Component-based Software Process Support

Kevin Gary, Tim Lindquist, and Harry Koehnemann
Computer Science and Engineering Department
Arizona State University
Tempe, Arizona 85287-5406

Jean-Claude Derniame
Laboratoire lorrain de Recherche en Informatique et Applications
LORIA : Bd des Aiguillettes
BP 239 54 506 Vandoeuvre Cedex

Abstract

Only recently has the research community started to consider how to make software process models interoperable and reusable. The task is difficult. Software processes are inherently creative and dynamic, difficult to define and repeat at an enactable level of detail. Additionally, interoperability and reusability have not been considered important issues. Recent interoperability and reusability solutions advocate the development of standard process model representations based on common concepts or generic schemas, which are used as a basis for translating between heterogeneous process representations. In this paper we propose an alternative approach through the development of process-based components. We present the Open Process Components Framework, a component-based framework for software process modeling. In this approach, process models are constructed as sets of components which interact in meaningful ways. Interoperability and reuse are obtained through encapsulation of process representations, an explicit representation of process state, and an extendable set of class relationships.

1.0 Introduction

Since Osterweil's proposal[12] for automating the software process a decade ago, there has been significant debate about how to best define, execute, and support software processes.The Software Process community has proposed various modeling formalisms including Petri nets [9], rule-based formalisms [1,8,13], process programming languages [15], event-based representations [3,6,10], object-oriented approaches [6,10] and hybrids. Despite their benefits, systems based on these formalisms create enactable process models which are not interoperable nor reusable with one another. The prevailing solution is to advocate an intermediary standard process representation and provide translations for interoperability and reuse. We do not believe this approach is scalable and defeats the purpose of using heterogeneous process representations.

We advocate an object-oriented, component-based philosophy for providing software process interoperability and reuse. This paper presents *Open Process Components*, a component-based framework for software process definition and enactment. In this framework, components are well-encapsulated representations of process entities that interact in meaningful ways. The framework is solidly founded on mature concepts in the software process field, and yet is extendable so that process models may be customized in a particular domain. A componentized view of process representations results in easier process definition, modularized process enactment, natural interoperability, and greater potential for reuse.

This work was supported in part by the Office of Naval Research under Award number N00014-97-1-0872

2.0 A Component-based Process Philosophy

Enactable software processes are typically not repeatable. Instances vary according to constantly changing demands of specific projects. Fully elaborating a software process model to an enactable level of granularity is often too tedious, time-consuming, and costly[4].

Motivated by the need for interoperability and reuse, we advocate applying component-based techniques to software process modeling. Constructing software process models as sets of interacting components allows interoperability by encapsulating the semantics of existing representations. Reuse is achieved by brokering for predefined components.

A component-based approach:

- avoids deep integration of semantic models
- handles the natural complexity of software processes,
- responds to dynamic software processes, and
- facilitates reuse, minimizing one-shot process models.

Component-based process modeling requires a framework for developing components. The framework must identify process entities, define meaningful interactions between entities, and be able to incorporate new representations.

2.1 The Open Process Components Framework

The Open Process Components (OPC) framework provides a foundation for developing, integrating, maintaining, and reusing a variety of process representations. The framework defines basic abstractions of the problem space that can be specialized. Yet, the framework must make some commitments regarding the representation and manipulation of processes. The OPC framework is constructed upon three categories of abstractions, 1) a meta-model that identifies fundamental process entities and relationships, 2) a per component representation of process states and transitions, and 3) a set of class relationships layered in a structured fashion to produce type definitions for components.

2.1.1 The Basic Meta-model

The OPC Framework is derived from a set of basic abstractions contained in a meta-model. Instead of using this meta-model as a basis for translation between process models, we use it as a foundation for identifying elements of the process space for componentization, and for defining meaningful ways in which process components interact.

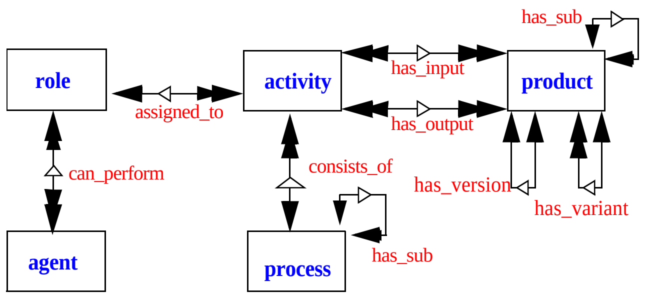

FIGURE 1. The Open Process Components Framework Meta-Model

The meta-model is centered around process entities representing work (Process and Activity), people (Role and Agent), and artifacts (Product). The meta-model defines the “rules of engagement” for components. It identifies what component types interact with what other component types under what relationships. These relationships are not static; process components and component relationships are highly dynamic during the course of the component’s life cycle.

2.1.2 Process Component States

The elements of the meta-model appear in most process models. Each model requires a different enactment service to interpret the representation and execute the process. Regardless of the formalism employed and the interpreter used, all models define actions on the entities within the process domain, which effect the states of those entities. The result of these actions can be modeled using a traditional *finite state machine*. The OPC framework includes finite state machines as part of its basic abstractions.

Finite state machines are represented through a *state transition graph*. Using state transition graphs explicitly for process modeling is not unique[7,11]. The OPC framework defines a basic set of states and transitions for Process and Activity components. These include states such as *executing*, *suspended*, and *aborted*, with corresponding transitions between states defined by actions such as *startProcess*, *suspendProcess*, and *abortProcess*.

Each component may dynamically modify its state transition graph or even construct a new one. A new state transition graph reflects the object’s unique behavior when interacting with other components within the framework. The current class definition for state transition graphs include operations to add and remove states and allowable transitions between states, making a component’s state and behaviors affecting state explicit and manipulable.

2.1.3 Component Class and Interface Definitions

The OPC framework identifies classes and interfaces in a layered, three-tier software architecture (Figure 2). The Framework Layer defines classes and interfaces modeling process entities derived from the OPC meta-model. The Representation Layer classes encapsulate process representations behind well-defined interfaces so that heterogeneous representations can interoperate. The Component Layer extends representations to particular domains. It is from this layer that actual Process component objects are instantiated. A process model in the OPC framework is a set of components, realized as objects of Component Layer classes, and a set of relations between those components, created under the constraints of the Framework Layer, implemented using Representation Layer semantics.

Figure 2 shows example classes at each layer of abstraction for the meta-model element *Process*. The Framework Layer contains the class *Process*, derived from the meta-model Process entity of Figure 1. The Representation Layer is comprised of class definitions for specific process representations, such as event-based processes (*ECAProcess*), sequencing processes (*OrderedProcess*), Petri Nets (*PetriNetProcess*), rule-based representations (*RuleProcess*), process definition languages (*PDLProcess*), and any other representations we wish to encapsulate. The Compo-

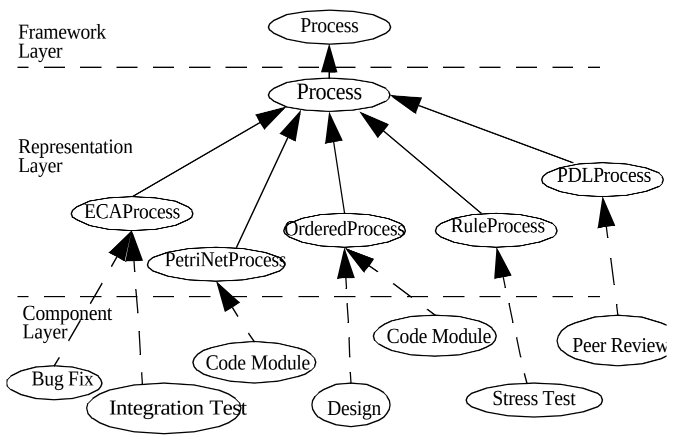

FIGURE 2. Object-oriented class diagram for Process components

nent Layer contains type definitions for actual process types. The dashed lines between layers in Figure 2 denotes that the Representation and Component Layers in fact can have many levels. This allows for multiple ways in which to extend and specialize the framework.

The first step identifies a base set of classes and interfaces at the Framework Layer. The next step is to construct encapsulations of process representations in the Representation Layer. The semantics of implementing Processes are encapsulated behind the interfaces inherited from the Framework Layer. For example, the implementation may come from a COTS process tool. Finally, components defined at the Component Layer delegate their implementation to Representation Layer objects. Delegation is used since inheritance would tie the component's type to its implementation. Component Layer objects are configurable. Component Layer classes represent generic process models. Actual components, or instances, represent customized process models. A component who has its relationships to other components (Process, Activity, Product, Role, or Agent) fully specified and bound is part of an instantiated process model[5].

Extending the OPC framework is straightforward. Subclassing the Framework Layer provides specialized implementations according to the semantics of given representations. Delegating Representation Layer classes provides implementations for the Component Layer. Create components from Component Layer classes to customize process models. Defining relationships between components instantiates process models. This methodology is a straightforward object-oriented approach for providing Conradi's levels of process specialization[5].

3.0 PCIS2 Process Services

The Open Process Components framework is currently used as a basis for the PCIS2 process services specification. PCIS2 is an architecture for supporting wide-area software development support. PCIS2 services include Configuration Management, Traceability, and Process Support. Services are integrated and distributed via CORBA.

The process support services in PCIS2 are based on the OPC specification and the Workflow Management Coalition (WfMC) sponsored Workflow Facility specification, known as jFlow[11], submitted to the OMG. The jFlow specification is largely an "object-ization" of existing WfMC interfaces[16]. This is not a drawback, but one of the strengths of the OMG's approach to adopting and adapting existing technology. The jFlow specification improves upon the original WAPI specifications by defining appropriate interactions between objects to gain interoperability and maintainability of workflow systems. The PCIS2 specification is object-oriented from the ground up, but has borrowed some of the jFlow concepts in order to maintain compliance with emerging standards.

PCIS2 and the jFlow specification differ in three areas. First, PCIS2 supports dynamic processes through ad hoc process support, and the ability to modify process definitions during enactment. Second, PCIS2 supports shared and circulating products, providing a mechanism for reasoning about data artifacts of the process. Finally, PCIS2 incorporates support for the metaprocess, by defining views on its services for controlling, defining, performing, and monitoring processes. jFlow only defines interfaces for performing (enacting) and monitoring workflows.

It should also be pointed out that jFlow identifies concepts not provided in PCIS2, such as the ability to assign arbitrary resources (including human and computer performers) to a process. Despite the differences in these specifications, they are largely complementary and both provide important contributions to process standardization.

4.0 Related Work

The component philosophy espoused in this paper is similar in motivation to the recent work presented by Avrilionis, Belkhatir, and Cunin[2] on the Pynode project. The authors propose the construction of software process components for producing process artifacts. A "software process component" is essentially a process model fragment written in some Process Modeling Language (PML). Components are dynamically combined to construct complete process models through interface types and their respective "connectors ports". The authors correctly motivate the need to eliminate monolithic process systems and instead provide reuse and integration capabilities for process representations. However, the approach lacks adherence to foundational concepts, such as those used in OPC (see Section 2.1). The three-tier layering of the OPC framework provides a structured, extendable way to develop interoperable and reusable process-oriented components. Despite their differences, the Pynode component approach is similar in philosophy and motivation to the OPC framework, and appears to be at roughly the same level of maturity. Results of these two experiments will be very useful to the software process modeling community.

A more methodological object-oriented approach to process modeling is taken by Shams-Aliee and Warboys[14]. The authors view the object space and the process space at different levels. The object space is data-oriented, whereas the process space emphasizes process as a set of state transformations. The authors go on to argue for a layer that brings together the object level and the process level together. Shams-Aliee and Warboys[14] also advocate modeling a process as a collection of objects or components. However, we find the distinction between the object level and the process level unnecessary. In particular, we do not agree that the object level is a data-oriented model. In the OPC framework, we view process entities themselves as objects, and are concerned with the behaviors of these objects as defined by their interfaces. OPC merges objects and processes into components through an explicit representation of process state contained in the component. We propose a full object-oriented framework that includes class definitions, inheritance, and rules for component interaction. This merging of objects and processes into a complete component-based model allows OPC the full potential to achieve interoperability and reuse by being independent of any process modeling formalism.

5.0 Summary and Future Work

In this paper we have presented a component-based framework for constructing interoperable and reusable software processes. This framework identifies common concepts in the research community and defines an object-oriented framework for applying these concepts. This framework is currently employed in the construction of a software architecture for support distributed software development.

This approach, together with related efforts in the field of workflow, makes the important contribution that the software process automation field is maturing to the point that efforts such as the one described herein can be attempted. Despite whether the reader agrees with the design of this framework, providing interoperability and reusability will overcome one of the serious hurdles preventing wide scale deployment of software process automation technology.

6.0 References

[1.] Arbaoui, S., Mouline, S., Oquendo, F., and Tassart, G. PEACE: Describing and Managing Evolving Knowledge in Software Process. Proc. of EWSPT ‘92, Trondheim, Norway. September, 1992.

[2.] Avrilionis, D., Belkhatir, N., and Cunin, P. A Unified Framework for Software Process Enactment and Improvement. Proc. of the 4th International Conference on the Software Process (ICSP4). December, 1996.

[3.] Ben-Shaul, I. and Kaiser, G. An Interoperability Model for Process-Centered Software Engineering Environments and its Implementation in Oz. Technical Report CUCS-034-95, Computer Science Department, Columbia University. 1995.

[4.] Christie, A., Levine, L., Morris, E., Zubrow, D., Belton, T., Proctor, L., Cordelle, D., Ferotin, J. A Study into the Current Usage of Software Process Automation. Proc. of the NSF Workshop on Workflow and Process Automation in Information Systems, Athens, GA, May, 1996.

[5.] Conradi, R. Fernstrom, C., Fuggetta, A. and Snowdon, R. Towards a Reference Framework for Process Concepts. Proc. of EWSPT’92, pp. 3-17, Trondheim, Norway. September 1992.

[6.] Conradi, R., et. al. Object-Oriented and Cooperative Process Modelling in EPOS. In Software Process Modeling and Technology, A. Finklestein, J. Kramer, and B.Nuseibeh (Eds.), pp. 33-70. John Wiley. 1994.

[7.] Derniame, J.C. Life Cycle Process Support in PCIS. Proc. of the PCTE ‘94 Conference. 1994.

[8.] Derniame, J.C., and Gruhn, V. Development of Process-Centered IPSEs in the ALF Project. Journal of Systems Integration, vol. 4, pp. 127-150. 1994.

[9.] Emmerich, W. and Gruhn, V. FUNSOFT nets: A Petri-net Based Software Process Modeling Language. Proc. of the 6th International Workshop on Software Specification and Design, Como, Italy. September 1991.

[10.] Melo, W.L. and Belkhatir, N. TEMPO: A Support for the Modeling of Objects with Dynamic Behavior. In A. Verbraeck, H.G. Sol, and P.W.G. Bots (Eds) *Dynamic Modeling and Information Systems*. Elsevier Science Publishers, North Holland. 1994.

[11.] Object Management Group. jFlow Joint Submission. OMG Document Number bom/98-06-07. July 4, 1998.

[12.] Osterweil, L. Software Processes are Software Too. Proc. of the 9th International Conference on Software Engineering, Monterey, CA. IEEE Computer Society Press. 1987.

[13.] Peuschel, B. and Schafer, W. Concepts and Implementation of a Rule-based Process Engine. Proceedings of the 14th International Conference on Software Engineering, pp. 262-279. May, 1992.

[14.] Shams-Aliee F., and Warboys, B. Applying Object-Oriented Modelling to Support Process Technology. Proceedings of the First World Conference on Design and Process Technology (IDPT-Vol.1). Ertag, A. et al (ed.). Society for Design and Process Science, Austin, TX. December 1995.

[15.] Sutton, S., Heimbigner, D., and Osterweil, L. Language Constructs for Managing Change in Process-centered Environments. Proc. of the 4th ACM SIGSOFT Symposium on Software Development Environments, Irvine, CA. December 1990.

[16.] Workflow Management Coalition. The Reference Model. WfMC Document Number TC00-1003, January 1995.