Process Enactment: A Foundation for Managing Knowledge Intensive Work Processes

John Noll
Computer Engineering Department
Santa Clara University
500 El Camino Real
Santa Clara, CA 95053-0566
jnoll@cse.scu.edu

Abstract
Attempts to extend process management to support dynamic, knowledge intensive activities have not been as successful as workflow for routine business processes. In part this is due to the dynamic nature of knowledge-intensive work: the tasks performed change continuously in response to the knowledge developed by those tasks. Also, knowledge work involves significant informal communications, which are difficult to capture.

We propose an approach to supporting knowledge-intensive processes that embraces these difficulties; rather than attempting to capture every nuance of individual activities, we seek to facilitate communication and coordination among knowledge workers to disseminate knowledge and process expertise throughout the organization.

1 Problem, Goals, and Transition

Goals Develop a process enactment architecture and deployment mechanism for
- Supporting the performance of knowledge-intensive work;
- Capturing essential facets of knowledge-intensive work processes;
- Facilitating knowledge transfer among members of an organization.

Transition The result of this work will be an architecture and mechanisms for a system to achieve the above goals, as well as a proof-of-concept implementation that demonstrates how the system would work in practice.

2 Approach

2.1 Introduction
The conventional process management approach employs a client-server architecture, in which a central engine executes process descriptions and stores documents produced by the processes. Process participants (actors) interact with the engine through web browsers, environments, or task-specific tools, receiving guidance on what activities to perform, and how to perform them. This approach has been successful for automating repetitive, routine processes, but attempts to extend it to support dynamic, knowledge intensive activities have not been as successful [1].

In part, this is due to the dynamic nature of such activities: actors in knowledge-intensive environments continually adapt their activities to reflect increasing understanding of the problem at hand, which understanding results from performing the knowledge-intensive activities. Thus, the performance or enactment of knowledge-intensive work processes involves a continuous cycle of planning, action, review, and refinement.
As an example of a knowledge-intensive problem domain, we consider the development of large software systems. Successful software development depends on the expertise of experienced designers and programmers, requires significant intellectual effort, and is difficult to describe in detail. In addition, modern commercial software development is subject to rapidly changing markets and short development cycles.

This presents several problems for process management:

- The activities performed in any cycle are difficult to describe in sufficient detail to be useful for conventional workflow;
- Experts perform these activities in a fluid, almost unconscious manner, rather than as discrete steps;
- The cycle is repeated rapidly and continuously, so the set of activities evolves rapidly; therefore, any description of the process is immediately out of date.

A significant amount of research has been devoted to specifying software development processes in sufficient detail to provide active process management support; so far, this has not produced the kind of productivity breakthrough seen in other domains where process management has been applied.

Therefore, rather than seeking to supplant domain experts with information systems, we propose a system targeted to facilitating communication and collaboration among knowledge workers to disseminate process expertise as widely as possible. In this approach, actors are given high-level guidance about what activities to perform, and how to perform them, but are free to carry out the details of those activities as their expertise dictates.

The approach comprises three key components:

1. A distributed process deployment and execution mechanism for enacting low fidelity process models
2. A Virtual Repository of artifacts providing access to distributed collections, repositories, and databases of information objects related to the work to be performed;
3. A data collection facility for monitoring, capturing, and replaying process enactment history, to be used for process and organizational analysis.

These components form the core of a comprehensive foundation for process management that integrates capture and modeling, analysis, monitoring, and performance measurement tools.

This approach relies on a meta-process and associated methods for identifying, capturing, and evaluating organizational processes to be deployed as low-fidelity models. This component will be realized through collaboration with Professor Walt Scacchi at the University of California, Irvine.

Dr. Scacchi has done extensive work in process discovery and redesign. His proposed research for MKIDS will provide the foundation and methods for capturing and analyzing processes. ¹

In contrast, the research proposed here provides an enactment architecture and mechanism that support process capture and refinement via process discovery, recovery, and redesign techniques. Thus, the two efforts are complementary rather than redundant.

The following sections discuss, in turn, the context for process management; enactment as an integration concept; and the specific architecture proposed to provide the solution.

### 2.2 The Process Life-cycle

Process management involves more than simply specifying processes for deployment in a workflow system. The process life-cycle comprises a number of activities supporting the engineering of effective organizational processes [2, 3]. These activities include capture and analysis of current practices, redesign and simulation, deployment and management, history capture and auditing, and refinement in a continuing cycle of process improvement (see Figure 1).

¹See Walt Scacchi, *Dynamic Process Discovery, Recovery, and Redesign*, MKIDS white paper, August 2002
**Process Capture, Modeling, and Refinement** The process life-cycle begins with examination of existing processes. This activity, called process *capture*, establishes a baseline against which to compare subsequent redesigns, and as the initial documentation of processes for dissemination.

Process capture is traditionally a human-intensive activity, involving observation and interviewing of process performers [4]. However, there is ongoing research into automatic and semi-automatic process capture, that may reduce the effort and time required to document existing processes [5].

Process *modeling* uses the data acquired during process capture to develop models of focal organizational processes; these models serve as the basis for analysis, simulation, and ultimately enactment.

Process *refinement* relies on informal and formal techniques that measure and improve the structure and performance of a modeled process.

Our previous work with process modeling demonstrates the value of *low-fidelity* models for documenting and analyzing knowledge-intensive work [3, 4, 6, 7]. A low-fidelity model does not seek to capture every detail and nuance of a knowledge-intensive process; rather, it documents the major activities of a process, and the primary sequence in which they are performed.

Modeling processes using low-fidelity models yields several benefits:

- Low-fidelity models are easy to specify, and can be generated rapidly.

- A low-fidelity model still captures the essential facets of a process, especially the resources consumed and artifacts produced by a given set of activities.

- Because they seek to represent only high-level detail, low-fidelity models are relatively stable; that is, they continue to be accurate descriptions of the high-level process, even as the details of process activities evolve in response to knowledge and experience gained with the problem.

Figure 2 shows such a fragment of such a model, specified in the PML process modeling language. This fragment describes the test portion of a software development process dealing with the design-implementation-test cycle, shown in Figure 3.

Note that, despite being a “low-fidelity” model, it still contains considerable detail about the process, including the major steps required, the resources consumed, the artifacts produced, and the actors that should perform the process.

**Analysis, Simulation, Redesign** Once existing processes have been documented, they can be analyzed in various ways to uncover inefficiencies and other flaws and construct models of the organization and its processes. This includes analysis of the sequence of process steps, flow of resources (tools and products) through the process [8], actors involved in the process, and verification of the accuracy of the description. Results of the analysis phase can include not only assessment of the process itself, but also models of the organization and domain that form the process setting.
sequence test {
    action WriteTestPlan {
        requires { analysis }
        provides { test_plan }
        agent { TestEngr }
    }
    action WriteTests {
        requires { test_plan }
        provides { test_suite }
        agent { TestEngr }
    }
    action RunTests {
        requires { test_suite && code }
        provides { code.status == tested }
        agent { TestEngr }
    }
}

Figure 2: Software Development Process Fragment.

**Deployment and Enactment** Processes can be deployed in a variety of ways. The simplest is to document the process in a handbook or manual to be distributed to process performers, possibly augmented by training to familiarize performers with the process before they are required to perform it. However, if the process description is sufficiently formal, it can be executed by an interpreter to provide guidance to process performers in real-time, termed *enactment*.

There are various enactment approaches. Workflow systems focus primarily on task guidance and document routing. They also rely on programmers to specify and code the workflow in a low-level implementation form. Process performers thus do not have access to the process model in a form that they can comprehend, tailor, and redesign. But process enactment can be made more useful than this, especially when the specification (a low fidelity process model in PML) and implementation (an interpreted run-time rendering of PML that binds user roles, tools, required and provided resources) are cast in separate, distinct representations. This is a distinguishing feature of the process enactment approach proposed here.

This Model-based process enactment approach can be used as a foundation that supports and integrates many process life cycle activities. A process enactment mechanism can support process visualization, process prototyping walkthroughs and enactment simulators, semi-automated generation of on-demand process training materials, and integrated process run-time enactments that can invoke external tools and application programs or internal data entry forms [2, 4, 6]. Model-based process enactment also provides a strategy for monitoring, capturing, and replaying the history of events and contexts that characterize how a particular process enactment instance occurs. These histories then serve as a basis for analyzing and recovering process enactment instances, as one form of adaptive and continuous process improvement. Previously, these capabilities could only be demonstrated using high-fidelity process models and a centralized process enactment server. In contrast, the proposed efforts investigates how to design an architecture and run-time mechanism that enables a global peer-to-peer, Internet-based process deployment and enactment capability, in a manner that enables the monitoring, capture and replay of enactment events, resource states, and contexts (as specified in a low-fidelity PML models). Such a capability will therefore enable a physically decentralized, but logically centralized, process enactment and deployment architecture.

### 2.3 Enactment Mechanism

Enactment is driven by events, which are classified as either process events or resource events. These are summarized in Table 1.

Process events signal task initiation and completion, and are generated directly by actors as a consequence of performing tasks. Resource events reflect changes in the environment, such as creation, deletion, and modification of resources, and time events such as deadlines or alarms. All events occur within one or more process enactment contexts, as specified in a low-fidelity process model.
An actor requests instantiation of a new process instance.

- An actor has begun a task.
- The actor has suspended an active task.
- The actor has completed a task.
- The actor aborts a task that can’t be completed.

- A new resource has been created (or detected).
- An existing resource has been changed.
- An existing resource has been destroyed or removed.
- A time event (deadline, milestone, alarm) has passed.

The actor or organization can respond to events as follows:

1. Proceed to the next task; the process description specifies the order in which tasks should be performed. A process event can trigger the transition from one task to the next, for example the completion of the Design task in Figure 3 causes transition to the Implement task.

2. Re-prioritize existing tasks to reflect new information encapsulated by the event. In some cases, this could be done automatically, based on knowledge of how similar events were handled in the past. Examples from software development include requests for minor changes or enhancements to a system’s functionality.

Other events require human input, because they either represent previously unseen circumstances, or represent such a significant change in the operating environment that major changes are required. For example, in the late 1990s many large organizations were involved in large client-server development efforts. The explosion of interest in the World Wide Web required many of these efforts to be abandoned in favor of development of Web-based systems.

3. Create a new instance of an existing process, to handle known events. An example is the Run Tests task shown above.

4. Develop a new process to handle a new event not seen before. This response requires human expertise.

5. Ignore the event; the event is recorded, but no response is deemed necessary.

The enactment mechanism is depicted in Figure 4. The User Interface allows actors to generate process events, and invoke tools to perform tasks. The Event Manager handles Process and Resource events, responding as described above. (The Event Manager is similar to the “Scheduler” shown on the MKIDS Management Schematic). The Virtual Machine interprets process descriptions to determine the appropriate next task in response to a process event. The Resource Interface (similar to the “Controller” on the Management Schematic) is responsible for detecting resource events, and translating them into a representation independent form suitable for interpretation by the Event Manager.

The Resource Interface is a key component of this mechanism. In any given organization, information objects will come from a wide variety of sources: email, data and knowledge bases, organizational intranets, the World Wide Web, etc.
The Resource Interface provides a logically centralized view of a set of distributed, heterogeneous information repositories. This enables the enactment mechanism to treat a variety of resources, such as email messages, Web pages, documents, etc. as information objects with a uniform format and access interface [9].

The enactment mechanism is organized around a peer-to-peer architecture. Each actor runs an instance of the enactment system; these instances are entirely independent. It is assumed that the resources provided and required by actors’ processes are available from a shared distributed repository, such as the World Wide Web.

An example, showing enactment of the implementation-test process discussed above, is shown in Figure 5. In this example, an analyst, a programmer, and a tester coordinate to implement and test a body of code. They share resources through a global code repository.

The arcs labeled “notify(analyst)” and “notify(code)” signify notification of a change in the analysis and code resources, events detected by the Resource Interface of the programmer’s and tester’s enactment systems.

In order to preserve the highest possible degree of autonomy and flexibility for actors, coordination of activities is achieved by synchronizing on shared resources. Activities can block until a resource (product, document, or other artifact) is produced by another activity performed by a different actor. A predicate in the process description specifies the state that the required resources must satisfy before the activity can proceed (the mechanism is described fully in [8]).

An example of how this works for the software development process depicted above is shown in Figure 3. This process is decomposed into three “fragments”: one bound to the Analyst, comprising the requirements analysis task; one bound to the Programmer, comprising design and implementation tasks, and the last bound to the Tester, involving test design and implementation.

These process fragments are synchronized through the shared “analysis” and “code” resources; as a result, the Design task doesn’t begin until the Analyst completes the Analyze task, thus making the “analysis” resource available for Design and Write Plan. Likewise, the Run Tests task cannot begin until the “code” resource has been produced by the Programmer.

Note that because the processes are indirectly coupled, it is not necessary for all activity to be modeled, or enacted; enacted processes can be coordinated with ad-hoc work or activities in another organization, through a shared resource. Thus, the Run Tests task can begin as soon as the “code” resource is available; but this resource can be produced by
any process, including a completely spontaneous ad-hoc process.

2.4 Plan

Work on this project will proceed in an iterative fashion designed to produce tangible results as early as possible. Project milestones and associated deliverables will be determined in part through project coordination and guidance from the research sponsors.

2.5 Conclusion

The approach described herein has several distinctive features.

First, since individual actors each have their own enactment system, they can bring process support to groups to which they belong, without imposing on other group members.

Second, because the enactment system assumes that resources and products are stored in an external database, it works with, rather than replaces, existing data repositories. As a result, coordination can be achieved across organizational boundaries, and between peers and ad-hoc processes, as long as the shared resources are accessible to both.

Further, because the coupling between coordinating actors is indirect, through a shared resource, there is no requirement for trust (or even awareness) between coordinating peers. This enables extremely fluid, dynamic organizations in which participants can join at will without requiring administrative approval or action.

Finally, processes can be performed off-line, and synchronized periodically when desired or possible. This is especially important for mobile actors, who often work with intermittent network connections.
3 Background and References Cited


4 Cost

Following is a preliminary budget based on a two-year project duration.

Salaries, Wages & Benefits
Principal Investigator: 2 mo. SUM x 100% $ 36378
Graduate Research Assistant: 3 mo SUM @ 100%, 9 mo AY @ 50% $ 54183
Total Salaries, Wages & Benefits $ 90561

Other Direct Costs
Travel $ 4000
Materials, fees, etc. $ 1000
Tuition Remission $ 27109
Total Direct Costs $ 122670

Overhead (HHS Negotiated Rate @ 57.6% S&W) $ 46589

Total Project Cost $ 169260