Proceedings of the 10th WSEAS International Conference on COMPUTERS, Vouliagmeni, Athens, Greece, July 13-15, 2006 (pp1-6)

Modular Software Process Simulation Models Through Metamodeling

MERCEDES RUIZ[1], ISABEL RAMOS[2], MIGUEL TORO[2]
Department of Computer Languages and Systems
[1]University of Cádiz
C/ Chile, 1, 11003 – Cádiz
[2]University of Seville
Avda. Reina Mercedes, s/n, 41012 – Seville
SPAIN

Abstract: - In this paper we present the main concepts and principles of a multilevel architecture to help in the development of modularized and reusable software process models under the System Dynamics approach. The conceptual ideas of the multilevel architecture have been formalized using UML as a notation. Metamodeling is used to support the process of abstract modules development. The architecture proposed is also based on ISO's Information Resource Dictionary System. The principles of the architecture and overall guide to develop software process simulation models are described in this work.

Key-Words: - Software process simulation modeling, simulation models reusability, simulation modeling architecture

1 Introduction

One of the factors limiting the development and wide application of software process simulation modeling in the industry has traditionally been the inability to deal with the conceptual complexity of formulating, building, calibrating and debugging complex models. A well-recognized method for reducing complexity involves structuring models as a set of distinct modules with well-defined interfaces. This has been a proven technique in the software development field, which has been and still is strongly influenced by the application of concepts such as modularization, encapsulation, reutilization, and, definitely, abstraction. Currently, there is a growing interest in defining and applying a methodology to formally develop software process simulation models. This significant interest has been shown during the last editions of ProSim where a good number of papers dealt, direct or indirectly, with this subject [1, 2, 6, 7, 10].

In recent papers, our research has been focused on the application of dynamic modeling and simulation as an effective tool for software process improvement. As a consequence of our research efforts, a Dynamic Integrated Framework for Software Process Improvement (DIFSPI) has been developed. This framework combines static algorithmical methods assumed as traditional techniques in the planning, monitoring and management fields of the software engineering [3, 4], with the dynamic methods of the software process modeling and simulation under the System Dynamics approach [11].

Since the mentioned framework was oriented to help in the process improvement field, the Capability Maturity Model (CMM) [8] was used as the guiding model to assess software process maturity. The internal hierarchical structure of this reference model served as a model upon which develop a multi-tier architecture for the development of the dynamic models of the framework.

This paper shows an overview of the steps followed to develop and implement this architecture. It is organized as follows: Section 2 shows how the proposed architecture can be formally modeled using UML as a notation. Section 3 describes how these concepts can be used to develop a software process simulation model. Finally, Section 4 summarizes the paper and draws the conclusions and further works.

2 Modeling the architecture using UML

2.1 Why UML

As the strategic value of software process simulation modeling increases, it is necessary to look for techniques to automate the production of software process simulation models and to improve quality and reduce both cost and time-to-develop. These

techniques should include component technology, patterns and frameworks, techniques that have proven valid to deal with the same problems in the software development arena. When dealing with complex systems such as software processes it is also important to seek techniques to manage the complexity of systems as they increase in scope and scale.

Bearing in mind that the development of software process simulation models can be thought of as a kind of software development project too, the techniques or approaches that proved successful in the software development field could help to solve the same problems in the field of simulation modeling. One of the techniques that were designed to respond to these needs in the software engineering field is the Unified Modeling Language (UML) [12].

The UML is a standard visual language mainly aimed to help designers and developers to specify, develop and document software systems. UML also serves as a tool to model business and processes.
The three main primary benefits of using UML in the field of software process simulation modeling are:

1. Provide system dynamics modelers with an expressive visual modeling language that works at a higher level of abstraction than stock and flow notation. UML can complement the causal diagrams adding the meaning of the conceptual architecture than would support the dynamic model.
2. Modelers can benefit of the concepts of specialization that help implement the principles of vertical aggregation of dynamic models.
3. Models built this way are independent of the final simulation language. In fact, it is easy and quick to develop a software tool to implement and run a simulation model created this way.
4. It helps to design making use of reusability of previous models already developed. It provides a mechanism to develop a library of modules to be used in the construction of complex dynamic models.

2.2 Why a metamodeling approach

Once the notation to guide and represent the software process simulation model has been chosen, the next step before starting the development of models is to determine the modeling approach. Traditionally, simulation models are developed following an iterative process that transforms the mental model into a running one by using a certain modeling approach and simulation language. It is often difficult to separate or avoid the influence that the modeling approach and the simulation language have over the model developer and, consequently, the final model.

Metamodeling supports the development of conceptual models that offer abstract views on certain aspects of the real world and the system to be implemented. They can be used for different purposes, such as a communication medium between users and model developers, for managing and understanding the complexity within the application domain, and, for making experiences reusable, easing the path to collaborative model building.

The reasons for using metamodeling in the software process simulation discipline are, among others:

1. The complexity of the software process requires a decomposition of the modeling task into subtasks.
2. It helps gather together different modelers expertise in a conceptual view that can be easily translated into a running model.
3. The community of software process simulation modelers have acknowledged the need of analyze, specify and document software process simulation models [2]. The metamodeling approach offers the tools to carry out these activities.

There are many approaches and architectures based on the concepts of metamodeling. Most of them were developed in the information system domain. For the purpose of this study, we use as a starting point the approach proposed by the International Standard Organization (ISO). ISO's Information Resource Dictionary System (IRDS) [5] proposes an architecture that combines information systems use and evolution. These two concepts are applicable to the field of software process simulation modeling since we need to represent models functionality, that is, the way they are going to be used and what the users can obtain from them, and their evolution, that is, the different versions of models that appear when working following the incremental model building approach. The core of the architecture proposed in this paper is composed of a multilevel repository based on the ISO's IRDS Standard. However, while this standard recommends a four-level structure for organizing the information, a three-level architecture is considered to be adequate to represent all the information associated with a software process simulating model. Figure 1 shows what is expected to find at every level of the architecture. A brief description of what is intended to be contained in each level follows:

1. *Scenario Level*. The Scenario Level contains objects which cannot have instances. The main

goal of a model building process is to produce a valid and running simulation model. This final model is precisely the main element of this level. Following an object-oriented terminology, it can be said that the final model is obtained by the instantiation of several classes in a set of objects. It is that set of objects and their interactions what constitute the final working simulation model. For a model to effectively work, some kind of information is also needed since all its parameters must receive an initial value. The set of initial values that populate the parameters of a model constitute the information that defines the scenario that the model will simulate.

2. *Model Level*. The Model Level represents the classes of the objects at the instance level. Those classes define the dynamic models as well as rules for their manipulation and inter-model communication. At the same time, these classes are themselves instances of the schema defined at the modeling language level. Using concepts from the field of the object-oriented programming, in this level a dynamic model can be represented by a set of classes all implementing an interface that collects the common behavior and services of what a dynamic model must offer.
3. *Metamodeling Level*. The Metamodeling Level contains the metaclasses that define the structure of the classes of the former level. In this level, the abstractions of inheritance and interface are plenty used to design modular dynamic models and dynamic modules that, when instantiated, will become a part of a running modular dynamic model. In our case, the metamodel level helps to describe the behavior of a dynamic simulation model. In order to do this, the concept of interface is used. Every dynamic model must implement the operations of the metaclass which are:
    a. *runOutput()*. It computes the current state of the level variables of the dynamic model using the current values, the auxiliary variables, and the value of the integration step.
    b. *runDifferential()*. It computes the value of the differential step.
    c. *runNextState()*. It computes the values for the auxiliary variables, closing the loop of computation.

3 Using the architecture to design and develop a software process model

This section illustrates how the proposed architecture can be used to develop a software process simulation model.

3.1. Structure of the model

The basic software process simulation model is composed of four dynamic models that implement the management processes as well as the system development engineering ones that take place in the software process. A description of these models follows:

- Development model. Software development process is aimed to the construction of a software product. The development model encapsulates all the cause-and-effect relations that determine the software production process, as well as the detection and correction of defects cycle.
- Plan model. This model collects the information related to the initial project plan as well as the current progress of the project. It helps the control model to determine the actions needed to keep the project on time and within budget.
- Control model. Using the information generated by the plan model, the control model makes the decisions aimed to improve the progress of a project.
- Human Resource model. The Human Resource model collects the causal relations that model the human resource management activities within an organization and/or a project.

As Figure 1 shows, a software process simulation model can be made of these four modules. This figure also shows that a software process simulation model can be composed of other software process simulation models.

This reflexive composition association helps to extend the software process model to develop complex models and provides flexibility of use of the architecture. For instance, for software organizations

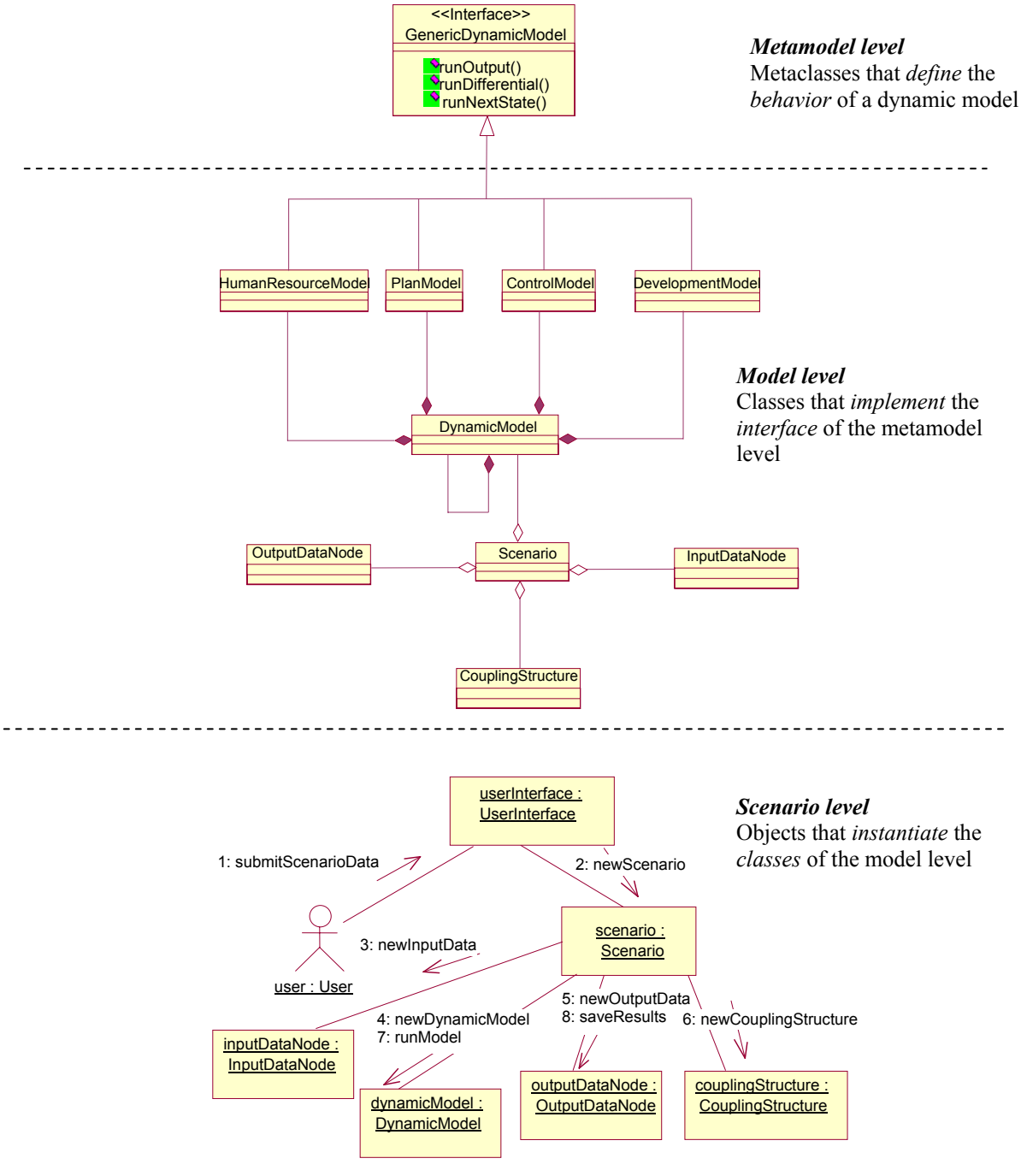

Fig. 1 Formalized structure of the software process simulation model

working with immature, not defined processes, it can be useful to model the software process at a high level of abstraction. For these kinds of organizations, using a simulation model that is helps visualize the evolution of the project and the main variables of the four models described above can be useful. However, for organizations that have defined, repeatable and managed processes, such a simple model is not very useful. As the maturity of the software organization increases, the simulation model that help model and simulate the processes carried out within the organization must also evolve and show a more complex behavior. In order to provide a flexible mechanism to allow software process simulation grow in complexity, the self-composition association is used. Hence, a software process simulation model can not only be composed of the four main models, but of other process simulation models.

This feature is especially useful when a hierarchical model such as CMM is used. In the framework developed, the software process simulation models for level 3 organizations are made of as many process simulation models as main activities figure at the top level of the WBS of a project. Each of these process simulation models are also made of the composition of the main four models described at the beginning of this section. This helps, for instance, gain control over the resource allocation to each activity.

3.2. Designing a scenario

In order to run a simulation model, it is necessary to define the scenario in which the software process takes place. Scenarios are defined at the Model Level of the architecture too. To define a scenario, the value for each of input parameter of the model must be provided. Input parameters are the mechanism to customize the software process to the own features of a software organization. Aspects such as the hiring delays, assimilation rates, etc. are to be defined as input parameters. As the maturity level of the organization increases, the number of input parameters required for the simulation model also increases because the amount of information that high level organizations have about their processes is expected to be bigger too.

3.3. Running simulations

Once the scenario is defined, it is possible to run a simulation of the model. Different policies or decisions can be simulated by changing the values of the adequate parameters or functions built in the equations of the dynamic modules that form the final simulation model.

The simulation results can then be used for different purposes. Simulation runs can be graphically visualize to analyze the qualitative behavior of the resulting variables. In the case that the equations and functions of the simulation model have been validated, the simulation outputs can also be used in the quantitative field. In this way, they can help to make decisions or select the software process improvement action that offered the better results. Finally, simulation runs offer also a good opportunity to study the results of different improvement actions using the techniques coming from the field of machine learning, as simulation can be used to generate datasets that act as inputs to these kinds of algorithms [9].

4 Conclusion

In this work, we have presented a description of a multilevel architecture for building dynamic models. The models built under this architectural pattern are made of the integration of different dynamic modules that can be easily added to the existing set of models and can be either enabled or disabled during a simulation course. The core of the architecture is based on ISO IRDS.

The architecture has been formally modeled using the UML notation that provides not only formality to the approach but the ability to direct translation to the programming constructors that implement the conceptual ideas of the architecture.

The software classes that result from this translation implement the principles of data encapsulation, reutilization, and operational abstraction in the field of software process modeling under the approach of System Dynamics. We are now working on the enhancement of the framework by adding a set of dynamic modules that model the engineering processes under different software development approaches.

Our future work is mainly concentrated on the full development of new dynamic modules In addition, although the experiments carried out with the current modules prove that they reproduce the expected behavior from a qualitative point of view, we intend to obtain real data to validate them from a quantitative perspective.

References:

[1] Angkasaputra, N., Pfahl, D., Making Software Process Simulation Modeling Agile and Pattern-based. *Proceedings of the 5th. International Workshop on Software Process Simulation and Modeling*, pp. 222-227. Edinburgh, Scotland (UK) 2004.

[2] Ahmed, R., Hall, T., Wernick, P., Simulation Modelling Practices of ProSim03 participants: a survey. *Proceedings of the 5th. International Workshop on Software Process Simulation and Modeling*, pp. 67-76. Edinburgh, Scotland (UK) 2004.

[3] Boehm B., *Software Engineering Economics*. Prentice-Hall Inc. 1981.

[4] Boehm B., E. Horowitz, R. Madachy, D. Reifer, BK. Clark, B. Steece, AW. Brown, S. Chulani, and C. Abts, *Software Cost Estimation with COCOMO II*. Prentice-Hall Inc. 2000.

[5] ISO/IEC International Standard. Information Resource Dictionary System (IRDS) -Framework ISO/IEC 10027, 1990.

[6] Kirk, D., Tempero, E., Proposal for a Flexible Software Process Model. *Proceedings of the 5th. International Workshop on Software Process Simulation and Modeling,* pp. 161-170. Edinburgh, Scotland (UK) 2004.

[7] Neu H, Rus I., Reuse in Software Process Simulation Modeling, *Proceedings of the 4th International Workshop on Software Process Modeling and Simulation*. Paper nº 17 Portland, OR (USA) 2003.

[8] Paulk M., SM. Garcia, MB. Chrissis, and M. Bush., Key practices of the capability maturity model. Version 1.1 *Technical Report CMU/SEI-93-TR-25*. Software Engineering Institute, Carnegie Mellon University, Pittsburg, PA.1993.

[9] Quinlan JR. *C4.5: Programs for machine learning*. Morgan Kauffman, 1993.

[10] Raffo D, Spehar G, Nayak U., Generalized Simulation Models: What, Why and How? *Proceedings of the 4th International Workshop on Software Process Modeling and Simulation*. Paper nº 26. Portland, OR (USA) 2003.

[11] Ruiz M. Ramos I. Toro M., A Dynamic Integrated Framework for Software Process Improvement. Software Quality Journal, Vol. 10, Nº2, pp.181-194 2002.

[12] Object Management Group. Unified Modeling Language. http://www.uml.org/

Acknowledgements:

The authors wish to thank the Comisión Interministerial de Ciencia y Tecnología, Spain, (under grant TIN2004-06689-C03-03) for supporting this research effort.