Journal of Computer Science 2 (4): 333-336, 2006
ISSN 1549-3636


# Software Process Modeling Using Role and Coordination

[1]Atil Fadila, [2]Ghoul Said and [1]Bounour Nora
[1]LRI Laboratory, University of Badji Mokhtar, BP 12, Annaba, Algeria
[2]Institute of Computer Science, University of Philadelphia, Jordan

**Abstract:** The term software process joins all activities that have to be achieved in order to develop software. It has been shown that modeling such processes is difficult and expensive task. It's confirmed by diversity of software processes modeling approaches which are however, not satisfactory. This study deals with an area of growing importance and presents a role- and coordination- based approach to specify and model methodological aspects of this processes, by formally defining the policy that lead the process, such as rules which determine activities and their organization and the component mechanisms, such as tools that realize activities and operate on objects according to policy. The purpose of role modeling is to achieve separation of concerns, allowing the designer to consider different aspects, or the same aspect at different levels of detail. The originality of our approach is to consider a process as a coordination of a set of sub-processes. This have include profits; among which, the modular distribution of methodologies upon implicated sub-processes, the construction and the realization of component methodologies and the association of version of behaviors to the same process.

**Key words:** Activity, methodology and role modeling

## INTRODUCTION

As part of software engineering, a large community defines a software process as a partially ordered set of activities accomplished during development or evolution of software[1]. This definition implies that each set of software life cycle activities (and not necessarily every activities) forms a software process. For against, number of researchers defines a software process as the total set of software engineering activities. This definition is a particular case of the first because it is interested with a particular software process (covering all the set of life cycle activities).

Modeling, evaluation, improvement, formalizing description and progress of a software process have made the object of several research projects. Problem of this domain is that it calls at many technologies (knowledge representation, data bases, artificial intelligence, simulation…) and methodologies (sequential, cascade, expert system, prototyping…) of which the majority has not reach the stage of maturity and stability. This makes the software production a process with difficult approach to understand and with concepts difficult to unify. It also confirms the diversity of processes modeling approaches which are however not satisfactory[1-3].

In this study, we propose modeling software processes by a simple and natural way, using object approach. The problem that we have confronted is due to the fact that this approach doesn't allow the modeling of all dynamics and constant change of reality. To solve this problem, we attempt to use the role and coordination concepts to express how the object changes and to allow the definition of one or more methodologies controlling the behavior of the software process.

This approach presents many conceptual advantages with regard to actual works in the domain. In fact, a software process is regarded as a set of sub-processes, which cooperates for realizing the same objective. This vision is natural and present contribution concerning construction and reuse of software process's methodologies. In this approach, it's even possible to associate versions of roles to the same process.

**The software process:** The term software process joins all activities that have to be achieved in order to develop software. The methods for implementation of activities depend on the type and content of development projects and technology used. For the same type of projects, the same sequence of activities and the same methods for their implementations are used[4].

We can also define the software process as a sequence of operations required for building up various information objects (specifications, prototype documentation, test cases, code…) that compose a software product[5]. The software process can be split into sub-processes, but it is often very hard to find a good decomposition and to describe the complex way

**Corresponding Author:** Fadila Atil, LRI Laboratory, University of Badji Mokhtar, BP 12, Annaba, Ageria
Tel: +213 71 15 90 06 Fax : +213 38 51 56 22

in which they must communicate. Processes are dynamic, hard to comprehend and to reason about.

**Software process model:** A process model is the formal expression of a part of the process, with the goal to understand, communicate, improve, support or automate the process[6]. Process technology supports a process in order to consistently reach the goal within predefined time, budget and quality constraints.

A software process model (SPM) is a descriptive representation of the software process structure, used as a reasoning support, allowing its understanding and its progress[1].

Analysis of any process get appear two levels: structural level which represent objects on which process's activities perform and methodological level describing the policies which lead the process and its component methods.

SPM = ({Methodologies: Policies, Mechanisms},
{Structures})

**Role definition:** The role is a popular and powerful OO modeling concept. It is adopted as a means of associating human and other resources with tasks and processes.

It can be defined as an individual, group, Department, ad hoc team or system which has responsibility for some contribution to a process. This contribution is carried out through a set of partially ordered activities that share a common set of resource[7].

Examples of roles are Control System Design *Engineer*, *Safety Assessment Engineer*, *Chief Designer*, etc.

The mission of role modeling is to reduce complexity when doing "large-scale" design; i.e. complexity due to the size of the design task. This is done by supporting separation of concerns and reusable design[8].

**Limitations of object oriented approach:** The object oriented modeling present many advantages; nevertheless, many deficiencies could be taken up[9]. To model a software process according to the object approach, the principle is based one's argument on invocation of methods by sending messages to objects of a class hierarchy. The series of these methods must allow the correct and not ambiguous resolution of the given problem[3,10]. Such series of methods qualified with sensible and explicit, define a coordination of these methods[10,11].

If in object oriented programming languages, the semantic analysis allows verification of method invocation's validity by an object, nothing allows the verification of methods coordination's validity of the same object or of different objects. Nothing allows then to consider an object as a process and consequently, to verify its correct exploitation (according to this process). This is due to the total absence of an explicit formulation of coordination in actual object oriented formalism, which is a serious handicap for software processes modeling. We attempt to remedy to that by the integration of methodologies in the definition of objects. We note that actually there are needs in this way as part of formal specifications.

**Coordination paradigm:** We use a coordination model permitting expression of software processes methodologies. We consider a software process as a set of agents that cooperate for realizing the same objective. This approach is based on the set of the following concepts[1-3,12]:

**Process:** is a collection of interrelated steps/activities, leading to common objective and all of the elements necessary for their execution. Software process, consequently, includes activities for the development of software.

**Activity:** corresponds to a simple or compound action, which is executed by a human being or a machine.

**Dependency:** defines a relation between two or some activities. We say that an activity A1 depend on the activity A2 if the working of A1 require this of A2. Some dependencies come under intrinsic semantic of activities. They exist independently of any context (global objective to reach). For example, any "Consumer" activity depends on a "Producer" activity: It must ever check that the "Producer" activity is accomplished before its results are required by "Consumer" activity. Some other dependencies between activities come under a global objective to reach. These dependencies must be dynamically introduced (or separated) to satisfy this goal. A same objective can be reach with different manners, according to the applicable methodology. The set of dependencies between activities is open, in view of the infinity of contexts were they evolve and the changeable goals to reach. In our study, we are interested with two types of dependencies, namely, functional dependencies and organizational dependencies.

**Functional dependencies:** regroup all data flux and control flux dependencies, well known in procedural languages. They must be verified every time and are explicitly defined by the relation *Function that has a changeable* semantic.

The Function dependency expresses that a set of target activities TA depends on an optional set of initial activities IA under the optional constraint Ctr. When all activities of IA are executed, activities of TA could be executed under the constraint Ctr.

Formally, this dependency is defined with: ″ [*IA*] [*Ctr*] $\rightarrow$ *TA* ″, were *Ctr* is defined with <*condition;*

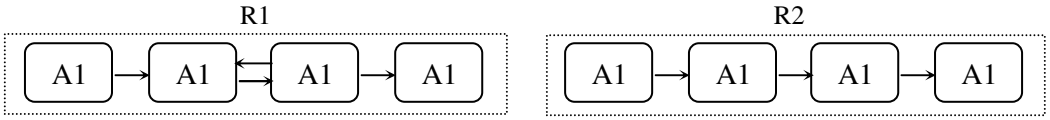

Fig. 1: Two roles R1 and R2 of P

*value; sense*>. *The Condition attribute* defines conditions that must be satisfying in order that dependency being valid. *Value attribute defines the data* flux required by this dependency. Finally, *Sense* attribute defines the semantic of dependency, which can be repetition (∗), implication (∧), exclusion (¬), equivalence (~), instantiation (∋), etc.

**Organizational dependencies**: allow an organization of activities during time (with Synchronous and Alternation dependencies) as well as their hierarchical organization (with Aggregation dependency). We note that organizational dependencies allow the modeling of behaviors of software processes.

**Synchronous:** This dependency allows ordering activities in time. It's expressed with: Syn a1, a2, ..., an Endsyn. Activities none implicated in a Syn dependency may be executed in any order.

**Alternation:** It's a dependency, which allows establishing a nil order between a set of activities. These activities are then alternated and could constitute a varying activity. By nil order, we imply that only one of concerned activities can be executed. This activity will be determined dynamically according to explicit or deduced contextual knowledge. It's defined with: Alt a1, a2, ..., an Endalt. Only one activity ai (i=1,n) must be executed and all the others will be ignored.

**Aggregation:** It allows constructing a complex activity with hierarchical composition (designed by an identifier) of different agent's activities. If the composition is designed with an identifier, this last will indicate the resulting activity. Such dependency will be expressed with: ″ {*IA1,IA2,...,IAn*} <∅;∅;U> → T*A* ″, were TA is the identifier of the resulting activity. None designed composition don't construct a complex activity.

**Role modeling:** The concept of role is intuitive and important to achieve a simple and natural modeling of process activities and to aid comprehension. It gives restricted, possibly complementary perspectives on a complex process and allow dynamic of such perspectives. A process has several roles that have been chosen in order to accomplish the objective of the modeling. The roles may change and they may exist simultaneously.

We consider real word concepts to consist of several mutually cooperating and interacting entities, not stand-alone entities existing independently of other entities in the same domain of interest. The design approach presented here is related to the ideas of considering objects as "playing" different roles in different contexts[8,12,13].

In a software process, sub-processes cooperate with each other to accomplish a global goal. So, they are related to each other in different way: Serving, using and communicating with each other. From the way in which they treat one another, processes have different perspectives of each other. These perspectives define the role that a process may play towards another. A role is formed as a set of behaviors of the process. Different roles exist for different purpose and the roles played by a process may change over time.

In our approach, process's activities can exist in many versions and can be organized during time in many different manners. Each acceptable organization of activities defines process behavior (a methodology of its working). In this way, behavior presents the associated process as a states machine[9]. The process's behavior according to a determined objective defines its role and the role is then a sensible series of activities.

In order to illustrate the use of this concept for modeling software processes, we present an example of a software process P. Behaviors present in P are defined by the set of activities {A1, A2, A3, A4}. We can assign to this process two distinct roles R1 and R2, schematically defined by the Fig. 1, allowing going back or no to A2 step from A3 step.

**Process modeling:** In our approach, a software process can be simple or complex, i.e., compound of a set of sub-processes which cooperate in order to achieve the same objective. The modeling of such process is essentially based on the definition of the set of composing sub-processes, of dependencies between its activities and of roles that it offers (Fig. 2).

*Process* <Process Name>;
*Interface* <Interface description: Identification of roles>
*Sub-Processes* <Definition of the set of composing sub-processes>
*Functional Dependencies*
  <Definition of functional dependencies>
Organizational Dependencies
  <Definition of the set of roles>
*End* <Process Name>

Fig. 2: Definition of a formal process.

A formal Process define a generic software process model, offering some alternatives, from which, we can generate specifics software processes (Real processes).

The generation is done according to an appropriate behavior and allows then the solving of a particular problem.

Owing to such model, we can define a formal process that can be independent of any problem (Fig. 3) and from which, we can generate a real process as an instance that can take part in development of specific software processes.

```
Process P;
  Interface R1, R2;
  Sub-Processes
       …

  Functional Dependencies
       …
  Organizational Dependencies
    R1 = ...; // Description of the first role
    R2 = ...; // Description of the second role
End P
```

Fig. 3: Specification of the formal process P

For example, we can generate from this formal process P two software processes, P1 and P2 respectively according to the roles R1 and R2.

Therefore, according to need, we can define or modify different methodologies (behaviors). The instance's methodology, generated from a process, imposes to this last a controlled behavior that can be automated. This vision offers a considerable benefit for software processes modeling.

We note that benefit of our approach is in the construction of methodologies of software processes that is done with a modular manner by reusing composing process's methodologies.

**Comparison with IA approach:** In the IA approach, the rule concerns an activity and its interface with others, which make the dependencies between activities implicit and informal. Against, in our approach, a rule of dependency relates two sets of activities according to a coordination constraint, that make the methodology more explicit, more formal and especially well structured (set, behavior, agent). This approach has the possibility of formal verification of methodologies and reasoning which it's the support.

In the IA approach, the inference's motor constructs the different possible alternatives when with the model proposed, the alternatives to consider may be imposed by an explicit selection mechanism (description of behavior). In this context, IA approach is purely analogue to an inference's motor with fixed strategy, when in our approach; it can correspond to a motor with a programmable strategy.

## CONCLUSION

In this study, we have presented a modeling approach of software processes based on integration of object oriented paradigm, role and coordination. The notion of coordination has allowed expressing the methodological aspect, by formally defining the policy that lead it (rules determining the activities and their organization,) and composing mechanisms (tools realizing activities and operating according to this policy). The purpose of role modeling is to achieve separation of concerns, allowing the designer to consider different aspects, or the same aspect at different levels of detail. A perspective that a process may play towards another defines a role. It is formed as a set of behaviors of the process. Different roles exist for different purpose and the roles played by a process may change over time. We have proved that the construction of complex methodologies can be done with modular manner by reusing the composing methodologies.

## REFERENCES


1. Ghoul, S., 1995. Methodological and structural aspects in software processes models", PhD dissertation, University of Annaba.
2. Atil, F., S. Ghoul, D. Meslati and N. Bounour, 2004. Modeling Software process using roles. 17th Intl. Conf. on Software & Systems Engineering and their Applications (ICSSEA) Paris.
3. Atil, F. *et al.*, 2005. Role based software process modeling. ISPS'2005, 7th Intl. Symp. on Programming and Systems, Algiers.
4. Horvat, R.V. *et al.*, 2000. SoPCoM-Model for Evaluation of the Software Processes Complexity. EuroSPI 2000, Copenhagen, Denmark, November.
5. Rueher, M. and C. Michel 1990. Using objects evolution for software processes representation. Proc. of 22nd Annual Hawaii Intl. Conf. on System. Sciences, vol. 11.
6. Estublier, J., 2005. Software are Processes Too. Software Process Workshop (SPW), Bejing.
7. Murdoch, J. and J.A. McDermid, 2000. Modelling Engineering Design Processes with Role Activity Diagrams. Society for Design and Process Science.
8. Andersen, E.P. and T. Reenskaug, 1992. System design by composing structures of interacting Objects. In Ole Lehrmann Madsen editor, Proc. of the 6th European Conference on Object-Oriented Programming.
9. McGregor, J.D. and D.M. Dyer1993. Inheritance and state machines. ACM/SIGSOFT, pp: 61- 69.
10. Blondo, L., 1998. Designing and programming with personality. Master's Thesis. University of Northeastern.
11. Malone, T.W. and K Rowston, 1994. The interdisciplinary study of coordination. ACM Computer Surveys, 26: 87-120.
12. Atil, F. and S. Ghoul, 1998. Object based software process modeling. 1st UK Colloquium on object Technology & System Re-engineering (COTSR), Demontfort University, Leicester.
13. Meslati, D. and S. Ghoul, 1997. Semantic classification: A genetic approach to classification in object-oriented models. J. Object-Oriented Programming, January.