Agent Based Software Process Modeling
Using Object Oriented Petri Nets

Sahnoun Zaidi  
LIRE Laboratory, Computer Sc. Dpt., University Mentouri of Constantine, ALGERIA.  
E-mail: sahnounz@yahoo.fr

Barkaoui Kamel  
Laboratoire CEDRIC, Equipe VESPA, CNAM, Paris, FRANCE  
E-mail : barkaoui@cnam.fr

Seghiri Ramla  
LIRE Laboratory, Computer Sc. Dpt., University Mentouri of Constantine, ALGERIA.

Abstract

The software process is a set of activities and associated results which produce a software product. Criticism of traditional life cycle models has been the subject of many studies arguing for new approaches of software process modeling.

In this paper, we show that the software development process is not a series of discrete activities. The activities are performed concurrently. For that purpose, we present:

1. A more comprehensive approach that captures the concurrency that exists among all activities of the software development process. The agent paradigm is used to represent each participant in the process and to capture the interaction between the different agents.

2. An Object Oriented Petri Net approach is used to model the agent-based software process.

1. Introduction

The software process is a set of activities and associated results which produce a software product. If the wrong process is used, this will probably reduce the quality of the software product to be developed (this is termed as the model adequacy). Criticism of traditional life cycle models has been the subject of many studies arguing for new approaches of software process modeling [4, 10, 12, 7]. In this paper, we show that the software development process is not a sequence of discrete activities. The activities are performed concurrently. For example, the spiral model describes the software development as a sequential process. It is true that there is sequential initiation of each activity, but once initiated, the activities are performed concurrently ("software development is a concurrent process") [7].

The adequacy, readability and ease of use, hierarchical decomposability, and amenability to formal analysis and reasoning are typical requirements for process models [15]. But we can hardly imagine a homogeneous process model capturing all these different aspects in a adequate way. In that respect, we present:

- A more comprehensive approach that captures the concurrency that exists among all activities of the software development process. The agent paradigm is used to represent each participant in the process and to capture the interaction between the different agents. The software process is thus considered as a multi agent system.

- An Object Oriented Petri Net approach is used to model the agent-based software process.

In the following, we will explain some concepts used in our study. The advantage of agent approach in software process model is the subject of section 3. In section 4, we distribute the process activities on different software process agents and propose the behavior scenario of each one. Section5, presents an interaction view between different agents. In section 6, we describe the proposed Object Oriented Petri net model. We terminate this study by a conclusion.

2. Basic Notions

2.1 The Software Process

The software process is the set of activities and associated results which produce a software product. These activities are mostly carried out by software engineers. There are four fundamental process activities which are common to all software processes. These activities are:

- Software specification : the functionality of the software and constraints on its operation must be defined,
- Software development : the software that meet the specification must be produced,
- Software validation: the software must be validated to ensure that it does what the customer wants, and
- Software evolution: the software must evolve to meet changing customer needs.

2.2 Petri Net

A Petri Net can be viewed as a directed bipartite graph composed of two kinds of nodes which are called S-elements and T-elements. The S-elements represent local states and the T-elements represent transitions. The arcs represent a causal flow relation between S- and T-elements, and vice versa [17].

Petri Nets are the most appropriate formalism and tool to model concurrent and dynamic activities. Among many well known advantages, these two are the main dominant ones[16]:

1. Petri Nets represent a static and dynamic specification of a system. For example systems specified as Petri nets can be evaluated directly using tools which execute a given net.
2. Petri Nets offer a graphic and a formal representation.

2.3 Agent Definition

Different definitions of agent have been used so far, ranging from primary intuitive to formal notions.

An Agent is a software module which has the following features [9]

1. Autonomous behavior: execute given service tasks in a given environment autonomously. For example to find the cooperative agents and the executable computer by itself at its run time.
2. Asynchronous execution: an event will be a trigger of agent service execution,
3. Cooperative achievement: cooperate with other agents by mutual message based communication.

3. Software Process and Agent Approach

Software development is a dynamic and distributed activity in which many cooperating participants may act partially independently of each other to iteratively transform an initial set of requirements into a validated operational system [15].

The purpose of this study is to present a more comprehensive model of software development that captures the concurrency that exists among all activities of software development and presents the distributed nature of the state process. A process model that is able to describe the synchronization and communication necessary to coordinate the different participants in the software process.

In order to represent each participant in the development process, we use the concept of agent. These agents have different role, knowledge and competencies. They are not working in isolated or sequential fashion, but they need to communicate, coordinate their actions and in general, to co-operate to produce a software.

Each agent possess a compatible goal with others agents goals. All together, they accomplish the global goal (common) [2].

Each software process agent [9] adopts a goal-oriented approach to describe and execute its activities. It decomposes its goal on sub-goals in order to find and accomplish a plan.

This approach permits the description of the dynamic aspect of software process. There is a possibility to modify the process activities (in execution) dynamically. That is, the order of activities is not static. It is possible to modify the actions plan or create new actions when user needs are modified.

The software process agents communicate between them with an explicit interaction model based on messages [15]. In this way, certain goals need the agents co-operation. This goal can be seen as a common or divided one.

The state process is represented by the states of whole agents, and the development process is represented by combined behaviors of different agents. Each agent is an actor that possesses its own life cycle and different states. That is the development process is parallel and incremental.

4. The software process agents

The proposed model is a network of agents that cooperate and coordinate their activities to accomplish a final goal: the production of a reliable software at a lower cost.

Our approach to design a multi-agent model is based on analyzing the software production process. That is, through this process, we identify the different activities, the different knowledge and tools used, and the different agents. To achieve this goal, we must identify the different actors and experts involved in the software production process.

The proposed model is essentially constituted of following agents: the developer (the software activity engineer), the software quality engineer, the user and the project manager. In the following, we will give an example of how to model the software development activities.

4.1 The developer

The main idea of this model is to develop a piece of software (PS). Clearly, one possible state of Software Engineering Activity is "NONE". That is there is no current activity underway. This is obviously the initial state. Once the activity starts, the software developer agent is busy developing a piece of software. Note that the transition from Developing a piece of software depends on a stimulus external to the developer. That
is the project manager is making a conscious decision to Start Requirement Specification. We will discuss the origin of this stimulus in the next section. Once under development, there are four types of activities in which an agent may be involved:

Performing Requirement Specification for that piece of software,
Doing Preliminary Design, then Developing The Subordinates of that piece of software,
Programing the source code of the piece of software,
or performing Software System Testing of the piece of software to verify that the software meets the requirements defined in the software requirement specification (SRS).

The Agent Developer is again modeled as a set of sub-agents. Each sub-agents deal with one of the four activities mentioned above. For instance:

The requirement analysis engineer is the responsible agent to accomplish the requirements specification activities where the principal role is to develop an adequate software. The Software Requirements Specification (SRS) activity is expanded as in Figure 1. We see that initially, the requirements specification is “UNDER DEVELOPMENT”. While in this state, all the usual requirements related activities are performed [5, 6, 4]. This includes:

Performing problem analysis,
writing an SRS,
documenting complex interfaces in an interface requirement specification (IRS),
and perhaps creating a draft version of a user’s manual.

We remain in this state until the SRS is completed where we declare that the requirements specification are “DONE”. Upon management agent directive, we now either BASELINE SRS or REVISE SRS. If the request is to revise the SRS, we return to “UNDER DEVELOPMENT”. If the request is to baseline the SRS, we declare the SRS to be “BASELINED”, and the requirement analysis engineer agent remain in this state until later revisions are necessary.

The designer is an agent that permits the transformation of requests into a representation near than source code.

5. Interaction between software processes agents

In this section we show how developer, user, project manager, and software quality engineer interact in general (see figure 2). Specifically, the project manager orchestrates the overall order of all other activities. Thus new and/or changing user needs drive the spawning of new and the modification of old manager decisions (arrow A). The software development decision drives the initiation of activities within developer (arrow B). The completion of various intermediate products developer triggers the initiation of specific reviews (arrow C). The passing or failure of any review within the software quality engineer signals the project manager (arrow D) to proceed with appropriate commands to developer (arrow B). Finally, the creation of software solution by developer changes the state of user needs satisfaction (arrow E).

6. Object-Oriented Petri net for a software process model

Software development is a dynamic and distributed activity in which many cooperating software process agents may act partially independently of each other to iteratively transform an initial set of requirements into a validated operational system. Different agents have different and selective knowledge about an evolving software system. The software system is typically characterized by a large set of software objects such as requirements definitions, design documentation, specification and program modules, test protocols,...etc. Semantic relationships between these objects and agents (participants into software product) influence the process dynamics and are themselves subject to dynamic changes.

In this context an adequate model should capture the distributed nature of information used by different software process agents in its various development states and the distribution of the changes it undergoes. Once concurrency has been identified as a central
issue, a synchronization and communication necessary to coordinate concurrent activities.

It is crucial to provide abstraction mechanisms that allow defining functional and structural properties of software objects and specified behavior and concurrency between agents of software process.

### 6.1 Concepts of proposed model:

Software objects are represented by pairs. The first component denotes a unique, system-provided object identifier and the second is a list of terms defining an object’s actual attribute values. For example, the form

```
"object module: (author: name, date: modification, spec: interface, body: implementation)"
```

defines objects of type module to have at least four attributes whose values are of type name, modification, interface, and implementation, respectively. These attributes might capture the properties of program modules relevant for configuration management. Further attributes can be added to an object during process execution as defined by the process model.

Software objects are created dynamically during process execution, and all its dynamic aspects (creation, destroy, changes of attribute values…) are captured by the Petri net (high level).

In our model, all S-elements are labeled with names of unary predicates that are defined on software objects types. T-elements are labeled with action names. Arc which represent a causal flow relation between S- and T-elements and vice versa, are labeled with sets of pairs of the form `<Id, Attr-list>` where each pair denotes an instance of object type.

The marking is given as a distribution of sets of objects over the S-elements of the net. Marking represent distributed development states.

The software process model in our proposition is a High Level Object Oriented Petri Net in which each software process agent is represented by a subnet.

### 6.2 Example

In this section, we try to illustrate the Petri net concept in the representation of software process agents by using the designer as example (see Figure 3). With a Software Requirement Specification (SRS) of piece of software (PS), the designer constructs the Preliminary Design (PD) of this piece. After this, we have a choice between design verification, or built the refinement of design when subordinates of piece of software appear. Upon the passing or failing the design review, the designer finishes its activities or modifies the design in order to rectify errors.

This model shows the non determinism hidden in the informal description in the form of conflicting actions, (verification request and design refine: two actions in conflict), explicates development states, visualizes the concurrency between activities (for example: we can modify the design of piece of software and at the same time construct the design of other piece), and visualizes the software objects flow between different actions. It also shows relations between the various types of documents and how they are updated during development activities.

The Object Oriented Petri Net model also supports stepwise refinement based on:

- Replacing actions by super sets whose border consists only of actions,
- Replacing places by super sets whose border contains only places, and
- Adding detailed abstract level representation of software objects.

![Fig 3: Petri Net model for the Designer](image-url)
1. A natural building of autonomous components as a hierarchy of Object Oriented Classes with explicit interfaces where each component behaves with respect to an intra-component evolution pattern that will exhibit intra as well as inter-object concurrcency.

2. While such components behave autonomously, they may (synchronously or asynchronously) interact with each other using their explicit interfaces following an inter-component interaction pattern, and

3. Verification/Validation based on Petri Net techniques.

Acknowledgements

We would like to acknowledge that this work is supported by a grant from the Algerian and French Governments (Accord Programme CMEP 01 MDU 512).

8- References


