45

PCS: A CASE Tool for Distributed Group Software Development

R.R. Huang and S. Jarzabek
Department of Information Systems and Computer Science
National University of Singapore, Singapore 119260
Tel: (+65)-7722864, (+65)-7722863
Fax: (+65)-7794580
Email: {huangr, stan}@iscs.nus.sg

Abstract

A large software project is usually developed by a group of people who may work for different companies and at different locations. Such a distributed development of software project needs support from both methods and CASE tools. In this paper, we suggest a software process based approach to support distributed group software development, and present a CASE tool called PCS that we developed to support the proposed method. PCS is based on Petri net process model, object-delegation role model and object-oriented repository model. All these models are discussed in the paper. In addition, a visual process language in PCS is also briefly introduced.

Keywords

CASE tool, group support software development, software process, Petri net, process model, role model, object repository model.

1 INTRODUCTION

With the wide spread use of high-performance workstations and computer networks, distributed group software development (DGSD) becomes increasingly popular. Software engineers working for different companies at different locations may collaboratively develop a common software project. They can work on their own computers and communicate with each other and share information through computer networks, ranging from LAN to Internet.

N. Terashima et al. (eds.), *Advanced IT Tools*

DGSD is closely related to the subjects of cooperative computing, groupware, and software process modeling. Cooperative computing mainly focuses on how to make a set of processes run on different processing nodes synchronously. These processes cooperate to reach a common goal and together they form a distributed program. Groupware mainly concentrates on modeling and coordinating the interactions among a group of human agents who work together for a common goal. Groupware software systems are combinations of network and cooperative computing. Software process modeling aims at conceptually modeling a software project. Both dynamic and static information about a software project, such as development processes and human roles, are captured in a software process. Distributed group software development involves many project developers who collaboratively develop a software project in a distributed environment.

DGSD brings new requirements for methods and CASE tool support. First, modeling means should be powerful enough to cover software project, human agents and distributed environment. Next, CASE tools are expected to support such a conceptual model more directly. There are some pioneering work on the subject of DGSD. For example, GSS (Hay, 1995) is a software system that supports group software development over a PC LAN. EPOS (Con, 1994) is an object-oriented cooperative process support system in UNIX-based workstation environment. SPADE (Ban, 1994) is a Petri net based software process support system. SOCCA is a specification for coordinated and cooperative activities.

PCS is a Petri net based CASE tool for the support of DGSD. Compared to the existing methods and systems, the advantages of PCS are that it supports software process modeling on high conceptual level and combines Petri net and object repository model to represent executable software processes. Most existing related systems allow users to model a software process with tool models or in some programming languages. This hampers the direct modeling of a problem situation, while forcing the process designer to take into account a lot of technical features of the modeling formalism. Our conceptual modeling method provides process designers with graphical modeling means, clear modeling steps and the way to map a conceptual software process to the system model of PCS. The existing Petri net based systems usually use traditional Petri net to model a software process. However, Petri net alone is not powerful enough to represent structure relationships among different condition nodes in Petri net. Therefore, we introduce large grain condition nodes ("repository") into Petri net.

In section 2, main characteristics of working with PCS are described. In section 3, a method for conceptually modeling a software process for DGSD is briefly introduced. In section 4, the system model of PCS is presented. In section 5, modeling facilities on process, role, repository and a visual process language is outlined. In section 6, brief description about PCS system is given. In section 7, the whole paper is concluded.

2 SOME CHARACTERISTICS OF WORKING WITH PCS

With PCS, a group of project developers can work together for the development of a common software project. The working environment is shown in Figure 1. Several human roles can be identified in this Figure. They are Process Designer, Process Manager, Group Leader, Developer, Object Server and Repository. Process Designer (PD) is responsible for modeling software processes. Process Manager (PM) is responsible for the control of process enactment. In PCS, some process control operations, such as "annealing", are

specifically defined for this role and can not be used by other roles. Group Leader (GL) is responsible for assigning roles and human agents to process activities. This role has some special privileges. He is allowed to dynamically change the relationship between human agents and process activities. Developers (DEs) are responsible for working on development tasks, such as GUI programming. This role can be further decomposed into roles on lower conceptual levels, for example, designer, programmer, QA engineer, etc. Object Server manages object repository and controls the access to it. Object repository contains all project related information and project deliverables. Actually, there are global and local repositories in PCS. Global information, such as software process specification is stored in global repository. Temporary information is stored in local repositories. Local repositories are attached to a single or a set of group members and are not illustrated in the Figure.

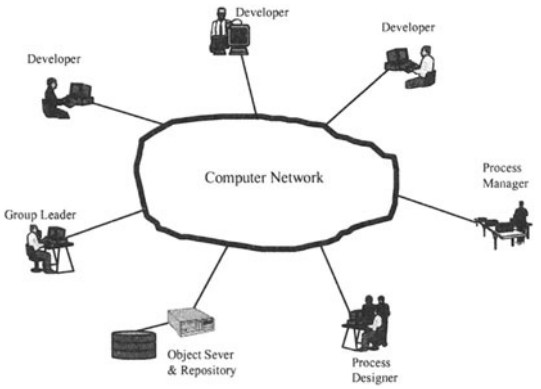


Figure 1 Working environment with PCS.

PCS supports cooperative working style. When GL, PD, PM work together, they have a common visual software process description on their screens. Whenever any changes to the process description by any one of them, other two roles will see the same changes on their screens instantly. Moreover, they can exchange their opinions about the process being designed by means of a message passing mechanism in PCS. The situation is the same while DEs are working together with GL and PM. The significant difference is that DEs can not change the predefined process. They have to work under the control of it. They also have the same view of a process execution and can also communicate with each other and share information resources in object repositories. In addition, when DEs work together for some activities defined as "simultaneous" in the software process, they can use a co-authoring tool, a text editor, to jointly write documents or programs.

3 CONCEPTUALLY MODELING A SOFTWARE PROCESS

Software process modeling aims at building a software project model which may capture both dynamic and static information about the development of a software project. The result of software process modeling is a software process. A software process is a set of software

engineering activities and associated information needed to transform a user's requirements into functioning software. Software engineering activities are completed by project developers with such roles as designer, programmer, QA engineer, etc. Some kinds of dependencies exist among these activities. They are, for example, sequential, alternative, parallel, simultaneous, precedent dependencies.

For conceptually modeling a software process, we first model four individual aspects of a software project, i.e., Project Activities, Project Developers, Project Deliverables and Project Constraints. After modeling these individual aspects, we integrate them together and thus form a complete software process specification. Modeling project activities is the key step in this process. It includes other three substeps, i.e., Modeling Project Activity Flow, Optimizing Project Activity Flow, Refining Project Activities. Figure 2 is a graphical representation of an incomplete software process for a compiler construction project. The objective of this project is to extend C++ with persistent objects. In Figure 2, big oval nodes represent project activities. Small oval nodes with hollow arrows indicate roles attached to project activities. Directed lines indicate data flows between project activities. Dashed lines mean refinement. Half-circles indicate alternative relationship and triangles indicate parallel relationship.

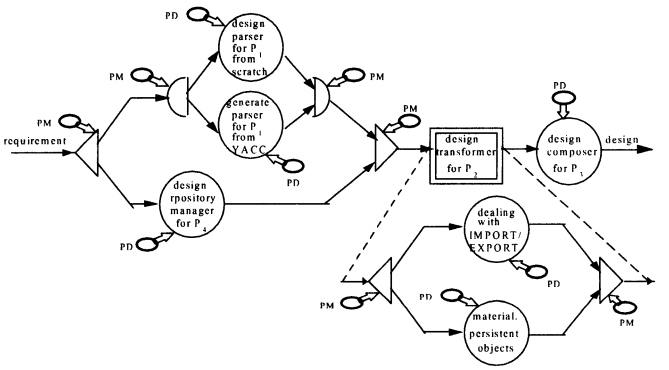


Figure 2 An example of a software process.

A software process specification consists of several sections. They are process section, activity section, role section, deliverable section, etc. Process section describes the construction of process flow, and the relationships between roles and project activities, as well as the relationships between project deliverables and project activities. Process section is an essential section in a software process specification. Details about modeling software process for other sections are omitted here because of space limitation. Figure 3 is the process section for above software process, which is composed of several process primitives. Below are all process primitives that can be used in process section.

(1) INT(D_1, Act, D_2): This is the first primitive to be used, which means the initial project activity is Act. Its input is D_1 and output is D_2. The activity can be further refined.

(2) SEQ(Act, {Act_1, Act_2, ... , Act_n}, {D_1, D_2, ... , $D_{(n-1)}$}): This primitive means Act can be further refined by a set of sequential activities {Act_1, Act_2, ... , Act_n} which are separated by a set of project deliverables {D_1, D_2, ... , $D_{(n-1)}$}.

(3) PAR(Act, {Act_1, Act_2, ... , Act_n}, {D_{11}, D_{12}, ... , $D_{1(n-1)}$}, {D_{21}, D_{22}, ... , $D_{2(n-1)}$}): This primitive means Act can be further refined by a set of parallel activities {Act_1, Act_2, ... , Act_n}. The input and output of Act_i are D_{1i} and D_{2i}, respectively.
(4) ALT(Act, {Act_1, Act_2, ... , Act_n}): This primitive means Act can be further refined by a set of alternative activities {Act_1, Act_2, ... , Act_n}.
(5) SIM({Act_1, Act_2, ... , Act_n}, Act): This primitive means a set of activities {Act_1, Act_2, ... , Act_n} should be done simultaneously and can be treated as a single activity Act.
(6) ITE(Act_1, Act_2): This primitive means there is an iteration from Act_1 to Act_2.
(7) ROL(Act, {R_1, R_2, ... , R_n}): This primitive means there is a set of roles {R_1, R_2, ... , R_n} attached to the activity Act.
(8) PRO(Act, P, I/O): This primitive means deliverable P is related to project activity Act. If it is an input of Act, then the third argument is I , otherwise the third argument is O.

```
Process_Section
Begin
   INT(requirement, design_RC++_compiler, design);
   SEQ(design_RC++_compiler,
       { design_parser_repository_manager,
         design_transformer,
         design_composer
       },
       { parser_design AND repository_manager_design,
         transformer_design
       });
   PAR(design_parser_repository_manager,
       { design_parser,
         design_repository_manager
       },
       { requirement_for_parser,
         requirement_for_repository_manager
       },
       { parser_design,
         repository_manager_design
       });
                    ⇨ turn right !

   PAR(design_transformer,
       { design_with_IMPORT_EXPORT,
         materializing_persistent_objects
       },
       { IMPORT_EXPORT_language_comp,
         PERSISTENT_language_comp.
       },
       { IMPORT_EXPORT_design,
         PERSISTENT_design
       });
   ALT(design_parser,
       { design_parser_from_scratch,
         generate_parser_from_YACC
       });
   ROL(design_parser_from_scratch, PD);
   .....
   PRO(design_parser_from_scratch,
       requirement_for_parser, I);
   .....
End
```

Figure 3 A fragment of a software process specification.

4 THE SYSTEM MODEL OF PCS

In PCS, four aspects of information, i.e., project activities, human roles, software tools, deliverables and certainly the relationships among them are considered in process modeling. The system model of PCS is a triple (P_M, R_M, O_M), in which P_M represents the process model, R_M represents the role model, O_M is an object repository model. Tools are modeled with role. Calls for tools are regarded as behaviors of a role and thus encapsulated in a role object. The elements modeled by P_M are probably also the elements modeled by R_M or O_M. For example, roles and deliverables are elements modeled by P_M. However, they are also the elements modeled by R_M and O_M, respectively. Therefore, the system model of PCS is a multidimensional model, as shown in Figure 4. In PCS, P_M is a Petri net model, R_M is an object-delegation model and O_M is an object-oriented repository model with single inheritance.

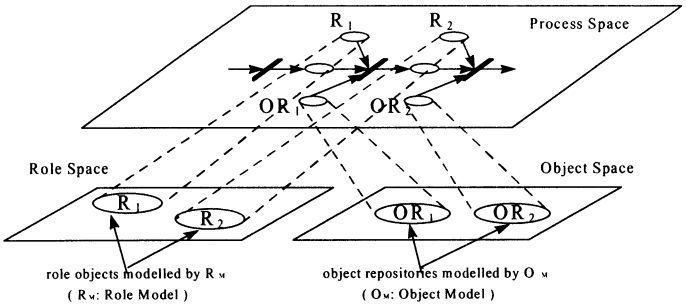


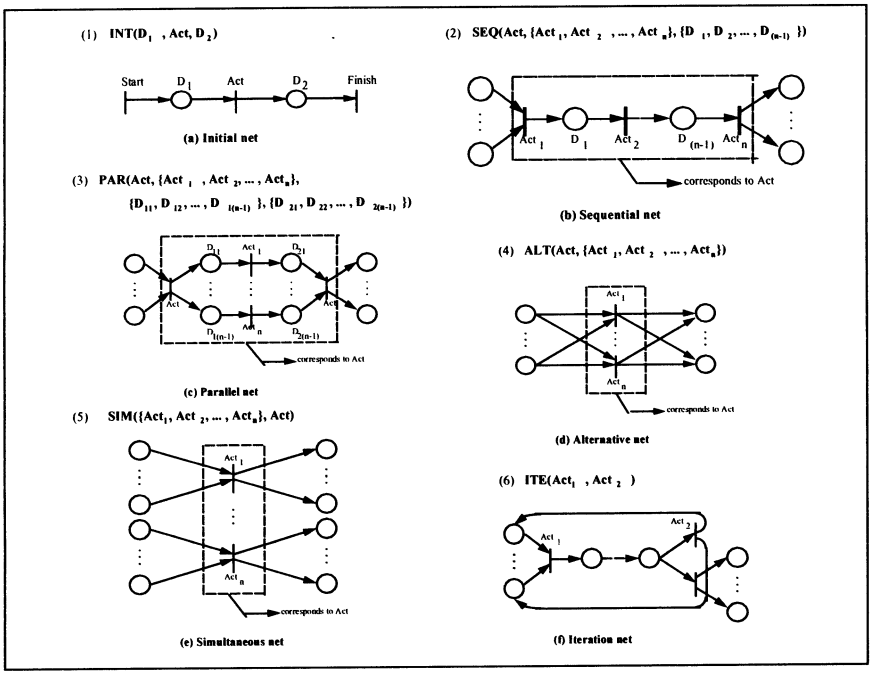
Figure 4 The system model of PCS.

5 MODELING FACILITIES IN PCS

In PCS, modeling facilities include Petri net, visual process language, role model and object repository model.

5.1 Petri Net Process Representation

For making a process specification executable, we map it onto a Petri net. In PCS, each process primitive corresponds to a specific Petri net (see Figure 5).



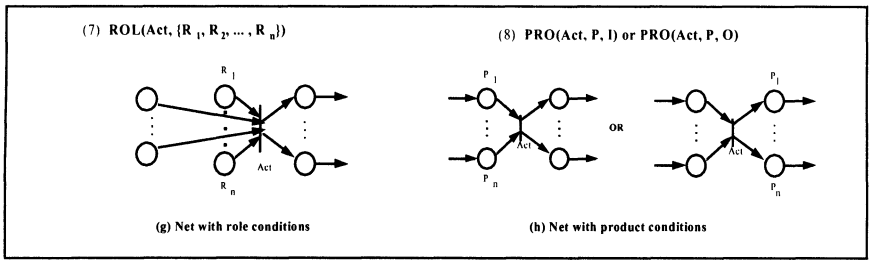


Figure 5 The correspondence between specification primitives and Petri nets.

In Petri net representation, roles and deliverables (/products) are condition nodes. Project activities are events. Because Petri net alone is not able to represent structure relationships among condition nodes, we introduce large grain condition nodes ("repository") into Petri net. With object repository model, structure relationships among condition nodes can be explicitly modeled.

5.2 Visual Process Language

The transformation from process specification to Petri net is automatic in PCS. Users are provided with a visual process language. They can draw the expected process on the screen. PCS will check the process diagram according to naming rules, connective rules, integrity rules and embedding rules, and then transform it from a process diagram to a specification and finally to a Petri net. An example of a process diagram is shown in Figure 2. For each process diagram, there is a special "start" node and a "finish" node. This rule is also true for sub-process diagram (the diagram for refined process activity). Note that all these nodes and some tokens about data flow between process activities are omitted in Figure 2.

5.3 Object Delegation Role Model

Roles are modeled in an object-oriented way. In PCS system, roles are modeled as classes. Objects, the instance of role classes, are human agents that take part in the process activities. Two special cases are considered in the design of role model. First, a human agent can belong to different role classes at different time. The membership is dynamically decided by GL. With the traditional inheritance based object-oriented method, such problems can be solved by multiple inheritance. However, in a more dynamic environment, this may cause the number of intersection subclasses increasing dramatically and mess up the class hierarchy. Next, the software tools encapsulated in a role class are dynamically shared by human agents with different roles. In traditional object-oriented systems, an object can only belong to a class. This restricts the variability of the behaviors of an object. Object-oriented systems based on object-delegation model can overcome these weaknesses (Lieberman, 1986). In PCS, we impose some restrictions on the role delegation mechanism. For example, delegation is not allowed to be nested; only behavior can be delegated, etc. In addition, we define some message patterns for interactions between human agents. For example, DLG(> role1, role_key2: func_name), which means a behavior of role1 is

delegated by role2 and the interactive interface will be displayed on role1's screen. Symmetrically, we also have DLG(role_key1, > role_key2: func_name).

5.4 Object Repository Model

Information related to the software project, such as software process, deliverables and human agents, are stored in global or local repositories. All repositories support an object-oriented repository model that has the concepts of class, object, single inheritance, complex object, set-valued attribute and object version. In addition, check-in/check-out mechanism is available for objects moving between global and local repositories. A SQL-like query language is also provided. Figure 6 is an example schema in PCS global object repository, in which big rectangles indicate classes and tokens with * are set-valued attributes. Directed lines represent "consist-of" relationships and hollow arrow indicates "is-a" relationship.

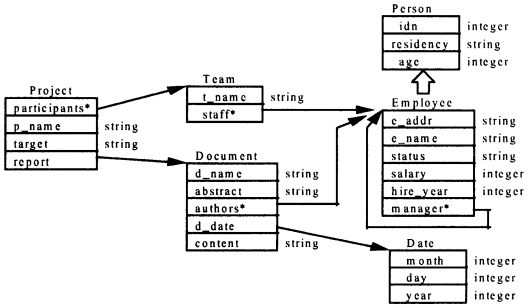


Figure 6 A schema example.

A query model is designed on the object repository model. Below are three query examples. Note that "SELECT t1; FROM t2; WHERE t3;" is abbreviated to "{t1; t2; t3}".

Example 1. Get all names of employees who are researchers in OMS research group.

{O.e_name; O/Employee; ∃p/Team(O.status="researcher" ∧ O∈p.staff ∧ p. name="OMS")}

Example 2. Get the name of the report on PCS project, submitted right after Dec. 1, 1994.

{ NEXT(O.report[01/12/94]).d_name; O/Project; O.p_name="PCS" }

Example 3. Get person's ID number. If he is employed, month salary higher than US$2000.

{ O.idn; O/Person; CH(O)=[Employee:O.salary>2000] }

6 SYSTEM OVERVIEW OF PCS

The current version of PCS system was implemented on the platform of SUN OS, X-Window Motif and LAN. The main components of PCS system are illustrated in Figure 7. PCS has a client-sever architecture. Servers include Tool server, Process server, Role server and Object server. For better efficiency, global/local servers and global/local repositories are used.

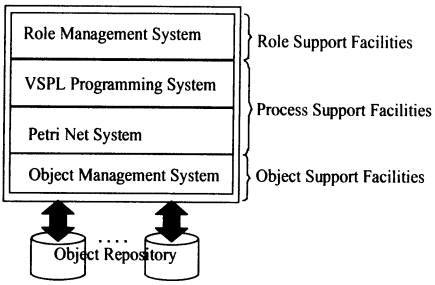


Figure 7 Main components of PCS.

7 CONCLUSION

In this paper, we introduce the objectives, methods, models of PCS which is a CASE tool for DGSD. The advantages of PCS are application independent and model based. However, further improvements are still needed in many directions. For example, heterogeneous platforms and Internet, more message patterns for collaborative software development, inference ability and support for multiple software development methods.

8 REFERENCES

Bandinelli, S. et al. (1994) SPADE: An Environment for Software Process Analysis, Design and Enactment, in *Software Process Modeling and Technology*, (ed. A. Finkelstein, J. Kramer, B. Nuseibeh), Research Studies Press Ltd., England, 223-247.

Conradi, R. et al. (1994) EPOS: Object-oriented Cooperative Process Modeling, in *Software Process Modeling and Technology*, (refer to the first reference), 33-70.

Engels, G. et al. (1994) SOCCA: Specifications of Coordinated and Cooperative Activities, in *Software Process Modeling and Technology*, (refer to the first reference), 71-102.

Hayne, S.C. et al. (1995) Experiences with Object-oriented Group Support Software Development. *IBM System Journal*, **34**, 1, 96-119.

Lieberman, H. (1986) Using Prototypical Objects to Implement Shared Behavior in Object Oriented System. *ACM SIGPLAN Notices*, **21**, 11, 214-223.

BIOGRAPHY

Dr. Riri Huang is a postdoctoral fellow at the Department of Information Systems and Computer Science, National University of Singapore. His research interests include distributed software engineering, software process modeling, object-oriented repository for software engineering and CASE tools. He received his Ph.D. degree from Peking University.

Dr. Stan Jarzabek is a Senior Lecturer at the Department of Information Systems and Computer Science, National University of Singapore. His research interests include business aspects of software engineering, software reuse, CASE tools, maintenance, re-engineering and business process modeling. He received his Ph.D. degree from Warsaw University.