A DOMAIN ONTLOGY FOR SOFTWARE PROCESS REUSING

Fadila Aoussat, Mourad Chabane Oussalah, Mohamed Ahmed-Nacer

A DOMAIN ONTLOGY FOR SOFTWARE PROCESS 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 Laboratoy, 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. Reuse the best practices and know-how capitalized from existing Software Process Models is a promising approach to model high quality Software Processes.

This paper presents a part of an approach for software processes reuse based on software architectures. This contribution is based on exploiting Software Process know-how and the solution proposed after the study of existing work on software process reuse field, our study focuses on approaches for reusing based on software architectures and based on domain ontology.

AoSP (Architecture oriented Software Process) approach, deals with the engineering "for" and "by" reusing Software Processes, it exploits the progress of two

research fields that promote reusing for the Software process reusing: domain ontologies and software architectures. AoSP exploits an domain ontology to retrieve, describe and deploy 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 is done.

Keywords: Software process (SP) reuse; SP architecture; SP structural styles; SP execution style; data flow connectors, control flow connectors; domain ontology, System and Software Process Engineering Metamodel (SPEM), ATL transformations, SP knowledge acquisition.

1 INTRODUCTION

The quality of the software product depends on the quality of the software process models that is used for the development and the maintenance of this software product [20].

Software Process (SP) models are complex structures used to define the steps performed during the 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 developed to cover the different development aspects. However, there are always difficulties to model SPs that deal with the software development preoccupations such as flexibility, dynamicity and executability [8].

Reuse SPs, is one of the promised approaches used to improve SPs. The objective is to exploit best practices and know-how capitalized from the precedent SP modeling and execution experiments. However, the diversity and the wide range of SP models in addition to the SP rigidity make 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 method 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 insufficienciesod the existing approaches for reusing SP and suggests a pertinent solution to reuse SP models. In order to cover engineering "by" reusing SPs, we focus our researches on the SP reuse approaches based on software architectures. We believe that the reusability, flexibility and abstraction of software architecture combined with the software architecture deployment techniques are relevant characteristics that can be used to provide a pertinent reusing approach to model high quality SP models. 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 Stakeholder

by capitalizing the best practices and know-how from heterogeneous SP models. We think that using a domain ontology can manage not only the heterogeneity of the used concepts, but also the heterogeneity of the used terminology.

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

To present the adopted solution to generate this domain ontology, our article is organized as follows: Section-2- presents AoSP approach and adopted the 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 SP ontology is designed and generated. To describe and deploy SP architectures, our ontology must capitalize different kinds of knowledge, thus, section-5- details how heterogeneous SP knowledge are capitalized. Section-6- presents a first evaluation of our contribution and details remaining work. Section -7- concludes the article and announces the future work.

2 AOSP (ARCHITECTURE ORIENTED SOFTWARE PROCESS APPROACH)

AoSP exploits the progress of two research fields for the SPs reusing: Ontology and software architectures. The proposed approach is constituted of two steps [4]:

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

The main objective of AoSP is:

- 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 handling SP models complexity.
- Increase the SP quality: by giving the essential characteristics for the SP model, such as comprehension, modeling and analyzing facilities, agility and execution control. These characteristics are often difficult to obtain as SPs are described as complex and depend on the comprehension of the used Process Modeling Language (PML) and terminology.

An Architecture oriented Software Process (AoSP) approach suggests a new vision to model SPs; describing SP architectures offers the possibility to separate the SP preoccupations: 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 configuration deployment mechanism reduce the SP models dependency on their environment and PMLs. Our objective is not to propose a new PML but to reuse existing tools and PMLs. According to software architectures specificities, AoSP suggests a particular SP modeling approach: SP modeling is decomposed of two steps:

- Pre modeling: Model the different SP preoccupations separately (structure, interaction and treatment). This step increases SP model comprehension and has a direct impact on SP modeling, analyzing and execution control facility.
- Final modeling: Deploy the SP architecture that can be done with different PMLs specific to different SPs kinds. The deployment must be in an automatic way by developing deployment programs. This possibility gives to our approach a generic aspect and increases the modeling facility.

3 SOFTWARE PROCESS ARCHITECTURE DESCRIPTION

3.1 Insufficiencies of the Reusing Approaches Based on Software Architectures

To describe SP architectures, we study existing approaches that exploits architectural elements to model SPs. Our SP architectural concepts identification is based on ADL (Architecture Description Language) approach [23][2], as the ADLs have a more pertinent semantics than the traditional architectures modeling approaches, the ADLs introduce explicitly architectural concepts, techniques and tools that allow describing software architectures rigorously. Our interest is to use existing software architectures tools to describe our SP architectures.

We study the next architectural elements commonly accepted by the software architecture community: component, connector, configuration, interface component, interface connector and architectural style.

In most reusing approaches based on software architectures [6] [7] [11] [13] 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 [24]. Configuration is in general implicit and not exploited formally, particularly, in the approaches based on components. Formal rules that describe the SP component assembling are not defined explicitly.

For the connector concept there is no consensus on its interpretation, the idea that emerges is that the connector is a dependency between activities, it can be a precedence link or a delegation link, and often depends on the used PML.

Approach	category	the used architectural element
PYNODE [6]	Component oriented.	Component, interface component, implicit connector.
APEL [14]	Component oriented.	Component, interface component (signature), implicit connectors.
RHODES [11]	Component oriented.	component, interface component, composite component, implicit connectors (function call).
SPEM [24]	component oriented.	Component (Activity), interface component (Work Product Ports), implicit connectors (Work Product Port Connectors.
Connectors for bridging SP models [22]	connector oriented	Predefined connectors, implicit component (SP models).
Supporting intensive SPs [1]	connector oriented	component, explicit connector, explicit configuration,
Acquisition process architectures [10]	Architecture oriented	Component (software process model), interface component, explicit connector (communication module), implicit configuration.
App. based on evolution process components [13]	Architecture oriented.	, component, predefined connectors (concurrent, selection, sequence), configuration.

Table 1. Architectural elements of the studied approaches based on Software Architecture.

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

- Limited reuse: The reusable elements such as SP Component, SP connector are defined to the 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 are not proposed.
- No general solution: Every approach deals with a particular problem and uses a particular PML (simulation [10], evolution [13], distribution [14], interaction [1]), there is no generic solution that can be applied for a large range of SPs.
- No SP architecture deployment: No deployment mechanism is proposed, even if there is some assistance; 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 objective 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 [**?**] and specific executions.

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 the SP 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 (Work Unit) that "facilitate and control" data and control transmissions between SP activities. SP Connectors do not create new products, but adapt, evaluate and control existing products.

The distinction between "creation" activities and "adaptation and control" Activities is the basis of the SP architectural concepts interpretation, thus, we define our SP architectural elements as follows (figure -1-):

- 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. The SP Port corresponds to the flow (Data or control) of the SP Component execution. Two kinds of SP ports are defined: Data Flow Port and Control Flow Port.
- SP Connector: The SP connector is an Activity that adapts or controls the SP transmissions and execution. It is independent from the SP Method Content but depends on the execution and the structure of the SP.
- SP Connector Role: The SP Connector interface is set of SP Connector Roles. It represents the flow (Data Flow or Control Flow) of the SP Connector. Two kinds of SP Connector Role 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 the capture of recurrent structures, but also the capture of recurrent execution strategies.

Figure-1- depicts the software architecture description according to the adopted semantics.

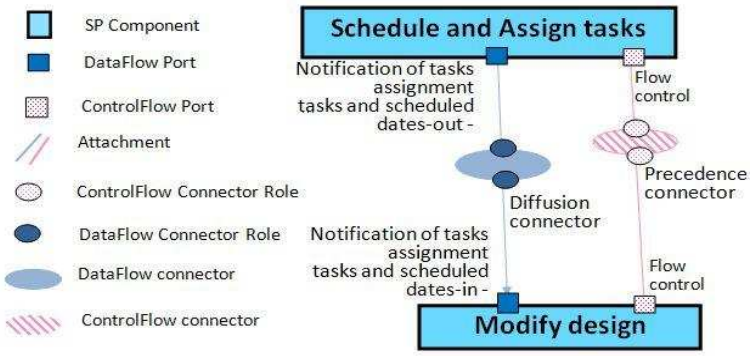

Fig. 1. Partial ISPW-6 example [19] described as ISPW-6 architecture according to AoSP approach.

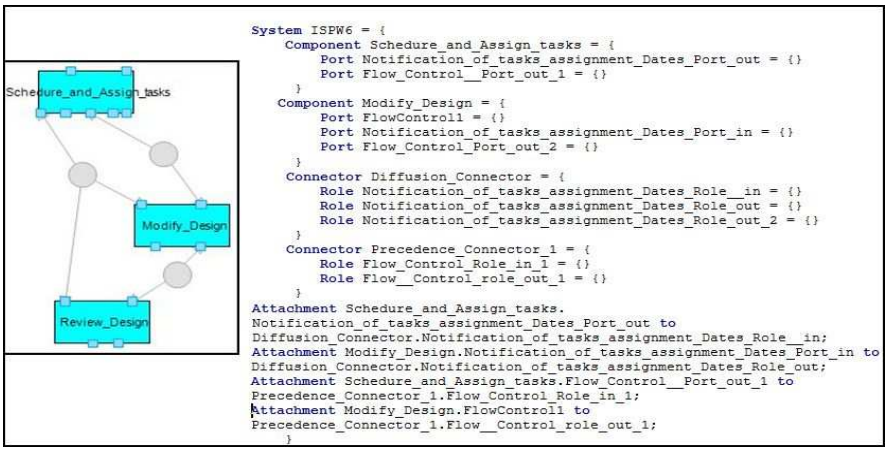

Fig. 2. Partial ISPW-6 example [19] described with ACME studio according to AoSP approach.

As the adopted semantics is based on ADL Reasoning, we can use existing ADL tools to describe SP architectures. Figure -2- depicts the partial view of ISPW6 example [19] described with ACME ADL [17]. As there are no ADLs specific to SPs we use ACME ADL. This choice is justifies by:

- ACME ADL is a generic ADL not specific to a particular domain.
- It allows the description of explicit connectors.
- It allows the description of architectural styles.

3.3 Explicit Connectors for SP architectures

By analyzing SP models behavior, we notice that some adaptation activities are recurrent. Work product adaptation activities like Work Products fusion or Work Products Fragmentation [14] are independent from SP model kind; these activities can be identified and reused. Thus, the Data Flow Connector manages the data transmission 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).

Introducing Control Flow Connectors formalizes explicitly the SP model execution policy; it allows controlling execution deviations by evaluating the SP execution flow. As figure -3- there are three kinds of Control Flow connectors:

- Execution order Connectors: These connectors explicit the execution order defined in the traditional SP models (Begin to Begin, Finish to Finish, Begin to finish and Finish to Begin). They assure the standard execution order without evaluate the SP execution. In our work, we use the Precedence connector (Finish to Begin) as basis connector to illustrate the execution order.
- Evaluation Connectors: These connectors assure the execution order and evaluate the execution quality by focusing on time, cost or quality aspects. They evaluate the required parameters and post the results. They are used in the dynamic execution human oriented (where the human decide the modification to make).
- Decision Connectors: These SP connectors are the same as the Execution Evaluation Connectors but emit decision and make modifications after a human validation. These SP connectors are used to define dynamic execution process oriented.

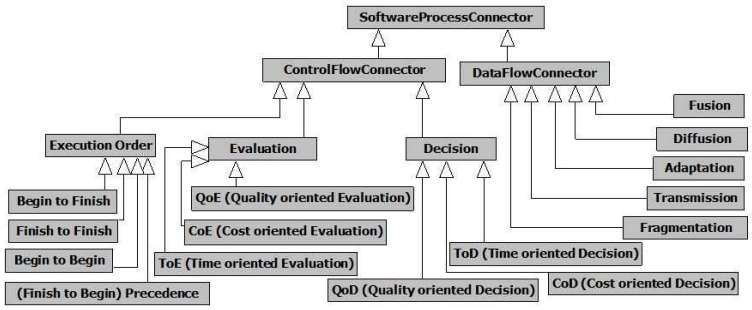

Fig. 3. Software process connector's taxonomy.

The SP connector can be a manual adaptation activity, in fact, some activities as fusion or fragmentation cannot be done automatically. The distinction between automatic and manual activities is important; we can formalize the human interaction responsibilities, and anticipate the execution deviations that can be caused by human errors.

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 define a SP architecture we can exploit these recurrent structures such as lifecycles and process patterns to define SP styles.

However, the SP has a characteristic that the software product has not: an SP model can be executed in different ways according to the development conditions. Indeed, during the SP model execution, the project manager may give priority to the development time, development cost or to the product quality.

Development priorities in addition to the unexpected events directly affect the SP model execution policy. Consequently, one SP Model can have multiple executions instances without being able to differentiate well from bad execution. Without a clear vision of the desired execution policy, it is difficult to control SP model execution and make the right adaptation decisions. The problem is that the execution policy is not explicitly described in the SP; execution policies are nor capitalized or reused.

In our work an SP architectural style must allow not only the capture of recurrent structures, but also the capture of recurrent execution policies to capitalize this expertise and give a tool for better execution control.

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 [16]. In our work, this definition remains valid, SP Component Types, SP Connector Types and invariants that are used to describe an architectural style are also used to describe a 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. Thus, a structural style can has different execution styles. We details the structural and the execution styles as follows:

- Structural styles describe recurrent SP structures. They focus on the Work Products treatments and transmissions that are described independently from the execution policy. A standard execution is assured by using precedence connector. We inspire from the existing life cycles and process patterns to define our structural styles.
- Execution styles describe recurrent SP execution policies. Every execution style is defined by the Control Flow connectors independently from the component

types. They formalize the adopted policy to evaluate and adjust the SP execution. An execution style are used combined with a structural style.

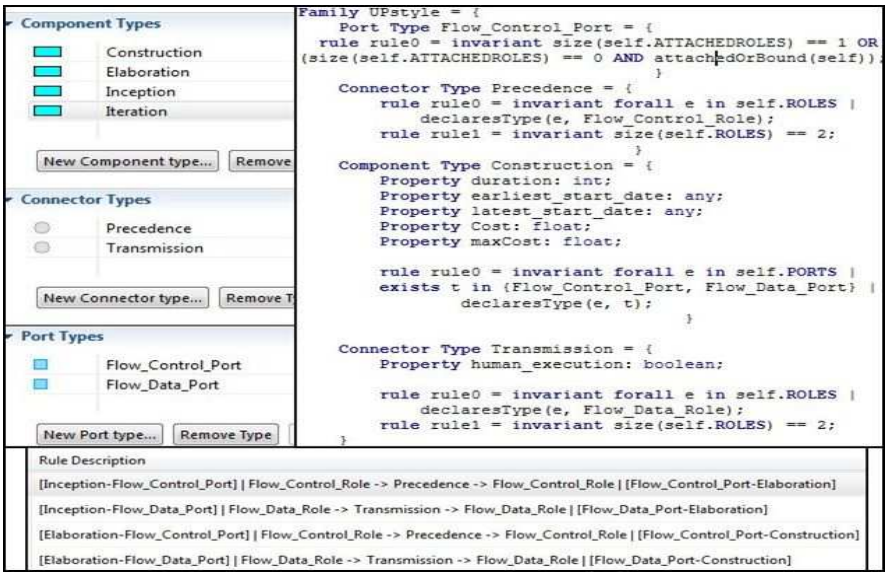

Fig. 4. Description of SP architectural style with ACME studio.

SP style can be described with ACME studio, to define SP structural style we follow the same steps as for defining software architecture styles, we define the types of the style elements, we also define properties, rules that govern this style.

Figure-4- depicts a partial view of the structural style UP described with ACME studio, we depicts the components types and the connectors types of the UP style, assembling rules and constraints are also described.

4 SOFTWARE PROCESS DOMAIN ONTOLOGY GENERATION

To capitalize the SP knowledge we use a domain ontology, our aim is to offer a repository to share the pertinent SP knowledge extracted from precedent successful SP models. In addition, our aim is to offer a tool to allow the reasoning and the emergence of new solutions. Our 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: Capitalize knowledge from various SP models can create ambiguities, indeed, even if there is consensus on the used terminology in the software development [15], 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.

4.1 Insufficiencies of the Reusing Approaches Based on Domain Ontology

To suggest a domain ontology, one of the first steps is to study the existing ones and consider there extension, fusion or adaptation. Many SP modeling approach based on domain ontology are defined [18][21][25][**?**]. These approaches use one or many ontologies to represent the SP model knowledge. However, these solutions are specific and deal with particular SP models. They do not suggest a general solution that can applied for a large range of SPs.

Appoach	**objective**	**Ontology structure**
OnSSPKR Framework [18]	Deal with CMM, CMMI, ISO/IEC15504, ISO9001 models.	Three ontologies (Process experiences, Personal skills, Knowledge artifacts)
SPO (Software Process Ontologie)[21]	Mapping between CMMI models and the ISO/IEC 15504 models	SP basic concepts
PCE based ontology [28]	Generate SP plans	Two ontologies (artifacts and activities)
Approach based descriptive logic [25]	Framework for software maintenance	Concepts that affect the software maintenance
Flexible PML based ontology [26]	Flexible SP models	Basic Process elements

Table 2. Approaches for reusing SPs based on domain ontology.

Table- 2- resumes the objectives and the structures of the used ontologies of the studied approaches. The domain ontologies defined by these approaches do not respond to our expectations, they do not deal with the concepts and terminology heterogeneity. Thus we can not exploit these ontologies.

4.2 Software Process Ontology Conceptualization

To suggest a SP domain ontology that deals with our preoccupations, we exploit SPEM (Systems and Software Process engineering Metamodel) conceptualization. SPEM is a standard metamodel adopted by the OMG to describe the concepts of a large range of software processes, 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 the tools and techniques offered 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 the reusing based on SP architectures that justify these lacks. Consequently, SPEM must be extended with the required SP elements.

4.3 SPEM profile extension

Having a complete semantic to describe a SP architecture, the extension of SPEM profile can be done, for this purpose, we introduce new stereotypes to describe the architectural elements of the SP architectures:

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". These stereotype describes the common behavior of SP style stereotypes and SP configuration stereotypes separatly.

We distinguish these two stereotypes as the "SP Style" is a "Method Content Package" and its elements are only a "Method Content Elements", however, The "SP Configuration" is a "Process Package" and its elements are only a "Process Elements" (Figure -5-).

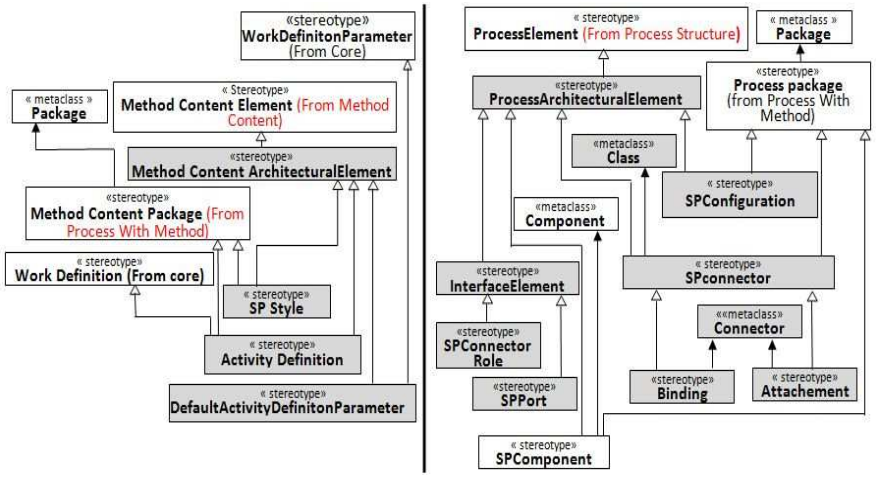

Fig. 5. Extention of Method Plugin profile with SP architectural Elements [5].

Figure-6- depicts classes and associations added to extend SPEM model. Two kinds of classes are added:

- **Classes that describes the SP architecture:** A SP Configuration is composed from SP Components and SP Connectors. The SP Component is an activity (Work Unit) that creates Work Products, its interfaces are a set of SP Ports. The SP connector is an activity that adapts and controls the SP Flow, its interfaces are SP connector roles. The SP components assembling are done via many attachments. An attachment is done between a SP Port and a SP connector that have the same kind (Data Flow or Control Flow).
- **Classes that describes the SP style:** the SP style is composed from Activity Definitions. Activity Definition class identifies the SP connector and the SP component types at the same time and Work Product Definition describes the SP ports and the SP Connector role types. The default assembling is described with Default Activity Definition Parameter.

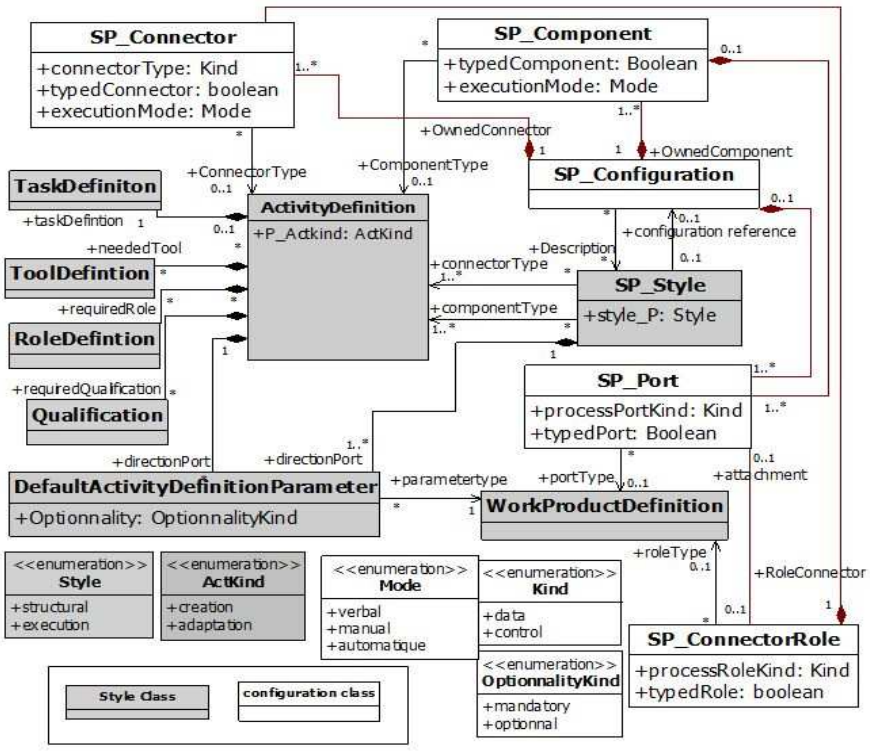

Fig. 6. SPEM model extention with SP architectural Elements.

4.4 SPEMOntology Generation

SPEM being extended, we can generate our ontology that allows describing and deploying SP architectures. Our ontology OWL is generated by applying many model transformations. To achieve this aim, we use ATL (Atlantique Transformation Language) modules UML2OWL and UML2COPY modules [3].

UML2OWL module is developed to transform an UML model to an OWL model. As SPEM is an UML profile, we cannot directly use this module; in fact, in UML2OWL ATL module there are no ATL rules that allow the transformation of profiles, stereotypes and tagged values to OWL elements.

So, we develop an ATL module "applySPEMprofile2SPEMmodel" [4] (figure-7-) that apply the SPEM profile to the SPEM model. The not stereotyped elements are copied directly with UML2COPY ATL module. The result is used to execute UML2OWL module and generate our ontology "SPEMOntology".

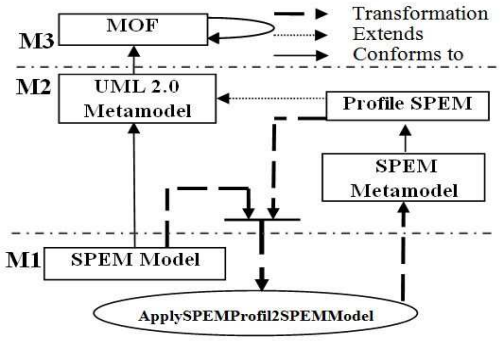

Fig. 7. "applySPEMprofile2SPEMmodel" ATL transformation.

Figure-7- depicts the "applySPEMprofile2SPEMmodel" transformation, the source models are SPEM model (confoms to UML metamodel) and SPEM profile (an UML profile), the results is a stereotyped SPEM model that constitutes the SPEM meta-model.

4.5 SPEMOntology Structure

SPEMOntology is the result of successive ATL transformations. It is constitute 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 [24]. 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 packages classes.

Figure -8- depicts the main abstract classes of every SPEM package. We notice the existence of the concepts introduced to describe SP architectures. In our work we had not exploited Process Behavior package, thus it is not represented in figure-8-.

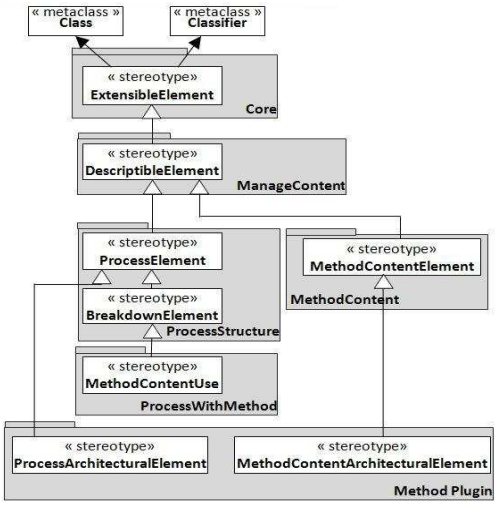

Fig. 8. Main abstract SPEM classes that describe the SPEM organisation (after the SPEM profile extension).

After the ATL transformations we can identify this organization (figure -9-). The SPEM packages view can be identified through the main concepts of SPEMOntology. To facilitate the ontology understanding, we have kept the same name as the SPEM classes; however, we have added the prefix "pro" to identify the stereotyped elements. This prefix is added during the execution of "applySPEMprofile2SPEMmodel" module.

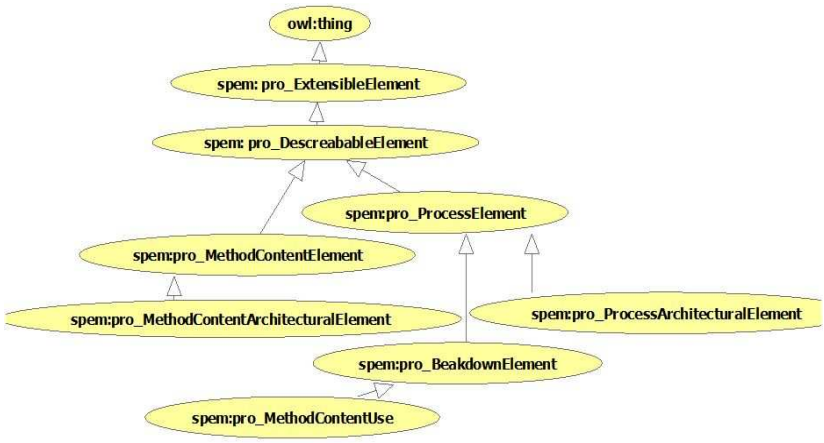

Fig. 9. Main abstract SPEMOntology concepts.

5 SOFTWARE PROCESS KNOWLEDGE AQUISITION

The concepts heterogeneity finds solution by exploiting a standard metamodel. The heterogeneity at instance level deals with 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 configuration and SP styles.
- **The Used Knowledge:** our ontology must allow to reuse exiting SP models. The capitalized knowledge concerns the know-how of existing SP models.
- **The Reference Vocabulary:** Our ontology must retrieve a comprehensible 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. Every SPEM package is used to store a kind of knowledge. In the next sections we detail how we capitalize every kind of SP knowledge.

5.1 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 strategies and practices. The knowledge can be recurrent SP configurations, well-

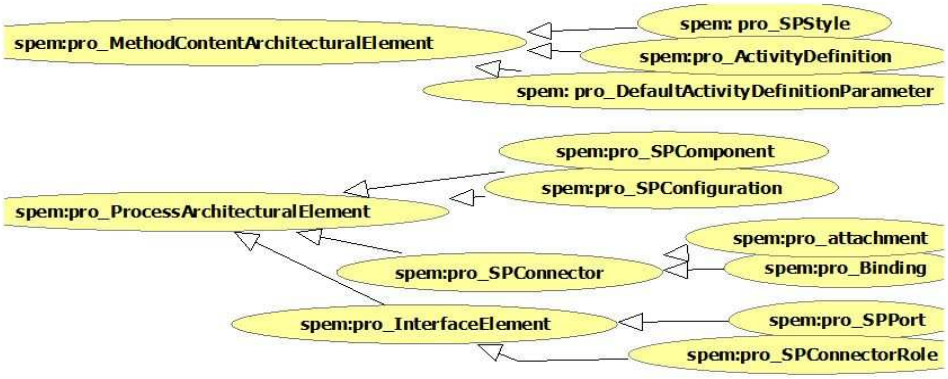

Fig. 10. SP architectural concepts of SPEMOntology.

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 concepts (figure-10-). These concepts are the added concepts to describe the SP architecture elements. 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.

5.2 The Used Knowledge Aquisition

We instantiate the concepts of "Process with Method" and "Process Structure" packages. In SPEM these packages are used to describe -respectively- the effective use of the content methods and the SP fragments independently from particular development method [24]. In our ontology, these concepts are used to capitalize the used knowhow that are collected from the existing SP models (Figure - 11-).

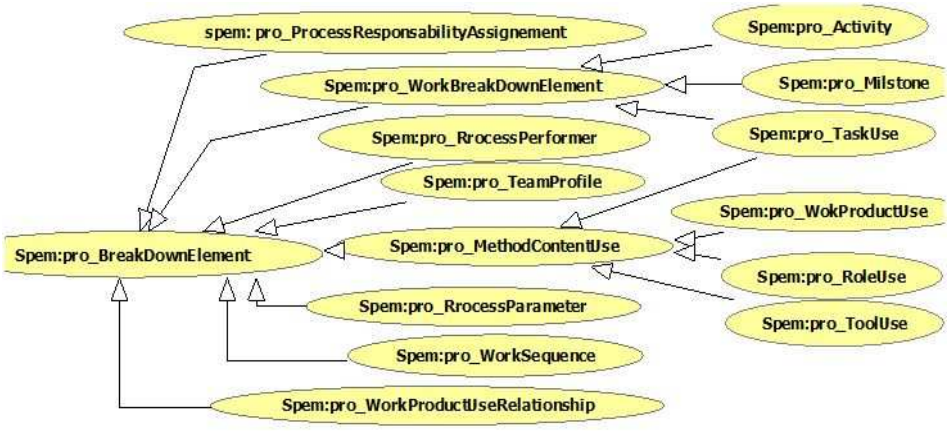

Fig. 11. The Used knowledge concepts of SPEMOntology.

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[15] and another for PBOOL+ models [12]. The knowledge aquisition from EPF models is direct as these models are conform to SPEM, however, for PBOOL+ models these is no "Task Use" concept, then, we suppose that every elementary activity of PBOOL+ model is constituted from one "Task Use" and do the instantiation.

5.3 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-).

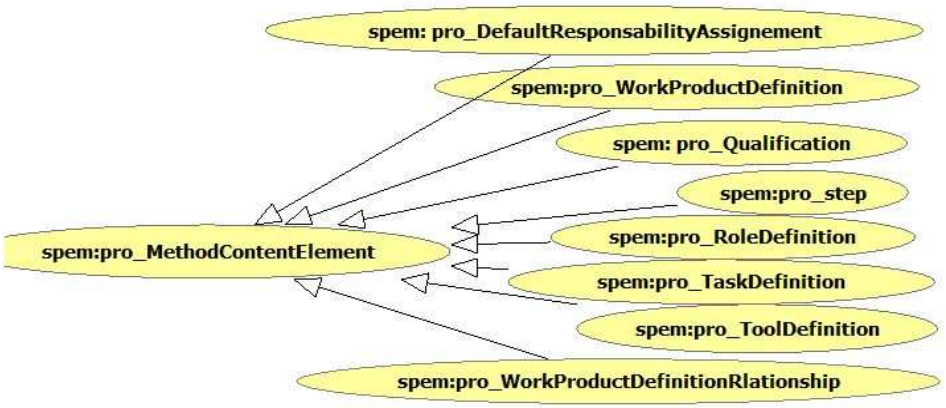

Fig. 12. The Vocabulary Reference concepts of SPEMOntology

The Method content concepts are solicited to describe many kinds of knowledge: Method Content Elements, Vocabulary Reference and Architectural Types. To distinguish between these kinds of knowledge, for each Method Content 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 allow to define formally the glossary of the company. It allows not only a better comprehension of the SP models, but also, constitutes a contribution to capitalize company know-how, that will be used and reused formally independently from the SP experts and its tacit knowledge.

5.4 The Instance Heterogeneity Management

To manage the instances heterogeneity, we must do a correspondence between the Used Knowledge and the Reference Vocabulary. This correspondence is done by using existing associations between Method Content Concepts and Process With Method concepts (figure -13-).

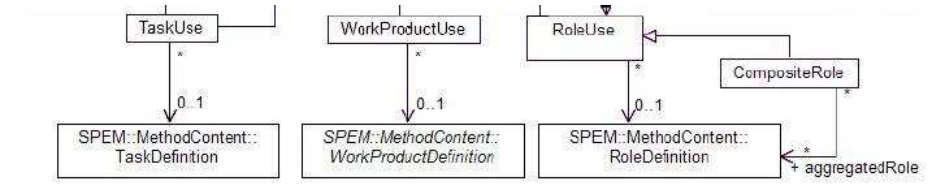

Fig. 13. SPEMOntology concepts.

Moreover, we define new associations to do the correspondence between the SP Architectural Concepts and the Vocabulary Reference concepts. The added associations are specific to our ontology and are not integrated on SPEM as we did with the SP Architectural concepts. Thus we add:

- An association between "SP Port" and "Work Product Definition".
- An association between "SP connector role" Work Product Definition".

6 AOSP APPROACH EVALUATION

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	optimize the instantiation program to integrate new SP models that are described with other PMLs.
Increase the reuse of existing SP models	developed a domain ontlogy "SPEMOntology" that capitalizes SP knowledge from most of 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 us: dynamic, distributed and using different PMLs.
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.

In this paper we presents only the engineering "by" reusing SPs of AoSP ap-

proach, the engineering "for" reusing SPs is not presented, thus, SP architecture retrieving, SP architecture inference and SP architecture deployment are not detailed. However, we can do a first evaluation of AoSP.

Both resuing 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 generale and can be used for many kinds of SPs. Also, unlike existing approaches, AoSP exploits all the opportunities offered by the software architecture domain and domain ontologies to model and execute SPs.

According to table -3- we had achieved globally 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. Describe SP architectures give the SP model more flexibility, comprehension and dynamicity.

The engineering "by" reusing SPs is not presented and can give more advantages to our approaches. The SP architectures knowledge retrieving, the SP architecture deployment and SP knowledge inference increase, also, the SP reusing.

7 CONCLUSION

AoSP is a SP reusing approach that offers a standard solution to increase the reuse and the reusability of SP models in order to model high quality SPs.

This paper presents the engineering "for" reusing SPs of AoSP (Architecture oriented software Process) approach. AoSP exploit 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. TIt 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 architectures 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 exploits the precedent good modeling and enactment experiments to model high quality SP models. 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 (Atlantique Transformation Language) modules

(UML2OWL and UML2COPY) and we develop an ATL module "applySPEMprofile2SPEMmodel" to apply SPEM profile to SPEM model.

To describe and deploy SP architectures, SPEMOntology must store different kinds of knowledge: The used know-how, the SP Architecture Knowledge and a Reference Vocabulary, in addition, it must do a correspondence between these kinds of knowledge. We exploit the SPEM structure (organized into packages) to store separately these kinds of knowledge. We add adequate properties to have to keep the knowledge coherence.

Actually we are working on the engineering "by" reusing; we are working on defining inference rules to infer tow kinds of knowledge: "equivalent SP configuration" to identify the SP configurations that can replace the required configuration and "equivalent SP components" to identify the components that can replace the required SP component. Also, first results of retrieving and deploying SP architectures are obtained, but must be refined before their 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, pages 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, N 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. and 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, pages 102-108.

[7] Belkhatir, N. Estublier, J.: Supporting reuse and configuration for large scale software process models. 10th International Software Process Workshop, 1996, page 35-40.

[8] 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.

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

[10] Choi J. , Scacchi W.: Modeling and Simulating Software Acquisition Process Architectures, J. of Systems and Software, vol 59, 2000, pp 343-354.

[11] Coulette, B. Thu, T. D. Crgut, X. Thuy, D. T. B.: Rhodes, a process component centered software engineering environment. In International Conference on Entreprise Information System, 2000, pages 253-260.

[12] 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.

[13] 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.

[14] 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

[15] EPF Composer: Eclipse Process Framework Composer. http://www.eclipse.org/epf/downloads/tool/tool$_d$*ownloads.php*

[16] 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.

[17] Garlan D., Wang Z.: Acme-Based Software Architecture Interchange. Third International Conference on Coordination Languages and Models, 1999, 340-354.

[18] 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.

[19] Kellner, M. I. Feiler, P. H. Finkelstein, A. Katayama, T. Osterweil, L. J. Penedo, M. H. Dieter Rombach, H. : ISPW-6 Software Process Example. ISPW '90, 5th international software process workshop on Experience with software process models, 1990.

[20] 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.

[21] Liao, L. Yuzhong Q. Leung H. K. N.: Software process ontology and its application. 4th International Semantic Web Conference Galway, 2005.

[22] Medvidovic, N., Grunbacher, P., Egyed, A., Boehm, B. W.: Bridging models across the software lifecycle. J. Syst. Softw., vol 68, 2003, pages 199-215.

[23] 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, N1, pp 2-57.

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

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

[26] Shen, B. Chen, C.: The design of a flexible software process language, In SPW/ProSim, 2006, pages 186-194.

[27] Sommerville, I. Rodden, T.: Human, Social and Organizational Influences on the Software Process, rends in Software: Software Process, 1996, 89-100.

[28] 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,pages 226-229.