Cooperating Process Components

Kevin A. Gary
Electrical Engineering and Computer Science Department
The Catholic University of America
Washington, D.C. 20064
garyk@cua.edu

Timothy E. Lindquist
Computer Science and Engineering Department
Arizona State University
Tempe, AZ 85287-5406
lindquist@asu.edu

Abstract

Modern automated process support systems can be considered monolithic in three ways. First, they model processes top-down, usually from a single perspective, such as the organizational perspective. Second, they are process-centered, in that they often require extensive knowledge of the users, data types, and applications in their environment. Third, they tie process implementation to a specific representation, making reuse and interoperability between process models difficult to achieve. This paper describes the application of component-based techniques to process modeling across overlapping process spaces. This approach, the Open Process Components\(^1\) approach, encapsulates process fragments as interoperable and reusable process components. This paper motivates a vision of cooperating components for automated process support, presents an overview of the Open Process Components approach, and shows the application of this technique to Humphrey’s Personal Software Process.

1. Introduction

Automated process support is the application of computer technology to assist and automate work on behalf of users. This includes applications of automated workflow\(^{[15]}\) and automated software process support\(^{[9]}\). This paper proposes a component-based framework for supporting process activities: building process models, executing process models, analyzing them, and evolving them. The proposed framework does not solve all of these problems in and of itself, but instead provides a platform by which automated process support issues may be addressed in further depth. This research is part of a larger vision of how processes are constructed, executed, monitored, and analyzed. This is just the first step, providing an infrastructure for realizing this vision by considering trends in software engineering that will effect process technology in the foreseeable future.

The recent boom in technologies such as the Internet, Java, Visual Basic, and CORBA has given rise to new buzzphrases like “reusable software component”, “distributed object middleware”, “n-tier architecture”, and “ubiquitous client”. The implications (once you get past the hype) these technologies hold for the future of software development and deployment includes “plug and play” software, “rent per use” software and “open marketplaces for distributed components”. Current approaches to automated process support do not lend themselves to these trends. Automated process support systems are large, monolithic creatures that often require extensive knowledge of the users, tools, and data types in their environment. The systems are usually tied to some underlying representation of process, and in that representation exist dependencies that make componentization of the process models themselves difficult. The research community has begun to recognize and propose solutions to these problems. It is sufficient to state that current automated process support systems are not the open systems that characterize prevailing trends in software development.

Open Process Components (\textit{OPC}) is a first step toward realizing a component-based environment for auto-

\(^{1}\) This work was supported in part by the Office of Naval Research under award number N00014-97-1-0872.
mated process support. It is an exercise in modularization and object-orientation applied in the extreme to process. OPC modularizes process models in order to define boundaries between process fragments. It enumerates objects in the process domain in order to encapsulate information and behaviors in the process space. It “componentizes” processes by capturing collections of objects that act together as a cohesive entity, and interact with other such collections in well-defined ways. By componentizing process, the environment in which the process acts is shielded from the complexity of specific process representations. Dependencies between processes are not eliminated, but moved to the boundaries between components where they are easily recognized and managed. In a component-based environment, there is hope ideas such as “plug and play processes”, “rent per use processes”, and “open marketplaces for distributed process components” can be realized. The Open Process Components vision is of an environment of distributed process components, where components can be located, adapted, and optimized on demand with respect to a set of process requirements.

This paper motivates the need for reusable, interoperable process support that spans multiple process spaces, and suggests a solution based on component-based process support. Section 2 and Section 3 provide background and motivation for a new approach to process automation. Section 3 overviews the component-based approach, suggesting that this technique is useful for realizing the vision discussed in Section 3. Section 4 demonstrates this approach on the Personal Software Process, and Section 5 provides some conclusions.

2. Background

Research in providing distributed, interoperable process support focuses on (1) describing low-level infrastructure support for distribution, such as distributed architectures[1][8][13][14] and distributed transactions[1], or (2) distributed process control integration protocols[1][11][13][17], or (3) distributed process data interchange protocols, usually in the form of a common language for translation[7][16]. In the context of current research, OPC contributes new ways of performing control and data integration through component-based design. The OPC approach is to support top-down process decomposition and bottom up synthesis via interacting components. Control integration is achieved by viewing a process model as a collection of interacting components. Data dependencies are reduced by encapsulating components behind well-defined interfaces, eliminating the need for translating between process languages.

OPC represents a fundamental departure from traditional approaches in distributed, interoperable process support. OPC promotes a fundamental shift from monolithic process-centered environments to flexible process-enabled environments. OPC is centered around the building blocks of enactable processes: work, people, and artifacts. Participants use their own tool sets and services to do the work. Process participants accept work items in the form of components, manipulate these components, and pass them along to the next participant or activate subsequent components. Strict adherence to prescriptive process definitions and tight integration of tools is not required. From the user’s perspective, work items, or process components, are received from any number of sources and integrated into the user’s personal work space.

3. Cooperating Components

In order to understand component-based process support, one must understand what it means to be a process component, and how the traditional activities of process modeling and enactment are performed in a component-based environment. OPC offers the following definitions:

- **Process Component** - a process component is an encapsulation of process information and behaviors at a given level of granularity.
- **Component-based Process Model** - a component-based process model is a collection of process components that interact in meaningful ways. A component-based process model is itself a process component.
- **Component-based Process Enactment** - component-based process enactment refers to the creation of a component, the dynamic evolution of the component, its interaction with other process components, and its interaction with users and an environment in which the component is situated.

These definitions are elaborated on in this section.

3.1 Process Components

A process component is an encapsulation of process information and behaviors at a given level of granularity. The process information is expressed in whatever formalism or language is used to represent the semantics of the component. OPC separates this information three ways: process schema, process states and transitions, and process implementation (see Figure 1). The process schema defines a vocabulary for the component, identifying entities and relationships that have meaning for a particular component. OPC defines a minimal, extendable, common schema for components. Process state and transitions
between states are represented as a finite-state machine. The set of process states and valid transitions are defined per component, with a common set of meta-states identified within OPC. The process implementation is the underlying representation of the component and the machinery which interprets this representation. This may be a petri net, a process programming language, or whatever other means are used to represent process semantics. It may also be a process tool that is wrapped to provide the process implementation. The implementation of a component is encapsulated in the component, so that components interact without requiring homogeneous representations.

A process component provides two additional abstractions. One is a process type, which is a means of categorizing processes. The schema, states and transitions, and implementation define the object implementation of a component, but they do not determine the type of process a particular component represents. Two components with the same exact schema, states and transitions, and implementation may represent two distinct processes, for example, a peer review process and a software development process. Likewise, two components with different schemas, states and transitions, and underlying implementations may both be software inspection processes. In OPC, a process type is defined for each component that indicates what type of process the component represents.

The second additional abstraction is meta-views. Meta-views define the allowable ways in which meta-roles expect to interact with process components. A meta-role identifies how the component is currently being used; for definition, enactment, ownership, or observation. A view is provided for each of the meta-roles that defines the way a given meta-role interacts with the component. OPC defines the four meta-roles just listed and their corresponding views. Additional meta-roles and views can be defined on a per component basis.

### 3.2 Component-based Process Models

A component-based process model is a collection of process components that interact in meaningful ways. OPC provides mechanisms for component interaction through well-defined interfaces, process events, and process messages. These mechanisms define a common contract by which all components abide. However, in order to prevent from prescribing a contract that is too rigid, these mechanisms are extendable and open.

A process model realized as a set of cooperating components is a step toward an environment of interoperable, reusable process fragments. Components are combined in an aggregation tree to support various levels of process granularity (see Figure 2). A component-based process model is itself a process component. OPC supports both top-down modeling and bottom-up reuse. A top-down process tree is constructed by decomposing process components at coarser levels to finer-grained process parts. Bottom-up reuse is supported by realizing a relation between two or more process parts as components. A subprocess component serves as part of the implementation of its parent process component. In terms of process granularity, a process component captures an abstraction of the process space at a particular level. The fact that the component may be composed of subprocess components is not exposed to the outside world.

### 3.3 Component-based Enactment

Component-based process enactment refers to component creation, dynamic component evolution, component interactions with other process components and with its environment (including end users). A process component in OPC is an active, “living” entity. It encapsulates its own process schema, state, and implementation, and allows dynamic modification of itself based on its own implementation. For example, a process component may extend its schema instance during enactment, or a compo-

![FIGURE 1. A process component](image1.png)

![FIGURE 2. A Process Component Tree](image2.png)
component may dynamically change its set of process states and transitions. Dynamic component modifications are managed by the component itself, reducing reliance on other components or global process information.

In OPC’s component-based approach, there is not a strict separation between process model and process instance. In that a component defines its own schema, states and transitions, and implementation, it defines its own process fragment. Process components evolve as they are enacted, activating subprocess components and instantiating itself as binding information becomes available. Hence each process component runs the gamut from creation to instantiation[2] during its active lifetime.

Process components interact with other components, users, and an environment during enactment. Interactions between components are provided through well-defined interfaces, events, and messages. Components also define interactions based on their relationship in the process tree. For example, in OPC a subprocess component notifies its parent process component when it changes state through a process event. Parent processes request subprocesses to perform actions through process messages.

Process components interact with users and the external environment through meta-views. As shown in Figure 3, a user interacts with a component by assuming a meta-role. The user is part of an environment that defines a context in which the component is activated. The environment provides the component with situation-specific information the component requires when interacted with under a given meta-view. For example, tool invocation may be required during process component enactment. The process component may determine what type of tool to invoke and when to invoke it, but asks the environment for a specific tool to bind to at the given time.

Meta-views and environments are the bridge between abstract process components and an architecture for distributing and interacting with real components. A process component defines the boundaries of process information, while meta-views define ways in which components interact with the external world of users, tools, and middleware platforms. In the sense that components interact with each other in meaningful ways that define a contract between components, meta-roles and meta-views define a contract between process components and the outside world. A component-supporting architecture provides the services components require but cannot provide for themselves. For example, components require support for registering interest in, and receiving notification of, categories of events. Components also require a repository for locating, naming, and storing process components. While these can largely be considered implementation issues, they do force one to reconsider the traditional view of process support architectures like the Workflow Reference Model [15] in the context of component-based process support. The OPC framework provides these types of services in support of distributed process components.

This section has provided a brief overview of the general approach of OPC. Details of the framework support for these concepts is beyond the scope of this paper. Further information may be found in [3].

4. Example: The Personal Software Process

The Personal Software Process (PSP)[4][6], developed by Watts Humphrey at the Software Engineering Institute (SEI), is the application of a disciplined software process at the individual level in software engineering. The PSP is an entire discipline that structures the way individuals develop software, monitor their work, and improve their methods of performing tasks. The PSP includes defined processes for guiding how developers perform the PSP. This section introduces the PSP, and applies the component-based approach to produce an implementation of the PSP in OPC.

4.1 PSP Specification

Although the PSP discusses general process improvement principles for any individual process domain, it is the Personal Software Process, and as such recommends practices specific to software process development. These practices include a process script for guiding the sequence of individual activities.

The PSP introduces a fair amount of overhead into the daily activities of software developers by adding estimating and recording activities. “Overhead” does not imply these activities are not worth doing; they are precisely the activities which provide the value-added benefits of the PSP. However, they are activities that software developers do not usually perform on a daily basis. Humphrey...
acknowledges the extra time requirements and tediousness of these activities, but argues that in the end performing these activities actually saves time and increases quality.

4.2 Automating the PSP

The time consuming and tedious tasks the PSP adds to a personal software development process are ideal candidates for process automation. Humphrey realized as much and created a specification for automating the PSP [5]. The specification is quite large and detailed, specifying a dataflow between forms presented in [6]. The implementation described here is based on the process scripts specified in [4]. Process automation can reduce the time spent on these tasks by automatically recording the various kinds of tracking data the PSP requires. More importantly, automated process support can ensure that the individual consistently and faithfully follows the PSP. This section presents the implementation of the PSP as a set of process components based on the process script presented in [4]. The PSP is modeled as a set of cooperating components in OPC that form a process tree as shown in Figure 4.

The PSP specifies a number of forms for collecting process-related data during process enactment. These forms are associated with PSP activities tracking time and defects, and provide input for the PSP Project Plan Summary. PSP enactment also involves the usual products of a software development process such as requirements documents, design documents, and source code. The OPC implementation of the PSP supports these products as part of the schema of the components in Figure 4.

A component-supporting environment for OPC has been implemented in Java. This environment includes mechanisms for component creation, modification, extension, enactment, monitoring, reuse, and interoperability. The components in Figure 5 are defined within this environment to provide PSP support. The PSP process tree in the OPC environment is shown in Figure 5.

FIGURE 4. PSP process tree

FIGURE 5. PSP process tree in OPC
5. Summary

Open Process Components provides a framework for component-based support for automated software processes. Component-based process support addresses some of the restrictive characteristics of existing process environments. It allows processes to be viewed as dynamically interacting components that can be reused across process models. These components are also interoperable in the sense they allow process fragments created in heterogeneous process tools to interact with each other without requiring a translation from one process representation to another. The result is an open, process-enabled environment as opposed to a closed, monolithic one.

This paper has presented the vision Open Process Components, provided an overview of the foundation of the approach, and described an implementation of components for the Personal Software Process. OPC has been implemented in Java, providing capabilities for creating dynamic, reusable process components. Our plan to use this environment to implement repositories of components for a variety of software processes. For example, in addition to the PSP, the ISPW-6 Software Process Scenario has also been implemented in OPC [3].

The Open Process Components framework is the first step toward a vision of distributed, dynamic process components. Powerful new functionality, such as dynamic lookup of components (brokering) and dynamic scheduling of components can be investigated using this framework. Research is already under way on applying brokering capabilities for process components [12].

6. References


