ISBN 978-83-60810-22-4

Extending the Descartes Specification Language Towards Process Modeling

Joseph E. Urban[3], Vinitha H. Subburaj[1] , Lavanya Ramamoorthy[1]
[1]Computer Science Department
[3]Industrial Engineering Department
Texas Tech University
Lubbock, Texas 79409 USA
{vinitha.subburaj,joseph.urban,lavanya.ramamoorthy}@ttu.edu

Abstract—**With current complex real time software problems, the need for reliable software specification becomes crucial. This paper overviews the use of formal methods to specify requirements and the advantage of using an executable formal specification language processor to develop a process model for the development of a software system. The paper presents how a software process can be described using the Descartes specification language, an executable specification language, and the language extensions made to Descartes to make it suitable to describe a software process.**

Index terms—software specification; software process model;

I. Introduction

Software development is a complex and creative effort. A software process is a sequence of steps that are used to manage the development of software. A software process should be handled effectively to deliver the software on time and with good quality. Time, cost, and quality are some reasons why software development should be automated. Software process modeling focuses on what occurs during software creation and evolution. The basis for process models includes the individuals involved in the development of software, the assigned work, tools required to do the work, and the final results of the work.

Requirements analysis is a significant phase of software development because if any fault in the specification is left undetected, it can be carried over into the next phase. Thus, later correction of the fault would involve fixing the fault and fixing the effects of the fault in subsequent phases. Apart from consumption of a large amount of resources, for a change in a specification, the entire code will have to be rewritten by the developers. Also, since there is no actual system to verify the requirements provided by the user in a conventional software life cycle, development of such unverified requirements can cause errors in the program code.

The remainder of this paper is structured as follows. In Section II, related work that has been done in developing agent systems using formal methods is discussed. Section III discusses the research methodology used in this research effort. Section IV describes the extensions made to the Descartes specification language for specifying process modelling. Section V gives a comparative study with several existing methodologies. Section VI concludes the paper with a summary and future research.

II. Related work

Software process models are expressed formally by process modeling languages (PML). Many process modeling languages have been developed. The following section briefly describes the earlier work done similar to this research effort. This section includes APER-2, CSPL, DPEL, Marvel, Merlin, VRPML, and YAWN.

APER-2 [3] is a developer centered and object-oriented process language. A process program in APER-2 is composed of classes. CSPL (Concurrent Software Process Language) [2] is a process modeling language that uses Ada95-like syntax. CSPL integrates object-oriented Ada95 for modeling support and UNIX shell scripts for enactment support.

DPEL (Decentralized Process Enactment Language) [4] is a process modeling language which is used to model activities and activity synchronization. In DPEL, modeling of a process is based on developers. The process programs are converted into DPEL segments for enactment in DPEM. Marvel [8, 9, 1] is a process modeling technique based on rules. Marvel has three types of rules: project rule set, project type set, and project tool set. Process specific issues are described by project rule set. Project type set is used to specify the data with object-oriented classes.

MERLIN [7, 6] has a knowledge base which describes a process that is built using a rule based technique. Rules and facts are interpreted as forward and backward chaining. Backward rules and facts are interpreted in a Prolog-like manner. VRPML (Virtual reality process modeling language) [12] is a process modeling language that uses graphs to specify a software process. YAWN (Yet Another Workflow Notation) [11] is a graphical PML which uses directed graphs to represent a process interaction model.

A process model should enable effective communication, facilitate reuse, support evolution, and facilitate management [5]. Extensions were made to Descartes, such that it represents all the process entities and their relationships. The Descartes constructs are to support the basic elements of the software process.

Descartes [14] is an executable specification language that is based on three data structuring methods proposed by Hoare: direct product, discriminated union, and sequence. The language uses a tree structure notation to perform analysis and synthesis within a specification.

III. Extensions to the Descartes Specification Language for Process Modeling

In order to describe a software process, basic elements of the software process must be modeled. The basic elements of the software process are activities, products, role, human, and tool. The following describes extensions to Descartes that support software processes.

A. Activities and products

An activity is a step of a process that produces changes to the software product. The product is the set of artifacts developed and maintained in a project. The product is the input and output of an activity. A pre-pended unary reserve word "activity" is included in a Descartes module for declaring an activity module. After the module title, the module contains the specification for the other process elements. The input PRODUCTS consist of software products.

```
activity ACTIVITY_NAME_USING_(PRODUCTS)
        PRODUCTS
              inputs
```

B. Human actor

An actor is a person who is responsible for a software process activity. The reserved keyword "actor" is used to specify the person responsible for that activity. Consider the example of "modify_design" activity which is performed by the actor "design_eng1". The "design_eng1" is responsible for performing the activity "modify_design".

```
activity MODIFY_DESIGN_USING_(PRODUCTS)
        PRODUCTS
              requirement_change
                    FILE
              design_document
                    FILE
        actor
              'design_eng1'
```

C. Role

A role is the rights and responsibility of a person who is going to perform a software process activity. The "role" construct introduced by Medina and Urban [10] is used for specifying the role of the actor in an activity.

```
activity MODIFY_DESIGN_USING_(PRODUCTS)
        PRODUCTS
              requirement_change
                    FILE
              design_document
                    FILE
        actor
              'design_eng1'
        role
              'design_engineer'
```

In the above example, the activity "modify_design" is performed by a "design_eng1" who is a "design_engineer".

D. Tool

The tools used in the software production, such as textual editors and case tools, should be represented. The reserved keyword "tools" is used to specify the tools used in that activity. Consider the example of "modify_activity" in which "texual_editor" is used to edit the "design_document".

```
activity MODIFY_DESIGN_USING_(PRODUCTS)
        PRODUCTS
              requirement_change
                    FILE
              design_document
                    FILE
        actor
              'design_eng1'
        role
              'design_engineer'
        tools
              'textual_editor'
```

E. Process example

In this example, the process of designing a library management system is used. Suppose a library management system consists of two subsystems 'subsystem1' and 'subsystem2'.

When the process starts, subsystem1 is designed by 'design_eng1' and subsystem2 is designed by 'design_eng2'. Then the design of subystem1 is reviewed by 'design_rev1' and the design of subsystem2 is reviewed by 'design_rev2'. If the subsystem1 fails the review, the changes are made to subsystem1 by 'design_eng1'. If the subsystem2 fails the review, the changes are made to subsystem1 by 'design_eng2'. Changes are made to subsystem1 and subsystem2 until the subsystems pass the review. If subsystem1 and subsystem2 passes the review, the whole system is reviewed by the 'design_rev1' and 'design_rev2'. If the whole system fails the review, the changes are made to the system by 'design_eng1' and 'design_eng2'. Changes are made to the system until the system passes the review.

At the start of the process, the DES_SS1 and the DES_SS2 activities are started concurrently as shown in the specification below. The "parallel" construct introduced by Sung [13] is used to express the concurrent execution.

```
DESIGN_LIBRARY_SYSTEM
        return
            parallel
                DES_SS1_USING_(PRODUCTS)
                DES_SS2_USING_(PRODUCTS)
```

In the DES_SS1 activity, the 'design_eng1' takes the requirement_document as input and designs the subsystem1_design_document.

The subsystem1_design_document is modified using the tools 'textual_editor' and 'case_tool'. After subsystem1 is designed, the REVIEW_SS1_DESIGN activity module is executed.

```
activity DES_SS1_USING_(PRODUCTS)
         PRODUCTS
               products
                       requirement_document
                               FILE
                       ss1_design_document
                               FILE
         actor
               'design_eng1'
         role
               'design_engineer'
         tools
               'textual_editor'
               'case_tool'
return
        SS1_DESIGN_DOCUMENT
        ' designed_by '
        ACTOR
        ' using '
        TOOLS
        REVIEW_SS1_DESIGN_USING_(PRODUCTS)
```

In the DESIGN_SS2 activity, the 'design_eng2' takes the requirement_document as input and designs the ss2_design_document. The ss2_design_document is modified using the tools 'textual_editor' and 'case_tool'. After the ss2 is designed, the REVIEW_SS2_DESIGN activity module is executed. The specification written for the DES_SS2_USING_(PRODUCTS) activity is similar to the DES_SS1_USING_(PRODUCTS) activity.

In the REVIEW_SS1_DESIGN activity, the 'design_rev1' takes the requirement_document and the ss1_design_document as input and reviews the design of ss1. The design of subsystem1 is reviewed using the tool 'textual_editor'. If subsystem1 passes the review, the REVIEW_SYSTEM_USING_(PRODUCTS) activity is called. If subsystem1 fails the review, the MODIFY_SS1 activity module is executed.

```
activity  REVIEW_SS1_DESIGN_USING_(PRODUCTS)
        PRODUCTS
                products
                        requirement_document
                                FILE
                ss1_design_document
                                FILE
                        review+
                                review_pass
                                        STRING
                                review_fail
                                        STRING
                        feedback
                                FILE
        actor
                'design_rev1'
        role
                'design_reviewer'
        tools
                'textual_editor'
return+
        REVIEW_PASS
                SS1_DESIGN_DOCUMENT
                'reviewed_by '
                ACTOR
                ' using '
                TOOLS
                'subsystem1_passed_the_review '
                REVIEW_SYSTEM_USING_(PRODUCTS)
        REVIEW_FAIL
                SS1_DESIGN_DOCUMENT
                'reviewed_by '
                ACTOR
                'using '
                TOOLS
                'ss1_failed_the_review '
                MODIFY_SS1_USING_(PRODUCTS)
```

In the REVIEW_SS2_DESIGN activity, the 'design_rev2' takes the requirement_document and the ss2_design_document as input and reviews the design of ss2. The design of ss2 is reviewed using the tool 'textual_editor'. If ss2 passes the review, the REVIEW_SYSTEM_USING_(PRODUCTS) activity is called. If ss2 fails the review, the MODIFY_SS2 activity module is executed. Similarly we can write the specifications for REVIEW_SS2_DESIGN_USING_(PRODUCTS) activity, MODIFY_SS1 activity, MODIFY_SS2 activity, REVIEW_SS_DESIGN_USING_(PRODUCTS) activity, and MODIFY_SYSTEM activity.

IV. Comparison

A process modeling language can be compared through the following criteria: interface and style. Table 1 shows the comparison of several process modeling languages based on interface and styles along with the Descartes specification language used for process modeling.

Process modeling languages	Interface		Style			
	Graphical	Textual	Rule based	Object Modeling	State Transitions and petrinets	Programming language
APER-2		X		X		
CSPL		X				X
DPEL		X				X
Marvel		X	X			
MERLIN		X	X			
VRPML	X				X	
YAWN	X				X	
Descartes Specification Language		X	Formal Specification Style			

Table 1 Comparison of process modeling languages

The Descartes specification language could be used in both specification development and for describing a software process. With this advantage, the Descartes specification language has an edge over the other process modeling languages mentioned because, time and cost in training the personnel could be saved by using the Descartes specification language in both specification development and for describing a software process.

V. Summary and future work

Managing a process manually will consume more time, cost more, and can result in low quality software. Thus, automation of a software process will save time and reduce extra work. This paper introduced the extended Descartes specification language for automating a software process. Extensions were made to the Descartes specification language for describing a software process. Extensions made to the Descartes specification language were identified to be helpful in modeling the basic elements of a software process.

The future research effort can focus on providing tool support for designing, modifying, analyzing, simulating, and verifying a process. Tool support would be more helpful for a user to organize the work and to keep track of what is going on in a process. A tool could be provided for designing and modifying the specifications for a process. Simulating a process specification before it is executed could help in checking whether the process performs what is intended. At present, the Descartes specification language does not support simulation. Future research can concentrate on adding a simulation feature to the Descartes specification language. The tool support could be provided for simulation.

References

[1] N. Barguthi and G. Kaiser, "Scaling up Rule-based Software Development Environment," *Proceedings of the 3rd European Software Engineering Conference (ESEC'91),* Milan, Italy, Oct 1991, pp. 380-395.

[2] J. J. Chen, "CSPL: An Ada95-Like, Unix-Based Process Environment," *IEEE Transactions on Software Engineering*, Vol. 23, No. 3, March 1997, pp. 171-184.

[3] J. Y. Chen, S. C. Chou, and W. C. Liu, "APER-2: A Developer Centered, Object-Oriented Process Language," *Proceedings of theInternational Symposium on Multimedia Software Engineering*, December 2000, pp. 297-303.

[4] S. C. Chou, "DPEM: A Decentralized Software Process Enactment Model," *Information and Software Technology* 46, 2004, pp. 383–395.

[5] B. Curtis, M. I. Kellner, and J. Over, "Process Modeling," *Communications of the ACM*, Vol. 35, No. 9, September 1992, pp. 75-90.

[6] W. Emmerich, G. Junkermann, and W. Schafer, "MERLIN: Knowledge Based Process Modeling," *Proceedings of the First European Workshop on Software Process Modeling*, Milan, Italy, May 1991, pp. 181-187.

[7] H. Hunnekens, G. Junkermann, B. Peuschel, W. Schafer, and J. Vagts, "A Step Towards Knowledge-Based Software Process Modeling," *Proceedings of the First Conference on System Development Environments and Factories*, 1990, pp. 49-58.

[8] G. Kaiser, N. Barghuti, and M. Sokolsky, "Preliminary Experience With Process Modeling in the Marvel SDE Kernel," *Proceedings of the IEEE 23rd Hawaii ICSS*, Software Track, 1990, pp. 131-140.

[9] G. Kaiser and P. Feiler, "An Architecture for Intelligent Assistance in Software Development," *Proceedings of the 9th International Conference on Software Engineering* Monterey, April 1987, pp. 180-188.

[10] M. A. Medina and J. E. Urban, "An Approach to Deriving Reactive Agent Designs from Extensions to the Descartes Specification Language," *Proceedings of the Eight International Symposium on Autonomous Decentralized Systems*, March 21-23, 2007, pp. 363-367.

[11] D. Rossi and E. Turrini, "Using a Process Modeling Language for the Design and Implementation of Process-Driven Applications," *International Conference on Software Engineering Advances (ICSEA)*, 2007, pp. 55-61.

[12] K. Z. Zamli, "The Design and Implementation of the VRPML Support Environments," *Malaysia Journal of Computer Science,* Vol. 18, No. 1, June 2005. pp. 57-69.

[13] K.-Y. Sung and J. E. Urban, "A Real-Time Specification Method for Specifying and Validating Real-Time Concurrent Systems," *Proceedings of the Twelfth IEEE International Phoenix Conference on Computers and Communications*, Tempe, Arizona, March 24-26, 1993, pp. 578-584.

[14] V. H. Subburaj and J. E. Urban, "Issues and Challenges in Building a Framework for Reactive Agent Systems," *Proceedings of the Third International Workshop on Frontiers in Complex Intelligent and Software Intensive Systems (FCISIS-2010),* Krakow, Poland, February 15-18, 2010.