, our article is organized as follows: Section 2 presents AoSP approach and the adopted steps to model reusable SPs. AoSP describes SP architectures, thus, Section 3 provides the adopted semantics to describe SP architectures. We present briefly the explicit connectors and architectural styles defined for the SP architectures. Section 4 details how our domain ontology is designed. The SPEMOntology generation is detailed in section 5. To describe and deploy SP architectures, our ontology must capitalize different kinds of knowledge; thus, section 6 details how heterogeneous SP knowledge are capitalized. Section 7 presents a first evaluation of our contribution and details the remaining works. Section 8 concludes the article and announces the future work.

2 AOSP (ARCHITECTURE ORIENTED SOFTWARE PROCESS) APPROACH

AoSP exploits the progress of two research fields to improve the SPs reusing: ontology and software architectures. The proposed approach has two steps [4]:

- Knowledge capitalization by reverse engineering applied to existing SP models. For this aim we use domain ontology that capitalizes the pertinent SP knowledge. The capitalized knowledge is used to do the SP pre-modeling.
- Effective SP knowledge reusing by describing and deploying the extracted SP knowledge as software architectures. This step constitutes the SP final modeling.

The main objectives of AoSP are as follows:

- Suggest a general solution: that can be applied for different kinds of SP models.
- Increase the SP reuse: by exploiting the precedent SP modeling and enactment experiences.
- Increase the SP re-usability: by modeling reusable SP models and dealing with the SP models complexity and rigidity.
- Increase the SP quality: by giving the essential characteristics, such as comprehension, modeling and analyzing facilities, agility and evolution. These characteristics are often difficult to obtain as SPs are described as complex and depend on the kind of the used Process Modeling Language (PML) and the adopted terminology.

Architecture oriented Software Process (AoSP) approach suggests a new vision to model SPs; describing SP architectures offers the possibility to separate the SP preoccupations such as the execution control, the interaction and the SP model structure. Also, it allows greater flexibility: separating the process content from the process structure and exploiting the deployment mechanisms reduce the SP models dependency on their environments and PMLs.

The objective of AoSP is not to suggest a new PML but to reuse existing tools and PMLs. According to software architectures specificities, the SP modeling is decomposed into two steps:

- Pre_modeling: In this step we model the different SP preoccupations separately (structure, interaction and treatment). This step increases SP model comprehension and has a direct impact on SP modeling and analyzing facilities.
- Final modeling: In this step we deploy the SP architecture that can be done with different PMLs specific to different SP kinds. The deployment must be in an automatic way by developing code generators. This possibility gives a general aspect to our approach and increases the SP reusability.

3 SOFTWARE PROCESS ARCHITECTURE DESCRIPTION

To describe SP architectures, we study existing approaches that exploit architectural elements to model SPs. Our SP architectural concepts identification is based on ADL (Architecture Description Language) approach [24, 2], as the ADLs have more pertinent semantics than the traditional architecture description approaches. The ADLs introduce explicitly architectural concepts, techniques and tools that allow describing the software architectures rigorously. Our motivation is to exploit existing software architectures tools to describe our SP architectures. Thus, in our work, we study the architectural elements commonly accepted by the software architecture community [25]: component, connector, configuration, interface component, interface connector and architectural style.

3.1 Insufficiencies of the Reusing Approaches Based on Software Architectures

In most reusing approaches based on software architectures [6, 8, 12, 14] the central concept is the SP Component. A SP component in an activity (Works Unit) or an activities sequence. The SP Component is explicit in most approaches. The SP component interface is the work product required or given by the SP component [26, 12]. Configuration is in general implicit and not exploited formally, particularly, in the approaches based on components. Formal rules that describe the SP components and SP connectors assembling are not defined explicitly [26, 12, 1, 14]. For the connector concept there is no consensus on its interpretation, the SP connector can be:

- implicit and describes a dependency between activities, it can be a precedence link or a delegation link [12];
- predefined and depends the used PML [23, 14];

- explicit but considered as software connector and not specific to the SP domain (communication unit, communication protocols) [11].

Approach	**Category**	**The used architectural element**
PYNODE [6]	Component oriented.	Component, interface component, implicit connector.
APEL [15]	Component oriented.	Component, interface component (signature), implicit connectors.
RHODES [12]	Component oriented.	Component, interface component, composite component, implicit connectors (function call).
SPEM [26]	Component oriented.	Component (Activity), interface component (Work Product Ports), implicit connectors (Work Product Port Connectors).
Connectors for bridging SP models [23]	Connector oriented.	Three predefined connectors, implicit component (SP models).
Supporting sensitive SPs [1]	Connector oriented.	Component, explicit connector, implicit configuration.
Acquisition and simulation of process architectures [11]	Architecture oriented.	Component (SP model), interface component, explicit connector (communication module), implicit configuration.
App. based on evolution process components [14]	Architecture oriented.	Component, predefined connectors (concurrent, selection, sequence), configuration.

Table 1. Architectural elements of the studied approaches based on software architectures

Table 1 summarizes the architectural elements used in the existing approaches for reusing SPs based on software architectures. The objective of these approaches is to reuse their own SP components with their own PMLs, thus, the proposed semantics are specific to each approach and depend on the used PMLs; this explains the personal interpretations and the lack of consensus on the architectural elements. We summarize the insufficiencies of these approaches as follows:

- Limited reuse: The reusable elements such as SP components and SP connectors are defined for internal use, they are described with particular PMLs and cannot be reused by other environments.
- Under exploitation of architectural elements: Configuration and assembling constraints are not exploited; architectural styles and explicit reusable connectors specific to SP models are not defined.
- No general solution: Every approach deals with a particular problem and uses a particular PML (simulation [11], evolution [14], distribution [15], interaction [1]), there is no generic solution that can be applied for a large range of SPs.

- No SP architecture deployment: No deployment mechanisms or code generation are proposed, even if there is some assistance [12]; the final version of the SP model is modeled by the SP developer.

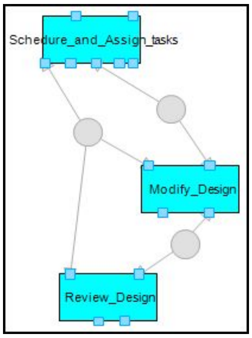
3.2 Software Process Architecture Description

Based on existing SP reusing approaches insufficiencies, combining with ADL approaches, we define a complete semantic to describe SP architectures. Our aim is to describe the SP model as software architecture and exploit the advantages offered by the software architecture domain but by respecting the SP characteristics such as human dimension [29] and unexpected events during the SP execution. Thus a SP Architecture is set as software architecture of SP components that interact through SP connectors, it describes the principles and guidelines that govern there design and evolution.

The interactions have a central place in the SP model [1]; moreover, the SP is human centered; thus, it is important to manage the different kinds of interactions. Our analysis is oriented to give a solution to handle the different kinds of SP interactions. Defining SP connectors that can adapt and facilitate the SP interactions is the adopted solution. We define our SP connector as an activity that facilitates and controls the transmission flow between the SP activities. A SP connector does not create new products, but adapts, evaluates and controls existing products. The distinction between “creation” activities and “adaptation and control” activities is the basis of the SP architectural concepts interpretations; thus, we define our SP architectural elements as follows:

- SP component: the SP component is a treatment done on input work products to “create” out work products.
- SP port: The SP component interface is a set of SP ports. It represents connection points that allow sending or receiving the flow (data flow or control flow) of the SP component. Two kinds of SP ports are defined: data flow ports and control flow ports.
- SP connector: The SP connector is an activity that assures and adapts the SP data transmissions. It also allows evaluation and control of the execution flow. It is independent from the used developement method but depends on the execution and the structure of the SP model.
- SP connector role: The SP connector interface is set of SP connector roles. It represents connection points that allow sending or receiving the flow (data flow or control flow) of the SP connector. Two kinds of SP connector roles are defined: data flow connector roles and control flow connector roles.
- SP configuration: as software configuration, it describes an assembly of SP components and SP connectors by determining explicitly the connection and the assembly constraints that must be respected.

- SP style: The SP architecture style is defined as a structural style that the execution policy can be formalized by combining an adequate execution style. SP architectural styles allow not only to capture recurrent structures, but also to capture recurrent execution policies.



```
System ISPW6 = {
    Component Schedure_and_Assign_tasks = {
        Port Notification_of_tasks_assignment_Dates_Port_out = {}
        Port Flow_Control__Port_out_1 = {}
     }
   Component Modify_Design = {
        Port FlowControl1 = {}
        Port Notification_of_tasks_assignment_Dates_Port_in = {}
        Port Flow_Control_Port_out_2 = {}
     }
    Connector Diffusion_Connector = {
        Role Notification_of_tasks_assignment_Dates_Role__in = {}
        Role Notification_of_tasks_assignment_Dates_Role_out = {}
        Role Notification_of_tasks_assignment_Dates_Role_out_2 = {}
     }
    Connector Precedence_Connector_1 = {
        Role Flow_Control_Role_in_1 = {}
        Role Flow__Control_role_out_1 = {}
     }
Attachment Schedure_and_Assign_tasks.
Notification_of_tasks_assignment_Dates_Port_out to
Diffusion_Connector.Notification_of_tasks_assignment_Dates_Role__in;
Attachment Modify_Design.Notification_of_tasks_assignment_Dates_Port_in to
Diffusion_Connector.Notification_of_tasks_assignment_Dates_Role_out;
Attachment Schedure_and_Assign_tasks.Flow_Control__Port_out_1 to
Precedence_Connector_1.Flow_Control_Role_in_1;
Attachment Modify_Design.FlowControl1 to
Precedence_Connector_1.Flow__Control_role_out_1;
    }
```

Figure 1. Partial ISPW-6 example [20] described with ACME studio according to AoSP approach

As the adopted semantics is based on ADL reasoning, we can use existing ADL tools to describe our SP architectures. Figure 1 depicts the partial view of ISPW6 example [20] described with ACME ADL [18]. As there are no ADLs specific to SP architectures we use ACME ADL, as ACME ADL is a generic ADL not specific to a particular domain, it allows the description of explicit connectors and architectural styles.

3.3 Explicit Connectors for SP Architectures

By analyzing the SP models behaviors, we notice that some adaptation activities are recurrent. Work product adaptation activities like work products fusion or work products fragmentation [15] are independent from the SP model type; these activities can be identified and reused. Thus, the data flow connector manages the data transmissions between SP components; it represents an activity that adapts the work product to be used by the connected SP components.

On the other hand, project management activities are defined to manage and evaluate the SP model execution; these activities can be also considered as SP connectors: the control flow connector is an activity that manages and controls the execution flow (order and quality). It allows controlling execution deviations by evaluating the SP execution flow. As shown in Figure 2, there are three kinds of control flow connectors:

1. connectors that provide only the execution order,
2. connectors that provide the execution order and evaluate the execution time, cost or quality result, and
3. connectors that evaluate and make decision about the SP execution progress.

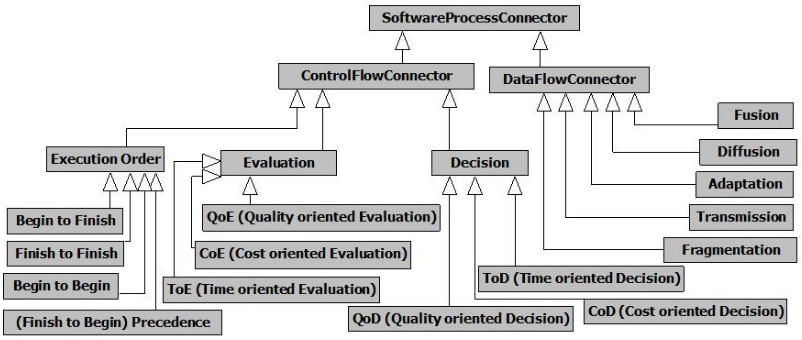

SoftwareProcessConnector
ControlFlowConnector
DataFlowConnector
Fusion
Diffusion
Adaptation
Transmission
Fragmentation
Execution Order
Evaluation
Decision
Begin to Finish
Finish to Finish
Begin to Begin
(Finish to Begin) Precedence
QoE (Quality oriented Evaluation)
CoE (Cost oriented Evaluation)
ToE (Time oriented Evaluation)
ToD (Time oriented Decision)
QoD (Quality oriented Decision)
CoD (Cost oriented Decision)

Figure 2. Software process connectors' taxonomy

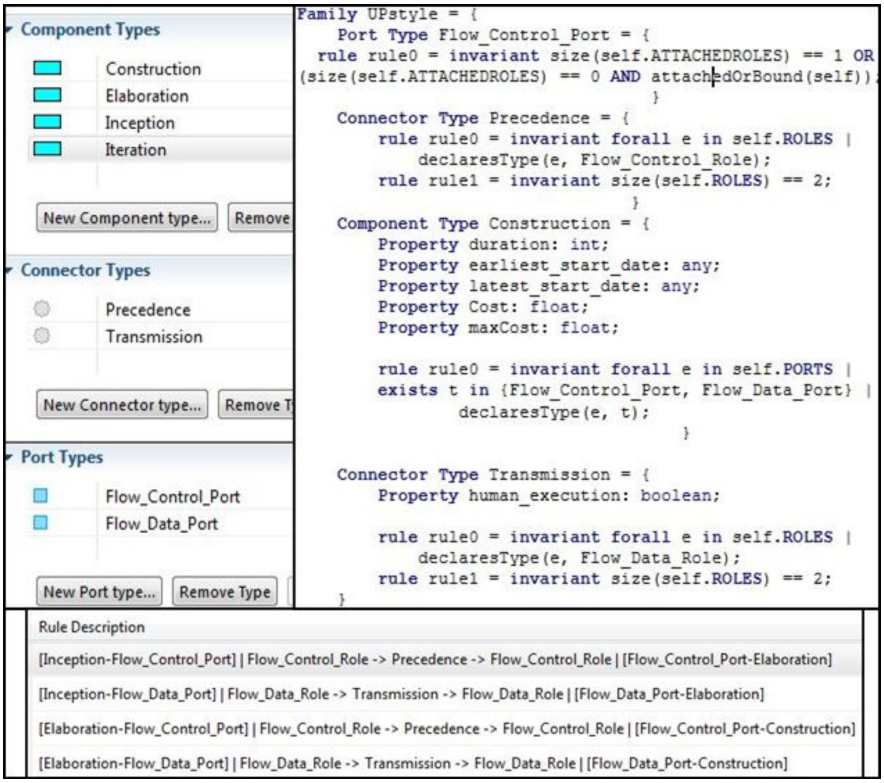
The SP connector can be a manual adaptation activity; in fact, some activities such as fusion or fragmentation cannot be done automatically. The distinction between automatic and manual activities is important; we can formalize the human interactions and anticipate the execution deviations that can be caused by human errors. To identify the human centered execution, we define the property "execution kind" for each SP connector that can have tow values: "human" or "automatic".

3.4 Architectural Styles for SP Architectures

For better modeling, existing approaches focus mainly on capturing recurrent SP structures and identifying best activities sequences. To improve the SP architecture description we can exploit these recurrent structures (lifecycles and process patterns [7]) to define our SP architectural styles.

However, the SP has a characteristic that the software product lacks: an SP model can be executed in different ways according to the development conditions. Development priorities (time, cost, quality of the result) in addition to the unexpected events directly affect the SP model execution policy. Consequently, the same SP model can have multiple executions instances without being able to differentiate bad execution well.

Without a clear vision on the desired execution policy, it is difficult to control the SP model execution and to make the right adaptation decisions. The problem is that the execution policy is not explicitly described in the SP; the execution policies are neither capitalized nor reused. In our work SP architectural styles must allow not only the capture of recurrent structures, but also of recurrent execution policies.



```
Family UPstyle = {
    Port Type Flow_Control_Port = {
  rule rule0 = invariant size(self.ATTACHEDROLES) == 1 OR
(size(self.ATTACHEDROLES) == 0 AND attachedOrBound(self));
                                       }
    Connector Type Precedence = {
        rule rule0 = invariant forall e in self.ROLES |
            declaresType(e, Flow_Control_Role);
        rule rule1 = invariant size(self.ROLES) == 2;
                                   }
    Component Type Construction = {
        Property duration: int;
        Property earliest_start_date: any;
        Property latest_start_date: any;
        Property Cost: float;
        Property maxCost: float;

        rule rule0 = invariant forall e in self.PORTS |
        exists t in {Flow_Control_Port, Flow_Data_Port} |
                declaresType(e, t);
                                         }

    Connector Type Transmission = {
        Property human_execution: boolean;

        rule rule0 = invariant forall e in self.ROLES |
            declaresType(e, Flow_Data_Role);
        rule rule1 = invariant size(self.ROLES) == 2;
    }
```

Rule Description
[Inception-Flow_Control_Port] | Flow_Control_Role -> Precedence -> Flow_Control_Role | [Flow_Control_Port-Elaboration]
[Inception-Flow_Data_Port] | Flow_Data_Role -> Transmission -> Flow_Data_Role | [Flow_Data_Port-Elaboration]
[Elaboration-Flow_Control_Port] | Flow_Control_Role -> Precedence -> Flow_Control_Role | [Flow_Control_Port-Construction]
[Elaboration-Flow_Data_Port] | Flow_Data_Role -> Transmission -> Flow_Data_Role | [Flow_Data_Port-Construction]

Figure 3. Description of SP structural style with ACME studio

An architectural style is a coordinated set of architectural constraints that restricts the role of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style [17]. In our work, this definition remains valid; SP component types, SP connector types and constraints that are used to describe an architectural style are also used to describe our SP architectural style. However, to formalize the execution style, the SP architectural style is defined as structural style that the execution policy can be identified by combining a particular execution style. As a result, every structural style can have different execution styles. We summarize the structural and the execution styles as follows:

- The structural styles describe recurrent SP structures. They focus on the work products treatments and transmissions and are independent from an execution policy. A standard execution is assured by using the “precedence connector”. We inspire from the existing life cycles and process patterns to define our structural styles; however, we can define particular structural styles that capture recurrent but unknown structures.

- The execution styles describe recurrent SP execution policies. Every execution style is defined by the control flow connectors only. The SP component types have no incidence on the SP execution policy.

The SP architectural style can be described with ACME studio. Figure 3 depicts a partial view of the structural style UP described with ACME studio, we depict the components types and the connectors types of the structural style. Assembling rules and some constraints are also described.

4 OUR SOFTWARE PROCESS DOMAIN ONTOLOGY

Using a domain ontology has multiple advantages:

- It offers a repository that share the SP knowledge extracted from precedent successful SP models. The SP knowledge can be extracted from heterogeneous SP models; the domain ontology provides a unified representation for both SP models.
- It allows to reuse SP models that were not dedicated for reusing (not oriented components for example) and that are reused manually via reports for example, that increases the SP model reusing.
- Ontologies are well-suited to combine information from various sources; it allows reasoning and inferring new facts based on this, and contributes to analyzing and emerging of new solutions for improving SP models.
- Ontology is designed to retrieve SP architectures; however, reasoning can be applied to reuse this knowledge for other purposes.

Our SP ontology must:

- be coherent, not ambiguous and commonly accepted;
- offer a conceptualization to store and retrieve SP architectures knowledge;
- manage the SP concept heterogeneity: the ontology must have a conceptualization that can be exploited for different SP models, without focusing on a particular SP kind;
- manage the heterogeneity at the instance level: capitalizing knowledge from various SP models can create ambiguities, indeed, even if there is consensus on the used terminology in the software development, the developers can use their own vocabulary;
- restore a comprehensible knowledge: a vocabulary reference that represents the vocabulary of the final user must be defined and stored;

- provide a tool for the analysis and the emergence of new solutions for improving SP models.

To develop a domain ontology, one of the first steps is the study the existing ones and considering their extension, fusion or adaptation. Many SP modeling approaches based on domain ontology are defined [19, 22, 27, 10]. These approaches use one or many ontologies to represent the SP knowledge. However, these solutions are specific and deal with particular SP models. They do not suggest a general solution that can be applied for a large range of SPs.

Appoach	**Objective**	**Ontology structure**
OnSSPKR Framework [19]	Deal with CMM, CMMI, ISO/IEC15504, ISO9001 models.	Three ontologies (Process experiences, Personal skills, Knowledge artifacts)
SPO (Software Process Ontology)[22]	Mapping between CMMI models and the ISO/IEC 15504 models	SP basic concepts
PCE based ontology [30]	Generate SP plans	Two ontologies (artifacts and activities)
Approach based descriptive logic [27]	Framework for software maintenance	Concepts that affect the software maintenance
Flexible PML based ontology [28]	Flexible SP models	Basic process elements

Table 2. Approaches for reusing SPs based on domain ontology

Table 2 summarizes the objectives and the structures of the existing ontologies. The ontologies defined by these approaches do not give answer to our expectations; they do not deal with the concepts and terminology heterogeneity, thus we cannot exploit these ontologies in our approach.

4.1 Software Process Ontology Conceptualization

To suggest a SP domain ontology that deals with our preoccupations, we exploit SPEM (Systems and Software Process Engineering Metamodel)[26] conceptualization. SPEM is a standard metamodel adopted by the OMG (Object Management Group) to describe the concepts of a large range of SPs. This choice is justified by many reasons:

- SPEM is a standard accepted by the SP community.
- SPEM covers a large range of SP concepts without focusing on a particular SP.
- SPEM is a UML profile so we can use existing tools and techniques such as model transformation in order to generate our ontology.

However, to describe and deploy SP architectures SPEM lacks important architectural concepts such as "explicit connector", "configuration" and "style". SPEM allows reusing based on components, but does not deal with reusing based on SP architectures. Consequently, SPEM must be extended with the required SP architectural elements.

4.2 SPEM Profile Extension

Having a complete semantics to describe a SP architecture the extension of SPEM profile can be done; for this purpose, we introduce new stereotypes to describe the required architectural elements.

The added stereotypes are organized into two categories: stereotypes that describe the SP style elements and the stereotypes that describe the SP configuration elements. Consequently, two abstract stereotypes are introduced: "process architectural element" and "method content architectural element".

We distinguish these two stereotypes as the "SP style" is a "method content package" and its elements are only "method content elements"; however, the "SP configuration" is a "process package" and its elements are only "process elements" (Figure 4).

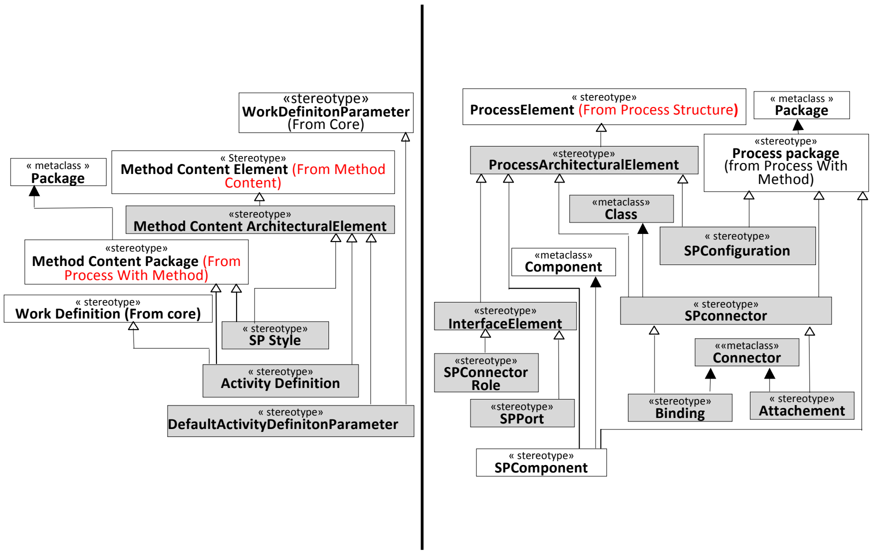

«stereotype» WorkDefinitonParameter (From Core)
« metaclass » Package
« Stereotype» Method Content Element (From Method Content)
«stereotype» Method Content ArchitecturalElement
«stereotype» Method Content Package (From Process With Method)
« stereotype» Work Definition (From core)
« stereotype» SP Style
« stereotype» Activity Definition
« stereotype» DefaultActivityDefinitonParameter
« stereotype» ProcessElement (From Process Structure)
« metaclass » Package
«stereotype» ProcessArchitecturalElement
«stereotype» Process package (from Process With Method)
«metaclass» Class
« stereotype» SPConfiguration
«metaclass» Component
«stereotype» InterfaceElement
« stereotype» SPconnector
««metaclass» Connector
«stereotype» SPConnector Role
«stereotype» SPPort
«stereotype» Binding
« stereotype» Attachement
« stereotype» SPComponent

Figure 4. Extension of method plugin profile with SP architectural elements [5]

Figure 5 depicts architectural elements added to extend the SPEM model. Two kinds of UML elements (classes, associations, attributes, etc.) are added:

UML elements that describe the SP architecture: A SP configuration is composed from SP components and SP connectors. The SP component is an activity that creates work products; it is composed of a set of SP ports. The SP connector is an activity that adapts and controls the data flow or control flow, it is composed of a set of SP connector roles. The SP components assembling is done via attachments. An attachment is done between a SP port and a SP connector that have the same kind (data flow or control flow).

UML elements that describe the SP style: the SP style is composed from activity definitions. Both SP components and SP connectors are activities; thus, activity definition identifies the SP connector types and the SP component types at the same time. In the same manner, work product definition describes the SP ports and the SP connector role types. The default assembling is described with default activity definition parameter.

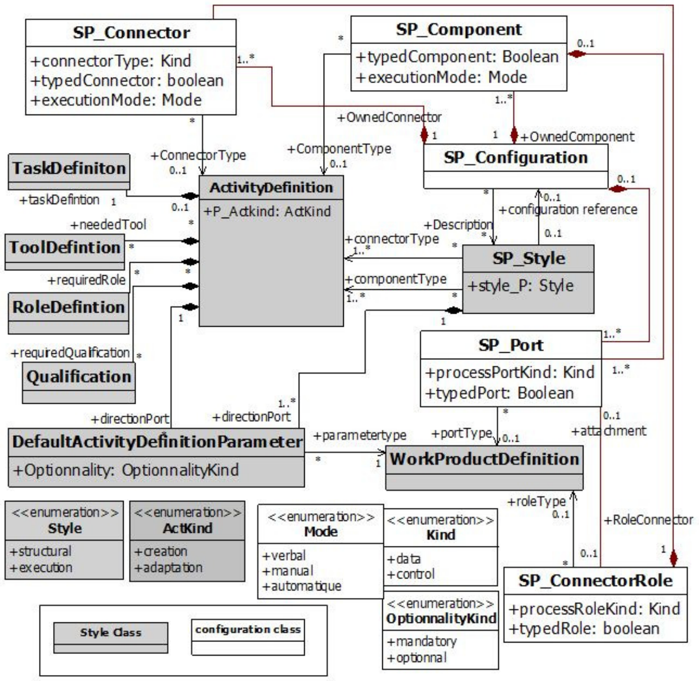


Figure 5. SPEM model extension with the required SP architectural elements

5 SPEMONTOLOGY GENERATION

SPEM being extended, we can generate our ontology that will allow describing and deploying SP architectures. Our ontology OWL is generated by applying many model transformations. To achieve this aim, we use ATL (Atlas Transformation

Language) modules UML2OWL and UML2COPY [3]. UML2OWL module is developed to transform an UML model to an OWL model. As SPEM is an UML profile, we cannot use this module directly; in fact, in UML2OWL ATL module there are no ATL rules that allow the transformation of profiles, stereotypes and tagged values to OWL elements. Thus we had developed an ATL transformation module (ApplySPEMProfile2SPEMModel Module) that applies SPEM profile to SPEM model before applying UML2OWL transformation.

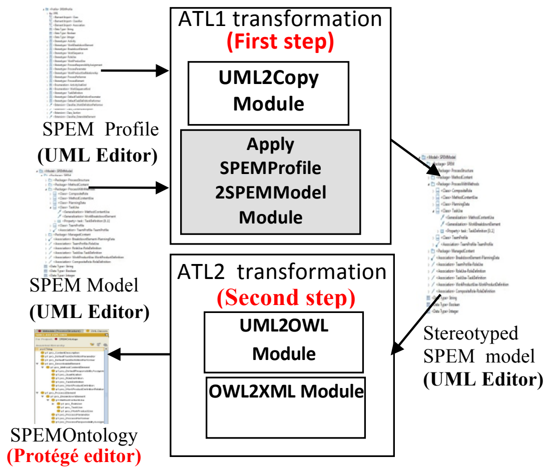


Figure 6. ATL transformations for genrating SPEMontology

Consequently the transformation of SPEM metamodel to an OWL Ontology is performed by two ATL transformations ATL1 and ATL2 (Figure 6)

- ATL1 is an ATL transformation used to prepare the SPEM metamodel to be transformed into an OWL model. It applies the SPEM profile to the SPEM model in order to have a stereotyped UML model (stereotyped SPEM model). Knowing that SPEM has stereotyped and non-stereotyped elements, ATL1 is composed from two modules:
  - UML2COPY module: copies directly the non-stereotyped elements of the SPEM model into the output model (stereotyped SPEM model).
  - ApplySPEMProfile2SPEMModel module: applies the stereotypes of SPEM profile to the corresponding elements of SPEM model. Unlike the other modules, this module is specific to SPEM model; in fact, the rules are written according to the name of the SPEM model elements and the name of SPEM profile elements.

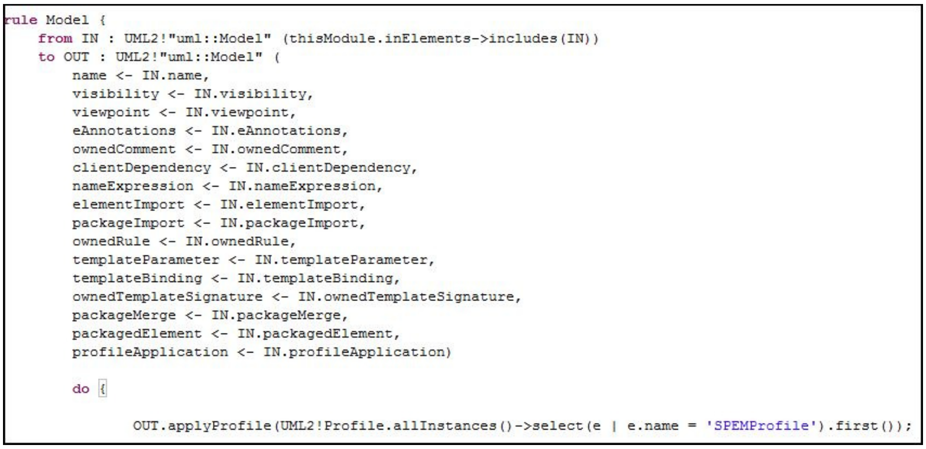
  As there is no ATL rule for the same element in the two modules at the same time, these modules are executed in parallel.

- ATL2 is the UML2OWL transformation. This exisiting transformation is used to generate an OWL ontology (SPEMOntology) from the stereotyped SPEM model (the result of ATL1 transformation).

5.1 ApplySPEMProfile2SPEMModel Module

The execution of ApplySPEMProfile2SPEMModel Module is as follows:

- Definition of the SPEM profile and SPEM model and their metamodels with adequate tools. The extension of the files must be ".uml" and that of the metamodels must be ".ecore".
- Application of the SPEM profile to the SPEM model: This step is performed using the rule "ApplyProfile2Model" (Figure 7). This rule is a matchedRule applied when an instance of the element "UML2!Model" is recognized in the source model. It allows copying the SPEM model and all its characteristics (name, visibility, etc.). The rule has an imperative part introduced by the keyword "do" in which the SPEM profile is applied to the SPEM model using the "applyProfile()" method.

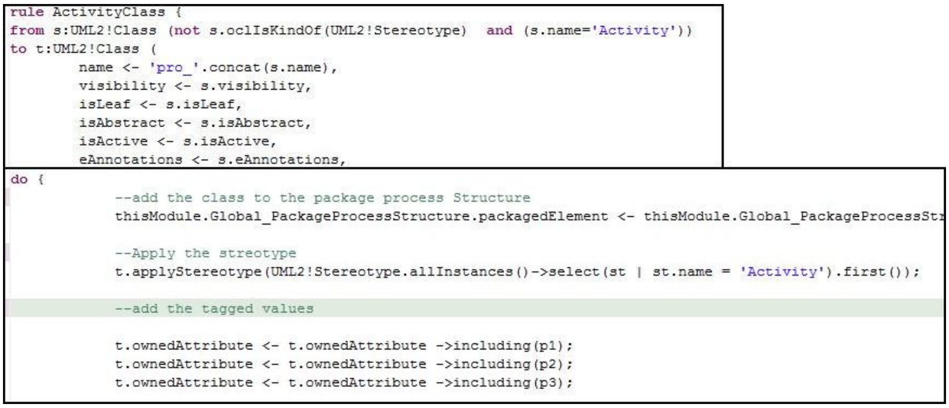


```
rule Model {
    from IN : UML2!"uml::Model" (thisModule.inElements->includes(IN))
    to OUT : UML2!"uml::Model" (
        name <- IN.name,
        visibility <- IN.visibility,
        viewpoint <- IN.viewpoint,
        eAnnotations <- IN.eAnnotations,
        ownedComment <- IN.ownedComment,
        clientDependency <- IN.clientDependency,
        nameExpression <- IN.nameExpression,
        elementImport <- IN.elementImport,
        packageImport <- IN.packageImport,
        ownedRule <- IN.ownedRule,
        templateParameter <- IN.templateParameter,
        templateBinding <- IN.templateBinding,
        ownedTemplateSignature <- IN.ownedTemplateSignature,
        packageMerge <- IN.packageMerge,
        packagedElement <- IN.packagedElement,
        profileApplication <- IN.profileApplication)

        do {

               OUT.applyProfile(UML2!Profile.allInstances()->select(e | e.name = 'SPEMProfile').first());
```

Figure 7. ATL rule for applying SPEM profile to SPEM model with "applyProfile" method

- Application of SPEM stereotypes to the SPEM elements: For each SPEM element (class or association) we associate the following matchedRule: in the declarative part, the element and all its characteristics are copied, and in the imperative part, the generated element is placed in the corresponding package; then the corresponding stereotype is applied using the "applyStereotype()" method (Figure 8).

  In order to distinguish stereotyped elements from the on-stereotyped elements in the OWL ontology, we add the prefix "Pro" to the name of the stereotyped element.



```
rule ActivityClass {
from s:UML2!Class (not s.oclIsKindOf(UML2!Stereotype)  and (s.name='Activity'))
to t:UML2!Class (
        name <- 'pro_'.concat(s.name),
        visibility <- s.visibility,
        isLeaf <- s.isLeaf,
        isAbstract <- s.isAbstract,
        isActive <- s.isActive,
        eAnnotations <- s.eAnnotations,
do {
            --add the class to the package process Structure
            thisModule.Global_PackageProcessStructure.packagedElement <- thisModule.Global_PackageProcessStr

            --Apply the streotype
            t.applyStereotype(UML2!Stereotype.allInstances()->select(st | st.name = 'Activity').first());

            --add the tagged values

            t.ownedAttribute <- t.ownedAttribute ->including(p1);
            t.ownedAttribute <- t.ownedAttribute ->including(p2);
            t.ownedAttribute <- t.ownedAttribute ->including(p3);
```

Figure 8. Partial view of ATL rule for applying SPEM stereotype to SPEM element (Activity class) using the "applyStereotype" method

- Tagged values transformation that will be detailed in the next section.

The methods "applyProfile()" and "applyStereotype()" are provided by the UML2.0 plugin in the Eclipse environment. This plugin is essential for carrying out such transformations. It also provides the UML2.0 meta-model described in Ecore to which UML models (SPEM model, SPEM profile and the stereotyped SPEM model) conform.

5.2 Tagged Values Transformation

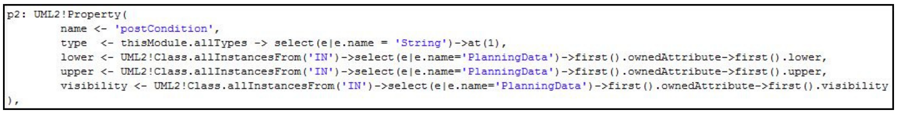
To transform the "tagged values", we proceed as follows:

We notice that a tagged value can only be "data type property" at ontology level; in fact, the "data type property" links an OWL concept to an "OWL data type", which is the case of the "tagged value" that links an UML class to an "UML data type". In the UML2OWL ATL transformation an OWL "data type property" is generated only from an attribute of an UML class. Consequently, at the SPEM model each tagged value is considered as an attribute of an UML class.

The "tag definition" of a particular stereotype, once applied to the corresponding class, corresponds to a tagged value. Thus, for each class that the corresponding stereotype includes tag definition, an UML attribute that corresponds to its tagged value will be created.

To achieve this, we add an adequate instruction to the matchedRule (declarative part) of each class whose stereotype has "tag definitions" (Figure 9). These instructions will create an UML attribute for each tag definition. In the imperative part of the rule, the UML attribute is added to the stereotyped class (Figure 8). When creating an UML property the name, type, multiplicity and visibility are specified as follows (Figure 9):

- Name: the name of the corresponding tag definition.
- Type: the type of tag definition, which may be an UML data type or may take values of an enumeration. If the tag definition takes the values from an enumeration, the type of the UML attribute will be the UML DataType “string”.
- The visibility of the properties of SPEM model is “public”, so any generated Property has a “public” visibility.
- The multiplicity of an attribute indicates the number of values that can be associated to it. In our case the maximum number of values associated with an attribute is at most one. Thus, lower and upper values of any generated tagged value property are “zero” and “one” respectively.



```
p2: UML2!Property(
        name <- 'postCondition',
        type  <- thisModule.allTypes -> select(e|e.name = 'String')->at(1),
        lower <- UML2!Class.allInstancesFrom('IN')->select(e|e.name='PlanningData')->first().ownedAttribute->first().lower,
        upper <- UML2!Class.allInstancesFrom('IN')->select(e|e.name='PlanningData')->first().ownedAttribute->first().upper,
        visibility <- UML2!Class.allInstancesFrom('IN')->select(e|e.name='PlanningData')->first().ownedAttribute->first().visibility
),
```

Figure 9. Partial view of ATL rule for defining tagged values

5.3 SPEMOntology Structure

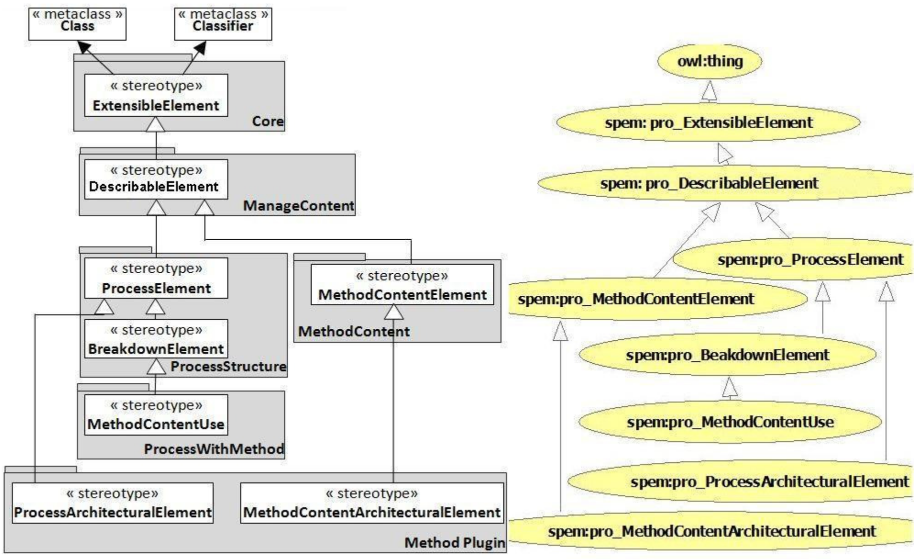

« metaclass » Class
« metaclass » Classifier
« stereotype» ExtensibleElement
Core
« stereotype» DescribableElement
ManageContent
« stereotype» ProcessElement
« stereotype» BreakdownElement
ProcessStructure
« stereotype» MethodContentElement
MethodContent
« stereotype» MethodContentUse
ProcessWithMethod
« stereotype» ProcessArchitecturalElement
« stereotype» MethodContentArchitecturalElement
Method Plugin
owl:thing
spem: pro_ExtensibleElement
spem: pro_DescribableElement
spem:pro_ProcessElement
spem:pro_MethodContentElement
spem:pro_BeakdownElement
spem:pro_MethodContentUse
spem:pro_ProcessArchitecturalElement
spem:pro_MethodContentArchitecturalElement

Figure 10. SPEMontology structure (class view/concept view)

SPEMOntology is the result of successive ATL transformations. It is constituted from 56 concepts and an important number of data and object properties. In order to facilitate its understanding, it is important to describe its organization. SPEM is structured into seven packages [26]. By analyzing the SPEM packages (after the extension), we notice that every SPEM package has its abstract class that regroups the common behavior of the package classes.

Figure 10 (left) depicts the main abstract classes of every SPEM package. We notice the existence of the concepts introduced to describe SP architectures; all the classes (abstract or not) are also transformed into OWL concepts. In our work we had not exploited the Process Behavior package, thus it is not represented here. After the ATL transformations this organization can be identified (Figure 10 (right)). The SPEM packages view can be identified through the main concepts of SPEMOntology.

6 SOFTWARE PROCESS KNOWLEDGE AQUISITION

The concepts heterogeneity finds solution by exploiting a standard metamodel. The heterogeneity at instance level is treated by separating every kind of knowledge. Indeed, our ontology must store four kinds of knowledge:

The SP architecture knowledge: our ontology must allow retrieving SP architectures. The capitalized knowledge concerns SP configurations and SP styles.

The used knowledge: our ontology must allow the reuse of exiting SP models. The capitalized knowledge concerns the know-how of existing SP models.

The reference vocabulary: Our ontology must retrieve a comprehensible SP knowledge. The capitalized knowledge concerns the vocabulary used by the final stakeholders.

The instance heterogeneity management: our ontology must manage the heterogeneous vocabulary. Thus, we must do the correspondence between the used knowledge and the reference vocabulary.

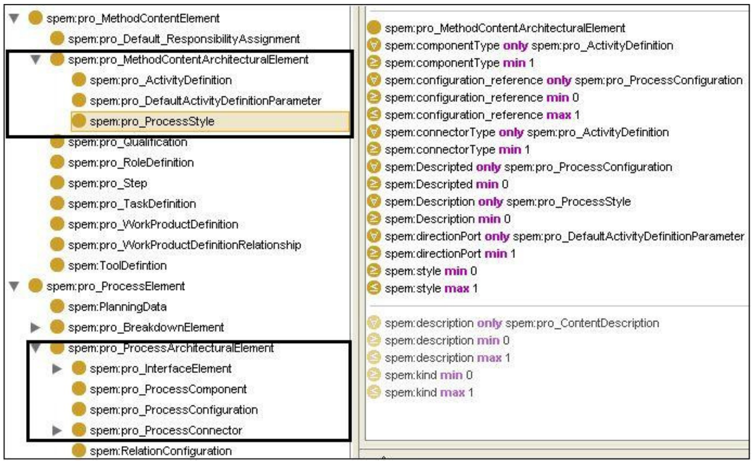
We exploit the structure of SPEM to deal with the instances heterogeneity. Each SPEM package is used to store a kind of knowledge. In the next sections we detail how we capitalize each kind of SP knowledge.

6.1 The SP Architectures Knowledge Capitalization

The SP expert stores the SP configurations and the SP styles of the company. This step is very important as it allows describing formally the company development policies and practices.

The knowledge can be recurrent SP configurations, well-known life cycles or particular processes that the company may use as SCRUM or XP processes.

The instantiation is done on the "process architectural element" concepts and the "method content architectural element" concepts (Figure 11). This step is done manually by a SP expert of the company. However, the advantage is that it is done once and it will be reused independently from the SP expert intervention.



```
▼ spem:pro_MethodContentElement
    spem:pro_Default_ResponsibilityAssignment
  ▼ spem:pro_MethodContentArchitecturalElement
      spem:pro_ActivityDefinition
      spem:pro_DefaultActivityDefinitionParameter
      spem:pro_ProcessStyle
    spem:pro_Qualification
    spem:pro_RoleDefinition
    spem:pro_Step
    spem:pro_TaskDefinition
    spem:pro_WorkProductDefinition
    spem:pro_WorkProductDefinitionRelationship
    spem:ToolDefintion
▼ spem:pro_ProcessElement
    spem:PlanningData
  ▶ spem:pro_BreakdownElement
  ▼ spem:pro_ProcessArchitecturalElement
    ▶ spem:pro_InterfaceElement
      spem:pro_ProcessComponent
      spem:pro_ProcessConfiguration
    ▶ spem:pro_ProcessConnector
      spem:RelationConfiguration
```

```
spem:pro_MethodContentArchitecturalElement
spem:componentType only spem:pro_ActivityDefinition
spem:componentType min 1
spem:configuration_reference only spem:pro_ProcessConfiguration
spem:configuration_reference min 0
spem:configuration_reference max 1
spem:connectorType only spem:pro_ActivityDefinition
spem:connectorType min 1
spem:Descripted only spem:pro_ProcessConfiguration
spem:Descripted min 0
spem:Description only spem:pro_ProcessStyle
spem:Description min 0
spem:directionPort only spem:pro_DefaultActivityDefinitionParameter
spem:directionPort min 1
spem:style min 0
spem:style max 1

spem:description only spem:pro_ContentDescription
spem:description min 0
spem:description max 1
spem:kind min 0
spem:kind max 1
```

Figure 11. SP architectural concepts of SPEMOntology

6.2 The Used Knowledge Aquisition

We instantiate the concepts of "process with method" and "process structure" packages. In SPEM these packages are used to describe the effective use of the content methods and the SP fragments, respectively, independently of particular development method [26]. In our ontology, these concepts are used to capitalize the used know-how that are collected from the existing SP models (Figure 12).

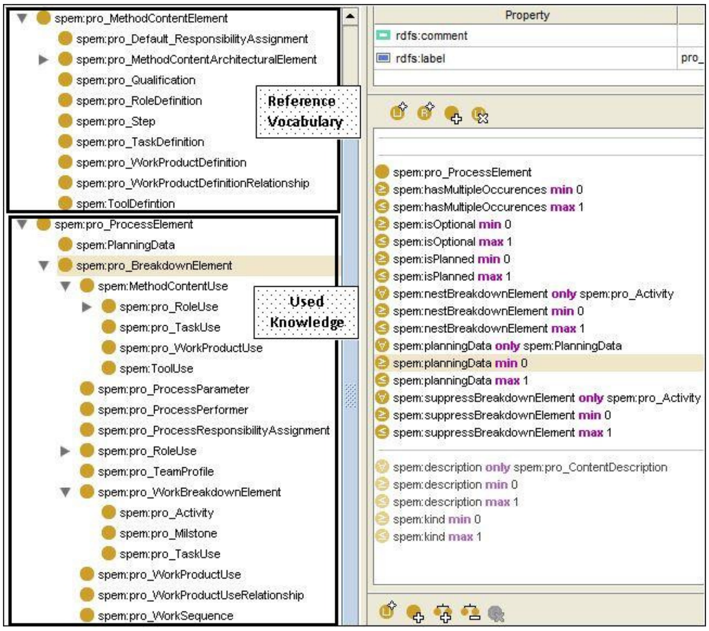
This step is done automatically; we apply a reverse engineering on every SP model that will be reused. For each PML we develop an instantiation program that identifies the pertinent concepts and allows the extraction of the pertinent knowledge.

As example we develop an instantiation program for EPF (Eclipse Process Framework) models [16] and another for PBOOL+ models [13]. The knowledge acquisition from EPF models is direct as these models conform to SPEM; however, for PBOOL+ models there is no "task use" concept, so then, we suppose that every elementary activity of PBOOL+ model is constituted from one "task use" and do the instantiation.

6.3 The Reference Vocabulary Capitalization

In SPEM the method content package is dedicated to describe development methods independently from their use: "... The method content package defines the core elements of every method such as roles, tasks, and work product definitions..." [6]. We use these concepts to describe the vocabulary reference (Figure 12).

The method content concepts are solicited to describe many kinds of knowledge: method content elements, vocabulary reference and architectural types. To



```
▼ spem:pro_MethodContentElement
    spem:pro_Default_ResponsibilityAssignment
  ▶ spem:pro_MethodContentArchitecturalElement
    spem:pro_Qualification
    spem:pro_RoleDefinition
    spem:pro_Step
    spem:pro_TaskDefinition
    spem:pro_WorkProductDefinition
    spem:pro_WorkProductDefinitionRelationship
    spem:ToolDefintion
▼ spem:pro_ProcessElement
    spem:PlanningData
  ▼ spem:pro_BreakdownElement
    ▼ spem:MethodContentUse
      ▶ spem:pro_RoleUse
        spem:pro_TaskUse
        spem:pro_WorkProductUse
        spem:ToolUse
      spem:pro_ProcessParameter
      spem:pro_ProcessPerformer
      spem:pro_ProcessResponsibilityAssignment
    ▶ spem:pro_RoleUse
      spem:pro_TeamProfile
    ▼ spem:pro_WorkBreakdownElement
        spem:pro_Activity
        spem:pro_Milstone
        spem:pro_TaskUse
      spem:pro_WorkProductUse
      spem:pro_WorkProductUseRelationship
      spem:pro_WorkSequence
```

```
rdfs:comment
rdfs:label                                     pro_

spem:pro_ProcessElement
spem:hasMultipleOccurences min 0
spem:hasMultipleOccurences max 1
spem:isOptional min 0
spem:isOptional max 1
spem:isPlanned min 0
spem:isPlanned max 1
spem:nestBreakdownElement only spem:pro_Activity
spem:nestBreakdownElement min 0
spem:nestBreakdownElement max 1
spem:planningData only spem:PlanningData
spem:planningData min 0
spem:planningData max 1
spem:suppressBreakdownElement only spem:pro_Activity
spem:suppressBreakdownElement min 0
spem:suppressBreakdownElement max 1

spem:description only spem:pro_ContentDescription
spem:description min 0
spem:description max 1
spem:kind min 0
spem:kind max 1
```

Figure 12. The Used knowledge concepts of SPEMOntology

distinguish between these kinds of knowledge, for the "method content element" concept we add a data type property "concept role" that can have the next values: "MC" for method content knowledge, "VR" for vocabulary reference and "AT" for architectural type.

The weakness of this step is that the instantiation is done manually; it depends on the experience of the SP expert and on the users groups. Every company has its own vocabulary and its own abbreviation and terminology convention.

However, the advantage of this manual step is that it formally defines the terminology of the company. It allows not only a better comprehension of the SP models, but also constitutes a contribution to capitalize company's know-how that will be used and reused formally, independently from the SP experts and their tacit knowledge.

6.4 The Instance Heterogeneity Management

To manage the instances heterogeneity, we must provide a correspondence between the used knowledge and the reference vocabulary. This correspondence is provided by using existing associations between method content concepts and process with method concepts (Figure 13).

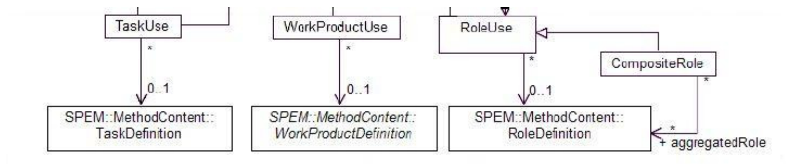

TaskUse
WorkProductUse
RoleUse
CompositeRole
*
0..1
SPEM::MethodContent::TaskDefinition
SPEM::MethodContent::WorkProductDefinition
SPEM::MethodContent::RoleDefinition
+ aggregatedRole

Figure 13. SPEM associations used to manage the SP knowledge

7 AOSP APPROACH EVALUATION

In this paper we present only the engineering "by" reusing SPs of AoSP approach, the engineering "for" reusing SPs is not presented; thus, SP architecture retrieving, SP architecture inference and SP architecture deployment are not detailed.

However, even if the AoSP appoach is not presented in full and having a first operationnal result (the SPEMOntology), we can do a first evaluation of AoSP.

The initial goal	**to this aim we have...**	**as future work we can...**
Suggest a general solution	used a standard metamodel that describes concepts of a large range of SPs.- give a uniform representation for different kinds of SP models	develop a deployment program for each SP model kind that we need to generate
Increase the reuse of existing SP models	developed a domain ontlogy "SPEMOntology" that capitalizes SP knowledge from existing SP models independently from their PML or metamodel	infer new solutions and new SP architectures
Increase the reusability of the modeled SPs	exploited the abstract structure at the modeling step; defined reusable SP connectors; defined SP styles; defined execution styles	many SP architecture deployments such as: dynamic, distributed and using different PMLs; describe SP components ans SP connectors with XML files
Increase the SP model quality	used the SP architectures that allow modeling flexible, comprehensible SP models; used a domain ontology that allows exploiting the precedent SP modeling experiments	retrieve optimal solution according to the development context (time, cost or quality oriented)

Table 3. AoSP approach evaluation

Both reusing approaches based on software architecture and approaches based on ontology suggest particular solutions for particular problems (simulation, evolution, interaction, software maintenance...); however, our solution is general and can be used for many kinds of SPs (capitalize or retrieve SP models). Also, unlike existing approaches, AoSP exploits all the opportunities offered by the software architectures and domain ontologies to model and execute SPs.

According to Table 3 we had globally achieved our primary goals; in fact, to increase the SP reusing and reusability, we have defined an ontology that can capitalize heterogeneous SP models and we have exploited the software architecture principles to model reusable SPs. The described SP architectures give the SP model more flexibility, comprehension and dynamicity.

However, this evaluation is not complete; in fact, the engineering "by" reusing SPs is not presented and can give more advantages to our approach. The SP architectures knowledge retrieving, the SP architecture deployment and SP knowledge inference increase, as well as the SP reusing.

7.1 The SP Knowledge Inference

Infer new knowledge is one of the advantages of a domain ontology. The large size of SPEMOntology gives us the ability to infer various SP knowledge. The inference of new knowledge must have a clear goal and follow a coherent reasoning. In our case, our inference rules are organized into two kinds:

- Inference rules that manage the consistency of knowledge base. These rules must manage the consistency and coherence of the SP knowledge extracted from different sources. In fact, the extraction of knowledge from many sources can create "gaps" in the knowledge base.
- The inference rules that allow the emergence of new solutions (new activities or components assembling for example), and allow exploring optimal solutions according to specific criteria. For example two kinds of knowledge can be inferred: "equivalent SP configuration", the rule identifies the SP configurations that can replace a particular SP configuration and "equivalent SP components": the rule identifies the SP components that can replace a particular SP component. We can define other rules that infer equivalent configuration that optimizes the execution time, execution cost or quality results.

We can give numerous examples of rule inference; however, this work is in progress and will be subject of future articles.

Example 1. $SPEM : \mathit{predecessor}(?SPEM : W, ?SPEM : Y) \rightarrow SPEM : \mathit{successor}(?SPEM : Y, ?SPEM : W)$: If W is a predecessor of Y then Y is a successor of W.

Example 2. $SPEM : \mathit{linkedTaskUse}(?SPEM : x,\ ?SPEM : y) \wedge SPEM : \mathit{ownedProcessParameter}(?SPEM : y,\ ?SPEM : z) \wedge SPEM : \mathit{ownParameters}(?SPEM : y, ?SPEM : w) \rightarrow SPEM : \mathit{isPerformed}(?SPEM : x,\ ?SPEM : w)$: if the performer x is associated to the task use y and y is a task use of activity w then x is a performer of activity w.

8 CONCLUSION

AoSP is a SP reusing approach that offers a standard solution to increase the reuse and the reusability of SP models.

This paper presents the engineering "for" reusing SPs of AoSP (architecture oriented software process) approach. AoSP exploits the software architecture principles to model SPs. By separating the SP modeling preoccupations: work product treatments (components), work product transmissions (data flow connectors) and execution control (control flow connectors), AoSP offers an innovative vision of the SP modeling. It allows modeling more comprehensible, flexible and controllable SP models.

Based on existing approaches insufficiencies, we define a complete semantics to describe SP architectures. We model SP architecture as software architecture but we respect the SP specificities such as human dimension and the characteristics of the SP execution. In addition, we define explicit connectors and architectural styles specific to the SP architectures. On the other hand, AoSP uses a domain ontology to capitalize the best practices of the software development domain. It exploits the capitalized knowledge to retrieve and deploy SP architectures.

The ontology conceptualization is discussed, it is based on SPEM. We extend SPEM by introducing required architectural concepts. Indeed, SPEM deals with SP reusing based on components and lacks important architectural elements to describe SP architectures. The ontology was generated by transformation model techniques; to this aim, we use ATL modules (UML2OWL and UML2COPY) and we develop our ATL module "applySPEMprofile2SPEMmodel" to apply SPEM profile to SPEM model.

To describe and deploy the SP architectures, SPEMOntology must store different kinds of knowledge: The used know-how, the SP architecture knowledge and the reference vocabulary. In addition, it must provide a correspondence between these kinds of knowledge. We exploit the SPEM structure (organized into packages) to separately store these kinds of knowledge. We add adequate properties to keep the knowledge coherence.

Actually we are working on the engineering "by" reusing: first results of inferring, retrieving and deploying SP architectures have been obtained, but must be refined before publishing.

REFERENCES

[1] Alloui, I.—Oquendo, O.: Supporting Decentralised Software-Intensive Processes Using ZETA Component-Based Architecture Description Language. International Conference on Entreprise Information Systems 2001, pp. 207–218.

[2] Atli, A.—Khammaci, T.—Smeda, A.: Integrating Software Architecture Concepts into the MDA Platform with UML Profile. J. of Computer Science, Vol. 3, 2007, No. 10, pp. 793-802.

[3] Atlas Transformation Language, ATL: ATL transformations list. 2007, `http://www.eclipse.org/m2m/atl/atlTransformations/`.

[4] Aoussat, F.—Ahmed-Nacer, M.—Oussalah, M.: Reusing Approach for Software Processes based on Software Architectures. International Conference on Entreprise Information Systems 2010, pp. 366–369.

[5] Aoussat, F.—Oussalah, M.—Ahmed Nacer, M.: SPEM Extention with Software Process Architectural Concepts. COMPSAC 2011, pp. 215–225.

[6] Avrilionis, D.—Belkhatir, N.—Cunin, P.-Y.: A Unified Framework for Software Process Enactment and Improvement. 4th International Conference on the Software Process 1996, pp. 102–108.

[7] Barchetti, U.—Capodieci, A.—Lisa Guido, A.—Mainetti, L.: Modelling Collaboration Processes Through Design Patterns. Computing and Informatics, Vol. 30, 2011, No. 1, pp. 113–135.

[8] Belkhatir, N.—Estublier, J.: Supporting Reuse and Configuration for Large Scale Software Process Models. 10th International Software Process Workshop 1996, pp. 35–40.

[9] Bendraou,R.—Gervais, M. P.—Blanc, X.: UML4SPM: An Executable Software Process Modelling Language Providing High-Level Abstractions. 10th IEEE International Enterprise Distributed Object Computing Conference 2006, pp. 297–306.

[10] Bermejo-Alonso, J.—Sanz, R.—Rodriguez, R.—Hernandez, C.: Ontology-Based Engineering of Autonomous Systems. In Proceedings of the Sixth International Conference on Autonomic and Autonomous Systems (ICAS) 2010.

[11] Choi, J.—Scacchi, W.: Modeling and Simulating Software Acquisition Process Architectures. J. of Systems and Software, Vol. 59, 2000, pp. 343–354.

[12] Coulette, B.—Thu, T. D.—Cregut, X.—Thuy, D. T. B.: Rhodes, a Process Component Centered Software Engineering Environment. In International Conference on Entreprise Information Systems 2000, pp. 253–260.

[13] Cregut, X.—Coulette, B.: PBOOL: An Object-Oriented Language for Definition and Reuse of Enactable Processes. J. Software – Concepts and Tools, Vol. 18, 1997, No. 2, pp. 47–62.

[14] Dai, F.—Li, T.—Zhao, N.—Yu, Y.—Huang, B.: Evolution Process Component Composition Based on Process Architecture. International Symposium on Intelligent Information Technology Application Workshops 2008, pp. 1097–1100.

[15] Dami, S.—Estublier, J.—Amiour, M.: APEL: A Graphical Yet Executable Formalism for Process Modeling. J. Automated Software Engineering, Vol. 5, 1997, pp. 61–96.

[16] EPF Composer: Eclipse Process Framework Composer. `http://www.eclipse.org/epf/downloads/tool/tool_downloads.php`

[17] Fielding, R. T.: Architectural Styles and the Design of Network-based Software Architectures. Ph. D. on Information and Computer Science, University of California, Irvine, USA 2000.

[18] Garlan D.—Wang, Z.: ACME-Based Software Architecture Interchange. Third International Conference on Coordination Languages and Models 1999, pp. 340–354.

[19] He, J.—Yan, H.—Liu, C.—Jin, M.: A Framework of Ontology Supported Knowledge Representation in Software Process. 2007, `http://www.atlantispress.com/php/download_paper.php?id=1180`.

[20] Kellner, M. I.—Feiler, P. H.—Finkelstein, A.—Katayama, T.—Osterweil, L. J.—Penedo, M. H.—Rombach, D. H: ISPW-6 Software Process Example. 5th international software process workshop on Experience with software process models (ISPW) 1990.

[21] Li, J.—Li, J.—Li, H.: Research on Software Process Improvement Model Based on CMM. 2008, `http://www.waset.org/journals/waset/v39/v39-70.pdf`.

[22] Liao, L.—Yuzhong, Q.—Leung, H. K. N.: Software Process Ontology and Its Application. 4th International Semantic Web Conference Galway 2005.

[23] Medvidovic, N.—Grunbacher, P.—Egyed, A.—Boehm, B. W.: Bridging Models Across the Software Lifecycle. J. Syst. Softw., Vol. 68, 2003, pp. 199–215.

[24] Medvidovic, N.—Rosenblum, D. S.—Redmiles, D. F.—Robbins, J. E.: Modeling Software Architectures in the Unified Modeling Language. J. ACM Transaction on Software Engineering and Methodology, Vol.11, 2002, No. 1, pp. 2–57.

[25] Medvidovic, N.—Taylor, R. N.: A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Trans. Softw. Eng., Vol. 26, 2000, pp. 70–93.

[26] Object Management Group (OMG): Software Systems Process Engineering Meta Model (SPEM), v2.0, 2008, `http://www.omg.org/cgi-bin/docFormal/2008-04-01`.

[27] Rilling, J.—Zhang, Y.—Meng, W. J.—Witte, R.—Haarslev, V.—Charland, P.: A Unified Ontology-Based Process Model for Software Maintenance and Comprehension. In Models in Software Engineering: Workshops at MoDELS, Vol. 4364, 2007, pp. 56–65.

[28] Shen, B.—Chen, C.: The Design of a Flexible Software Process Language. In SPW/ProSim 2006, pp. 186–194.

[29] Sommerville, I.—Rodden, T.: Human, Social and Organizational Influences on the Software Process. Trends in Software: Software Process 1996, pp. 89–100.

[30] Tomohiko, K. M.—Mori, K.—Shiozawa, T.: Process-Centered Software Engineering Environment Using Process and Object Ontologies. Second Joint Conference on KnowledgeBased Software Engineering 1996, pp. 226–229.

Fadila Aoussat is an Assistant at Saad Dahlab Blida University, Algeria. She received her Doctor degree from USTHB University (in collaboration with Nantes University, France) in 2013. She is an active member on the software engineering team of LSI-USTHB laboratory. Her research interests include software process modeling and software architectures.

Mourad Oussalah is a Full Professor of computer science at the University of Nantes and the chief of the software architecture modelling team. His research concerns software architecture, object architecture and their evolution. He worked on several European project (Esprit, IST, ...). He has been acting as the leader of national projects (France Telecom, Bouygues telecom , Aker-Yard-STX, ...).

Mohamed Ahmed Nacer is a Full Professor at USTHB (Algier's University). He received his PhD degree in computer science from the High School – Polytechnic National Institute (INPG) of Grenoble (France) in 1994. He is an expert at the UNDP in the enhancement of quality assurance and institutional planning. He is the Director of the Computer Engineering Laboratory at USTHB and is in charge of the software engineering team. His current research interests include process modelling, software architecture based components, web services and database systems.