Towards Distributed and Composable Process Components

By
Timothy Lindquist
Arizona State University
Lindquist@asu.edu

and
Jean-Claude Derniame
CRIN
Derniame@loria.fr

May 1997

1.0 Motivation

In this proposal, we describe a component approach to process support for distributed team-based software development, we characterize what is meant by process components and we discuss the opportunities of process components in terms of further development. The component approach to process support [4] reacts to two changes in software development. First, increasingly software systems are developed with distributed teams and are developed by multiple organizations - such as a prime contractor with subcontractors[1], [9],[10] Second, the move to component based tools is enabling deeper tool integration but at the same time removing process constraints.

Distributed Software Development. Participants in a software group are more often geographically distributed relying on the services of the internet to assist in communication. Distribution ranges from telecommuting to virtual software development teams, in which participants migrate from project to project and even from organization to organization. Distributed participants should not be required to migrate from tool-set to tool-set to be productive. Distributed projects are more likely to involve heterogeneous platforms and tool-sets. This brings about the problem of interoperability of products. If a designer uses Rational Rose and Rose is not on the design reviewer’s desktop, then how can the review be performed?

Distributed teams accent the problems associated with activity coordination and information flow in a software development. Team coordination is already a problem with co-located development teams. These problems are only magnified with distributed development.

Component Based Tools. Tool integration and interoperability are becoming more prevalent thanks to approaches that can best be described as “Tool-components”. Componentized tools open the opportunity for integration, reuse and better cooperation. While components give us the added advantages of integration, tool componentization presents an
interesting opportunity. Tool vendors have always sold pre-packaged (statically packaged) compositions of functionality. As such, tools embody mechanisms that largely constrain or bias the process governing their use. For example, the closed nature of the persistent representation of a design document makes it very difficult to support use in a process requiring fine-grained relationships between design components, source code and requirements.

With componentized tools, the bias toward process is replaced with no constraints whatsoever. Componentization brings about a positive aspect of integration, but results in composed functionality that imposes no or little process constraint. The opportunity then is to provide some form of process infrastructure, without which loosely composed tool components open the door to less constrained processes. A good example of process-less infrastructures is JavaBeans. JavaBeans may evolve to be an excellent mechanism for coordinating and composing tool components, but without including a process infrastructure JavaBeans encourages a travesty of component compositions when considered from a process perspective.

**Process Components.** Process components respond to these needs by providing process composition and interoperation infrastructure that allows applying tool components to define software processes. We envision an infrastructure accomplishing for process what JavaBeans does for tool components. Depending on organizational needs, processes can be modeled, modified, reused and enacted. Activities can be process sensitive or can be derived from a void process definition.

Process interoperability is accommodated by a general model so that process fragments (whose underlying notation and engine may differ) can be composed. Process components holds the potential to provide the same form of integration among process notations that OMG’s CORBA [8] enables among clients and servers written in different programming languages. Where OMG uses IDL to accomplish this, process components uses a general meta-model.

What are the basic manipulations defined for process components?

- Modeling a process according to the meta-model,
- Identifying a process component by clustering and applying changes to an existing process or component definition,
- Specifying composition - defining how a process component interacts with related process components,
- Completing a process component by specifying the necessary bindings that allow it to be enacted,
- Enacting a process component to carry out activities.

### 2.0 Defining Process Components

Processes can be modeled from a set of basic abstractions relying on the PCIS LCPS model [3], the Workflow Management Coalition reference model [11] and others [2].
use the model, shown in Figure 1 as a basis for identifying and clustering elements of a process that can be reused and that can be defined to interact with other components.

Conceptually, we consider the entities in Figure 1 as abstract classes in an object-oriented sense. Each entity has a set of properties and methods. Each includes definition of a pre-defined default behavior. The abstract classes are realized with a specific approach to process, such as event-condition-action, or Petri-based. The realization extends the abstract to accommodate a specific approach to process modeling. An organization creates process models by specializing the realized classes. The process to define a process fragment is described by a process model and is governed by a process definition right. Associated with activity states, it allows us to build evolution support [6].

Thus, to create a new process model, for say a review process, we may conceptually specialize the process entity of Figure 1 to create a review process entity. The review process inherits all the methods and relationships of its parent. The specific properties of review can be created dynamically, as modeling of a review continues. Process components are subsequently identified to structure process models for reuse and enactment.

3.0 Derived Process Components

To get to the point of enactment, there are manipulations that can be performed to describe a process and its components. Manipulations may include addition of product types, association with directions or definition of roles, for example. Products may be defined by type (for example, text document or design document) with bindings to specific instances of the type deferred to enactment. Or, a product may be fully specified to an instance of a

FIGURE 1. Process Components Meta-Model
type at the time the process is defined (as may be the association between a role and an
agent to carry out the role). A process component’s definition can be used to define a new
process component with a similar or related functionality.

The most common scenario for creating a process component is not by specializing from a
model entity. Instead, we view reuse of existing process components to be a common way
to create components. A process component may consist of a single activity or a single
process (with all or a portion of its subprocesses and sub-activities). A component may
consist of any meaningful grouping of activities and processes. A process component
includes all the appropriate roles and products that are associated with included processes
and activities. The grouping can be detached from a model and applied by augmenting a
new set of properties (products, roles or activities) to create a derived process component.
In addition to adding new properties, a derived component can modify its properties by
further constraining existing properties. For example, a code review process component
may be derived from a review process in which a text input product is further constrained
to be source code (a specialization of text).

4.0 Interacting Tool and Process Components

We can specify connections among process components that provide for their composition
and interaction, and tool components may be associated with a process component. Tool
components provide a way to specify how products are to be manipulated by a process.
Tool components provide integration and interoperability in several ways [7], including:

- Exposing tool functionality to be used by other components through invokable meth-
  ods. For example, FrameMaker has defined an application programmers interface (API) (called the Frame Development Kit) that exposes internal functionality to other
vendors. Rather than divulge the details of proprietary internal structures, tool vendors
expose functionality for outside use through an API.

- Extending functionality. Tools can viewed as components when new functionality can
be added after delivery of the tool. For example, Visio allows third party vendors to
extend its functionality through new menu selection functionality. FrameMaker has
been extended in a similar way by Quadralay Corporation through its html generator
for Frame documents.

- Sun has defined interfaces for componentization of tools through its JavaBeans specifi-
cation. JavaBeans defines how component methods and attributes can be shared. One
component can register itself as a listener for attribute changes or as a listener for
events defined by other components. Beans also includes a mechanism whereby one
component can be notified about attempts to change attributes. Most importantly, Java-
Beans provides a framework whereby components can be easily and visually com-
posed. The BeanBox allows relationships between components to be made explicit.

Process components can interact in exactly the same ways as tool components can inter-
act. A process component has properties which describe its roles, directions and products.
A process component can be constructed in such a way that changes or other manipula-
tions that may occur with these properties can be signaled to other process components
who select to listen to these properties. In a similar way, a process component can define
events, which may rely on process relevant data, that can be listened to by cooperating
process components. In addition, process components can interact by exchanging infor-
mation on their state (as in ProcessWall [5]). States are defined from an organizational
point of view (for example, defined, active, suspended, closed) and can be used with the
right to define a component and right to control that a process component has fulfilled its
directions as a support to describe responsibilities in an organization.

5.0 Creating a Workspace

Process components are fragments that describe activities of sufficient importance to an
organization to have persistent and evolvable definitions. But, process components are
foremost intended to be enacted. This means to place them into a workspace allowing the
corresponding activities to be carried out in accomplishing work. We envision that a pro-
cess defining (and controlling) the activities and interactions of a working group is made
up of components. Each component may be enacted using a different engine and may be
described in a different notation. Nonetheless, components may interact according to cer-
tain rules. When a process component is enacted, all outstanding bindings must be com-
pleted. This means that the products, directions and agents must be bound. The bindings,
together with the component (and its associated engine) are then delivered to the appropriate
workstation. Together the bindings, activity, engine and agent’s preferences make up the
workspace for the process component.

6.0 Open Opportunities

A component approach to process, which we refer to as “Process Beans” allows heteroge-
neous fragments to be coalesced into a higher level process. There are several outstanding
issues regarding this approach.

Research is needed to identify diverse ways to describe, reuse and compose components.
It is easy to envision a visual component editor to aid in these manipulations. Defining
interactions and compositions among components is an open research question consider-
ing the components we wish to cooperate may rely on very different underlying notations
and engines. It may not be possible to define a sufficiently useful set of interactions that
are equally applicable to all different process notations. This question of process model
integration is an important related question.

Another area of research leads from the idea that process components can be described
from a level that allows them to be widely reused. If we can arrive at a representation inde-
dependent specification of a process component, then research in a process service broker
should be pursued. In the same way that we go to the Telephone Book, we should be able
describe needs to a broker for process service providers. The service broker, containing
the registration of available services, would negotiate with a provider and employe the ser-
vices on behalf of an agent.
Anther opportunity opened with process components is for the integration of individual personal processes with individual professional process (personal software process) with workgroup and organization processes. Process component integration and process beans interoperability could be of great help.

7.0 References


[7.] Nierstrasz, O., Dami, L.; Component-Oriented Software Technology in Object-Oriented Software Composition, The Object-Oriented Series, Prentice Hall, 1995.


